Project 2

TMA4300 Computer Intensive Statistical Methods V2021

Casper Stenersen & Tinius Mellbye

19 March, 2021

Problem A

Here we will analyse the coal data set available as the variable `coal` after importing the *boot* library, `library(boot)`. The data set contains the date of coal mining disasters in the UK, the first and last instances are the start and end of the period, respectivle, and at those dates no disaster occured.

1)

Here we plot the cumulative number of distaters from the coal data set.

```
# Here we we start with the first year with 0 accidents and count one more
# accident per row in the data.frame The last year in the data do not represent
# another accident. Hence the two last elements in cumulative number of accidents
# are equal
cum_num = c(c(0:189), 189)
cumulative_plot = ggplot(data = coal, aes(x = date, y = cum_num)) + geom_path() +
    ylab("Cumulative number of disasters") + xlab("Year") + labs(caption = "Figure 1")

cumulative_plot
```

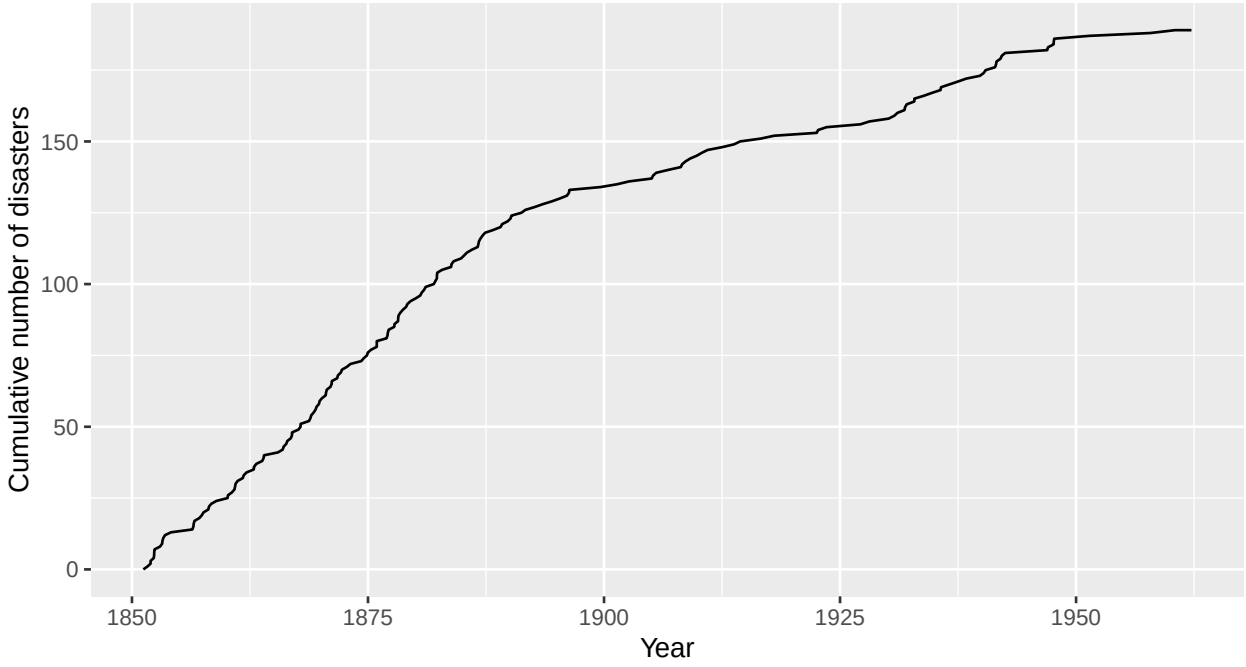


Figure 1

Figure 1 shows a steep linear trend until around 1990, then the growth of the cumulative decreased. i.e. time between accidents increased, it became safer.

