Comparsion of R packages for CPUE standardisation

Darcy Webber

12 August 2021

Contents

1 Introduction

This document is illustrates several different R packages useful for CPUE analyses. It includes examples of non-spatial, spatial, and spatio-temporal CPUE standardisation models. It will also illustrate delta models, hurdle models, and zero-inflated models when I get around to it.

A population-level effect is the Bayesian version of a fixed-effect, and a group-level effect is the Bayesian word for a random-effect.

2 Non-spatial models

Five R packages are tested using a basic CPUE model that includes `year` and `month` as fixed-effects. The dependent variable is `cpue` which in this case is the catch per tow (for tows where the catch > 0).

2.1 glm

The R function `glm` (from the `stats` package which comes with base R and is automatically loaded) is likely the most widely used. The `glm` function does not include a `lognormal` family, so a `gaussian` distribution must be used and a log transform done on the `cpue`:

```
fit1 <- glm(log(cpue) ~ year + month, data = data, family = "gaussian")
```

2.2 gam

The R function `gam` (from the `mgcv` package) is also resonably common.

```
library(mgcv)

fit2 <- gam(log(cpue) ~ year + month, data = data, family = "gaussian")
```

2.3 brms

The R function `brm` (from the `brms` package) is fully Bayesian, making use of Stans Hamiltonian Monte Carlo (HMC) MCMC algorithm. The `bmrs` package does not include a `lognormal` family making a lognormal model easy to fit:

```
library(brms)
options(mc.cores = parallel::detectCores())

fit3 <- brm(cpue ~ year + month, data = data, family = lognormal())
```

The `brms` package also provides several functions and features that makes it very easy to use. For example, extracting the coefficients by year is very easy using:

```
newdata <- data.frame(year = sort(unique(data$year)), month = NA)
year_coefs <- fitted(fit3, newdata = newdata)
```

Notice the `month = NA` declaration in the `newdata` data frame that is being passed to the `fitted` function. This is unique to `brms` and allows average predictions to be made each fishing year, with uncertainty, and no need for any further processing of the coeficients to derive a CPUE series. The posterior predictive distribution by year can also be computed easily using:

```
year_pp <- predict(fit3, newdata = newdata)
```

The `brms` package includes many other families above and beyond the standard families. For example `student`, `bernoulli`, `weibull`, `wiener`, `Beta`, `dirichlet` and many more. Several hurdle models are easily coded using `hurdle_poisson`, `hurdle_negbinomial`, `huedle_gamma`, `hurdle_lognormal`. And several zero inflated models `zero_inflated_beta`, `zero_inflated_poisson`, `zero_inflated_negbinomial`, `zero_inflated_binomial`. And if that is not enough, then custom families can be defined by writing your own distribution using Stan code.

What's more, several packages have been developed to help with model comparison and model averaging (i.e. using the `loo` package), plotting diagnostics (the `rstan` package), and model fits (the `bayesplot` pacakge):

```
library(rstan)

fit3 <- add_criterion(fit3, criterion = c("loo", "waic"))
stan_trace(object = fit3, pars = "month")
pp_check(object = fit3)
```

Speed is an issue with brms. Small models with just a few population-level or group-level effects and few data points run very quickly. However, big data sets or complex models with splines become very slow. The software includes spatial modelling features such as CAR models, but these are also very slow.

2.4 INLA

INLA is also Bayesian (using the Laplace approximation). INLA also includes the `lognormal` family:

```
library(INLA)

fit4 <- inla(cpue ~ year + month,
             data = data,
             family = "lognormal",
             control.compute = list(config = TRUE))
```

INLA seems to have a limit on what it can do though as it flops with large data sets. Spatial models are relatively fast using the SPDE. Data manipulation becomes complex very quickly with INLA. But it is worth the learning curve for spatial modelling.

2.5 glmmTMB

The glmmTMB package is also Bayesian (using the Laplace approximation).

```
library(glmmTMB)

fit5 <- glmmTMB(log(cpue) ~ year + month,
                data = edw_pos,
                family = "gaussian")
```

Seems like it has some spatial modelling features.

2.6 tmbstan

Need to add this example.

```
library(tmbstan)

# fit6 <- tmbstan(log(cpue) ~ year + month, data = edw_pos, family = "gaussian")
```

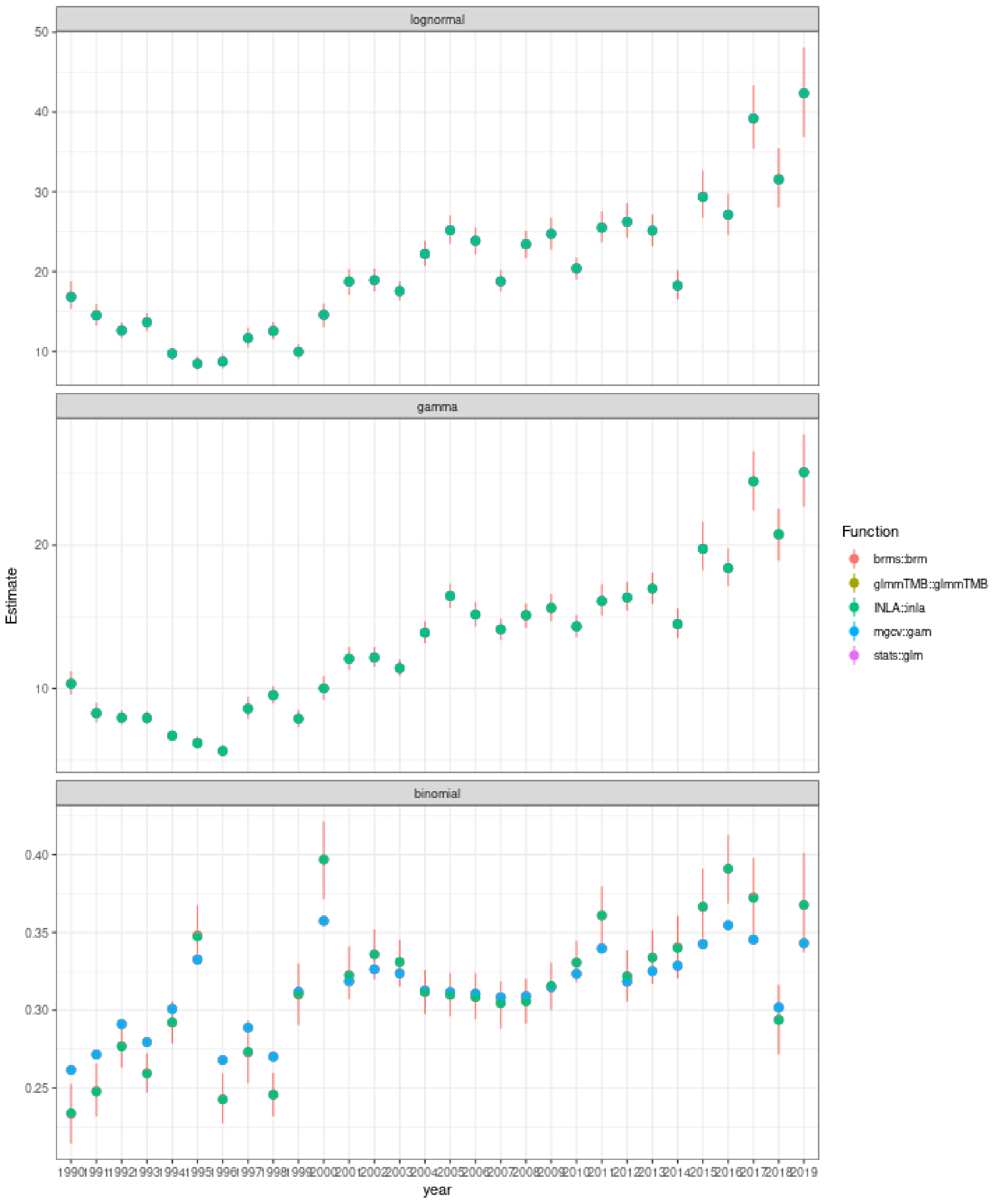


Figure 1: Comparison of different packages in R that can be used for CPUE analysis for three distributions. The lognormal and Gamma distributions use the log link and the binomial distribution uses the logit link. Uncertainty is provided for the brms model runs only, all series are scaled to the geometric mean of the brms model runs. A simple model that included fishing year and month was done. The binomial INLA model used a reduced data set (70% of the total data set) because the original data set was too big.

