TMA4300 Computer Intensive Statistical Methods
Exercise 2, Spring 2019

Group members: Henrik Syversveen Lie, Mikal Solberg Stapnes

12.03.2019

We wish to carry out a spatial analysis on mortality rates of oral cavity cancer in males in Germany during a 5-year period, 1986–1990.

We assume that the observed counts are conditionally independent Poisson

$$\lambda_i|\eta_i \sim Poisson(E_i \exp \eta_i)$$

The log relative risk $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)^T$ is decomposed into

$$\boldsymbol{\eta} = \mathbf{u} + \mathbf{v},$$

where the component $\mathbf{u} = (u_1, \ldots, u_n)$ is spatially structured with smoothing parameter κ_u. The component $\mathbf{v} = (v_1, \ldots, v_n)$ is unstructured white noise with precision parameter κ_v, i.e. $\mathbf{v} \sim N(\mathbf{0}, \kappa_v^{-1} I)$

For $\mathbf{u}$, we assume that neighboring districts are more similar than distant districts. We define that two districts are neighbours if they share a common border. $\mathbf{u}$ then becomes an intrinsic Gaussian Markov random field with density

$$p(\mathbf{u}|\kappa_u) \propto \kappa_u^{(n-1)/2} \exp\left(\frac{-\kappa_u}{2} \sum_{i \sim j} (u_i - u_j)^2\right)$$

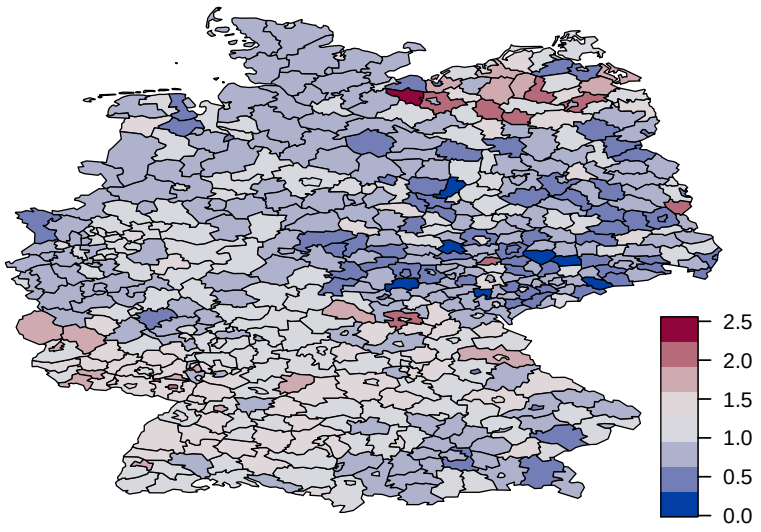

Figure 1: Standardised mortality rates (SMR)

We can define R as the matrix

$$R_{ij} = \begin{cases} n_i, & i = j \\ -1, & i \sim j \\ 0, & \text{otherwise.} \end{cases}$$

and get

$$p(\mathbf{u}|\kappa_u) \propto \kappa_u^{(n-1)/2} \exp\Big(\frac{-\kappa_u}{2}\mathbf{u}^T R\mathbf{u}\Big),$$

where n_i is the number of neighboring districts of district i and $i \sim j$ denotes that district i is a neighboring district of district j. The distribution of $\boldsymbol{\eta}$, conditional on the spatial component $\mathbf{u}$ and κ_v, is

$$\boldsymbol{\eta}|\mathbf{u}, \kappa_v \sim \mathcal{N}(\mathbf{u}, \kappa^{-1} I).$$

The precision terms κ_u and κ_v are assigned Gamma priors

$$\kappa_u \sim \text{Gamma}(\alpha_u, \beta_u), \kappa_v \sim \text{Gamma}(\alpha_v, \beta_v).$$

With $\alpha_u = \alpha_v = 1$ and $\beta_u = \beta_v = 0.01$.

We wish to analyse the data and its underlying spatial structure by implementing a Gibbs sampler with individual parameter updates based on the full conditional distributions.

Exercise 1 (Derivations)

a)

We start by computing the joint distribution from likelihood and prior distributions.

$$p(\mathbf{y}, \mathbf{u}, \boldsymbol{\eta}, \kappa_u, \kappa_v) = p(\mathbf{y}|\mathbf{u}, \boldsymbol{\eta}, \kappa_u, \kappa_v) \cdot p(\mathbf{u}, \boldsymbol{\eta}, \kappa_u, \kappa_v) = p(\mathbf{y}|\boldsymbol{\eta}) p(\boldsymbol{\eta}|\mathbf{u}, \kappa_v) p(\kappa_v) p(\mathbf{u}|\kappa_u) p(\kappa_u)$$

Inserting the Gamma density for κ_u and κ_v, the normal density for $\mathbf{u}|\kappa_u$ and the normal density for $\boldsymbol{\eta}|\mathbf{u}, \kappa_v$ we get

$$\begin{aligned} &p(\mathbf{y}, \mathbf{u}, \boldsymbol{\eta}, \kappa_u, \kappa_v) \propto \\ &p(\mathbf{y}|\boldsymbol{\eta}) \kappa_v^{n/2} \exp\Big(-\frac{\kappa_v}{2}(\boldsymbol{\eta} - \mathbf{u})^T(\boldsymbol{\eta} - \mathbf{u})\Big) \kappa_v^{\alpha_v - 1} \exp(-\beta_v \kappa_v) \kappa_u^{(n-1)/2} \exp\big(-\frac{\kappa_u}{2}\mathbf{u}^T R\mathbf{u}\big) \kappa_u^{\alpha_u - 1} \exp(-\beta_u \kappa_u). \end{aligned}$$

Inserting the Poisson density of the data $\mathbf{y}$, we get

$$\begin{aligned} p(\mathbf{y}, \mathbf{u}, \boldsymbol{\eta}, \kappa_u, \kappa_v) \propto &\Big(\prod_{i=1}^{n} \exp(\eta_i)^{y_i} \exp(-E_i \exp(\eta_i))\Big) \kappa_v^{n/2} \exp\Big(-\frac{\kappa_v}{2}(\boldsymbol{\eta} - \mathbf{u})^T(\boldsymbol{\eta} - \mathbf{u})\Big) \cdot \\ &\kappa_v^{\alpha_v - 1} \exp(-\beta_v \kappa_v) \kappa_u^{(n-1)/2} \exp\big(-\frac{\kappa_u}{2}\mathbf{u}^T R\mathbf{u}\big) \kappa_u^{\alpha_u - 1} \exp(-\beta_u \kappa_u), \end{aligned}$$

which we can rewrite to

$$p(\mathbf{y}, \mathbf{u}, \boldsymbol{\eta}, \kappa_u, \kappa_v) \propto \kappa_u^{(n-1)/2 + \alpha_u - 1} \kappa_v^{n/2 + \alpha_v - 1} \exp\Big(-\beta_u \kappa_u - \beta_v \kappa_v - \frac{\kappa_u}{2}\mathbf{u}^T R\mathbf{u} - \frac{\kappa_v}{2}(\boldsymbol{\eta} - \mathbf{u})^T(\boldsymbol{\eta} - \mathbf{u}) + \sum_i (y_i \eta_i - E_i \exp(\eta_i))\Big).$$

The posterior distribution of the parameters $p(\boldsymbol{\eta}, \mathbf{u}, \kappa_u, \kappa_v|\mathbf{y})$ is proportional to the joint distribution, giving us that

$p(\boldsymbol{\eta}, \mathbf{u}, \kappa_u, \kappa_v|\mathbf{y}) \propto p(\mathbf{y}, \mathbf{u}, \boldsymbol{\eta}, \kappa_u, \kappa_v) \propto$
$\kappa_u^{(n-1)/2+\alpha_u-1}\kappa_v^{n/2+\alpha_v-1} \exp\Big(-\beta_u\kappa_u - \beta_v\kappa_v - \frac{\kappa_v}{2}(\boldsymbol{\eta}-\mathbf{u})^T(\boldsymbol{\eta}-\mathbf{u}) - \frac{\kappa_u}{2}\mathbf{u}^T R\mathbf{u} + \sum_i (y_i\eta_i - E_i\exp(\eta_i))\Big).$

b)