2)

We will implement various versions of the MCMC algorithm and use the hiararchical Bayesian model framework assuming the coal-mining disasters follow an inhomogeenous Poisson process with intensity function

$$\lambda(t) = \begin{cases} \lambda_{k-1} & \text{for } t \in [t_{k-1}, t_k],\ k \in \{1, \dots, n\} \\ \lambda_n & \text{for } t \in [t_n, t_{n+1}]. \end{cases}$$

Here t_k for $k \in \{1, \dots, n\}$ are the break point for the intensity function while t_0 and t_{n+1} are the start and end time for the data set, respectivly. We will here consider the case where $n = 1$. Then one can derive that the likelihood function for the observed data, x, is

$$f(x|t_1, \lambda_0, \lambda_1) = \exp\Big(-\sum_{k=0}^{n} \lambda_k (t_{k+1} - t_k) \Big) \prod_{k=0}^{n} \lambda_k^{y_k} = e^{-\lambda_0(t_1 - t_0) - \lambda_1(t_2 - t_1)} \lambda_0^{y_0} \lambda_1^{y_1}$$

when y_k is the number of observed disasters from t_k to t_{k+1}. Apriori we make the following assumptions

- t_1 is uniformly distributed on the allowed values
- λ_0 and λ_1 comes from a gamma distribution with shape $\alpha = 2$ and scale β
- λ_0, λ_1 and t_1 are independent
- For β we use the improper prior

$$f(\beta) \propto \frac{\exp(-\frac{1}{\beta})}{\beta} \text{ for } \beta > 0.$$

We now find the posterior of θ (the vector of parameters), given x by using bayes rule and independence between variables,

$$f(\theta|x) = \frac{f(x|\theta)f(\theta)}{f(x)} \propto f(x|\theta)f(\theta) \tag{1}$$

$$\overset{\text{indep.}}{=} f(x|\theta)f(t_1)f(\lambda_0|\beta)f(\lambda_1|\beta)f(\beta) \tag{2}$$

note that $f(t_1)$ is uniformly distributed on the allowed values, meaning that it is constant on the allowed values and can be disregarded when considering proportionality relations. Hence,

$$f(\theta|x) \propto e^{-\lambda_0(t_1-t_0)-\lambda_1(t_2-t_1)}\lambda_0^{y_0}\lambda_1^{y_1}\frac{1}{\beta^2}\lambda_0 e^{-\frac{\lambda_0}{\beta}}\frac{1}{\beta^2}\lambda_1 e^{-\frac{\lambda_1}{\beta}}e^{-\frac{1}{\beta}}\frac{1}{\beta} \tag{3}$$

$$= \underline{\exp\left(-\lambda_0(t_1-t_0+\frac{1}{\beta})-\lambda_1(t_2-t_1+\frac{1}{\beta})-\frac{1}{\beta}\right)\frac{\lambda_0^{y_0+1}\lambda_1^{y_1+1}}{\beta^5}} \tag{4}$$

which is the posterior of $\theta|x$ up to a normalising constant.

3)

We now find the full conditionals for each element of θ. These can be extracted from the expression above, because the full conditional for the i^{th} element of $\theta = [\theta_1, \ldots, \theta_p]^T$ is

$$f(\theta_i|\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_p, x).$$

Thus, we get that

$$f(t_1|\lambda_0, \lambda_1, \beta, x) \propto \lambda_0^{y_0}\lambda_1^{y_1}e^{-t_1(\lambda_0-\lambda_1)} \tag{5}$$

which is not a named distribution.

Next we see that

$$f(\lambda_0|t_1, \lambda_1, \beta, x) \propto \exp(-\lambda_0(t_1-t_0+\frac{1}{\beta}))\lambda_0^{y_0+1} \tag{6}$$

$$= \lambda_0^{y_0+1}\exp\big(\frac{-\lambda_0}{(t_1-t_0+1/\beta)^{-1}}\big). \tag{7}$$

This is recognised to be proportional to the gamma distribution, i.e. $\lambda_0|t_1, \lambda_1, \beta, x \sim \text{Gamma}(y_0+2, t_1-t_0+\frac{1}{\beta})$, here we used the rate parametrisation. Similarly, $\lambda_1|t_1, \lambda_0, \beta, x \sim \text{Gamma}(y_1+2, t_2-t_1+\frac{1}{\beta})$. Lastly,

$$f(\beta|t_1, \lambda_0, \lambda_1, x) \propto \exp(-\frac{1}{\beta}(\lambda_0+\lambda_1+1))\frac{1}{\beta^5} \tag{8}$$

$$= \exp\big(\frac{-(\lambda_0+\lambda_1+1)}{\beta}\big)\frac{1}{\beta^{4+1}} \tag{9}$$

is proportional to the inverse gamma distribution, in other words, $\beta|t_1, \lambda_0, \lambda_1, x \sim \text{Inv. Gamma}(4, \lambda_0+\lambda_1+1)$ again with the rate parametrisation.