3 Spatial models

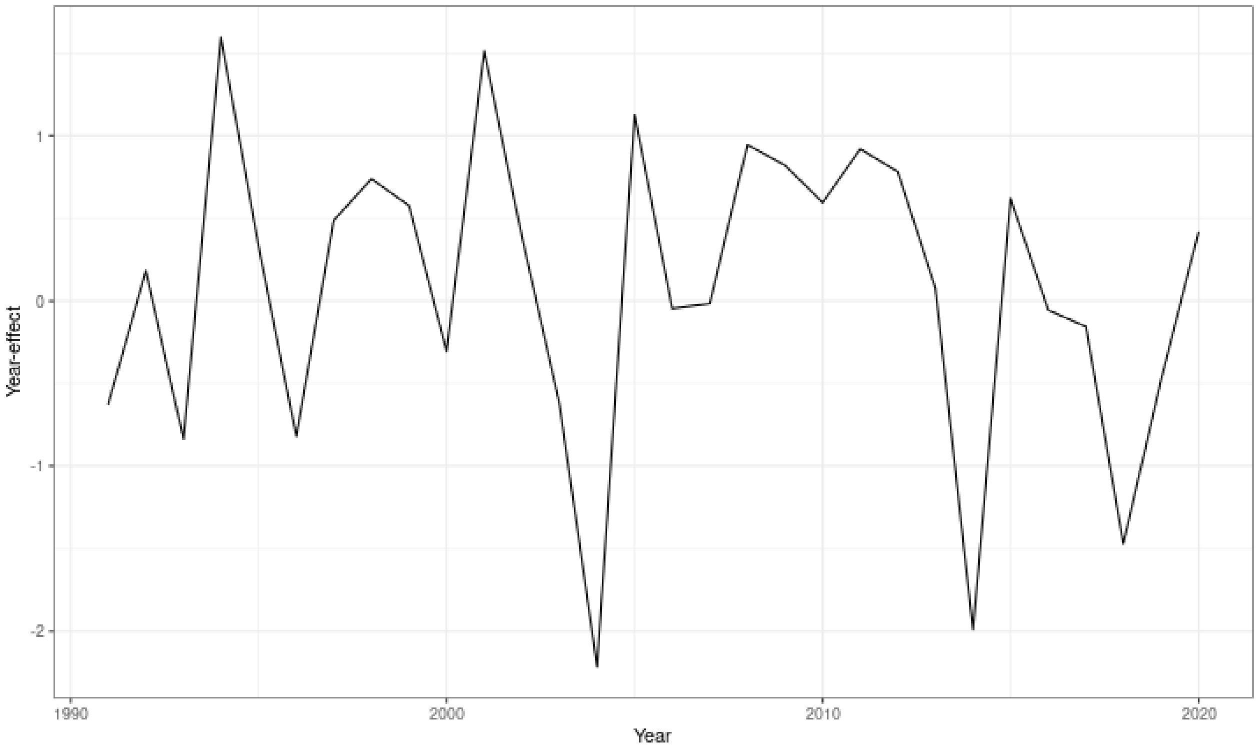
Now I test some spatial models (i.e. a spatial effect that is constant through time). First I create some fake data using the `volcano` data set in R. In this first example the spatial effect is the same each year, in other words the volcano does not change from year to year. The simulated data is

$$\log(z_{yi}) = \exp\left(\beta_y\right) + s(x,y) + \varepsilon_i$$

$$\varepsilon_i \sim N(0, 0.01)$$

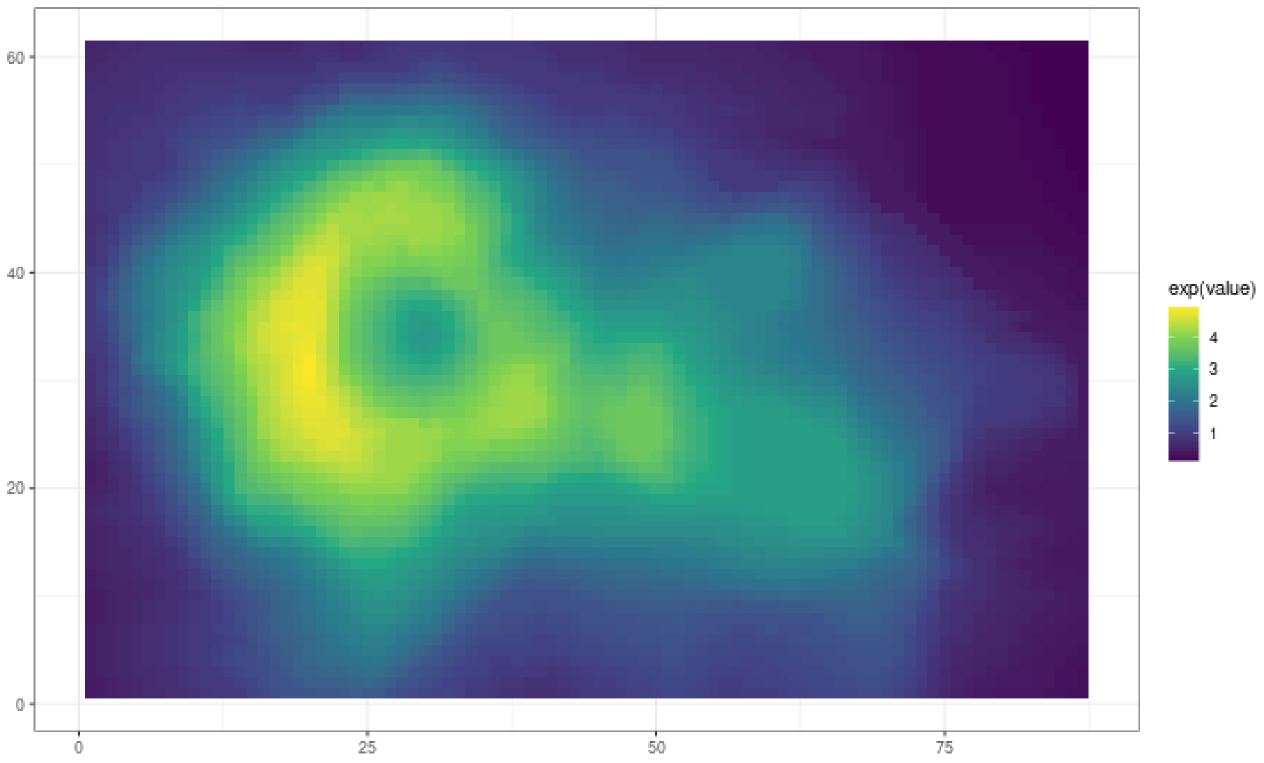
where z_{yi} is an observation of the CPUE for each fising event i each year y, and $s(x,y)$ is the volcano.

```r
library(reshape2)
library(tidyverse)
library(scales)
set.seed(1)

data(volcano)

# Create a year effect
year <- 1991:2020
n <- length(year)
year_eff <- rnorm(n = n, mean = 0, sd = 1)

ggplot(data = data.frame(year, year_eff)) +
  geom_line(aes(x = year, y = year_eff)) +
  theme_bw() +
  labs(x = "Year", y = "Year-effect")
```



```r
# Create a spatial effect using the volcano data set but rescale it to a similar scale
# to the year effect
volcano1 <- log(rescale(volcano, to = range(exp(year_eff))))
```

```r
volcano_df <- melt(volcano1) %>% mutate(Type = "Original")

ggplot(data = volcano_df) +
  geom_tile(aes(x = Var1, y = Var2, fill = exp(value))) +
  scale_fill_viridis_c() +
  theme_bw() +
  labs(x = NULL, y = NULL)
```



```r
data <- NULL
for (i in 1:n) {
  m <- round(runif(n = 1, min = 100, max = 2000), 0) # take a random number of samples (fishing events)
  err <- rnorm(n = length(volcano), 0, 0.01)
  d1 <- melt(volcano1) %>%
    rename(x = Var1, y = Var2) %>%
    mutate(yr = year_eff[i], year = factor(year[i]), err = err) %>%
    filter(ifelse(year %in% 1991:2000, x < 44, x >= 0)) %>%
    filter(ifelse(year %in% 2007:2015, y < 20, y >= 0)) %>%
    sample_n(m, replace = TRUE) %>%
    mutate(cpue = exp(yr + value + err))
  data <- rbind(data, d1)
}

head(data)
```

```
##    x  y       value         yr year           err      cpue
## 1  3 24 -0.31519013 -0.6264538 1991 -0.0068190959 0.3873359
## 2 35 17  0.98823837 -0.6264538 1991 -0.0096340516 1.4221225
## 3 37 58 -0.03222357 -0.6264538 1991 -0.0057311184 0.5145778
## 4 21 55  0.46295002 -0.6264538 1991  0.0032330873 0.8519131
## 5 33 32  1.18149206 -0.6264538 1991  0.0005096499 1.7428957
## 6  7 56 -0.19224328 -0.6264538 1991 -0.0024363807 0.4399327
```

