TMA4300Ex2

Karine H. Foss & August S. Mathisen

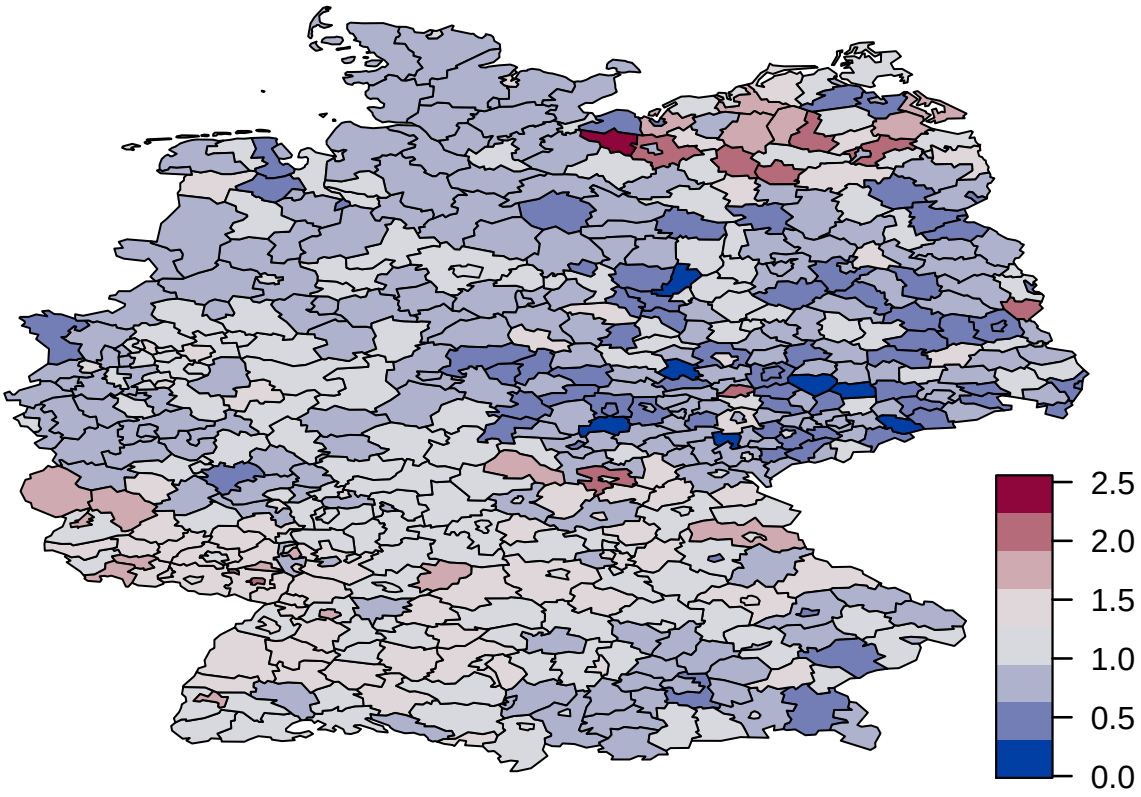
18 2 2019

Comments: * Need to explain better what $q(\eta)$ is. This is proposal distribution? * Which elements in the canonical form of mvnorm are the precision matrix etc? * ex3: says u and v in text; should be eta? * ex3: should we remove burn-in period before performing test? (and acf?) * ex5: says exp(u), but should be eta??

```
# Loading libraries
library(ggplot2)
library(gridExtra)
library(spam)  # load the data
str(Oral)  # see structure of data
```

```
## 'data.frame':    544 obs. of  3 variables:
##  $ Y  : int  18 62 44 12 18 27 20 29 39 21 ...
##  $ E  : num  16.4 45.9 44.7 16.3 26.9 ...
##  $ SMR: num  1.101 1.351 0.985 0.735 0.668 ...
```

```
# ???data.frame???: 544 obs. of 3 variables: $ Y : int 18 62 44 12 18
# 27 20 29 39 21 . . .  $ E : num 16.4 45.9 44.7 16.3 26.9 . . .  $
# SMR: num 1.101 1.351 0.985 0.735 0.668 . . .
attach(Oral)  # allow direct referencing to Y and E
# load some libraries to generate nice map plots
library(fields, warn.conflict = FALSE)
library(colorspace)


col <- diverge_hcl(8)  # blue - red
# use a function provided by spam to plot the map together with the
# mortality rates
germany.plot(Oral$Y/Oral$E, col = col, legend = TRUE)
```




```
load("additionalFiles/tma4300_ex2_Rmatrix.Rdata")

# Set seed so that the task can be reproduced
set.seed(42)
```

Exercise 1: Derivations

a)

From the definition of conditional probability and the nature of the assumptions in this exercise, we know that

$$\begin{aligned}\mathrm{p}(\boldsymbol{\eta}, \boldsymbol{u}, \kappa_u, \kappa_v|\boldsymbol{y}) &\propto \mathrm{p}(\boldsymbol{y}\,|\boldsymbol{\eta}, \boldsymbol{u}, \kappa_u, \kappa_v)\mathrm{p}(\boldsymbol{\eta}\,|\boldsymbol{u}, \kappa_u, \kappa_v)\mathrm{p}(\boldsymbol{u}\,|\kappa_u, \kappa_v)\mathrm{p}(\kappa_u\,|\kappa_v)\mathrm{p}(\kappa_v) \\ &\propto \mathrm{p}(\boldsymbol{y}\,|\boldsymbol{\eta})\mathrm{p}(\boldsymbol{\eta}\,|\boldsymbol{u}, \kappa_v)\mathrm{p}(\boldsymbol{u}\,|\kappa_u)\mathrm{p}(\kappa_u)\mathrm{p}(\kappa_v).\end{aligned}$$

By inserting the corresponding probabilities, this becomes

$$\begin{aligned}\mathrm{p} &\propto \left(\prod_{i=1}^{n}(E_i e^{\eta_i})^{y_i}\, e^{E_i e^{\eta_i}}\right) |\kappa_v \mathrm{I}|^{\frac{1}{2}}\, e^{-\frac{\kappa_v}{2}(\boldsymbol{\eta}-\boldsymbol{u})^T(\boldsymbol{\eta}-\boldsymbol{u})} \kappa_u^{(n-1)/2} e^{-\frac{\kappa_u}{2}\boldsymbol{u}^T\mathbf{R}\boldsymbol{u}} \kappa_u^{\alpha_u-1} e^{-\beta_u\kappa_u} \kappa_v^{\alpha_v-1} e^{-\beta_v\kappa_v} \\ &\propto \kappa_u^{\frac{n-1}{2}+\alpha_u-1} \kappa_v^{\frac{n}{2}+\alpha_v-1} \exp\left\{-\beta_u\kappa_u - \beta_v\kappa_v - \frac{\kappa_v}{2}(\boldsymbol{\eta}-\boldsymbol{u})^T(\boldsymbol{\eta}-\boldsymbol{u}) - \frac{\kappa_u}{2}\boldsymbol{u}^T\mathbf{R}\boldsymbol{u} + \sum_i (y_i\eta_i - E_i e^{\eta_i})\right\}.\end{aligned}$$

b)