4)

Now we will use the Metropolis-Hastings algorithm as our single site MCMC to sample the posterior distribution. The variable we are interested in is t_1. We will sample the remaining variables from their known full conditionals. We need to contruct a Markov chain with limiting distribution $\pi(t) = f(t|\lambda_0, \lambda_1, \beta, x)$,

the chain got to be irreducible (see below) and ergodic (aperiodic (see below) and positive recurrent). If a chain is not positive recurrent it will drift away, which the plots below illustrate is not the case. We have convergence to the limiting distribution when the detailed balance equation is satified,

$$\pi(t_1)P(t_1'|t_1) = \pi(t_1')P(t_1|t_1')$$

Hence, we want to find $P(t_1'|t_1)$ which satisfies this. To enforce this we impose the restriction to only accept a transition from t_1 to t_1' with probability $\alpha(t_1, t_1')$, leading to the updated detailed balance equation

$$\pi(t_1)q(t_1'|t_1)\alpha(t_1, t_1') = \pi(t_1')q(t_1|t_1')\alpha(t_1', t_1)$$

Here we set $P(i|j) = q(i|j)\alpha(j, i)$, where $q(i|j)$ is the proposal density of going from state j to state i. We will use the gaussian proposal $q(t_1'|t_1) \sim \mathcal{N}(t_1, \sigma^2)$ where σ^2 effectlivly is tuning parameter. Because the gaussian distribution can reach every possible state, in theory, the chain will be irreducible and aperiodic.

$\alpha(t_1, t_1')$ is selected to be

$$\alpha(t_1, t_1') = \min\left\{1, \frac{\pi(t_1')q(t_1|t_1')}{\pi(t_1)q(t_1'|t_1)}\right\}$$

and because we selected $q()$ to be a symmetric distribution $q(t_1|t_1') = q(t_1', t_1)$ making

$$\alpha(t_1, t_1') = \min\left\{1, \frac{\pi(t_1')}{\pi(t_1)}\right\} = \min\left\{1, \frac{\lambda_0^{y_0'}\lambda_1^{y_1'}e^{-t_1'(\lambda_0-\lambda_1)}}{\lambda_0^{y_0}\lambda_1^{y_1}e^{-t_1(\lambda_0-\lambda_1)}}\right\}$$

We compute fraction in α on log-scale to make the algorithm more stable.

```
# We make this support function for simplisity
get_y = function(x, t) {
    # the number of observations from t_k to t_{k+1}, but we only have two intervals,
    # hence
    y_0 = sum(x <= t)
    y_1 = length(x) - y_0
    return(list(y_0 = y_0, y_1 = y_1))
}

# The log full conditional of t_1
log_fc_t_1 = function(x, t) {
    return(get_y(x, t)$y_0 * log(lambda_0) + get_y(x, t)$y_1 * log(lambda_1) + t *
        (lambda_1 - lambda_0))
}

# The single-site algorithm
single_site_mcmc = function(t_1, lambda_0, lambda_1, beta, x, sigma, n = 10000) {
    set.seed(135642)

    # We initialise the data frame where we will store the values
    data_mcmc = data.frame(matrix(nrow = n, ncol = 4))
    colnames(data_mcmc) = c("t_1", "lambda_0", "lambda_1", "beta")
    data_mcmc[1, ] = c(t_1, lambda_0, lambda_1, beta)

    # We calculate the log full conditional once outside the loop, by doing so we do
    # not need to calculate the the log full conditional twice inside the loop
    log_fc_new = log_fc_t_1(x, t_1)

    # To calculate the success/acceptance rate
```

```
    success_count = 0
    for (i in 2:n) {
        # Generate proposals
        t_1_new = rnorm(1, mean = t_1, sd = sigma)

        # if the proposal is outside the interval [t_0, t_2]
        if ((t_1_new < t_0) | (t_1_new > t_2)) {
            log_accept = -Inf  # corresponds to alpha = 0 on non-log scale
        } else {
            # We store the log full conditional of the last iteration
            log_fc = log_fc_new
            # We get the y values for the new t_1
            y_new = get_y(x, t_1_new)

            # We calculate the log of the full conditional for t_1_new
            log_fc_new = log_fc_t_1(x, t_1_new)
            # Calculate the acceptance probability on log scale
            log_accept = log_fc_new - log_fc
        }

        # Accept or reject
        u <- runif(1)
        # If u < alpha, we accept the proposal
        if (u < exp(log_accept)) {
            t_1 = t_1_new
            success_count = success_count + 1
        }
        # else{ do nothing }

        # We add the selected values to the data frame
        data_mcmc[i, ] = c(t_1, lambda_0, lambda_1, beta)

        # Full conditional for the parameters
        beta = rinvgamma(1, shape = 4, rate = (lambda_0 + lambda_1 + 1))
        lambda_0 = rgamma(1, shape = get_y(x, t_1)$y_0 + 2, rate = (t_1 - t_0 + 1/beta))
        lambda_1 = rgamma(1, shape = get_y(x, t_1)$y_1 + 2, rate = (t_2 - t_1 + 1/beta))
    }
    return(list(chain = data_mcmc, success_rate = success_count/n))
}
```