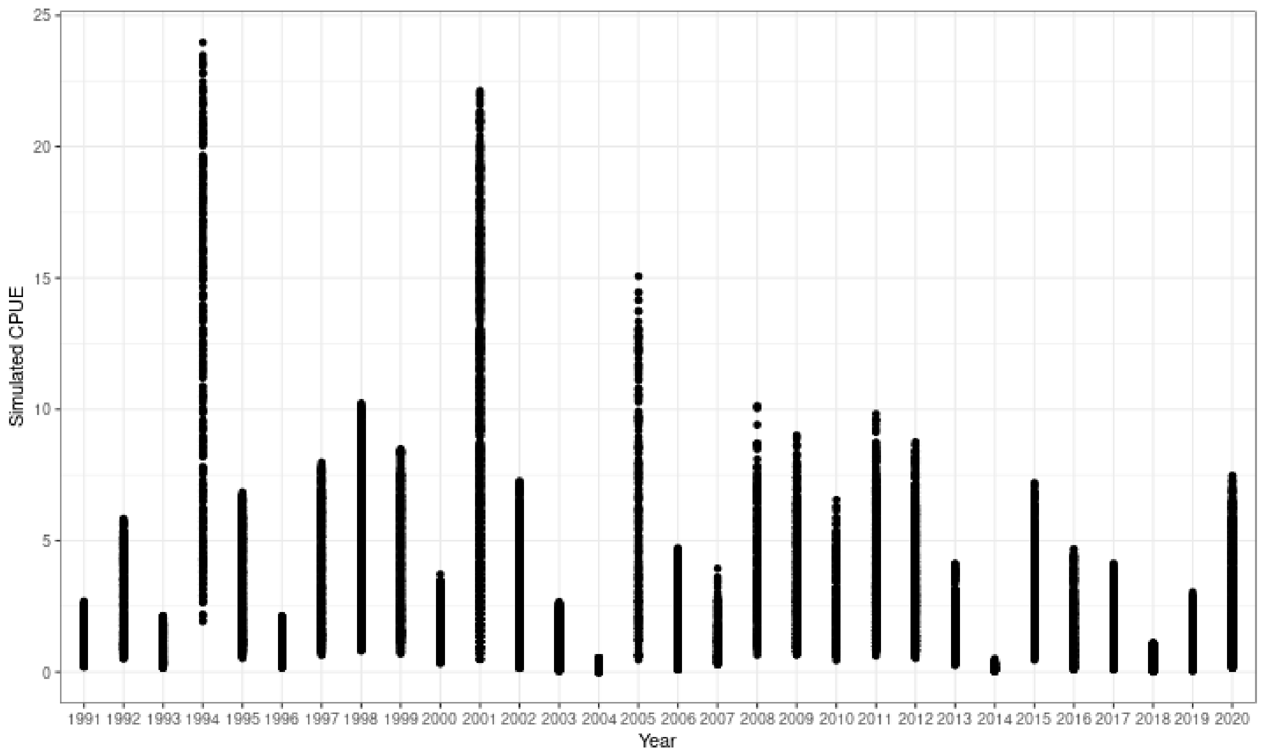
```
glimpse(data)
```

```
## Rows: 25,091
## Columns: 7
## $ x     <int> 3, 35, 37, 21, 33, 7, 25, 35, 21, 8, 23, 7, 32, 16, 25, 22, 28, ~
## $ y     <int> 24, 17, 58, 55, 32, 56, 25, 28, 34, 15, 24, 34, 6, 37, 55, 59, 5~
## $ value <dbl> -0.31519013, 0.98823837, -0.03222357, 0.46295002, 1.18149206, -0~
## $ yr    <dbl> -0.6264538, -0.6264538, -0.6264538, -0.6264538, -0.6264538, -0.6~
## $ year  <fct> 1991, 1991, 1991, 1991, 1991, 1991, 1991, 1991, 1991, 1991, 1991~
## $ err   <dbl> -0.0068190959, -0.0096340516, -0.0057311184, 0.0032330873, 0.000~
## $ cpue  <dbl> 0.3873359, 1.4221225, 0.5145778, 0.8519131, 1.7428957, 0.4399327~
```

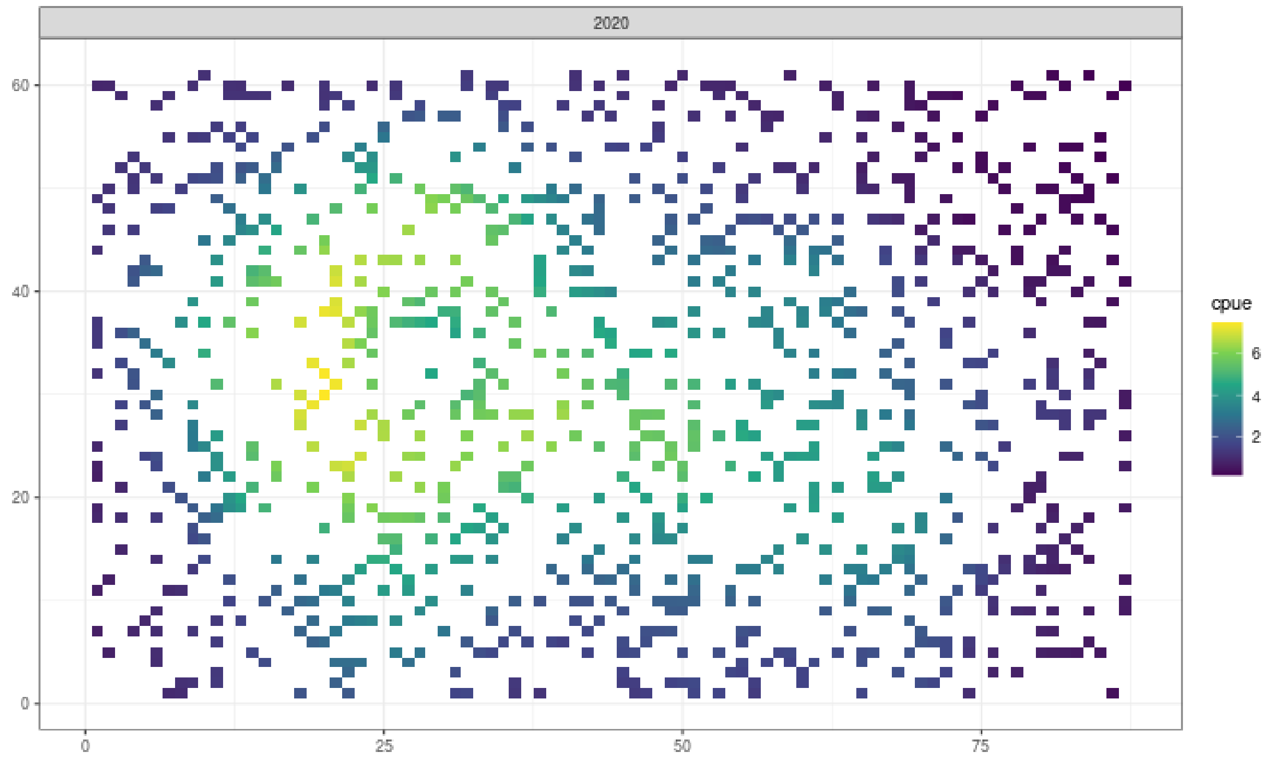
```
nrow(data)
```

```
## [1] 25091
```