The sum over e^{η_i} in the posterior means that the full conditional of η_i is difficult to sample from. We therefore want to approximate the distribution of $\boldsymbol{\eta}$ it with a multivariate normal in order to use Metropolis-Hastings steps for it. We define the function

$$f(\eta_i) = y_i\eta_i - E_i e^{\eta_i},$$

which has derivatives

$$\begin{aligned} f'(\eta_i) &= y_i - E_i e^{\eta_i} \\ f''(\eta_i) &= -E_i e^{\eta_i}. \end{aligned}$$

This yeilds the following Taylor series expansion of f around z_i,

$$\begin{aligned} \tilde{f}(\eta_i) &= y_i z_i - E_i e^{z_i} + (y_i - E_i e^{z_i})(\eta_i - z_i) + \frac{1}{2}(-E_i e^{z_i})(\eta_i - z_i)^2 \\ &= a(z_i) + b(z_i)\eta_i - \frac{1}{2}c(z_i)\eta_i^2, \end{aligned}$$

where $a(z_i) = E_i e^{z_i}(z_i - z_i^2/2 - 1)$, $b(z_i) = y_i + E_i e^{z_i}(z_i - 1)$ and $c(z_i) = E_i e^{z_i}$.

c)

As $\mathrm{p}(\theta_i|\boldsymbol{\theta}_{\setminus i}, \boldsymbol{y}) \propto \mathrm{p}(\boldsymbol{\theta}|\boldsymbol{y})$, we can quite easily find the full conditionals from the posterior. Using this, we see that

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

We recognise this as the core of a gamma distribution which means that the full conditional density of κ_u is gamma$(\frac{n-1}{2} + \alpha_u, \beta_u + \frac{1}{2}\boldsymbol{u}^T\mathbf{R}\boldsymbol{u})$. By the same reasoning, we see that gamma$(\frac{n}{2} + \alpha_v, \beta_v + \frac{1}{2}(\boldsymbol{\eta} - \boldsymbol{u})^T(\boldsymbol{\eta} - \boldsymbol{u}))$ is the full conditional density of κ_v. Similarly

$$\begin{aligned} \mathrm{p}(\boldsymbol{u}|\boldsymbol{y}, \boldsymbol{\eta}.\kappa_u, \kappa_v) &\propto \exp\left\{-\frac{\kappa_v}{2}(\boldsymbol{\eta} - \boldsymbol{u})^T(\boldsymbol{\eta} - \boldsymbol{u}) - \frac{\kappa_u}{2}\boldsymbol{u}^T\mathbf{R}\boldsymbol{u}\right\} \\ &\propto \exp\left\{-\frac{1}{2}\boldsymbol{u}^T(\kappa_u\mathbf{R} + \kappa_v\mathrm{I})\boldsymbol{u} + \kappa_v\boldsymbol{u}^T\boldsymbol{\eta}\right\}. \end{aligned}$$

We recognise this as the canonical form of a multivariate normal distribution. All these distributions are easy to sample from, and can be used in the Gibbs algorithm directly.

The full conditional distribution for $\boldsymbol{\eta}$ however takes the form

$$\mathrm{p}(\boldsymbol{\eta}|\boldsymbol{y}, \boldsymbol{u}, \kappa_u, \kappa_v) \propto \exp\left\{-\frac{\kappa_v}{2}(\boldsymbol{\eta} - \boldsymbol{u})^T(\boldsymbol{\eta} - \boldsymbol{u}) + \sum_i f(\eta_i)\right\}.$$

This does not correspond to any standard distribution, but by applying the approximation $\tilde{f}(\eta_i)$, we get

$$\begin{aligned} \mathrm{q}(\boldsymbol{\eta}|\boldsymbol{z}, \boldsymbol{y}, \boldsymbol{u}, \kappa_u, \kappa_v) &\propto \exp\left\{-\frac{\kappa_v}{2}\boldsymbol{\eta}^T\boldsymbol{\eta} + \kappa_v\boldsymbol{\eta}^T\boldsymbol{u} - \frac{1}{2}\boldsymbol{\eta}^T\mathrm{diag}(c(\boldsymbol{z}))\boldsymbol{\eta} + \boldsymbol{\eta}^T b(\boldsymbol{z})\right\} \\ &= \exp\left\{-\frac{1}{2}\boldsymbol{\eta}^T\Big(\kappa_v\mathbf{I} + \mathrm{diag}(c(\boldsymbol{z}))\Big)\boldsymbol{\eta} + \boldsymbol{\eta}^T(\kappa_u\boldsymbol{u} + b(\boldsymbol{z}))\right\}, \end{aligned}$$

where $\boldsymbol{z} = [z_1, ..., z_n]^T$ is the point around which we Taylor expand f, $b(\boldsymbol{z}) = [b(z_1), ..., b(z_n)]^T$ and $c(\boldsymbol{z}) = [c(z_1), ..., c(z_n)]^T$. q is the canonical form of a multivariate normal distribution, and can be used for Metropolis-Hastings steps for $\boldsymbol{\eta}$.

Exercise 2: Implementation of the MCMC sampler

Before we can implement the Metropolis-Hastings part of the sampler, we need to simplfy the expression for the acceptance propability α. If we first consider the ratio between true the full conditionals of $\boldsymbol{\eta}^*$ and $\boldsymbol{\eta}$ we can simply insert into the expression found in 1c)

$$\frac{\mathrm{p}(\boldsymbol{\eta}^*|\boldsymbol{y}, \boldsymbol{u}, \kappa_u, \kappa_v)}{\mathrm{p}(\boldsymbol{\eta}|\boldsymbol{y}, \boldsymbol{u}, \kappa_u, \kappa_v)} = \exp\left\{-\frac{\kappa_v}{2}\boldsymbol{\eta}^{*T}\boldsymbol{\eta}^* + \boldsymbol{\eta}^{*T}(\kappa_v\boldsymbol{u} + \boldsymbol{y}) - \exp(\boldsymbol{\eta}^*)^T\mathbf{E} + \frac{\kappa_v}{2}\boldsymbol{\eta}^T\boldsymbol{\eta} - \boldsymbol{\eta}^T(\kappa_v\boldsymbol{u} + \boldsymbol{y}) + \exp(\boldsymbol{\eta})^T\mathbf{E}\right\}.$$