Due to the non-normality, sampling from the posterior distribution will require a Metropolis–Hastings step. To obtain a proposal distribution that is easy to sample from, here a Gaussian, we note that we can approximate the function

$$f(\eta_i) = y_i\eta_i - E_i\exp(\eta_i)$$

by its 2nd order Taylor expansion, $\tilde{f}(\eta_i)$, around a point z_i. The approximation can be written as,

$$\begin{aligned}
\tilde{f}(\eta_i) &= y_i z_i - E_i\exp(z_i) + (y_i - E_i\exp(z_i))(\eta_i - z_i) - E_i\exp(z_i)\frac{(\eta_i - z_i)^2}{2} \\
&= E_i\exp(z_i)(z_i - 1 - \frac{z_i^2}{2}) + \eta_i(y_i + E_i\exp(z_i)(z_i - 1)) - \frac{\eta_i^2}{2}E_i\exp(z_i) \\
&= a_i + b_i\eta_i - \frac{c_i}{2}\eta_i^2,
\end{aligned}$$

where $a_i = E_i\exp(z_i)(z_i - 1 - \frac{z_i^2}{2})$, $b_i = y_i + E_i\exp(z_i)(z_i - 1)$ and $c_i = E_i\exp(z_i)$.

c)

The full conditional of κ_u can be computed by

$$p(\kappa_u|\boldsymbol{\eta}, \mathbf{u}, \kappa_v, \mathbf{y}) = \frac{p(\boldsymbol{\eta}, \mathbf{u}, \kappa_u, \kappa_v|\mathbf{y})}{\int p(\boldsymbol{\eta}, \mathbf{u}, \kappa_v, \kappa_u|\mathbf{y})d\kappa_u} \propto p(\boldsymbol{\eta}, \mathbf{u}, \kappa_u, \kappa_v|\mathbf{y})$$

We use the fact that any factor in the posterior not containing κ_u are now constants and get

$$p(\kappa_u|\boldsymbol{\eta}, \mathbf{u}, \kappa_v, \mathbf{y}) \propto \kappa_u^{\frac{n-1}{2}+\alpha_u-1}\exp\big(-\kappa_u(\beta_u + \frac{1}{2}\mathbf{u}^T R\mathbf{u})\big),$$

which we recognize to be the density of a gamma-distributed variable with parameters $\frac{n-1}{2} + \alpha_u$ and $\beta_u + \frac{1}{2}\mathbf{u}^T R\mathbf{u}$.

We follow along the same lines for κ_v, and get

$$p(\kappa_v|\boldsymbol{\eta}, \mathbf{u}, \kappa_u, \mathbf{y}) \propto \kappa_v^{\frac{n}{2}+\alpha_v-1}\exp\big(-\kappa_v(\beta_v + \frac{1}{2}(\boldsymbol{\eta}-\mathbf{u})^T(\boldsymbol{\eta}-\mathbf{u}))\big),$$

which we recognize to be the density of a gamma-distributed variables with parameters $\frac{n}{2} + \alpha_v$ and $\beta_v + \frac{1}{2}(\boldsymbol{\eta}-\mathbf{u})^T(\boldsymbol{\eta}-\mathbf{u})$.

Again, we do the same for $\mathbf{u}$, and get

$$p(\mathbf{u}|\boldsymbol{\eta}, \kappa_u, \kappa_v, \mathbf{y}) \propto \exp\big(\kappa_v\boldsymbol{\eta}^T\mathbf{u} - \frac{\kappa_v}{2}\mathbf{u}^T\mathbf{u} - \frac{\kappa_u}{2}\mathbf{u}^T R\mathbf{u}\big) = \exp(\kappa_v\boldsymbol{\eta}^T\mathbf{u} - \frac{1}{2}\mathbf{u}^T(\kappa_v I + \kappa_u R)\mathbf{u}\big),$$

which we recognize to be a normal density in canonical form with $\mathbf{b} = \kappa_v \boldsymbol{\eta}$ and $A = \kappa_v I + \kappa_u R$.

Finally, we repeat the procedure for $\boldsymbol{\eta}$, and get

$$p(\boldsymbol{\eta}|\mathbf{u}, \kappa_u, \kappa_v, \mathbf{y}) \propto \exp\left(\kappa_v \boldsymbol{\eta}^T \mathbf{u} - \frac{\kappa_v}{2}\boldsymbol{\eta}^T\boldsymbol{\eta} + \sum_i (y_i \eta_i - E_i \exp(\eta_i))\right) = \exp\left(\kappa_v \boldsymbol{\eta}^T \mathbf{u} - \frac{1}{2}\boldsymbol{\eta}^T(\kappa_v I)\boldsymbol{\eta} + \boldsymbol{\eta}^T\mathbf{y} - \exp(\boldsymbol{\eta})^T\mathbf{E}\right)$$

Using the Taylor expansion of $\mathbf{y}^T\boldsymbol{\eta} - \exp(\boldsymbol{\eta})^T\mathbf{E}$, as described in b), we may insert this into $p(\boldsymbol{\eta}|\cdot)$ to create the approximation $q(\boldsymbol{\eta}|\mathbf{z}, \cdot)$,

$$q(\boldsymbol{\eta}|\mathbf{z}, \mathbf{u}, \kappa_u, \kappa_v, \mathbf{y}) \propto \exp\left(\boldsymbol{\eta}^T(\kappa_v \mathbf{u} + \mathbf{b}) - \frac{1}{2}\boldsymbol{\eta}^T\big(\kappa_v I + \text{diag}(\mathbf{c})\big)\boldsymbol{\eta}\right),$$

where $\mathbf{b} = (b_1, \ldots, b_n)^T$, $b_i = y_i + E_i \exp(z_i)(z_i - 1)$, $\mathbf{c} = (c_1, \ldots, c_n)^T$, $c_i = E_i \exp(z_i)$, and $\text{diag}(\mathbf{c})$ is the matrix with the elements of $\mathbf{c}$ on the diagonal and zeros elsewhere.

We observe that $q(\boldsymbol{\eta}|\cdot)$ is a normal density written in canonical form and thus easily sampled from. We also observe that as the $\exp(\cdot)$ is a continuous transform, $q(\boldsymbol{\eta}|\mathbf{z}, \cdot)$ will converge to $p(\boldsymbol{\eta}|\cdot)$ as $\mathbf{z}$ gets increasingly close to $\boldsymbol{\eta}$. Thus, for a reasonably good $\mathbf{z}$, $q(\boldsymbol{\eta}|\mathbf{z})$ will be a good choice for a proposal density for $p(\boldsymbol{\eta}|\cdot)$.

Exercise 2 (Implementation of the MCMC sampler)

We implement a Gibbs sampling algorithm with individual parameter updates using the full conditional for κ_u, κ_v and $\mathbf{u}$. We update $\boldsymbol{\eta}$ using Metropolis-Hastings with $q(\boldsymbol{\eta}|\mathbf{z}, \cdot)$ as the proposal density. We set $\mathbf{z} = \boldsymbol{\eta}^{(m-1)}$, and do $N = 50000$ iterations.

```
# Setup
load("tma4300_ex2_Rmatrix.Rdata")
n = length(Oral$SMR)
alpha_u = 1
alpha_v = 1
beta_u = 0.01
beta_v = 0.01
```

```
set.seed(123)
source("dmvnorm.R")

sample_u = function(kappa_v, kappa_u, eta) {
    b = kappa_v * eta
    A = kappa_v * diag.spam(n) + kappa_u * R

    # Samples from the canonical normal
    res = rmvnorm.canonical(1, b, A)
    return(t(res))
}

sample_eta = function(kappa_v, u, z) {
    b_taylor = Oral$Y + Oral$E * exp(z) * (z - 1)
    c_taylor = Oral$E * exp(z)

    b_arg_sampler = kappa_v * u + b_taylor
    A_arg_sampler = kappa_v * diag.spam(n) + diag(c_taylor)
```

```
    # Samples from the canonical normal
    return(as.vector(rmvnorm.canonical(1, b_arg_sampler, A_arg_sampler)))
}

log_full_conditional_eta = function(eta, kappa_v, u, log = T) {
    # Corresponds to p(eta | ...) in the text

    res = kappa_v * eta %*% u - 1/2 * t(eta) %*% (kappa_v * diag.spam(n)) %*%
        eta + eta %*% Oral$Y - t(exp(eta)) %*% Oral$E
    if (log) {
        return(res)
    } else {
        return(exp(res))
    }
}

log_full_conditional_eta_approx = function(eta, z, u, kappa_v, log = T) {
    # Corresponds to q(eta | ...) in the text

    b_taylor = Oral$Y + Oral$E * exp(z) * (z - 1)
    c_taylor = Oral$E * exp(z)

    b_canonical = kappa_v * u + b_taylor
    A_canonical = kappa_v * diag.spam(n) + diag.spam(c_taylor)

    res = dmvnorm.canonical(eta, b_canonical, A_canonical)

    if (log) {
        return(res)
    } else {
        return(exp(res))
    }
}

# Set number of iterations
N_iter = 50000

# Allocate vectors and arrays
u = matrix(data = NA, nrow = N_iter, ncol = length(Oral$Y))
eta = matrix(data = NA, nrow = N_iter, ncol = length(Oral$Y))
kappau = vector(mode = "double", length = N_iter)
kappav = vector(mode = "double", length = N_iter)
updated_eta = vector(length = N_iter)
accept_prob = vector(length = N_iter)
times = vector(mode = "double", length = N_iter)

# Set start values
kappau_start = alpha_u/beta_u
kappav_start = alpha_v/beta_v
u_start = rep(0, length(Oral$SMR))
eta_start = log(Oral$SMR)

u[1, ] = u_start
```

```
eta[1, ] = eta_start
kappau[1] = kappau_start
kappav[1] = kappav_start



# Run the MCMC algorithm
for (i in 2:N_iter) {
    # Initialize clock to measure time
    t0 = as.numeric(Sys.time())
    # Sample kappau
    kappau[i] = rgamma(1, (n - 1)/2 + alpha_u, beta_u + 1/2 * t(u[i -
        1, ]) %*% R %*% u[i - 1, ])

    # Sample kappav
    kappav[i] = rgamma(1, n/2 + alpha_v, beta_v + 1/2 * t(eta[i - 1,
        ] - u[i - 1, ]) %*% (eta[i - 1, ] - u[i - 1, ]))

    # Sample u
    u[i, ] = sample_u(kappav[i], kappau[i], eta[i - 1, ])

    # Sample eta
    z = eta[i - 1, ]
    eta_prop = sample_eta(kappav[i], u[i, ], z)

    logp1 = log_full_conditional_eta(eta_prop, kappav[i], u[i, ])
    logp2 = log_full_conditional_eta(z, kappav[i], u[i, ])
    logp3 = log_full_conditional_eta_approx(z, eta_prop, u[i, ], kappav[i])
    logp4 = log_full_conditional_eta_approx(eta_prop, z, u[i, ], kappav[i])

    # Compute acceptance prob
    log_alpha = min(0, logp1 - logp2 + logp3 - logp4)
    accept_prob[i] = exp(log_alpha)

    # Accept if r<alpha, with r coming from a uniform[0,1] distribution
    r = runif(1)
    if (log(r) < log_alpha) {
        updated_eta[i] = TRUE
        eta[i, ] = eta_prop
    } else {
        eta[i, ] = eta[i - 1, ]
    }
    # Compute time elapsed in this iteration
    times[i] = as.numeric(Sys.time()) - t0
}
```