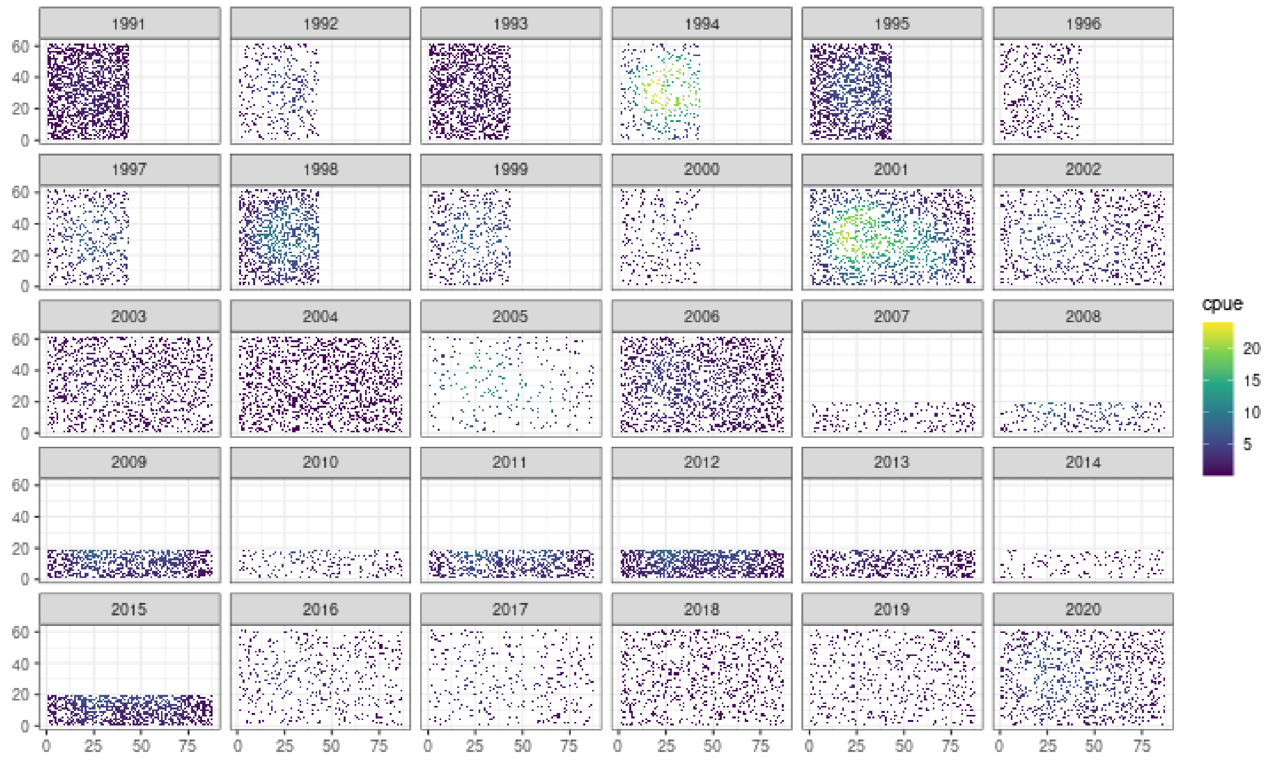
```
ggplot(data, aes(x = year, y = cpue)) +
  geom_point() +
  theme_bw() +
  labs(x = "Year", y = "Simulated CPUE")
```



```
ggplot(data = data %>% filter(year == 2020)) +
  geom_tile(aes(x = x, y = y, fill = cpue)) +
  facet_wrap(~year) +
  scale_fill_viridis_c() +
  theme_bw() +
  labs(x = NULL, y = NULL)
```



```
ggplot(data = data) +
  geom_tile(aes(x = x, y = y, fill = cpue)) +
  facet_wrap(~year) +
  scale_fill_viridis_c() +
  theme_bw() +
  labs(x = NULL, y = NULL)
```



```r
library(INLA)
library(reshape2)
library(sf)

loc <- data %>% select(x, y) %>% as.matrix()

mesh <- inla.mesh.2d(
  loc = loc,
  cutoff = 2,
  max.n.strict = 700)
mesh$n
```

```
## [1] 802
```

```r
plot(mesh)
points(loc, col = 2)
```

```r
spde <- inla.spde2.pcmatern(mesh = mesh, prior.range = c(0.5, 0.01), prior.sigma = c(1, 0.01))

A <- inla.spde.make.A(mesh = mesh, loc = loc)

stack <- inla.stack(
  tag = "space1",
  data = list(cpue = data$cpue),
  A = list(A, 1),
  effects = list(s = 1:spde$n.spde,
                 data.frame(year = data$year)))

if (do_estimate) {
  space1 <- inla(cpue ~ -1 + year + f(s, model = spde),
                 data = inla.stack.data(stack),
                 family = "lognormal",
                 control.predictor = list(A = inla.stack.A(stack)))

  blind1 <- inla(cpue ~ year,
                 data = data,
                 family = "lognormal")

  save(space1, blind1, data, loc, mesh, A, spde, stack, file = "space1.rda")
} else {
  load("space1.rda")
}

field <- space1$summary.random[['s']][['mean']]

xlim <- range(loc[,1])
ylim <- range(loc[,2])
proj <- inla.mesh.projector(mesh, xlim = xlim, ylim = ylim, dims = dim(volcano))

field_proj <- inla.mesh.project(proj, field)
rownames(field_proj) <- proj$x
colnames(field_proj) <- proj$y
pred_df <- melt(field_proj) %>%
  mutate(x = Var1, y = Var2, Type = "Reconstruction") %>%
```

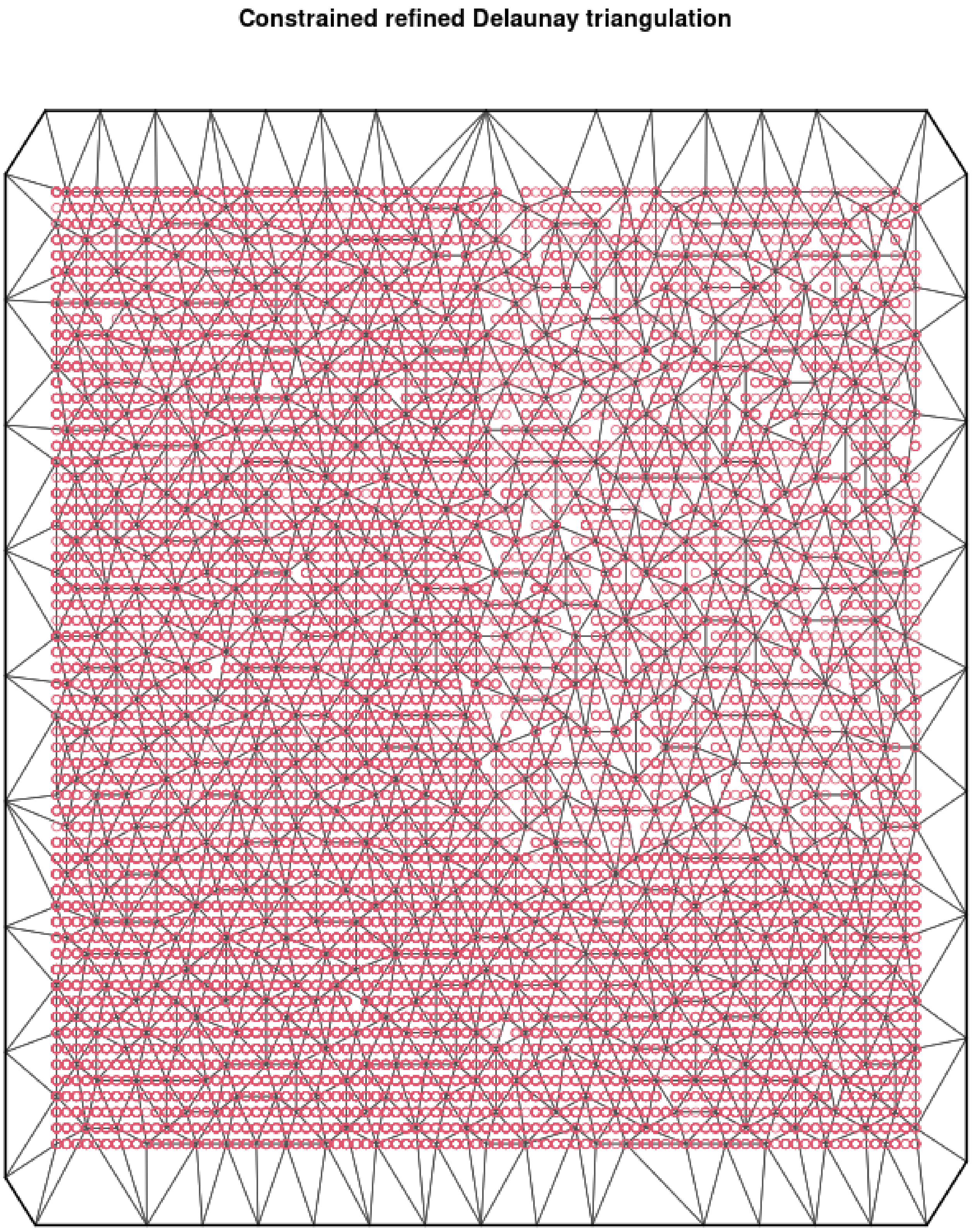


Figure 2: Comparison of different packages in R that can be used for spatial CPUE.

```r
  mutate(value = rescale(value, to = range(volcano_df$value)))

ggplot(data = bind_rows(volcano_df, pred_df)) +
  geom_tile(aes(x = Var1, y = Var2, fill = exp(value))) +
  facet_wrap(~Type, ncol = 1) +
  scale_fill_viridis_c() +
  theme_bw() +
  labs(x = NULL, y = NULL, fill = NULL)
```

```r
coefs1 <- exp(space1$summary.fixed$mean)
coefs1 <- coefs1 / geo_mean(coefs1)

coefs2 <- exp(c(0, blind1$summary.fixed$mean[2:n]))
coefs2 <- coefs2 / geo_mean(coefs2)

true_coefs <- exp(year_eff)
true_coefs <- true_coefs / geo_mean(true_coefs)

plot(true_coefs, type = "b")
lines(coefs1, col = 2)
lines(coefs2, col = 3)
```