Here $\boldsymbol{\eta}^*$ is the proposed m'th step, $\boldsymbol{\eta}$ the value of the $(m-1)$'th step while $\boldsymbol{u}$, κ_u and κ_v are the m'th step values. The ratio between the proposal distributions can also be found by insertion

$$\begin{aligned}\frac{\mathrm{q}(\boldsymbol{\eta}|\boldsymbol{\eta}^*, \boldsymbol{y}, \boldsymbol{u}, \kappa_u, \kappa_v)}{\mathrm{q}(\boldsymbol{\eta}^*|\boldsymbol{\eta}, \boldsymbol{y}, \boldsymbol{u}, \kappa_u, \kappa_v)} =& \frac{\left|\kappa_v\mathrm{I} + \mathrm{diag}(c(\boldsymbol{\eta}^*))\right|^{\frac{1}{2}}}{\left|\kappa_v\mathrm{I} + \mathrm{diag}(c(\boldsymbol{\eta}))\right|^{\frac{1}{2}}} \cdot \\ & \exp\Big\{-\frac{1}{2}\boldsymbol{\eta}^T\Big(\kappa_v\mathrm{I} + \mathrm{diag}(c(\boldsymbol{\eta}^*))\Big)\boldsymbol{\eta} + \boldsymbol{\eta}^T(\kappa_u\boldsymbol{u} + b(\boldsymbol{\eta}^*)) + \\ & \frac{1}{2}\boldsymbol{\eta}^{*T}\Big(\kappa_v\mathrm{I} + \mathrm{diag}(c(\boldsymbol{\eta}))\Big)\boldsymbol{\eta}^* - \boldsymbol{\eta}^{*T}(\kappa_u\boldsymbol{u} + b(\boldsymbol{\eta}))\Big\}.\end{aligned}$$

(dette er stygt, men jeg vet ikke hvordan det kan bli penere). Multiplying these two ratios and using the fact that $b(\boldsymbol{z}) = \boldsymbol{y} + \mathrm{diag}(c(\boldsymbol{z}))\boldsymbol{z} - c(\boldsymbol{z})$ and that $\Sigma c(\boldsymbol{z}) = \exp(\boldsymbol{z})^T\mathbf{E}$, we get

$$\alpha = \min\left\{1, \frac{\prod_i(\kappa_v + c(\eta_i^*))}{\prod_i(\kappa_v + c(\eta_i))}\exp\left[c(\boldsymbol{\eta})^T\Big(\mathrm{diag}(\boldsymbol{\eta}^*)(\frac{1}{2}\boldsymbol{\eta}^* - \boldsymbol{\eta}) + \vec{1}\Big) - c(\boldsymbol{\eta}^*)^T\Big(\mathrm{diag}(\boldsymbol{\eta})(\frac{1}{2}\boldsymbol{\eta} - \boldsymbol{\eta}^*) + \vec{1}\Big)\right]\right\}$$

```
# Define functions to be used in the MCMC

# Changed from z to exp(z) here
c_func = function(z, E) {
    exp(z) * E
}

b_func = function(z, y, E) {
    y + c_func(z, E) * (z - 1)
}

# Added 1/2 in the last term - taken away because it failed...?
drawKappaU = function(n, alpha_u, beta_u, u, R) {
    rgamma(1, (n - 1)/2 + alpha_u, rate = beta_u + t(u) %*% R %*% u)
}

drawKappaV = function(n, alpha_v, beta_v, eta, u) {
    rgamma(1, n/2 + alpha_v, rate = beta_v + t(eta - u) %*% (eta - u)/2)
}

# Dropped dividing with 2 in first term
```

```
drawU = function(n, kappa_u, kappa_v, eta, R) {
    t(rmvnorm.canonical(1, kappa_v * eta, kappa_u * R + diag.spam(kappa_v,
        n, n)))
}

# Added diag.spam(c_func(z,E)) in precision matrix
drawEta = function(n, z, y, kappa_v, kappa_u, u, E) {
    t(rmvnorm.canonical(1, kappa_u * u + b_func(z, y, E), diag.spam(as.vector(c_func(z,
        E))) + diag.spam(kappa_v, n, n)))
}

# Did not manage to check this...
calculateLogAcceptanceProb = function(etaProp, etaPrev, kappa_v, E) {
    c_prop = c_func(etaProp, E)
    c_prev = c_func(etaPrev, E)
    logProb = (sum(log(kappa_v + c_prop)) - sum(log(kappa_v + c_prev)))/2 +
        t(c_prev * etaProp) %*% (0.5 * etaProp - etaPrev) - t(c_prop *
        etaPrev) %*% (0.5 * etaPrev - etaProp) + sum(c_prev) - sum(c_prop)
}
```

```
# MCMC function
MCMCOral = function(M, y, starting_values, alpha_u, beta_u, alpha_v,
    beta_v, E, R) {
    # Get computing time
    time_start = Sys.time()

    # Fetching starting values
    eta_prev = starting_values[["eta"]]
    u = starting_values[["u"]]
    kappa_u = starting_values[["kappa_u"]]
    kappa_v = starting_values[["kappa_v"]]

    n = length(y)

    # Create matrices to contain all steps
    eta_values = matrix(nrow = n, ncol = M)
    u_values = matrix(nrow = n, ncol = M)
    kappa_u_vals = vector(length = M)
    kappa_v_vals = vector(length = M)

    eta_values[, 1] = eta_prev
    u_values[, 1] = u
    kappa_v_vals[1] = kappa_v
    kappa_u_vals[1] = kappa_u

    acceptance_rates = vector(length = M - 1)
    num_accepted = 0

    for (i in 2:M) {
        # Generate next step for u and kappas
        kappa_u = drawKappaU(n, alpha_u, beta_u, u, R)
        kappa_v = drawKappaV(n, alpha_v, beta_v, eta_prev, u)
        u = drawU(n, kappa_u, kappa_v, eta_prev, R)
```

```
        # prpose eta
        eta = drawEta(n, eta_prev, y, kappa_v, kappa_u, u, E)

        # generate acceptance variable
        decision_var = runif(1)

        # check for acceptance
        log_acceptance = calculateLogAcceptanceProb(eta, eta_prev, kappa_v,
            E)

        # Using log_acceptance further
        acceptance_rates[i] = min(0, log_acceptance)

        # Changed from log_acceptance to acceptance_rates[i]:
        if (log(decision_var) < acceptance_rates[i]) {
            eta_prev = eta
            num_accepted = num_accepted + 1
        }

        # Update value matrix
        eta_values[, i] = eta_prev
        u_values[, i] = u
        kappa_v_vals[i] = kappa_v
        kappa_u_vals[i] = kappa_u
    }
    time_end = Sys.time()
    # Return all values in a list
    return(list(eta = eta_values, u = u_values, kappa_u = kappa_u_vals,
        kappa_v = kappa_v_vals, acceptance_rates = acceptance_rates,
        num_accepted = num_accepted, time = time_end - time_start))
}
```

```
# Set parameters
M = 200
y = Oral$Y
E = Oral$E
alpha_u = 1
alpha_v = 1
beta_u = 0.01
beta_v = 0.01

# Set initial values
n = length(y)
u_start = matrix(0, ncol = 1, nrow = n)
eta_start = matrix(0, nrow = n, ncol = 1)
kappa_u_start = 100
kappa_v_start = 100
starting_values = list(eta = eta_start, u = u_start, kappa_u = kappa_u_start,
    kappa_v = kappa_v_start)
```