We note that the code takes 2313 seconds, or 38.6 minutes to run.

Exercise 3 (Convergence diagnostics)

To assess the convergence of our MCMC algorithm we conduct a few diagnostic summaries for the precision parameters κ_u and κ_v and three randomly chosen components of $\mathbf{u}$ and $\mathbf{v}$.

First, we show a trace plot of κ_u. The values in the burn-in period are really large, so we display both a plot of the whole chain, and a plot where the first 500 values are removed.

```
df1 = data.frame(kappa_u = c(kappau, kappau[501:N_iter]), N = c(1:N_iter,
    501:N_iter), type = c(rep("All iterations", N_iter), rep("First 500 discarded",
    N_iter - 500)))
ggplot(df1, aes(x = N, y = kappa_u)) + geom_point() + facet_wrap(. ~
    type, scales = "free")
```

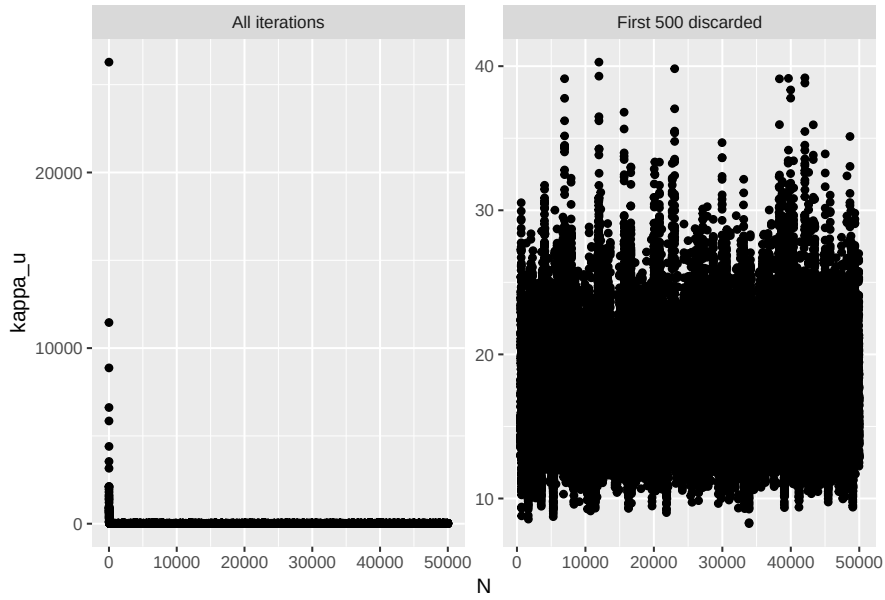

Figure 2: Trace plot of κ_u. All iterations plotted to the left, first 500 discarded plotted to the right.

From the left plot in figure 2 we see that the first samples for κ_u jump to very high values before settling in the range between 10 and 40. This indicates that the burn-in period might be somewhat over 500 iterations.

We display an autocorrelation plot for κ_u.

```
acf(kappau, lag.max = 60)
```

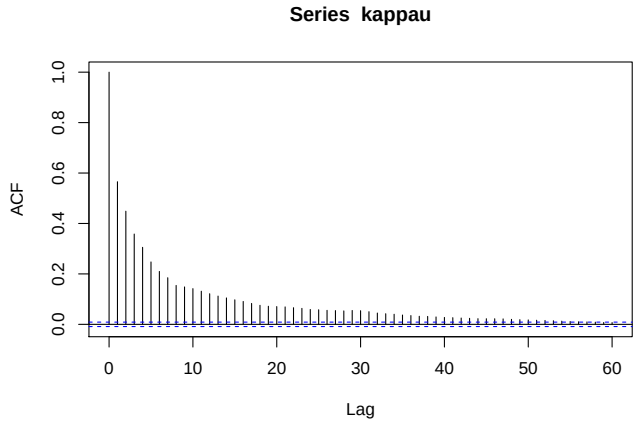

Figure 3: Autocorrelation plot for κ_u

From the plot in figure 3, we see that we can use only every $\sim 50^{\text{th}}$ iteration to get a correlation that is

not statistically significant. Taking into account the burn-in period, we would then only get about 1000 uncorrelated samples from our Markov chain of size $N = 50000$.

To conclude our analysis of the chain for κ_u, we check the chain for convergence by using the function `geweke.diag()` from the R package `coda`. The function computes a $z - score$ for the difference in mean between the first 10% and last 50% of the samples. The z-score can be used to compute a corresponding p-value. Large p-values indicate that the chain converged during the first 10% iterations. It could be reasonable to remove the burn-in period before using Geweke's diagnostic test, but we have chosen not to do this. Therefore our results from conducting Geweke's diagnostic may be too pessimistic.

```
gw = geweke.diag(kappau)
cat("zscores:", gw$z, "\n")
cat("pvals:", 2 * (1 - pnorm(abs(gw$z))))
```

```
## zscores: 1.138653
```

```
## pvals: 0.2548478
```

A p-value of 0.25 indicates that the chain did converge during the first 10% iterations, because this value is above any reasonable significance level.

We then move on to κ_v. As for κ_u, we display trace plots for the whole chain and where the first 500 values are removed.

```
df1 = data.frame(kappa_v = c(kappav, kappav[501:N_iter]), N = c(1:N_iter,
    501:N_iter), type = c(rep("All iterations", N_iter), rep("First 500 discarded",
    N_iter - 500)))
ggplot(df1, aes(x = N, y = kappa_v)) + geom_point() + facet_wrap(. ~
    type, scales = "free")
```

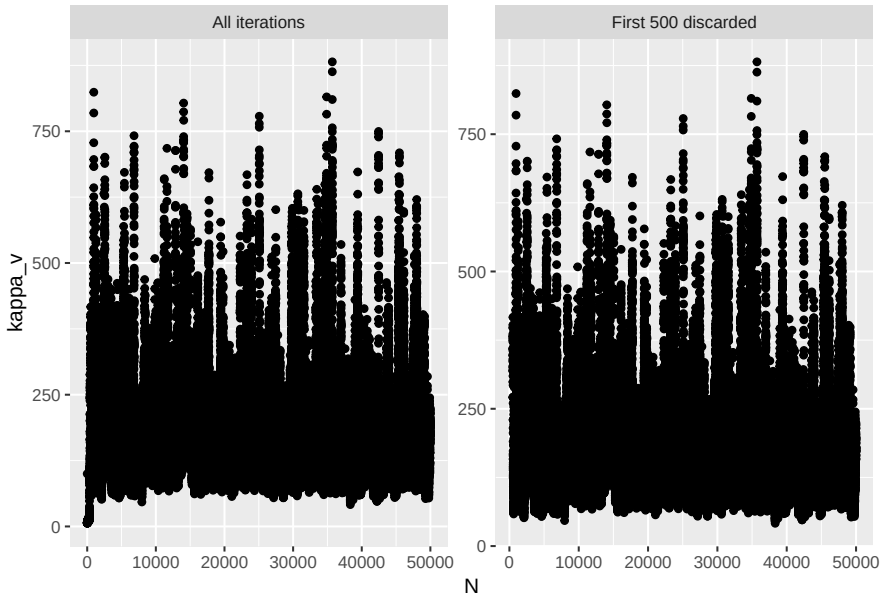

Figure 4: Trace plot of κ_v. All iterations plotted to the left, first 500 discarded plotted to the right.