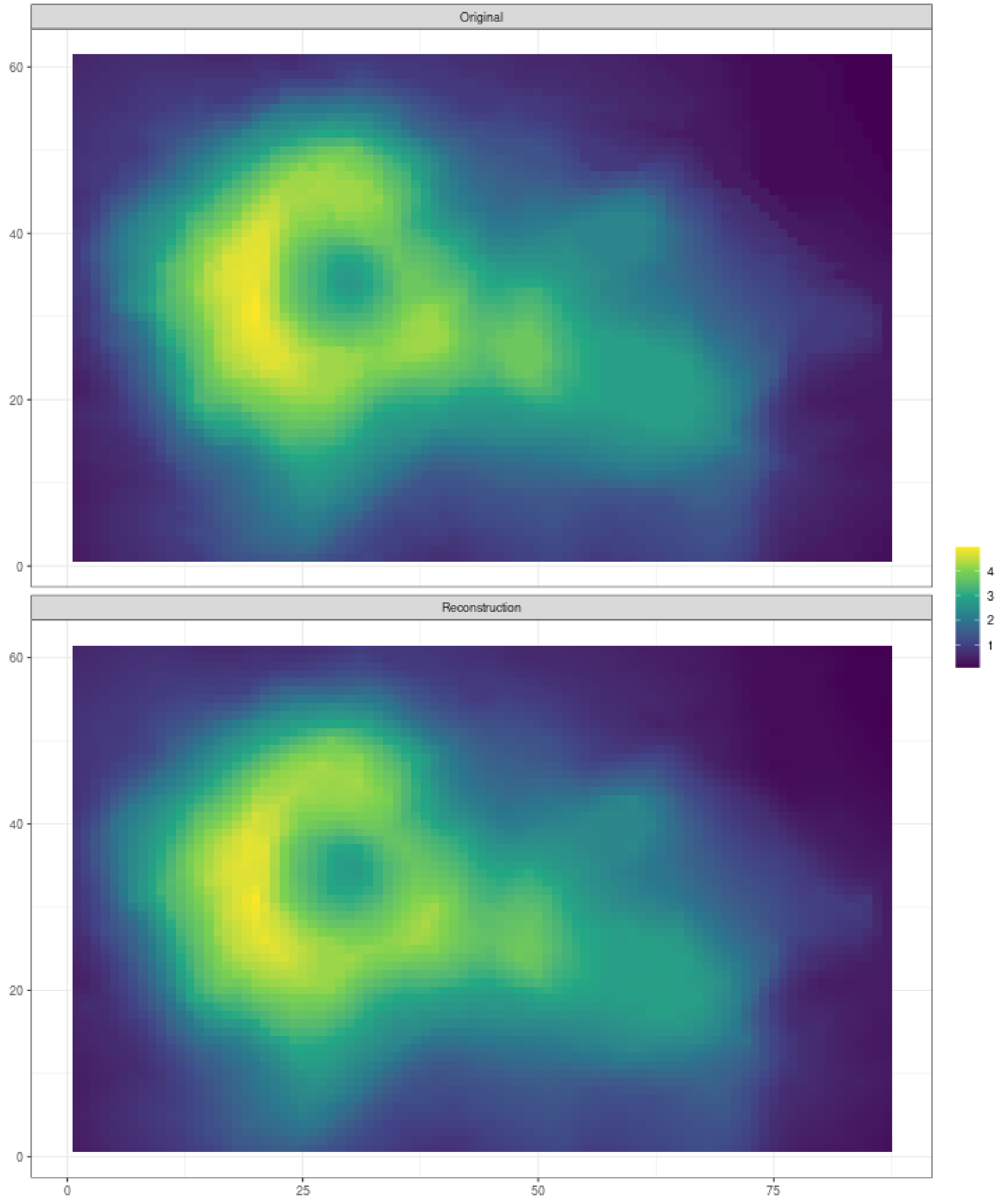


Figure 2: Comparison of different packages in R that can be used for spatial CPUE.

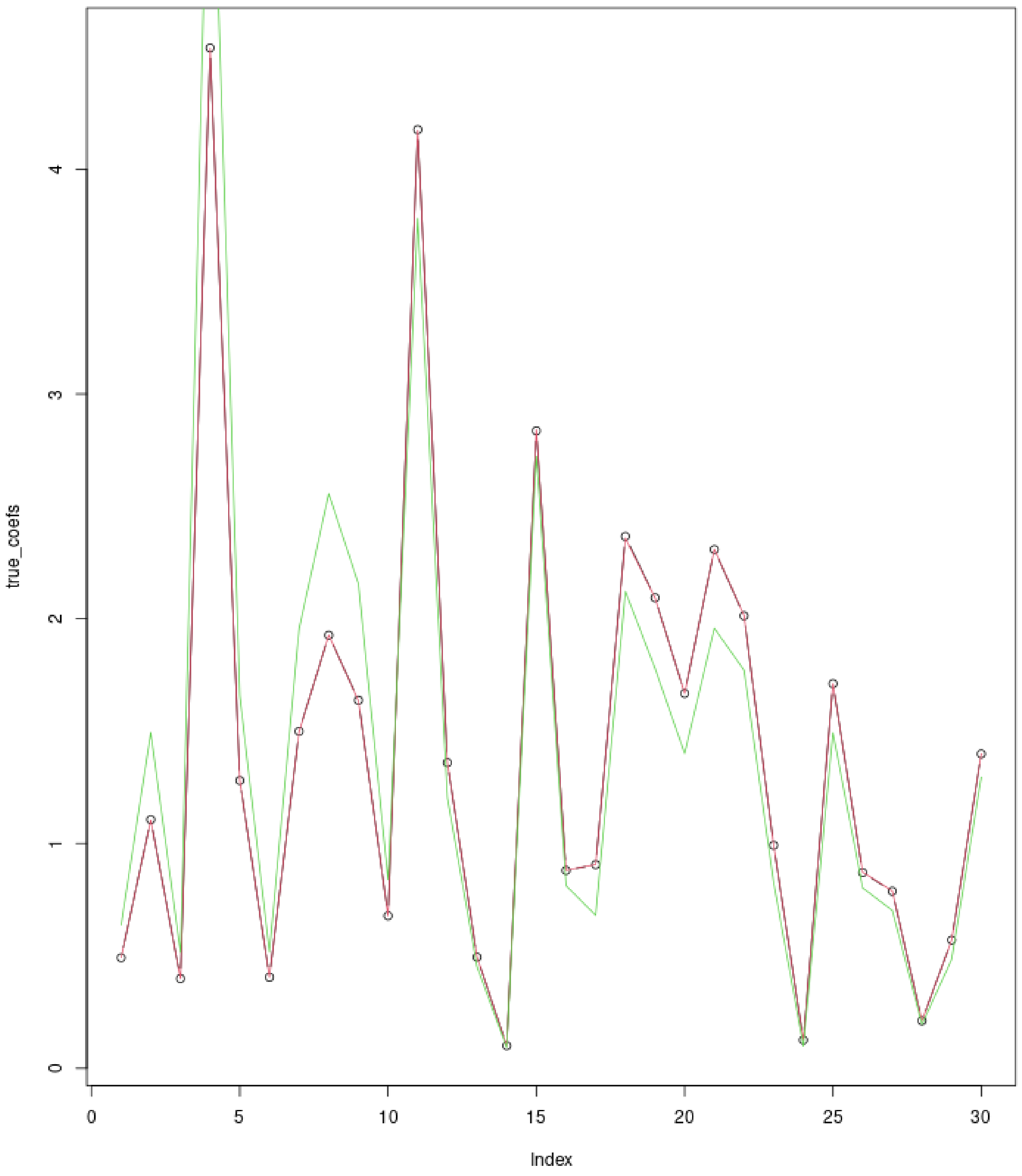
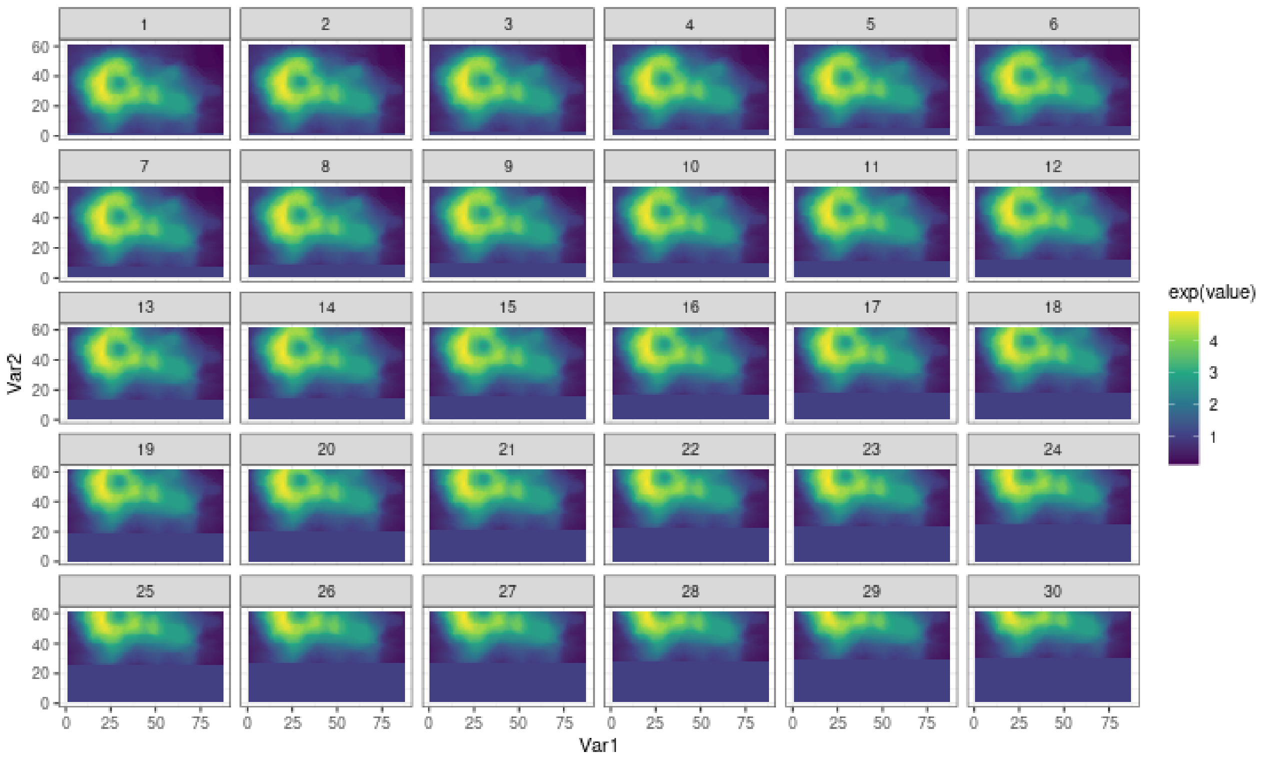