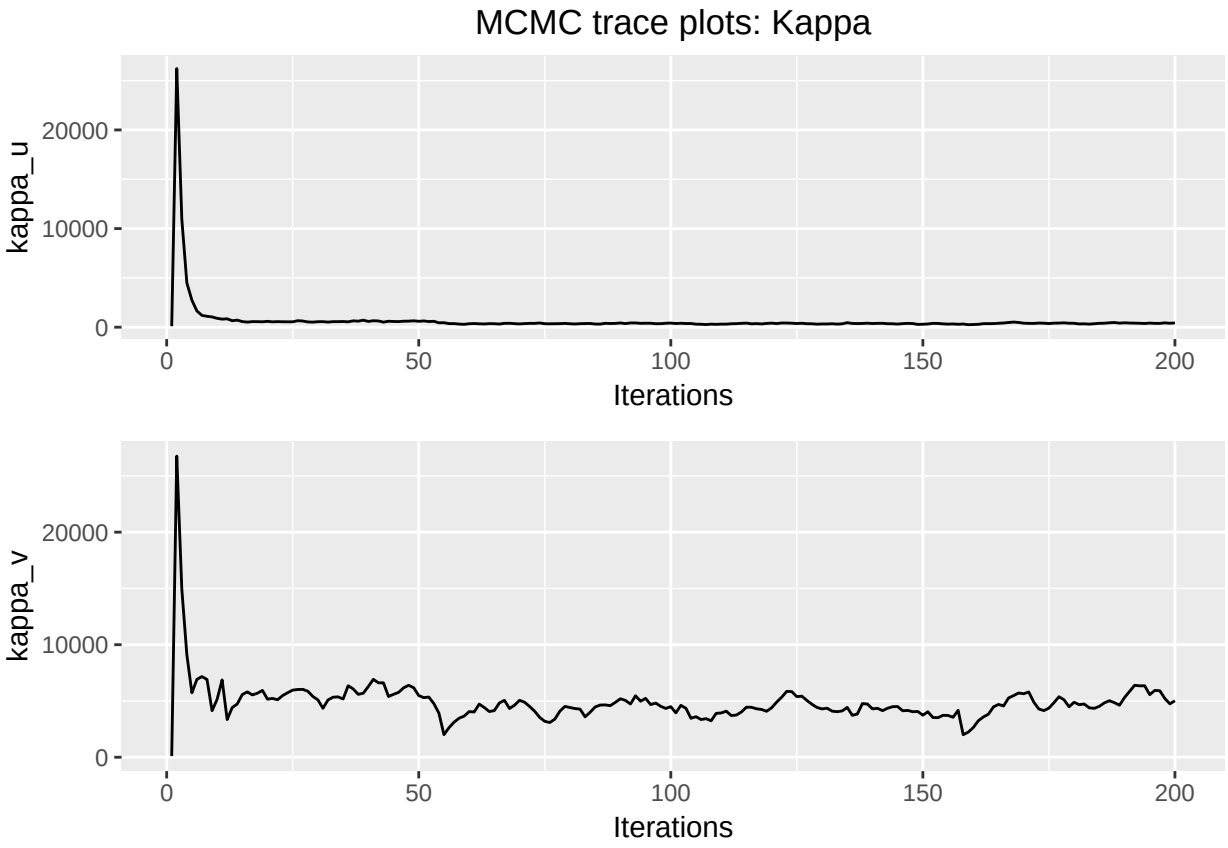
```
# Test function
set.seed(4300)
samples = MCMCOral(M, y, starting_values, alpha_u, beta_u, alpha_v, beta_v,
    E, R)
```

```
# Save and load, so we don't need to do it over and over again
# save(samples, file = 'MCMCsamples.RData', ascii=TRUE)
# load('MCMCsamples.RData') M = length(samples$kappa_u)
```

Exercise 3: Convergence diagnostics

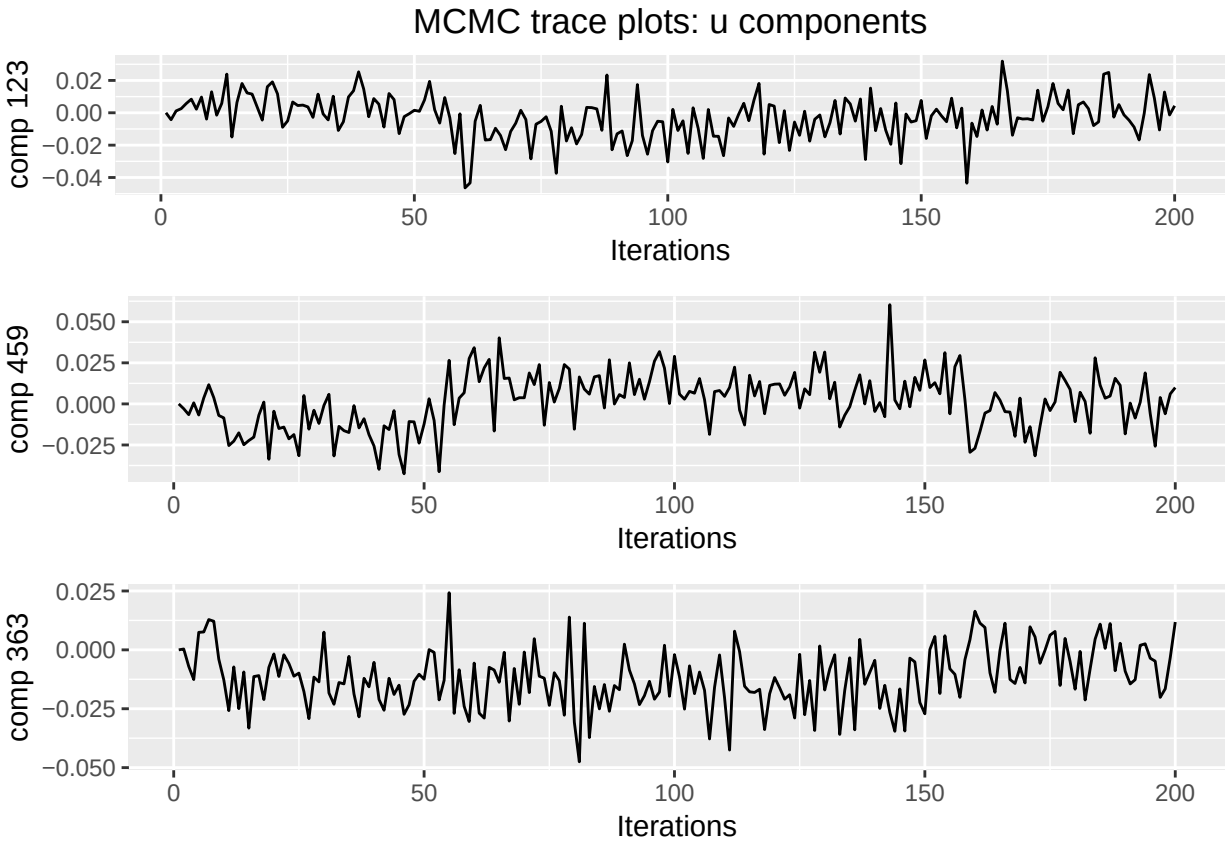
a) Trace plots

```
# Trace plots Kappa
trace.kappa_u = ggplot(data.frame(kappa = samples$kappa_u), aes(x = 1:M,
    y = kappa)) + geom_line() + ggtitle("MCMC trace plots: Kappa") +
    theme(plot.title = element_text(hjust = 0.5)) + xlab("Iterations") +
    ylab("kappa_u")

trace.kappa_v = ggplot(data.frame(kappa = samples$kappa_v), aes(x = 1:M,
    y = kappa)) + geom_line() + xlab("Iterations") + ylab("kappa_v")

trace.kappa_plot = grid.arrange(grobs = list(trace.kappa_u, trace.kappa_v),
    ncol = 1)
```