From the left plot in figure 4 we see that the samples of κ_v remain low for the first iterations before settling in the range between 50 and 700. Also here, we suspect that the burn-in period might be somewhat over 500 iterations.

We then display an autocorrelation plot for κ_v.

```
acf(kappav)
acf(kappav, lag.max = 1000)
```

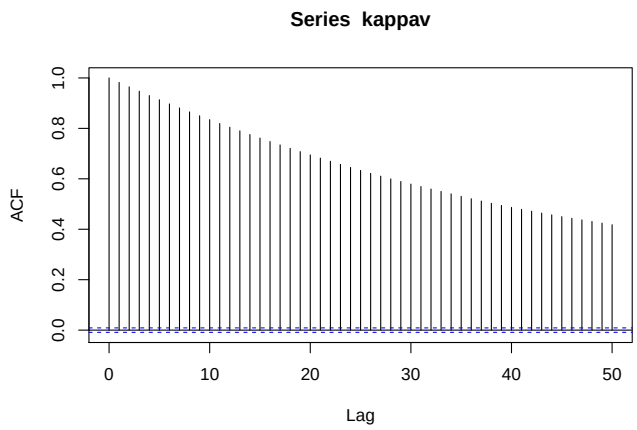

Figure 5: Autocorrelation plot of κ_v with max lag of 50.

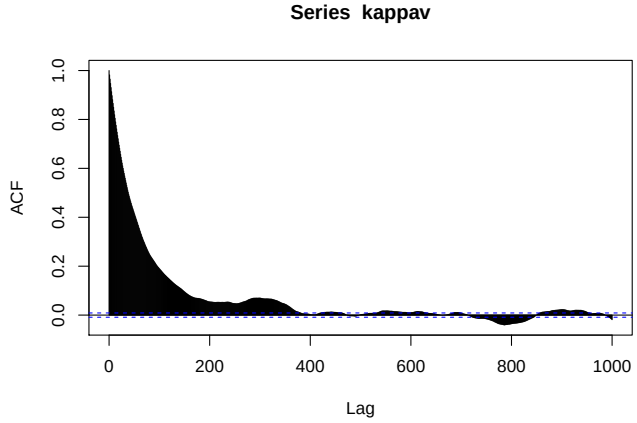

Figure 6: Autocorrelation plot of κ_v with max lag of 1000.

From the plot in figure 5 we see that the correlation between consecutive samples of κ_v is very high. From the plot in figure 6, we observe that the lag required to get non-significant autocorrelation is several hundred. The samples of κ_v being highly correlated is not ideal, as it will take the MCMC algorithm several iterations to generate the equivalent of an independent sample.

As a final diagnostic for κ_v we conduct Geweke's Diagnostic.

```
gw = geweke.diag(kappav)
cat("zscores:", gw$z, "\n")
cat("pvals:", 2 * (1 - pnorm(abs(gw$z))))
```

```
## zscores: -0.1159965
```

```
## pvals: 0.9076554
```

A p-value of 0.91 indicates that the chain converged during the first 10% iterations.

We then draw three random components of $\mathbf{u}$ and $\mathbf{v}$ and plot trace plots, autocorrelation functions and use the function `geweke.diag()` to check for convergence.

```
set.seed(123)
random_indexes = sort(sample(1:length(Oral$Y), 3, replace = F))
df4 = data.frame()
for (i in 1:3) {
    index = random_indexes[i]
    df = data.frame(u = u[, index], N = 1:N_iter, shape = as.factor(i),
        col = as.factor(index))
    df4 = rbind(df4, df)
}
ggplot(df4, aes(x = N, y = u, shape = shape, col = col)) + geom_point() +
    scale_shape(solid = F) + guides(shape = F) + ggtitle("Trace plot for u")
```

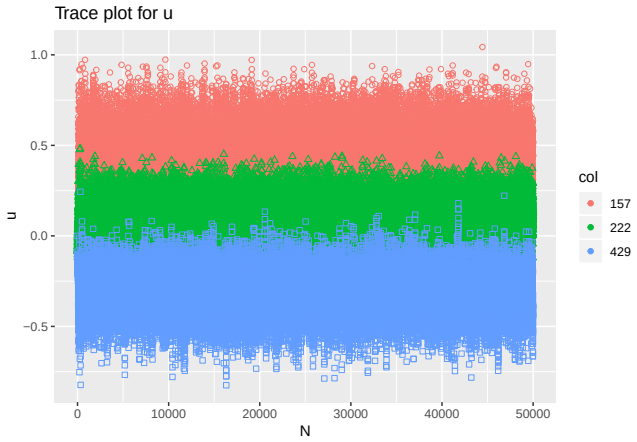

Figure 7: Trace plot for three random components of **u**.

From the trace plots in figure 7 of the three components of **u** we observe only a small burn-in period.

We then plot an autocorrelation plot for the same three components.

```
df5 = data.frame(u = u[, random_indexes])
acf(u[, random_indexes[1]], lag.max = 200)
acf(u[, random_indexes[2]], lag.max = 200)
acf(u[, random_indexes[3]], lag.max = 200)
```

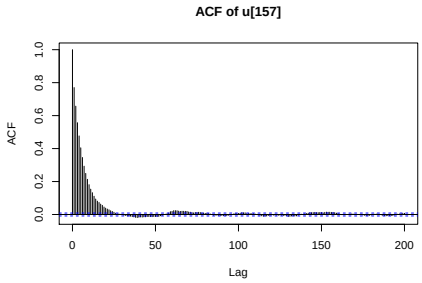

Figure 8: Autocorrelation plot for component 157 of **u**

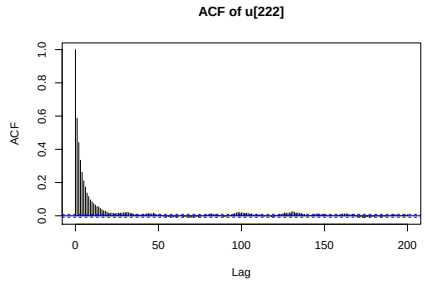

Figure 9: Autocorrelation plot for component 222 of **u**

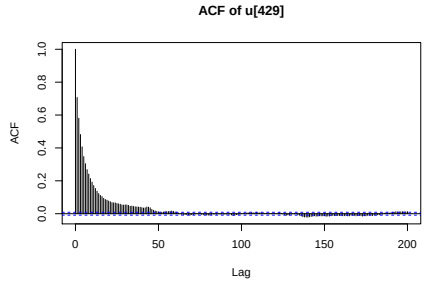

Figure 10: Autocorrelation plot for component 429 of **u**

From figure 8-10 we see that the three components of **u** show varying degree of autocorrelation. It takes roughly 50 iterations to reduce the autocorrelation of the three components to non-significance. Even so, the components are less correlated than κ_v.