5 & 6

Now we try the algorithm and investigate the burn-in and "mixing properties" for the algorithm. We initialise the algorithm with `t_1` to be the mean of `t_0` and `t_2`. Further, we set $\beta = 1$ and $\sigma = 1$ for convenience, additionally, λ_0 and λ_1 are drawn from their full conditionals as specified above.

```
set.seed(1997)
# These are global variables because we assume n = 1
t_0 = coal[1, ]
t_2 = coal[dim(coal)[1], ]
# Remove the first and last element of the coal data frame
x = coal[-1, ]
x = x[-length(x)]
```

```
################################### Initialising values
t_init = (t_0 + t_2)/2
beta = 1
lambda_0 = rgamma(1, shape = get_y(x, t = t_init)$y_0 + 2, rate = (t_init - t_0 +
    1/beta))
lambda_1 = rgamma(1, shape = get_y(x, t = t_init)$y_1 + 2, rate = (t_2 - t_init +
    1/beta))
# Number of iterations
n = 20000

# Now we run the algorithm for different sigmas
sigma1 = 1
mcmc_relisation1 = single_site_mcmc(t_init, lambda_0, lambda_1, beta, x, sigma = sigma1,
    n = n)

sigma2 = 5
mcmc_relisation2 = single_site_mcmc(t_init, lambda_0, lambda_1, beta, x, sigma = sigma2,
    n = n)

sigma3 = 10
mcmc_relisation3 = single_site_mcmc(t_init, lambda_0, lambda_1, beta, x, sigma = sigma3,
    n = n)

sigma4 = 30
mcmc_relisation4 = single_site_mcmc(t_init, lambda_0, lambda_1, beta, x, sigma = sigma4,
    n = n)
```

Here we have plot the first 1000 iterations to examin the burn-in.

```
par(mfrow = c(2, 2), oma = c(4, 2, 0, 0) + 0.1)
plot(mcmc_relisation1$chain$t_1[1:1000], type = "l", ylab = "Year", xlab = "Iterations",
    main = expression(paste(sigma, " = 1")))
plot(mcmc_relisation2$chain$t_1[1:1000], type = "l", ylab = "Year", xlab = "Iterations",
    main = expression(paste(sigma, " = 5")))
plot(mcmc_relisation3$chain$t_1[1:1000], type = "l", ylab = "Year", xlab = "Iterations",
    main = expression(paste(sigma, " = 10")))
plot(mcmc_relisation4$chain$t_1[1:1000], type = "l", ylab = "Year", xlab = "Iterations",
    main = expression(paste(sigma, " = 30")))
mtext(bquote("Figure 2: The first 1000 realisations for different " ~ sigma ~ " values"),
    side = 1, outer = TRUE, adj = 1)
```