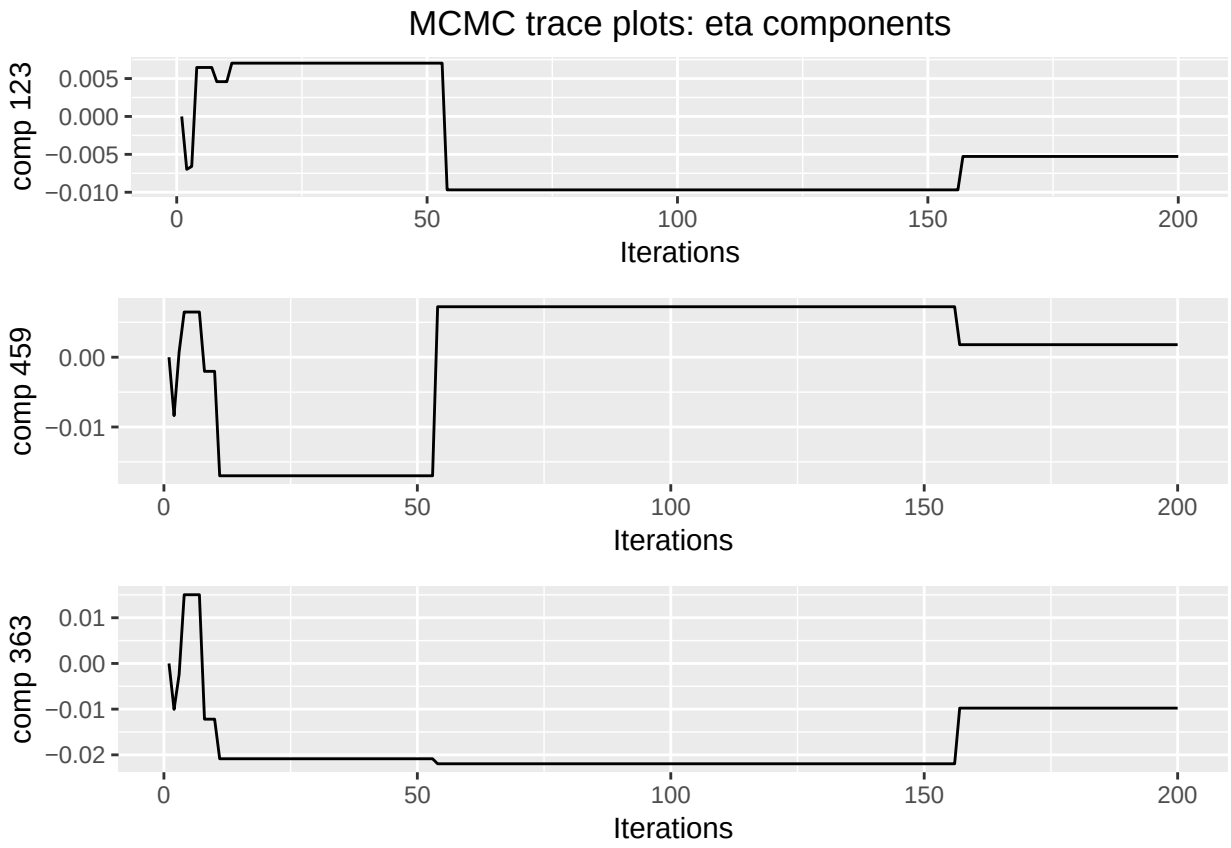



```
# Save trace plot
ggsave("figures/traceKappa.pdf", plot = trace.kappa_plot, device = NULL,
    path = NULL, scale = 1, width = 5.5, height = 2 * 4, units = "in",
    dpi = 300, limitsize = TRUE)
```

```r
# Trace plots somecomponents of u and eta Choose components:
set.seed(4301)
comp = ceiling(n * runif(3))
trace.u = list()
trace.eta = list()

# labels
lab = paste("comp", comp)

# Make first plots with title
trace.u[[1]] = ggplot(data.frame(u = samples$u[comp[1], ]), aes(x = 1:M,
    y = u)) + geom_line() + ggtitle("MCMC trace plots: u components") +
    theme(plot.title = element_text(hjust = 0.5)) + xlab("Iterations") +
    ylab(lab[1])

trace.eta[[1]] = ggplot(data.frame(eta = samples$eta[comp[1], ]), aes(x = 1:M,
    y = eta)) + geom_line() + ggtitle("MCMC trace plots: eta components") +
    theme(plot.title = element_text(hjust = 0.5)) + xlab("Iterations") +
    ylab(lab[1])

# Next plots do not need title
for (i in 2:length(comp)) {
    trace.u[[i]] = ggplot(data.frame(u = samples$u[comp[i], ]), aes(x = 1:M,
        y = u)) + geom_line() + xlab("Iterations") + ylab(lab[i])

    trace.eta[[i]] = ggplot(data.frame(eta = samples$eta[comp[i], ]),
        aes(x = 1:M, y = eta)) + geom_line() + xlab("Iterations") + ylab(lab[i])
}

trace.u_plot = grid.arrange(grobs = trace.u, ncol = 1)
```




```
trace.eta_plot = grid.arrange(grobs = trace.eta, ncol = 1)
```




```
# Save trace plots
ggsave("figures/traceU.pdf", plot = trace.u_plot, device = NULL, path = NULL,
    scale = 1, width = 5.5, height = 2 * 4, units = "in", dpi = 300,
    limitsize = TRUE)

ggsave("figures/traceEta.pdf", plot = trace.eta_plot, device = NULL,
    path = NULL, scale = 1, width = 5.5, height = 2 * 4, units = "in",
    dpi = 300, limitsize = TRUE)
```