As a final diagnostic, we conduct Geweke's Diagnostic to the three components.

```
gw = geweke.diag(df5)
cat("zscores:", gw$z, "\n")
cat("pvals:", 2 * (1 - pnorm(abs(gw$z))))
```

```
## zscores: -0.8298669 -1.666108 -0.1369014
```

```
## pvals: 0.406614 0.09569198 0.8911087
```

p-values of 0.41, 0.10 and 0.89 indicate that the chain did converge during the first 10% iterations only for all the three components in consideration.

```
df4 = data.frame()
for (i in 1:3) {
    index = random_indexes[i]
    df = data.frame(v = eta[, index] - u[, index], N = 1:N_iter, shape = as.factor(i),
        col = as.factor(index))
    df4 = rbind(df4, df)
}
ggplot(df4, aes(x = N, y = v, shape = shape, col = col)) + geom_point() +
    scale_shape(solid = F) + guides(shape = F) + ggtitle("Trace plot for v")
```

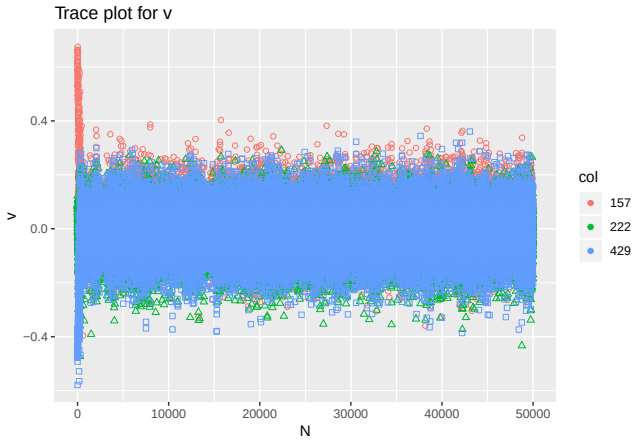

Figure 11: Trace plot for three random components of $\mathbf{v}$.

From the trace plots in figure 11 of the three components of $\mathbf{v}$ we observe only a small burn-in period.

```
acf(eta[, random_indexes[1]] - u[, random_indexes[1]], lag.max = 200)
acf(eta[, random_indexes[2]] - u[, random_indexes[2]], lag.max = 200)
acf(eta[, random_indexes[3]] - u[, random_indexes[3]], lag.max = 200)
```

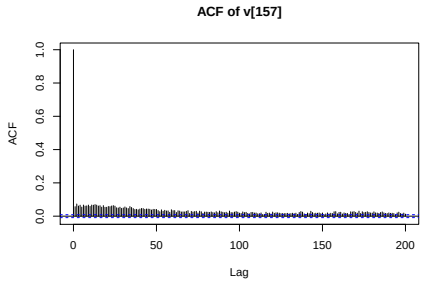

Figure 12: Autocorrelation plot for component 157 of $\mathbf{v}$

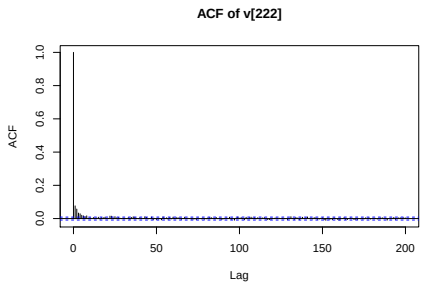

Figure 13: Autocorrelation plot for component 222 of $\mathbf{v}$

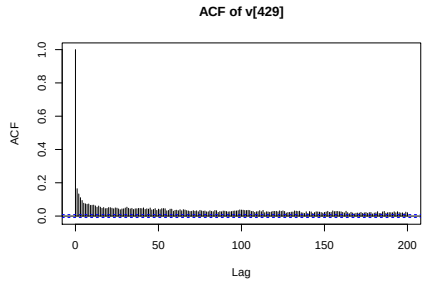

Figure 14: Autocorrelation plot for component 429 of $\mathbf{v}$

From figures 12-14 we see that the three components of $\mathbf{v}$ show varying degree of autocorrelation. As it takes roughly 100 iterations to reduce the autocorrelation to non-significance we observe that the three components are more autocorrelated than the components of $\mathbf{u}$, but still less correlated than κ_v.

Finally, the Geweke Diagnostic for the three components of $\mathbf{v}$ shows the following.

```
df7 = data.frame(v = eta[, random_indexes] - u[, random_indexes])
gw = geweke.diag(df7)
cat("zscores:", gw$z, "\n")
cat("pvals:", 2 * (1 - pnorm(abs(gw$z))))
```

```
## zscores: 0.9498505 -0.3609631 -1.162396
```

```
## pvals: 0.3421882 0.7181271 0.2450748
```

p-values of 0.34, 0.72 and 0.25 indicate that the chain converges during the first 10% iterations for all three components in consideration.

We have seen that we have a relative short burn-in period. To be safe, we consider the first 1000 iterations as burn-in when doing estimation. All in all, we have seen no evidence that our MCMC algorithm did not converge, nor do we have any prior belief that our problem might contain several modes. We therefore conclude that our Markov chain did converge. Still, most of the samples are highly autocorrelated, which is a critical flaw. High autocorrelation decreases the performance of the MCMC algorithm, implying that we have to run several iterations of the algorithm to produce the equivalent of an independent sample. Despite this weakness, the chain has in fact converged, and we believe it can be used.

Exercise 4 (Effective sample size)

To assess the quality of the samples from our Markov chain, we calculate the effective sample size for the precision parameters κ_u and κ_v by using the function `effectiveSize()` from the `coda` library.

```
cat("ESS of kappa u: ", effectiveSize(kappau), "\n")
cat("ESS of kappa_v: ", effectiveSize(kappav))
```

```
## ESS of kappa u:  4969.173
```

```
## ESS of kappa_v:  450.0652
```

The ESS is the estimated number of independent samples needed to obtain a parameter estimate with the same precision as the MCMC estimate based on N dependent samples. In our case, $N = 50000$, which means that the quality of our κ_u samples is the equivalent of 4969 independent samples. Our κ_v samples are the equivalent of 450 independent samples. This means that if we want very accurate estimates of the posterior marginals of κ_u we will have to run the MCMC chain for a very long time. The κ_u samples are better than the κ_v samples, which also could be seen by the fact that the κ_u samples are much less correlated than the κ_v samples.

To improve the ESS of our MCMC algorithm we need to decrease the autocorrelation of the samples in our algorithm. One method is to implement blocking, which is done by sampling some or all of the parameters together. In this case it would be natural to sample κ_u and $\mathbf{u}$ together and κ_v and $\boldsymbol{\eta}$ together. We would then accept or reject $(\kappa_u, \mathbf{u})$ / $(\kappa_v, \boldsymbol{\eta})$ as if it were a single sample. To implement this we would have to use the joint density of $(\kappa_u, \mathbf{u})$ and $(\kappa_v, \boldsymbol{\eta})$. Naturally, sampling from these densities increases the complexity of the algorithm.

It could also be considered to use a more sophisticated approximation of the posterior marginal density of $\boldsymbol{\eta}$. However, we observe from figure 15 that the observed acceptance probabilities used for sampling $\boldsymbol{\eta}$, except for the burn-in period, are reasonably high.

```
df90 = data.frame(alpha = accept_prob, N = seq(1, N_iter))
ggplot(df90, aes(x = N, y = alpha, col = "acceptance probability")) +
    geom_line() + geom_smooth(aes(col = "smooth approx")) + ylab("r")
```

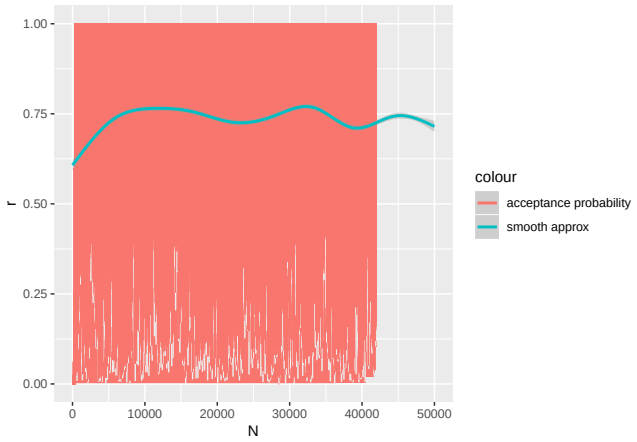

Figure 15: Acceptance probabilities for sampling $\boldsymbol{\eta}$.

We also compute the relative ESS, which is the ESS divided by the running time of the MCMC algorithm. The relative ESS can be interpreted as a measure for the number of independent samples produced by the algorithm per second. We get the following values for the relative ESS of κ_u and κ_v.

```
running_time = sum(times)

cat("Relative ESS of kappa u: ", effectiveSize(kappau)/running_time,
    "\n")
cat("Relative ESS of kappa v: ", effectiveSize(kappav)/running_time)
```

```
## Relative ESS of kappa u:  2.148294
```

```
## Relative ESS of kappa v:  0.1945742
```

This means that the MCMC algorithm produces the equivalent of 2.1482944 independent samples for κ_u per second, and the equivalent of 0.1945742 independent samples for κ_v per second. Also, to get one independent sample of κ_u, we need to run the algorithm for 0.47 seconds, while to get one independent sample of κ_v, we need to run the algorithm for 5.14 seconds.