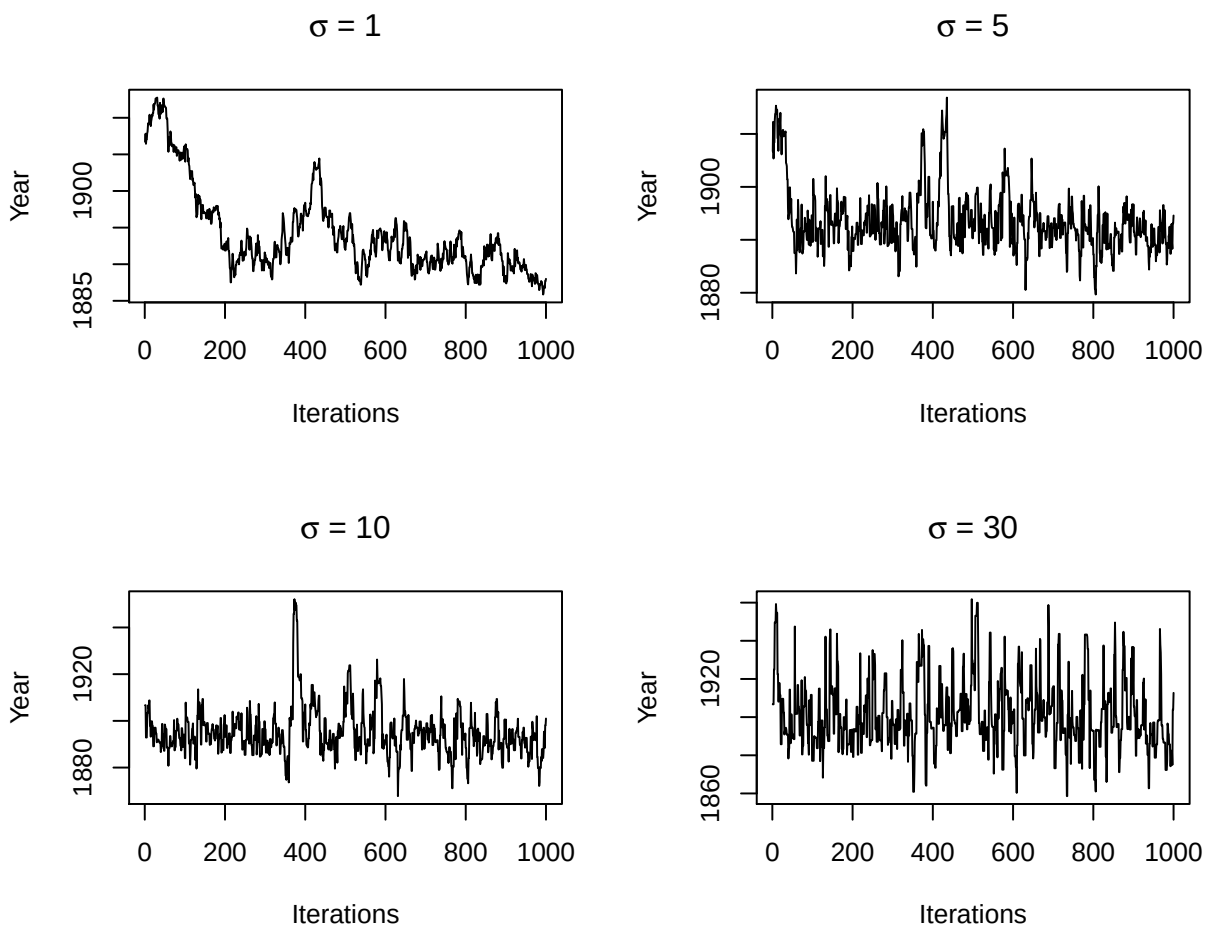


Figure 2: The first 1000 realisations for different σ values

Figure 2 illustrates that the burn-in is faster as σ increases, naturally. We see that the burn-in period for $\sigma = 1$ is approximatly 300, we remain slightly conservative and choose a burn-in of 500 (for convenience we use the same burn-in for all). Furthermore, all simulations appear to converge to a limiting distribution with mean around 1890.

```
burnin = 500

par(mfrow = c(2, 2), oma = c(4, 2, 0, 0) + 0.1)
hist(mcmc_relisation1$chain$t_1[burnin:n], freq = FALSE, col = "dark green", nclass = 20,
    main = expression(paste(sigma, " = 1")), xlab = "Year")
hist(mcmc_relisation2$chain$t_1[burnin:n], freq = FALSE, col = "dark green", nclass = 20,
    main = expression(paste(sigma, " = 5")), xlab = "Year")
hist(mcmc_relisation3$chain$t_1[burnin:n], freq = FALSE, col = "dark green", nclass = 20,
    main = expression(paste(sigma, " = 10")), xlab = "Year")
hist(mcmc_relisation4$chain$t_1[burnin:n], freq = FALSE, col = "dark green", nclass = 20,
    main = expression(paste(sigma, " = 30")), xlab = "Year")
mtext(bquote("Figure 3: The histograms of the chain with varying " ~ sigma ~ " values, excluding the bu
    side = 1, outer = TRUE, adj = 1)
```