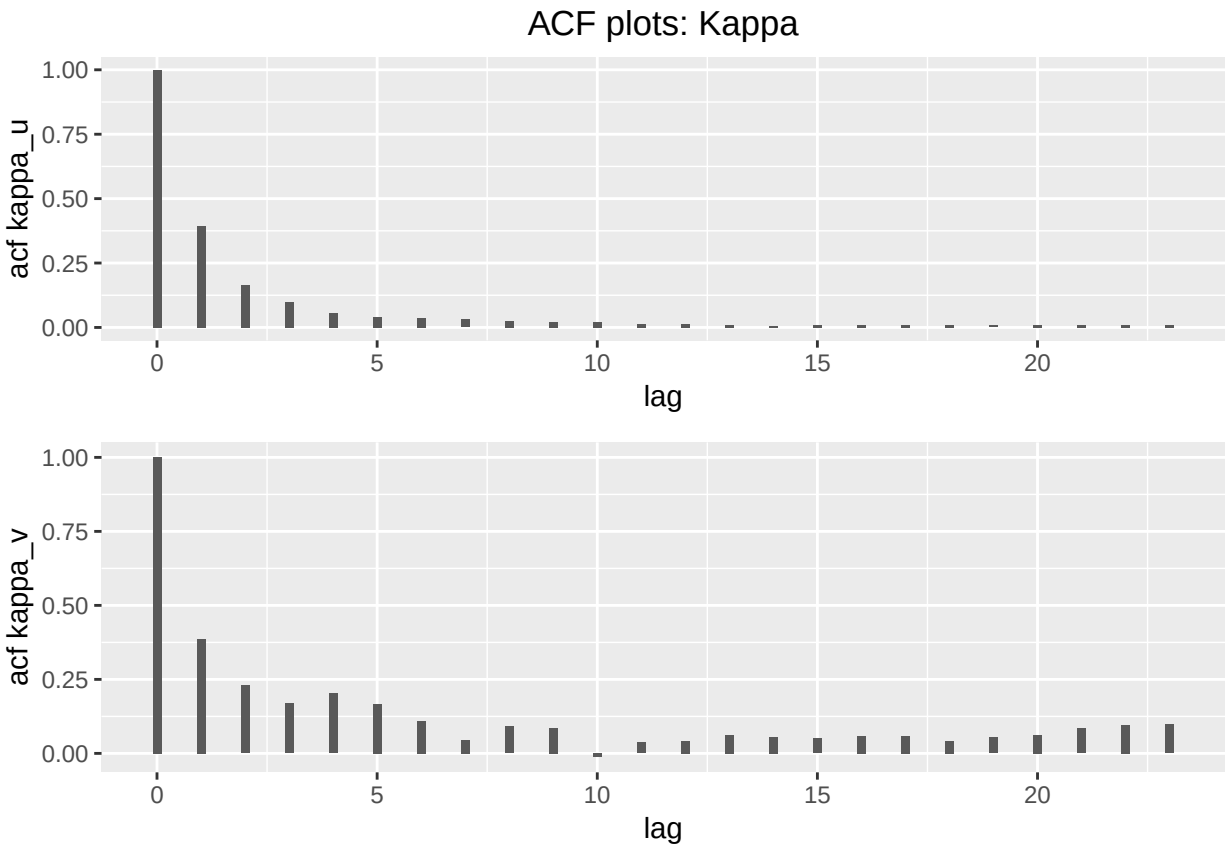
Remove burn-in here?

```
# Define burn-in area

# cut.u = cut.eta = cut.kappa_u = cut.kappa_v =

# Cut burn-in samples

# relevant.u = samples$u[,(cut.u+1):M] relevant.eta =
# samples$eta[,(cut.eta+1):M] relevant.kappa_u =
# samples$kappa_u[(cut.kappa_u+1):M] relevant.kappa_v =
# samples$kappa_v[(cut.kappa_v+1):M]
```

b) Autocorrelation plots

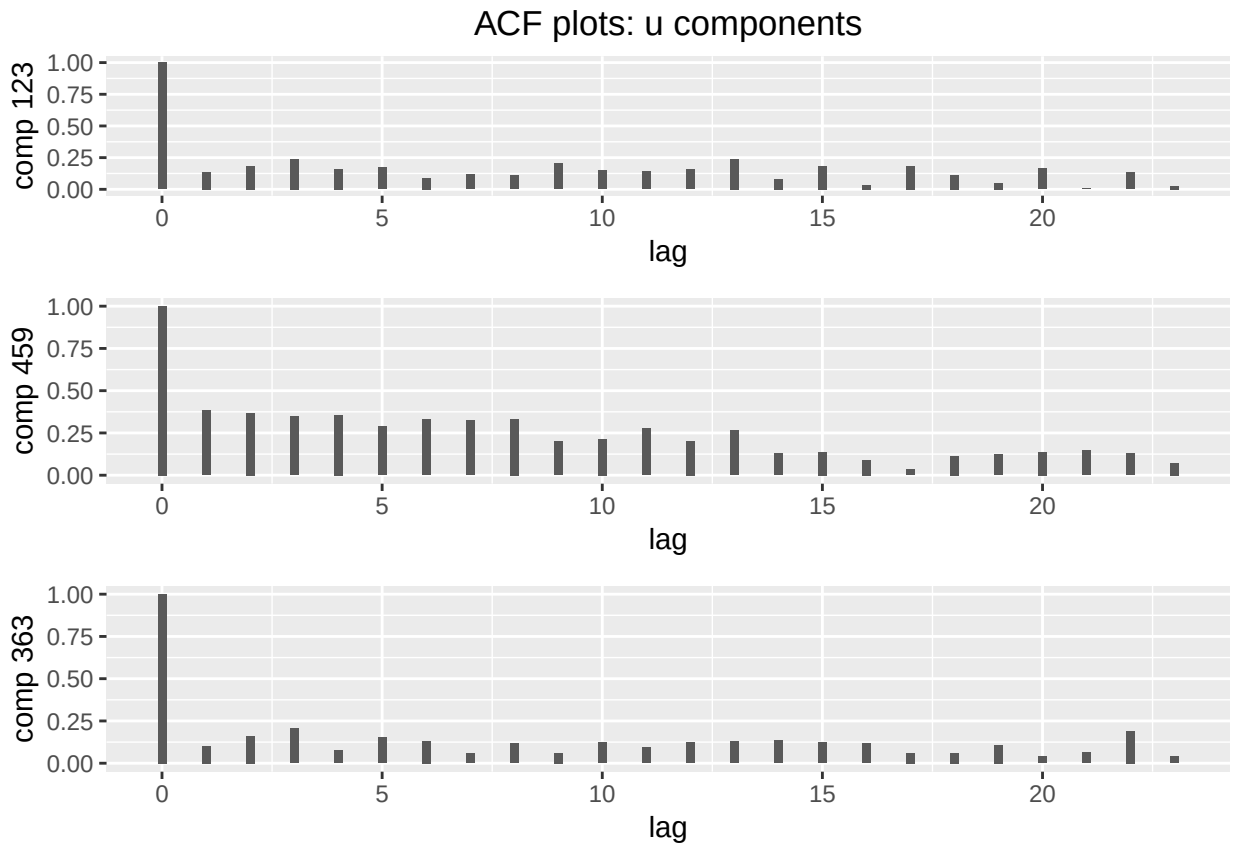
```
acf.kappa_u = acf(samples$kappa_u, plot = FALSE)
acf.kappa_v = acf(samples$kappa_v, plot = FALSE)
```

```
plot1 = ggplot(data.frame(acf = acf.kappa_u$acf, lag = acf.kappa_u$lag),
    aes(x = lag, y = acf)) + geom_col(width = 0.2) + ggtitle("ACF plots: Kappa") +
    theme(plot.title = element_text(hjust = 0.5)) + ylab("acf kappa_u")

plot2 = ggplot(data.frame(acf = acf.kappa_v$acf, lag = acf.kappa_v$lag),
    aes(x = lag, y = acf)) + geom_col(width = 0.2) + ylab("acf kappa_v")

acf.kappa_plot = grid.arrange(grobs = list(plot1, plot2), ncol = 1)
```