Exercise 5 (Interpretation of results)

Finally, we plot the posterior median of $\exp(\mathbf{u})$ for all regions using the function `germany.plot()`.

```
post_median_u = apply(exp(u[1000:N_iter, ]), 2, median)

germany.plot(post_median_u, col = col, legend = TRUE)
```

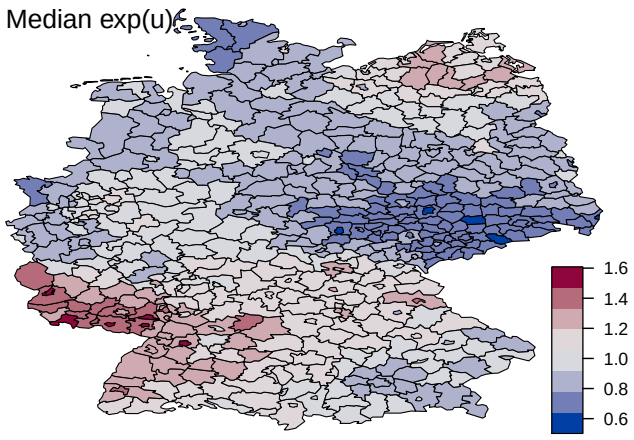

Figure 16: Posterior median of exp(**u**) for all regions in Germany.

We see from figure 16 that we have some spatial structure on the variable **u**. We get high values of exp(**u**) in the south-west regions and somewhat high values in regions in the north-east. Additionally, we get small values of exp(**u**) in the south-east, north-west, north and middle-east around the cities of Munich, Düsseldorf, Kiel and Leipzig. We also see some clear deviation from the standardised mortality rates in figure 1, especially in the south-west. exp(**u**) is also smoother than the SMR from figure 1.

Exercise 6 (Comparison of INLA and inclusion of covariate information)

a)

We now want to implement the same model using `R-INLA`. We set the same priors for κ_u and κ_v. The spatial structure requires the path to the geographical map of Germany. We therefore load the path to the Germany map.

```
library(INLA)
g = system.file("demodata/germany.graph", package = "INLA")
```

Then we use `R-INLA` to compute posterior marginals obtained by INLA for κ_u, κ_v and three ramdomly chosen components of **u** and $\boldsymbol{\eta}$. We can also get improved estimates of the posterior marginals for the precision parameters by applying `inla.hyperpar(result)` on the original INLA result object.

```
# Assert that Oral has region column
Oral$region = seq(1:length(Oral$Y))

# Create extra columns for INLA
Oral$region2 = Oral$region
```

```
# Create kappau prior
kappauprior = list(prior = "loggamma", param = c(alpha_u, beta_u))
# Create kappav prior
kappavprior = list(prior = "loggamma", param = c(alpha_v, beta_v))
```

```
# eta = u(region) + v(region)
formula <- Y ~ f(region, model = "besag", graph = g, hyper = list(prec = kappauprior),
    constr = T) + f(region2, model = "iid", hyper = list(prec = kappavprior))

# Oral$Y = Oral$E * eta
result <- inla(formula, family = "poisson", E = Oral$E, data = Oral,
    control.compute = list(dic = TRUE))

# We can use inla.hyperpar to get improved estimates of the marginals
# of the hyperparameters kappa_u and kappa_v
result_improved = inla.hyperpar(result)
```

```
# We also print time used by INLA:
cat("Time used by INLA: \n")
result$cpu.used
```

```
## Time used by INLA:
```

```
##  Pre-processing    Running inla Post-processing           Total
##       1.7783301       1.8221521       0.1239889       3.7244711
```

Having obtained the posterior marginals from INLA, we compare these estimates to the one obtained by MCMC, and plot the difference of $\exp(\mathbf{u})$ on the map of Germany.

```
germany.plot(exp(result$summary.random$region$`0.5quant`) - post_median_u,
    col = col, legend = TRUE)
```

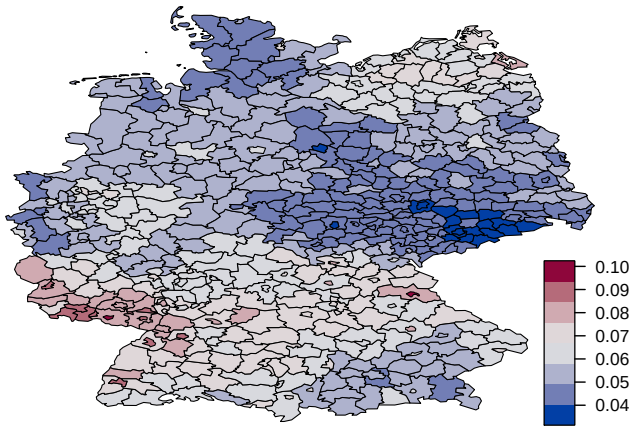

Figure 17: The difference between the MCMC estimate of the posterior median of $\exp(\mathbf{u})$ and the INLA estimate of $\exp(\mathbf{u})$ for all regions in Germany.

From figure 17, we see that we have reasonably small differences between the computed $\exp(\mathbf{u})$ from MCMC and INLA, with some larger outliers. However, we observe no negative differences, indicating that INLA might overestimate the effect of the spatial effect $\mathbf{u}$.

Now, we want to compare histograms of the MCMC-samples and the posterior marginals obtained by INLA for both precision parameters κ_u and κ_v, and the same for the three randomly chosen components of $\mathbf{u}$ and $\mathbf{v}$. We note that improved estimates of the posterior marginals for the precision parameters can be obtained by applying `inla.hyperpar(result)` on the original INLA results object.

```
df60 = data.frame(result$marginals.hyperpar$`Precision for region`)
df61 = data.frame(x = kappau[1000:length(kappau)])
```

```
df62 = data.frame(result_improved$marginals.hyperpar$`Precision for region`)

ggplot() + geom_histogram(data = df61, aes(x = x, y = ..density.., col = "MCMC"),
    bins = 100) + geom_line(data = df60, aes(x = x, y = y, col = "INLA")) +
    geom_line(data = df62, aes(x = x, y = y, col = "INLA - improved")) +
    xlim(6, 34) + xlab("kappa u")
```

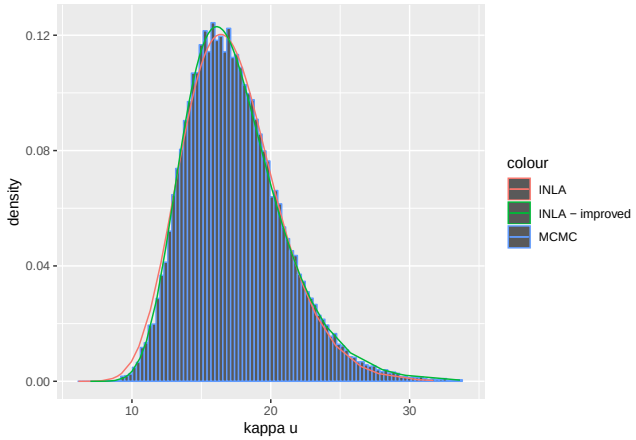

Figure 18: Comparison of computed marginal densities by INLA and histogram of MCMC sample for κ_u.

We see from figure 18 that all the three distributions are fairly similar. Still, the improved INLA estimate distribution is closer to the MCMC sample than the original INLA distribution.

```
df62 = data.frame(result$marginals.hyperpar$`Precision for region2`)
df63 = data.frame(x = kappav[1000:length(kappav)])
df64 = data.frame(result_improved$marginals.hyperpar$`Precision for region2`)

ggplot() + geom_histogram(data = df63, aes(x = x, y = ..density.., col = "MCMC"),
    bins = 100) + geom_line(data = df62, aes(x = x, y = y, col = "INLA")) +
    geom_line(data = df64, aes(x = x, y = y, col = "INLA - improved")) +
    xlim(40, 550) + xlab("kappa v")
```

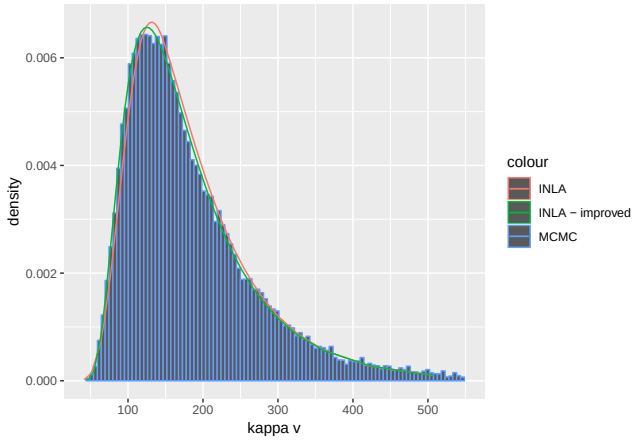

Figure 19: Comparison of computed densities by INLA and histogram of MCMC sample for κ_v.