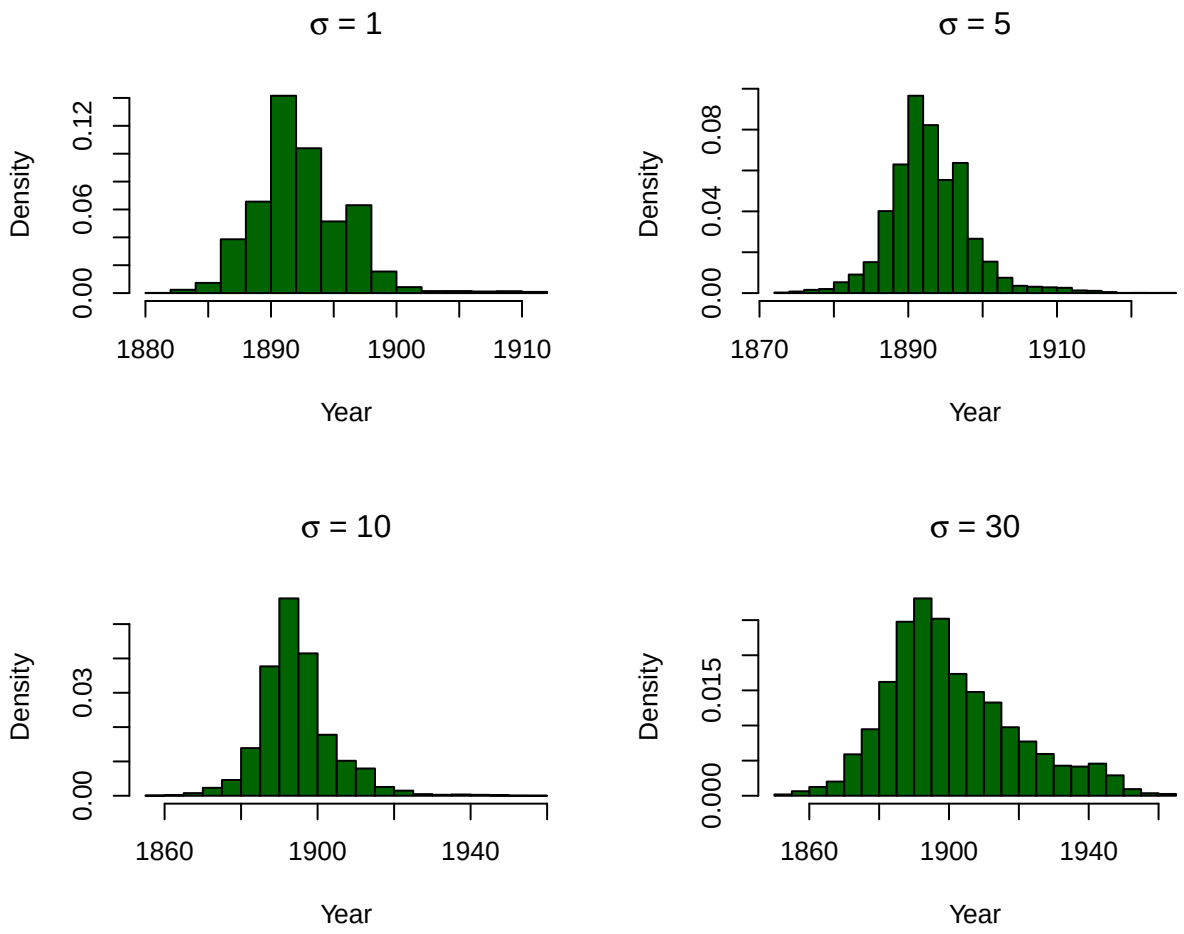


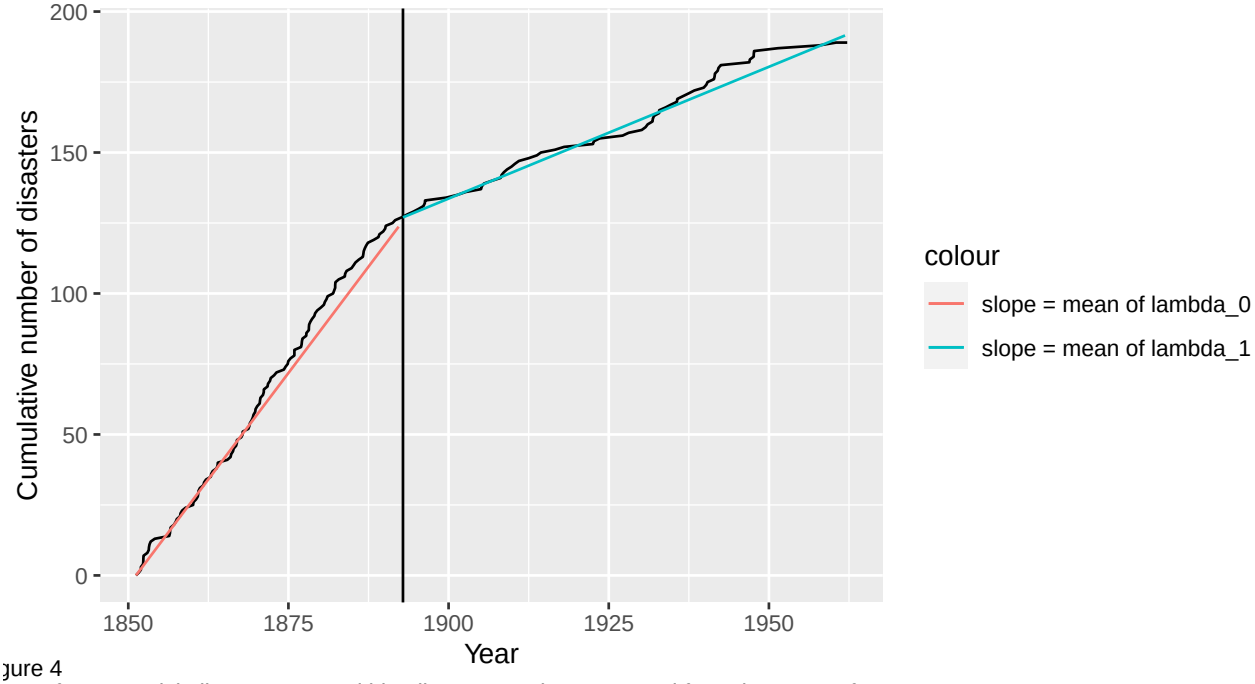
Figure 3: The histograms of the chain with varying σ values, excluding the burn−in.

Figure 3 support the claim that the limiting distribution of the means are the same for various values of σ. Though, when σ increase, the algorithm explores more of the space, and we see a heavy right tail when σ is 30 (Note that the x-axes vary between the plots). This is likely to be the result of the algorithm exploring the flatter right part of plot in A1 and gets stuck before retaining towards the dominant mean.

```
# Here we work with the second realisation
slope_0 = mean(mcmc_relisation2$chain$lambda_0[burnin:n])
slope_1 = mean(mcmc_relisation2$chain$lambda_1[burnin:n])
t_1_mean = mean(mcmc_relisation2$chain$t_1[burnin:n])

# To make the plots
slope_0_df = data.frame(list(x = seq(t_0, t_1_mean), y = slope_0 * (seq(t_0, t_1_mean) -
    t_0)))
slope_1_df = data.frame(list(x = seq(t_1_mean, t_2), y = slope_1 * (seq(t_1_mean,
    t_2) - t_1_mean) + get_y(x, t_1_mean)$y_0))

# We add lines to the plot from Problem A1
cumulative_plot + geom_vline(xintercept = t_1_mean) + geom_path(data = slope_0_df,
    aes(x = x, y = y, col = "slope = mean of lambda_0")) + geom_path(data = slope_1_df,
    aes(x = x, y = y, col = "slope = mean of lambda_1")) + labs(caption = bquote("Figure 4 \nMean of " ~
    t[1] ~ " = verticle line, orange and blue lines are estimates to and from the mean of " ~
    t[1]))
```



gure 4
ean of t_1 = verticle line, orange and blue lines are estimates to and from the mean of t_1