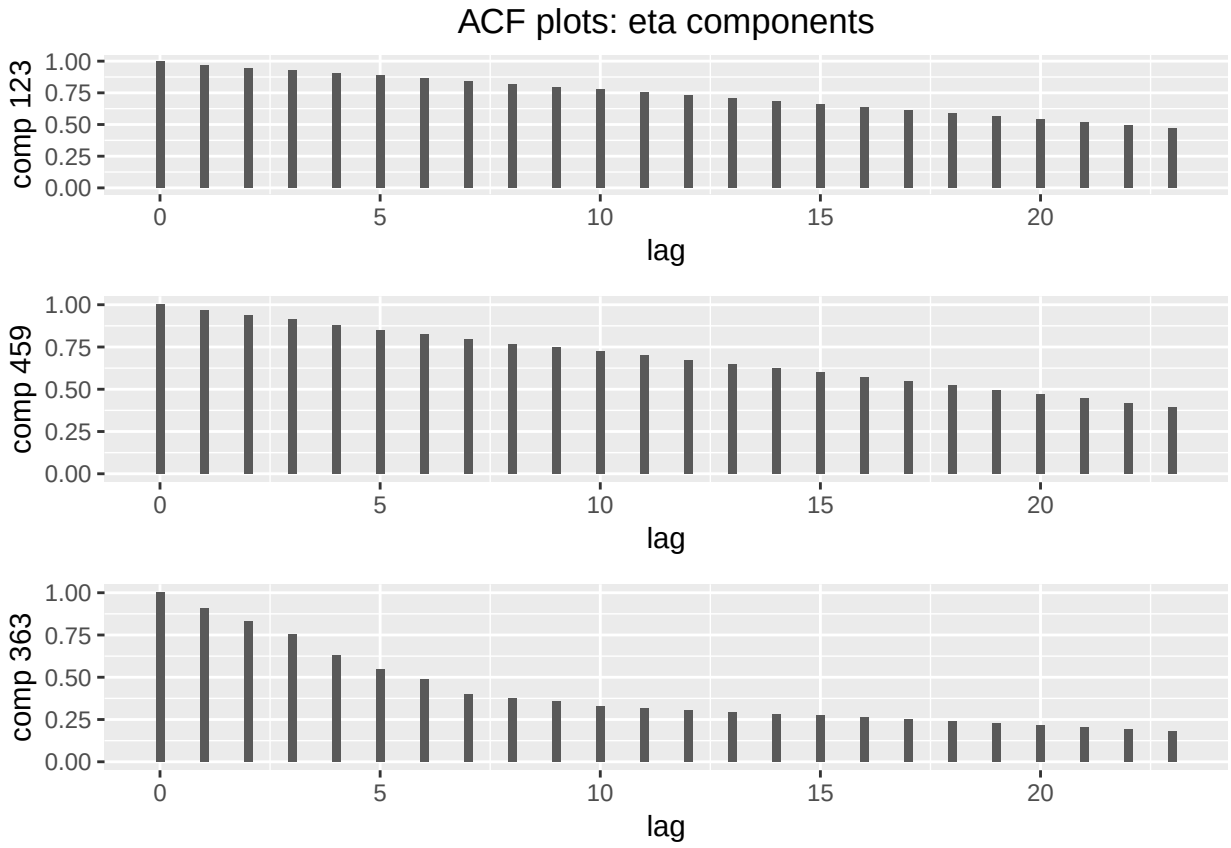



```
# acf for u and eta
plot_u = list()
plot_eta = list()

# Make first plots with title
acf.u = acf(samples$u[comp[1], ], plot = FALSE)
plot_u[[1]] = ggplot(data.frame(acf = acf.u$acf, lag = acf.u$lag), aes(x = lag,
    y = acf)) + geom_col(width = 0.2) + ggtitle("ACF plots: u components") +
    theme(plot.title = element_text(hjust = 0.5)) + ylab(lab[1])

acf.eta = acf(samples$eta[comp[1], ], plot = FALSE)
plot_eta[[1]] = ggplot(data.frame(acf = acf.eta$acf, lag = acf.eta$lag),
    aes(x = lag, y = acf)) + geom_col(width = 0.2) + ggtitle("ACF plots: eta components") +
    theme(plot.title = element_text(hjust = 0.5)) + ylab(lab[1])

# Next plots do not need title
for (i in 2:length(comp)) {
```

```
    acf.u = acf(samples$u[comp[i], ], plot = FALSE)
    plot_u[[i]] = ggplot(data.frame(acf = acf.u$acf, lag = acf.u$lag),
        aes(x = lag, y = acf)) + geom_col(width = 0.2) + ylab(lab[i])

    acf.eta = acf(samples$eta[comp[i], ], plot = FALSE)
    plot_eta[[i]] = ggplot(data.frame(acf = acf.eta$acf, lag = acf.eta$lag),
        aes(x = lag, y = acf)) + geom_col(width = 0.2) + ylab(lab[i])
}

acf.u_plot = grid.arrange(grobs = plot_u, ncol = 1)
```




```
acf.eta_plot = grid.arrange(grobs = plot_eta, ncol = 1)
```




```
# Save acf plots
ggsave("figures/acfKappa.pdf", plot = acf.kappa_plot, device = NULL,
    path = NULL, scale = 1, width = 5.5, height = 2 * 4, units = "in",
    dpi = 300, limitsize = TRUE)

# Save trace plots
ggsave("figures/acfU.pdf", plot = acf.u_plot, device = NULL, path = NULL,
    scale = 1, width = 5.5, height = 2 * 4, units = "in", dpi = 300,
    limitsize = TRUE)

ggsave("figures/acfEta.pdf", plot = acf.eta_plot, device = NULL, path = NULL,
    scale = 1, width = 5.5, height = 2 * 4, units = "in", dpi = 300,
    limitsize = TRUE)
```