Figure 2: Comparison of different packages in R that can be used for spatial CPUE.

4 Spatio-temporal models

Again we simulate a data set. I use the volcano data set again, but this time the volcano moves north/up over time.

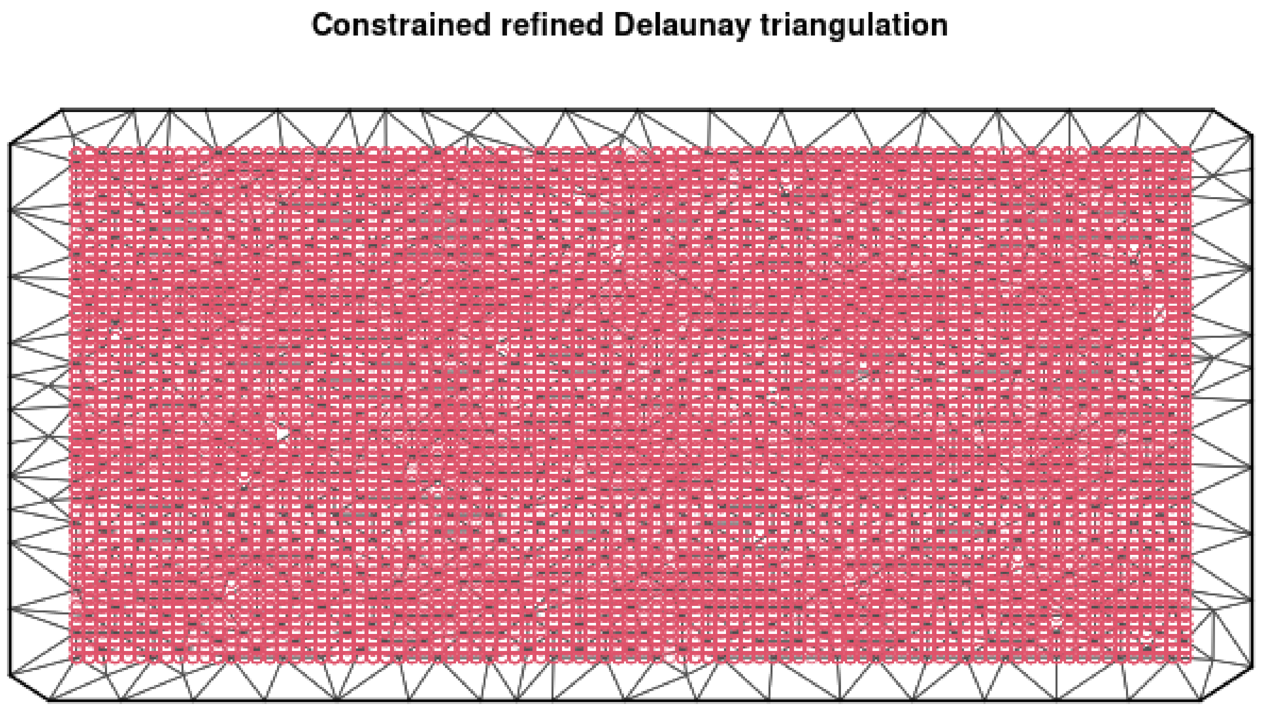
```r
# Create a spatial effect using the volcano data set but
# rescale it to a similar scale to the year effect
data(volcano)
volcano1 <- log(rescale(volcano, to = range(exp(year_eff))))

volcano_array <- array(NA, dim = c(n, dim(volcano)))
nshift <- 1
for (i in 1:n) {
  if (i == 1) {
    volcano2 <- volcano1
  } else {
    volcano2 <- volcano_array[i - 1,,]
  }
  volcano2[,(nshift + 1):ncol(volcano2)] <- volcano2[,1:(ncol(volcano2) - nshift)]
  volcano2[,1:nshift] <- 0
  volcano_array[i,,] <- volcano2
}
volcano_df2 <- melt(volcano_array) %>%
  mutate(Type = "Original") %>%
  rename(t = Var1, Var1 = Var2, Var2 = Var3)

ggplot(data = volcano_df2) +
  geom_tile(aes(x = Var1, y = Var2, fill = exp(value))) +
  facet_wrap(~t) +
  theme_bw() +
  scale_fill_viridis_c()
```



```r
data <- NULL
for (i in 1:n) {
  # take a random number of samples (fishing events)
  m <- round(runif(n = 1, min = 100, max = 2000), 0)
  err <- rnorm(n = length(volcano), 0, 0.01)
  d1 <- melt(volcano_array[i,,]) %>%
    rename(x = Var1, y = Var2) %>%
    mutate(yr = year_eff[i], year = factor(year[i]), err = err) %>%
    sample_n(m, replace = TRUE) %>%
    mutate(cpue = exp(yr + value + err))
  data <- rbind(data, d1)
}

loc <- data %>% select(x, y) %>% as.matrix()

mesh <- inla.mesh.2d(loc = loc, cutoff = 2, max.n.strict = 1000)
mesh$n
```