Also in figure 19 we see that the three distributions of κ_v are fairly similar, with improved INLA closer to the

MCMC sample than the original INLA.

```
index = random_indexes[1]
df1 = data.frame(result$marginals.random$region[index], idx = index)
colnames(df1) = c("x", "y", "idx")
df1_imp = data.frame(result_improved$marginals.random$region[index],
    idx = index)
colnames(df1_imp) = c("x", "y", "idx")
df1_samp = data.frame(x = u[, index], idx = index)

index = random_indexes[2]
df2 = data.frame(result$marginals.random$region[index], idx = index)
colnames(df2) = c("x", "y", "idx")
df2_imp = data.frame(result_improved$marginals.random$region[index],
    idx = index)
colnames(df2_imp) = c("x", "y", "idx")
df2_samp = data.frame(x = u[, index], idx = index)

index = random_indexes[3]
df3 = data.frame(result$marginals.random$region[index], idx = index)
colnames(df3) = c("x", "y", "idx")
df3_imp = data.frame(result_improved$marginals.random$region[index],
    idx = index)
colnames(df3_imp) = c("x", "y", "idx")
df3_samp = data.frame(x = u[, index], idx = index)

df = rbind(df1, df2, df3)
df_imp = rbind(df1_imp, df2_imp, df3_imp)
df_samp = rbind(df1_samp, df2_samp, df3_samp)

ggplot() + geom_line(data = df, aes(x = x, y = y, col = "INLA")) + geom_line(data = df_imp,
    aes(x = x, y = y, col = "INLA - improved")) + geom_histogram(data = df_samp,
    aes(x = x, y = ..density.., col = "MCMC"), bins = 100) + facet_wrap(~idx) +
    ylab("density") + xlim(-1, 1)
```

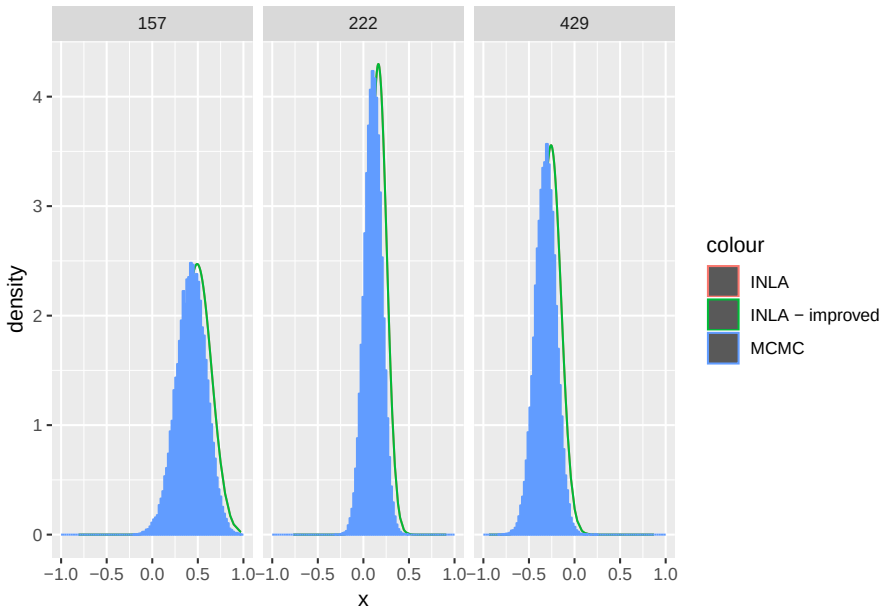

Figure 20: Comparison of computed densities by INLA and histogram of MCMC sample for three randomly chosen components of **u**.

For the three components of **u** in figure 20 we see that the improved INLA marginals coincide with the original INLA marginals. The INLA marginals seem to be somewhat positively shifted compared to the histogram of the MCMC samples, indicating that INLA may overestimate the effect of the spatial effect **u**.

```
inla_v = result$marginals.random$region2
inla_v_improved = result$marginals.random$region2

index = random_indexes[1]
df1 = data.frame(inla_v[index], idx = index)
colnames(df1) = c("x", "y", "idx")
df1_imp = data.frame(inla_v_improved[index], idx = index)
colnames(df1_imp) = c("x", "y", "idx")
df1_samp = data.frame(x = eta[, index] - u[, index], idx = index)

index = random_indexes[2]
df2 = data.frame(inla_v[index], idx = index)
colnames(df2) = c("x", "y", "idx")
df2_imp = data.frame(inla_v_improved[index], idx = index)
colnames(df2_imp) = c("x", "y", "idx")
df2_samp = data.frame(x = eta[, index] - u[, index], idx = index)

index = random_indexes[3]
df3 = data.frame(inla_v[index], idx = index)
colnames(df3) = c("x", "y", "idx")
df3_imp = data.frame(inla_v_improved[index], idx = index)
colnames(df3_imp) = c("x", "y", "idx")
df3_samp = data.frame(x = eta[, index] - u[, index], idx = index)

df = rbind(df1, df2, df3)
df_imp = rbind(df1_imp, df2_imp, df3_imp)
df_samp = rbind(df1_samp, df2_samp, df3_samp)
```

```
ggplot() + geom_histogram(data = df_samp, aes(x = x, y = ..density..,
    col = "MCMC"), bins = 100) + geom_line(data = df, aes(x = x, y = y,
    col = "INLA")) + geom_line(data = df_imp, aes(x = x, y = y, col = "INLA - improved")) +
    facet_wrap(~idx) + ylab("density")
```

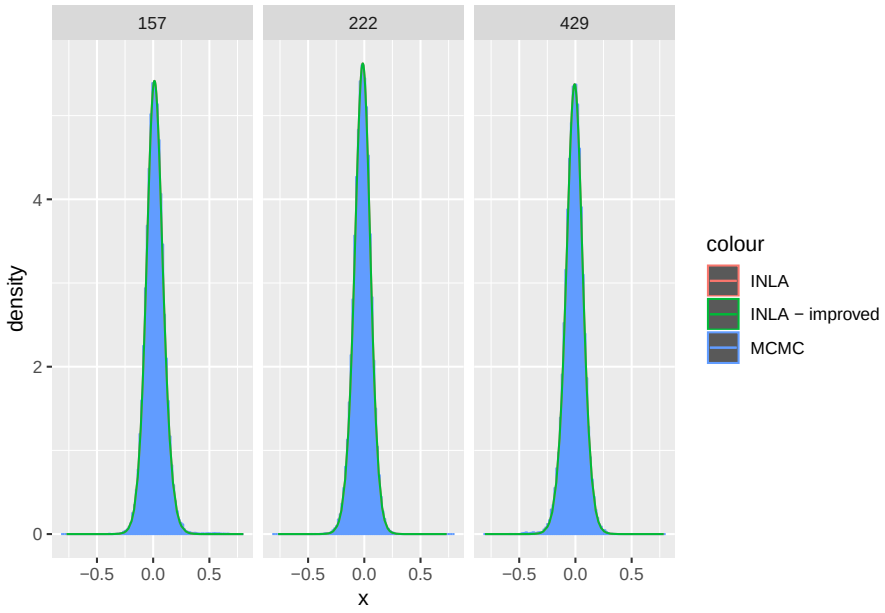

Figure 21: Comparison of computed densities by INLA and histogram of MCMC sample for three randomly chosen components of $\mathbf{v}$.