Convergence?

geweke.diag() performs a test for convergence of the markov chain. **Guess we have to remove the burn-in?** It takes one fraction of the beginning of the chain and one fraction of the end. If the chain has converged to the stationary distribution, the two means should be equal. The geweke test statistic is asymptotically standard normal distributed. The null hypothesis is that the chain has converged.**?? Have I done it right?**

```
library(coda)
# Apply geweke test:
gew.kappa_u = geweke.diag(samples$kappa_u)
gew.kappa_v = geweke.diag(samples$kappa_v)
```

```r
gew.u1 = geweke.diag(samples$u[comp[1], ])
gew.u2 = geweke.diag(samples$u[comp[2], ])
gew.u3 = geweke.diag(samples$u[comp[3], ])

gew.eta1 = geweke.diag(samples$eta[comp[1], ])
gew.eta2 = geweke.diag(samples$eta[comp[2], ])
gew.eta3 = geweke.diag(samples$eta[comp[3], ])

# p-values
p_vals = c(2 * pnorm(abs(gew.kappa_u$z), lower.tail = FALSE), 2 * pnorm(abs(gew.kappa_v$z),
    lower.tail = FALSE), 2 * pnorm(abs(gew.u1$z), lower.tail = FALSE),
    2 * pnorm(abs(gew.u2$z), lower.tail = FALSE), 2 * pnorm(abs(gew.u3$z),
        lower.tail = FALSE), 2 * pnorm(abs(gew.eta1$z), lower.tail = FALSE),
    2 * pnorm(abs(gew.eta2$z), lower.tail = FALSE), 2 * pnorm(abs(gew.eta3$z),
        lower.tail = FALSE))
conclusion = ifelse(p_vals < 0.05, "->reject", "-> do not reject")
```

```
## kappa_u:
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  var1
## 1.839
```

```
## kappa_u: test stat 1.839021  p-value 0.06591209 -> do not reject
```

```
## kappa_v: test stat 2.049901 0.04037414 ->reject
```

```
## u comp 123 : test stat 3.593422 0.0003263634 ->reject
```

```
## u comp 459 : test stat -3.145238 0.00165952 ->reject
```

```
## u comp 363 : test stat 0.4786851 0.6321627 -> do not reject
```

```
## eta comp 123 : test stat 5.671468 1.415791e-08 ->reject
```

```
## eta comp 459 : test stat -1.673486 0.09423167 -> do not reject
```

```
## eta comp 363 : test stat 0.5719936 0.5673263 -> do not reject
```

Conclusion? Heheee, so far it is not correct...

Exercise 4: Effective sample set

```r
effSize.kappa_u = effectiveSize(samples$kappa_u)
effSize.kappa_v = effectiveSize(samples$kappa_v)
```

interpret the values These values mean that out of the M samples of the MCMC, we "effectively" have xxx samples from the posterior distributions.

How could the MCMC be changed to increas the effective sample size? Choose the initial values more carefully?

```r
comp_time = as.numeric(samples$time)  #Check that this always gives seconds
# comp_time = 100 #For testing when the time was not computed
```

```
relEffSize.kappa_u = effSize.kappa_u/comp_time
relEffSize.kappa_v = effSize.kappa_v/comp_time
```

```
## Relative effective sample size for kappa_u is  75.72542
```

```
## Relative effective sample size for kappa_v is  76.89644
```

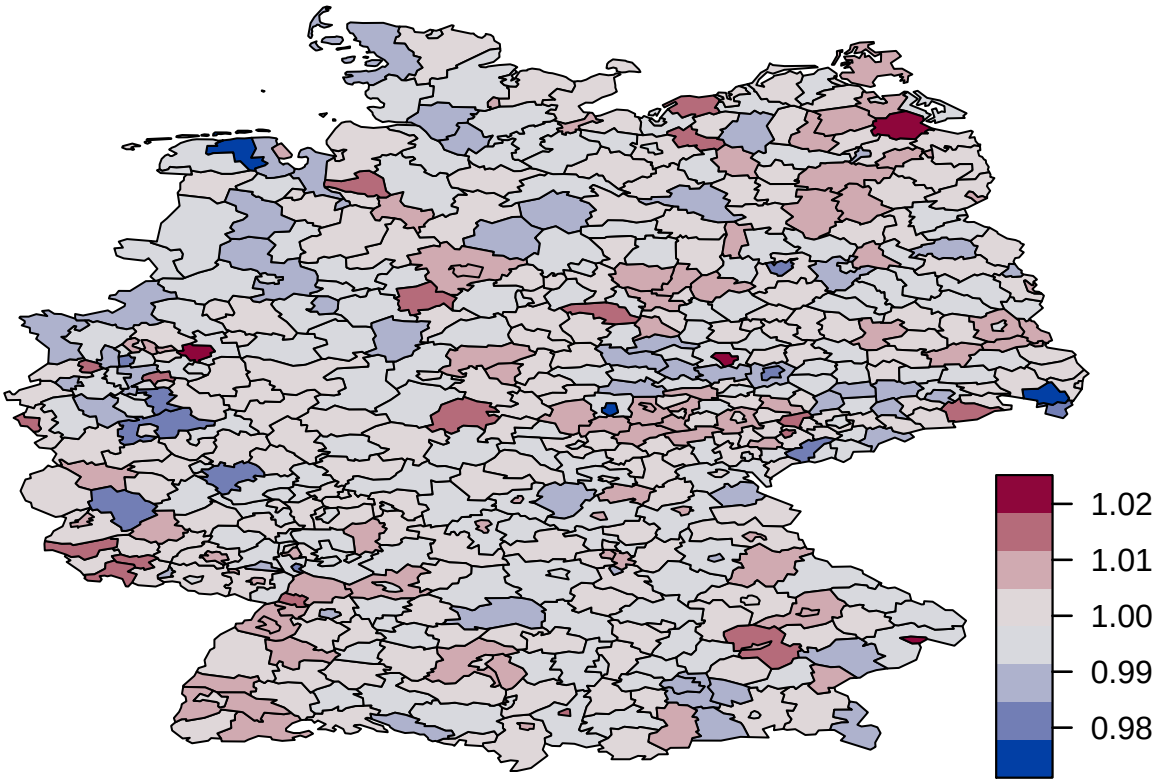
The relaive effective sample size is the effective sample size obtained per second of running the MCMC. This could be interesting, as it gives us some meaure of how efficient the MCMC algorithm is. Low computer time is not good in itself if the algorithm does not draw enough samples from the target distribution. Likewise, large effective sample size is of course desireble, but if the algorithm has to run for an unreasonable long time, some improvement of the algorithm or sampling approach should maybe be considered.

These numbers in themselves might not be too informative, but in relation to other approaches, they will give some measure of which to use.

Exercise 5: Interpretation of result

u or eta?

```
post_median = exp(apply(samples$u, 1, median))

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



Interpret

Exercise 6: INLA