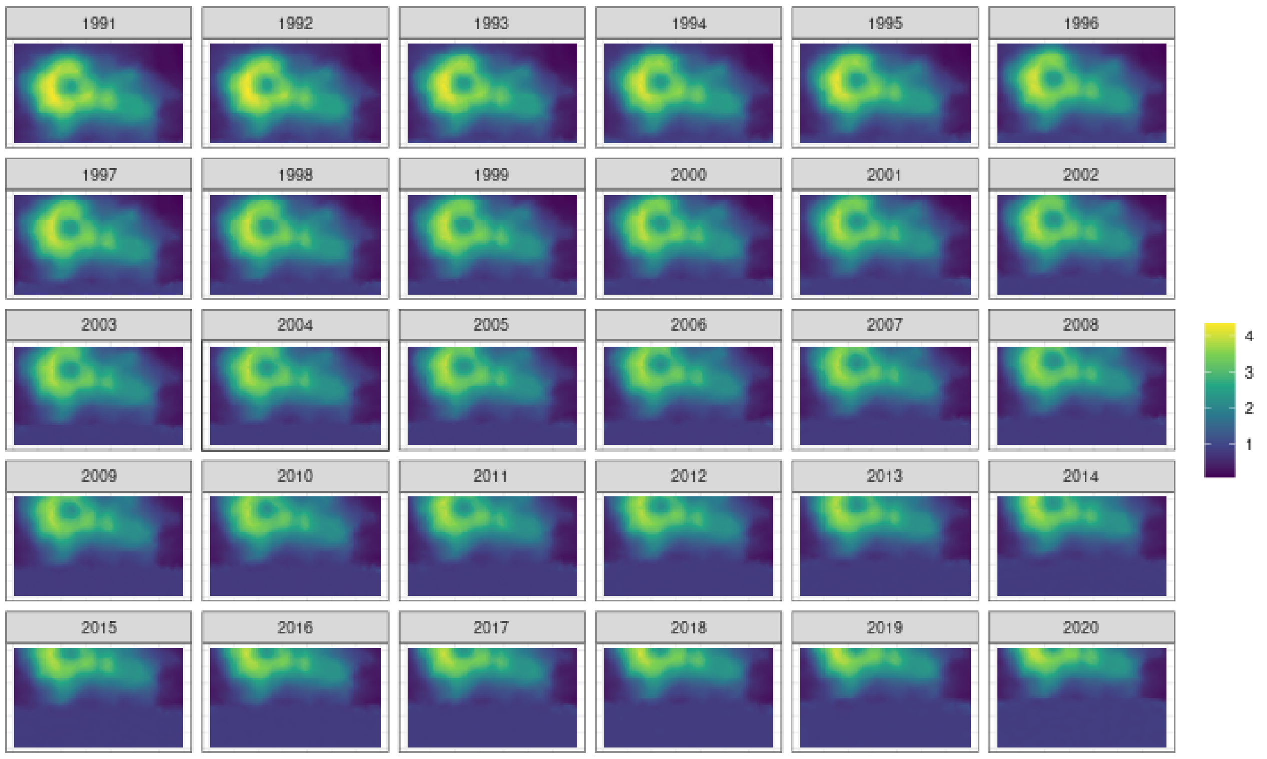
```
## [1] 808
```

```r
plot(mesh)
points(loc, col = 2)
```

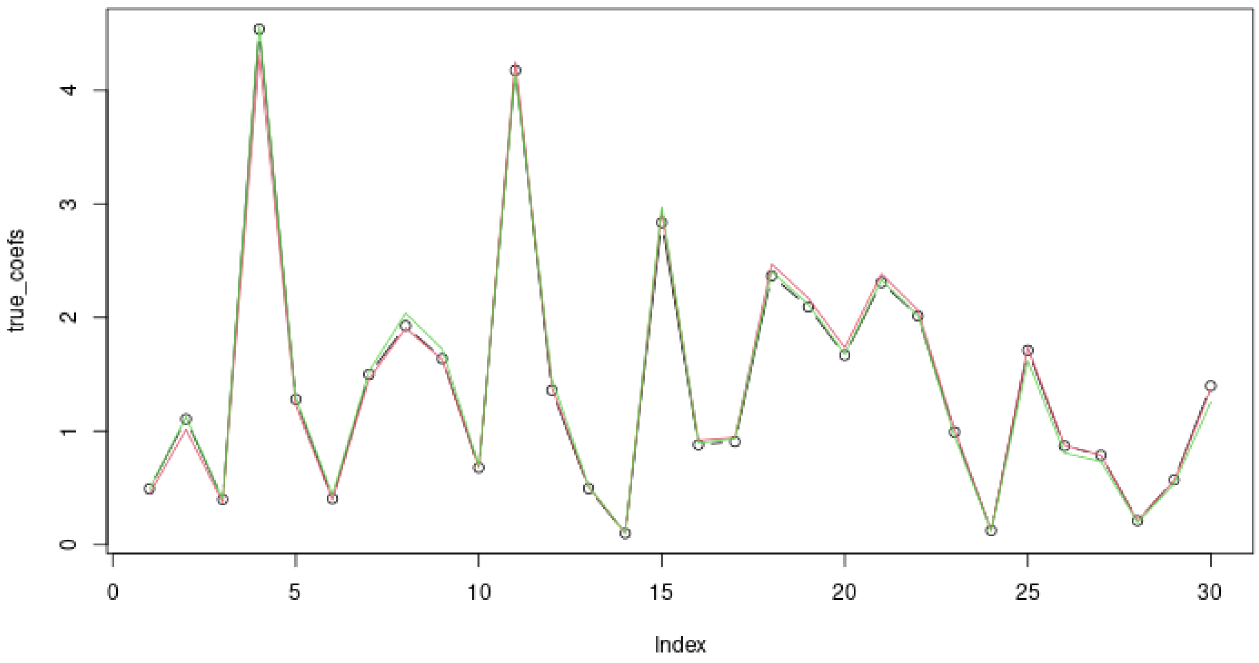


```
spde <- inla.spde2.pcmatern(mesh = mesh,
                            prior.range = c(0.5, 0.01),
                            prior.sigma = c(1, 0.01))

A <- inla.spde.make.A(mesh, loc = loc, group = as.numeric(data$year))

iset <- inla.spde.make.index('i', n.spde = spde$n.spde, n.group = n)

stack <- inla.stack(
  tag = "space2",
  data = list(cpue = data$cpue),
  A = list(A, 1),
  effects = list(iset,
                 data.frame(year = data$year)))

# We set the PC-prior for the temporal autoregressive parameter with P(cor > 0) = 0.9
h.spec <- list(theta = list(prior = 'pccor1', param = c(0, 0.9)))
prec.prior <- list(prior = 'pc.prec', param = c(1, 0.01))

if (do_estimate) {
  space2 <- inla(cpue ~ -1 + year + f(i, model = spde, group = i.group, control.group = list(model = "a
                 data = inla.stack.data(stack),
                 family = "lognormal",
                 control.predictor = list(A = inla.stack.A(stack)))

  blind2 <- inla(cpue ~ year,
                 data = data,
                 family = "lognormal")

  save(space2, blind2, data, loc, iset, mesh, A, spde, stack, file = "space2.rda")
} else {
```

```r
  load("space2.rda")
}

xlim <- range(loc[,1])
ylim <- range(loc[,2])
proj <- inla.mesh.projector(mesh, xlim = xlim, ylim = ylim, dims = dim(volcano))

fout <- NULL
for (i in 1:n) {
  j <- which(iset$i.group == i)
  field <- space2$summary.random[['i']][['mean']][j]
  # print(sum(field))
  # print(length(field))
  field_proj <- inla.mesh.project(proj, field)
  rownames(field_proj) <- proj$x
  colnames(field_proj) <- proj$y
  fout <- rbind(fout, melt(field_proj) %>% mutate(group = i, year = unique(data$year)[i]))
}

pred_df <- fout %>%
  mutate(x = Var1, y = Var2, Type = "Reconstruction")# %>%
  #mutate(value = rescale(value, to = range(volcano_df$value)))

ggplot(data = pred_df) +
  geom_tile(aes(x = Var1, y = Var2, fill = exp(value))) +
  facet_wrap(~year) +
  scale_fill_viridis_c() +
  labs(x = NULL, y = NULL, fill = NULL) +
  theme_bw() +
  theme(axis.title = element_blank(), axis.text = element_blank(), axis.ticks = element_blank())
```



```r
coefs1 <- exp(space2$summary.fixed$mean)
coefs1 <- coefs1 / geo_mean(coefs1)

coefs2 <- exp(c(0, blind2$summary.fixed$mean[2:n]))
coefs2 <- coefs2 / geo_mean(coefs2)

true_coefs <- exp(year_eff)
true_coefs <- true_coefs / geo_mean(true_coefs)

plot(true_coefs, type = "b")
lines(coefs1, col = 2)
lines(coefs2, col = 3)
```