For the three components of $\mathbf{v}$ in figure 21 we see that the improved INLA coincides with the original INLA. Also, the marginal densities from INLA coincide with the histograms of the MCMC samples.

```
inla_eta = result$marginals.linear.predictor
inla_eta_improved = result$marginals.linear.predictor

index = random_indexes[1]
df1 = data.frame(inla_eta[index], idx = index)
colnames(df1) = c("x", "y", "idx")
df1_imp = data.frame(inla_eta_improved[index], idx = index)
colnames(df1_imp) = c("x", "y", "idx")
df1_samp = data.frame(x = eta[, index], idx = index)

index = random_indexes[2]
df2 = data.frame(inla_eta[index], idx = index)
colnames(df2) = c("x", "y", "idx")
df2_imp = data.frame(inla_eta_improved[index], idx = index)
colnames(df2_imp) = c("x", "y", "idx")
df2_samp = data.frame(x = eta[, index], idx = index)

index = random_indexes[3]
df3 = data.frame(inla_eta[index], idx = index)
colnames(df3) = c("x", "y", "idx")
df3_imp = data.frame(inla_eta_improved[index], idx = index)
colnames(df3_imp) = c("x", "y", "idx")
```

```
df3_samp = data.frame(x = eta[, index], idx = index)

df = rbind(df1, df2, df3)
df_imp = rbind(df1_imp, df2_imp, df3_imp)
df_samp = rbind(df1_samp, df2_samp, df3_samp)

ggplot() + geom_histogram(data = df_samp, aes(x = x, y = ..density..,
    col = "MCMC"), bins = 100) + geom_line(data = df, aes(x = x, y = y,
    col = "INLA")) + geom_line(data = df_imp, aes(x = x, y = y, col = "INLA - improved")) +
    facet_wrap(~idx) + ylab("density")
```

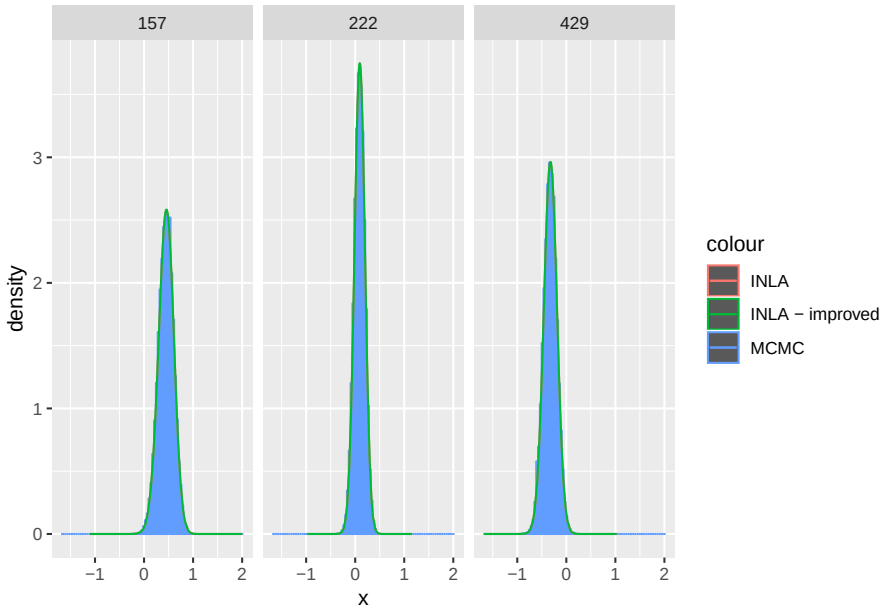

Figure 22: Comparison of computed densities by INLA and histogram of MCMC sample for three randomly chosen components of $\boldsymbol{\eta}$.

We note that even though INLA gives positively shifted estimates for $\mathbf{u}$, the INLA marginals of the three components of $\boldsymbol{\eta} = \mathbf{u} + \mathbf{v}$ perfectly overlap with the MCMC histograms, see figure 22.

In general, INLA and MCMC produce results that are at least qualitatively comparable. For κ_u, κ_v and $\boldsymbol{\eta}$, the resulting marginal densities perfectly overlap. For the spatial effect $\mathbf{u}$ we observe some positive shift in the INLA estimate. Worth noting is that the MCMC sampler uses 39 minutes, while INLA uses 3.7 seconds to run. Because they yield very similar results, we see that INLA is vastly superior if time consumption is a significant restriction.

b)

We now want to extend the INLA model formulation to incorporate a smoking covariate. First, we load information about smoking in each region.

```
# Attach smoking to Oral dataset
smoking = read.table("smoking.dat")
Oral$smoking = as.vector(smoking)$V1
```

Then, we make two new models. The first has smoking as a linear effect, and the second has smoking as a non-linear function using random walk of second order.

```r
# Add smoking as a linear covariate
formula_2 <- Y ~ f(region, model = "besag", graph = g, hyper = list(prec = kappauprior),
    constr = T) + f(region2, model = "iid", hyper = list(prec = kappavprior)) +
    f(smoking, model = "linear")

result_2 = inla(formula_2, family = "poisson", E = Oral$E, data = Oral,
    control.compute = list(dic = TRUE))

# Add smoking as a random walk of order 2
formula_3 <- Y ~ f(region, model = "besag", graph = g, hyper = list(prec = kappauprior),
    constr = T) + f(region2, model = "iid", hyper = list(prec = kappavprior)) +
    f(smoking, model = "rw2")

result_3 <- inla(formula_3, family = "poisson", E = Oral$E, data = Oral,
    control.compute = list(dic = TRUE))
```

We then compare the two extensions with the original model formulation by using the Deviance Information Criterion (DIC).

```r
cat("model without smoking:", result$dic$dic, "\n")
cat("model with smoking as linear component:", result_2$dic$dic, "\n")
cat("model with smoking as RW2:", result_3$dic$dic)
```

```
## model without smoking: 3286.91

## model with smoking as linear component: 3276.926

## model with smoking as RW2: 3276.924
```

We see from the printout that the two models with smoking included have lower DIC than the original model without smoking. Also, the linear and random walk model have almost the same DIC (difference of only 0.002). The DIC is a generalization of the Akaike Information Criterion, which maximizes likelihood while adding a penalty for the number of parameters. The best model is the model with lowest DIC. We observe that the inclusion of smoking in the model produces a reduction in loglikelihood that outweighs the parameter penalty, and can thus conclude that these models are better than the original.

However, we saw that the reduction for the random walk model and linear model were nearly equal. This means that between the flexible random-walk model and the somewhat rigid linear model, the reduction in loglikelihood permitted by the random-walk model is outweighed by the penalty on the number of parameters introduced. Thus, using DIC, we can argue that the linear and random-walk models are approximately equally good.

Finally, we plot the posterior median within 95% credible intervals of the non-linear covariate effect.

```r
df70 = data.frame(id = seq(1, length(result_3$summary.random$smoking$mean)),
    mean = result_3$summary.random$smoking$mean, median = result_3$summary.random$smoking$`0.5quant`,
    bot025 = result_3$summary.random$smoking$`0.025quant`, top975 = result_3$summary.random$smoking$`0.9

ggplot() + geom_ribbon(data = df70, aes(x = id, ymin = bot025, ymax = top975),
    alpha = 0.5) + geom_line(data = df70, aes(x = id, y = median, col = "median"),
    size = 1) + geom_line(data = df70, aes(x = id, y = bot025, col = "2.5 percentile"),
    size = 1) + geom_line(data = df70, aes(x = id, y = top975, col = "97.5 percentile"),
    size = 1) + ylab("f(smoking)") + xlab("smoking")
```

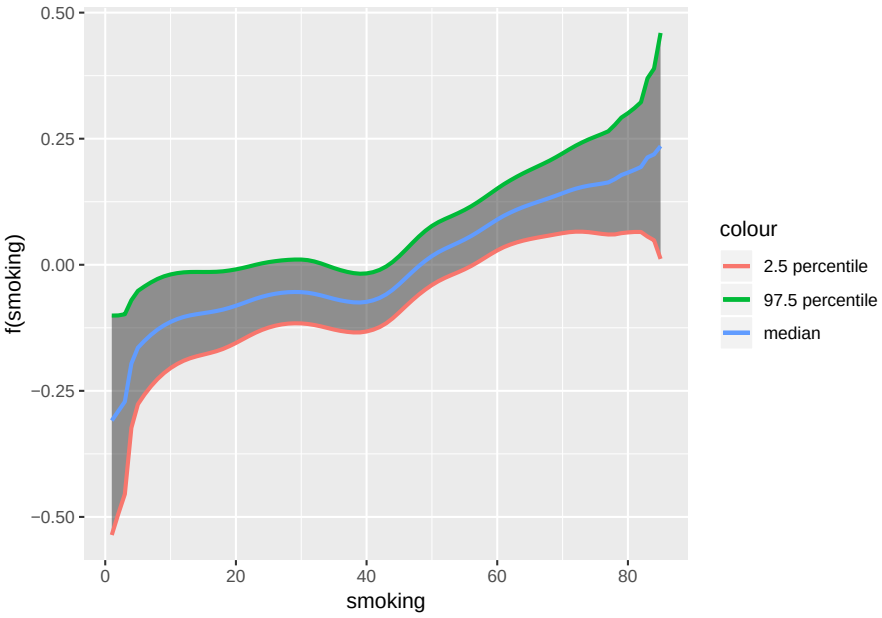

Figure 23: 95% Credible interval for posterior median of the non-linear covariate effect.

We observe that the credible interval for the random walk model does not contain the zero-hypothesis for all values of smoking but is very large. We also see that the credible interval of the random-walk smoking parameter is approximately linear. Comparing this with the DIC values for the two models we may hypothesize that the random-walk model will tend towards the linear model and that the true relationship is highly linear. In any case, the inclusion of smoking in the model yields a better model.