true_coefs
Index
0
1
2
3
4
0
5
10
15
20
25
30

4.1 glmmTMB

See this webiste for spatial models https://cran.r-project.org/web/packages/glmmTMB/vignettes/covstruct .html. I am not sure glmmTMB uses multiple cores, it seemd to only use one core on my machine. It does look to have likelihood profile methods. Too slow though by the looks of it.

```
# library(glmmTMB)
#
# data <- data %>% mutate(pos = numFactor(x, y), group = 1)
#
# space2 <- glmmTMB(log(z) ~ year + exp(pos + 0 | group), data = data)
#
# #as.data.frame(ranef(space2))
# coefs <- exp(c(0, fixef(space2)$cond[2:n]))
# coefs <- coefs / geo_mean(coefs)
#
# true_coefs <- exp(year_eff)
# true_coefs <- true_coefs / geo_mean(true_coefs)
#
# plot(coefs, type = "l")
# lines(true_coefs, col = 3)
#
# confint(space2, "sigma")
#
# newdata <- data.frame(pos = numFactor(expand.grid(x = 1:3, y = 1:3)))
# newdata$group <- factor(rep(1, nrow(newdata)))
# newdata$year <- factor(2020)
# newdata
# predict(space2, newdata, type = "response", allow.new.levels = TRUE)
#
# predict_col <- function(i) {
#     newdata <- data.frame(pos = numFactor(expand.grid(1:nrow(volcano), i)))
#     newdata$group <- factor(rep(1, nrow(newdata)))
#     newdata$year <- factor(2020)
#     predict(space2, newdata = newdata, type = "response", allow.new.levels = TRUE)
# }
# pred <- sapply(1:ncol(volcano), predict_col)
# pred_df <- melt(pred / sum(pred, na.rm = TRUE)) %>% mutate(Type = "Reconstruction")
#
# sum(volcano_df$value)
# sum(pred_df$value)
#
# # ggplot(data = pred_df) +
# ggplot(data = bind_rows(volcano_df, pred_df)) +
#   geom_tile(aes(x = Var1, y = Var2, fill = value)) +
#   facet_wrap(~Type, ncol = 1) +
#   scale_fill_viridis_c()
```

4.2 INLA

Simple spatial models are resonably easy to set up in INLA. INLA is set up to use multiple cores to help speed things up.

```
library(INLA)
```

```r
loc <- st_coordinates(edw_pos)

mesh <- inla.mesh.2d(
  loc = loc,
  cutoff = 23000,
  max.n.strict = 120)

spde <- inla.spde2.pcmatern(mesh = mesh,
                            prior.range = c(50, 0.01),
                            prior.sigma = c(1, 0.01))

A <- inla.spde.make.A(mesh = mesh, loc = loc)

stack <- inla.stack(
  tag = "space1",
  data = list(cpue = edw_pos$cpue),
  A = list(A, 1),
  effects = list(s = 1:spde$n.spde,
                 data.frame(year = edw_pos$year)))

if (do_estimate) {
  space1 <- inla(cpue ~ year + f(s, model = spde),
                 data = inla.stack.data(stack),
                 family = "gamma",
                 control.predictor = list(A = inla.stack.A(stack)))

  save(space1, file = "inla_space.rda")
} else {
  load("inla_space.rda")
}

field <- space1$summary.random[['s']][['mean']]

xlim <- range(st_coordinates(edw_pos)[,1])
ylim <- range(st_coordinates(edw_pos)[,2])
proj <- inla.mesh.projector(mesh, xlim = xlim, ylim = ylim, dims = c(250, 500))

field_proj <- inla.mesh.project(proj, field)
rownames(field_proj) <- proj$x
colnames(field_proj) <- proj$y
sfield <- reshape2::melt(field_proj) %>%
  st_as_sf(coords = c('Var1', 'Var2'), crs = st_crs(edw_pos)) %>%
  mutate(x = st_coordinates(.)[,1], y = st_coordinates(.)[,2])

ggplot(data = sfield) +
  geom_tile(aes(x = x, y = y, fill = value)) +
  # geom_sf(data = coast_sf_km, fill = "black", colour = NA) +
  coord_sf(xlim = xlim, ylim = ylim) +
  scale_fill_viridis_c(na.value = NA) +
  labs(x = NULL, y = NULL, fill = NULL) +
  theme_bw() +
  theme(legend.position = "bottom", legend.key.width = unit(3, "cm"))
```

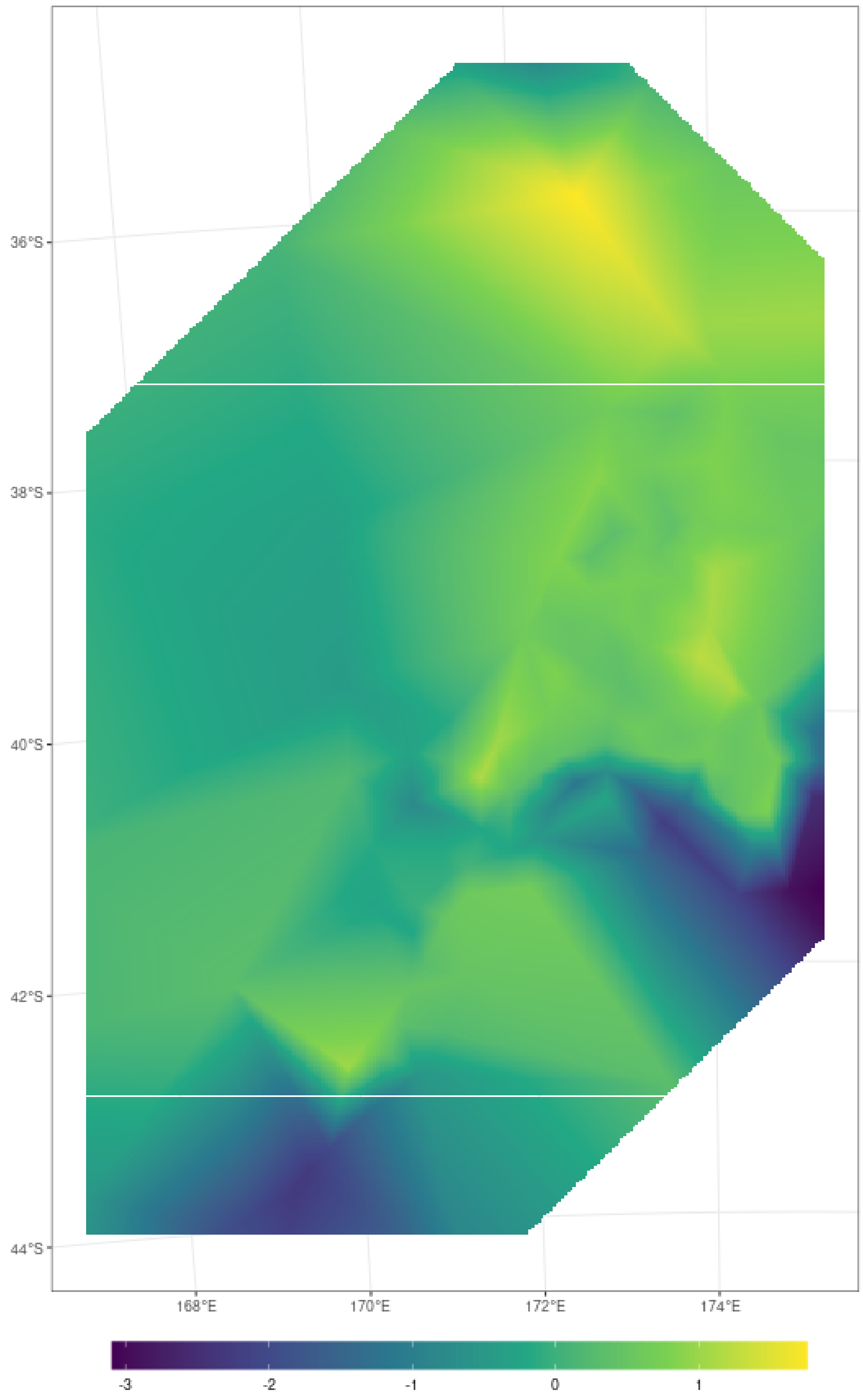


Figure 3: Comparison of different packages in R that can be used for spatial CPUE.

5 Discussion

What we really need is a version of INLA written in TMB. VAST attempted to fulfill this role, but it has become a black box.