Because λ_0 is the intensity parameter for the poisson process from t_0 to t_1 we use the mean of λ_0 as the slope for the orange line going from t_0 to t_1. From similar reasoning the mean of λ_1 is used as the slope for the blue line, going from t_1 to t_2. The plot illustrate that the algorithm has made a decent job and is likely to be implemented correctly. (Here $\sigma = 5$)

7

a)

We investigate a block Metropolis-Hastings algorithm with two block proposals. First we put forth a block proposal for $(t_1, \lambda_0, \lambda_1)$ with the "old" β. We generate new values $(t_1^{new}, \lambda_0^{new}, \lambda_1^{new})$ by first generating a $t_1^{new} \sim \mathcal{N}(t_1, \sigma_{t_1}^2)$ then we generate $(\lambda_0^{new}, \lambda_1^{new})$ from the joint full conditional using t_1^{new}. (Below we simplify the notation by substituting new with $'$)

We identify the relevant joint full consitional distribution is the distribution of (λ_0, λ_1), which is

$$f(\lambda_0, \lambda_1 | x, t_1', \beta) \overset{\text{indep.}}{=} f(\lambda_0 | x, t_1', \beta) f(\lambda_1 | x, t_1', \beta).$$

Following the same argumentation as in 4) we get that

$$\alpha_1(t_1, \lambda_0, \lambda_1, t_1', \lambda_0', \lambda_1') = \min\left\{1, \frac{\pi(t_1', \lambda_0', \lambda_1') q(t_1, \lambda_0, \lambda_1 | t_1', \lambda_0', \lambda_1')}{\pi(t_1, \lambda_0, \lambda_1) q(t_1', \lambda_0', \lambda_1' | t_1, \lambda_0, \lambda_1)}\right\} \tag{10}$$

Here

$$\pi(t_1', \lambda_0', \lambda_1') = f(\theta' | x)$$

is the limiting distribution of the markov chain (posterior found in 2) and

$$q(t_1, \lambda_0, \lambda_1 | t_1', \lambda_0', \lambda_1') = f(t_1' | t_1) f(\lambda_0', \lambda_1' | x, t_1', \beta) = f(t_1' | t_1) f(\lambda_0' | x, t_1', \beta) f(\lambda_1' | x, t_1', \beta)$$

and again, thanks to the symmetry of $f(t_1'|t_1)$ (we still use the symmetric gaussian distribution for proposing t_1) the terms cancel in the numerator and denominator. Then

$$\frac{\pi(t_1', \lambda_0', \lambda_1')q(t_1, \lambda_0, \lambda_1|t_1', \lambda_0', \lambda_1')}{\pi(t_1, \lambda_0, \lambda_1)q(t_1', \lambda_0', \lambda_1'|t_1, \lambda_0, \lambda_1)} = \frac{e^{-2\lambda_0'(t_1'-t_0+1/\beta)-2\lambda_1'(t_2-t_1'+1/\beta)-1/\beta}\lambda_0'^{2y_0'+2}\lambda_1'^{2y_1'+2}\frac{1}{\beta'^5}}{e^{-2\lambda_0(t_1-t_0+1/\beta)-2\lambda_1(t_2-t_1+1/\beta)-1/\beta}\lambda_0^{2y_0+2}\lambda_1^{2y_1+2}\frac{1}{\beta^5}} \tag{11}$$

and we end up with

$$\alpha_1(t_1, \lambda_0, \lambda_1, t_1', \lambda_0', \lambda_1') = \min\left\{1, \frac{e^{-2\lambda_0'(t_1'-t_0+1/\beta)-2\lambda_1'(t_2-t_1'+1/\beta)}\lambda_0'^{2y_0'+2}\lambda_1'^{2y_1'+2}}{e^{-2\lambda_0(t_1-t_0+1/\beta)-2\lambda_1(t_2-t_1+1/\beta)}\lambda_0^{2y_0+2}\lambda_1^{2y_1+2}}\right\}$$

b)

The block 2 part of the algorithm consists of proposing a block of $(\beta, \lambda_0, \lambda_1)$ keeping t_1 unchanged, using t_1 from block 1. We do this by generating a new β, β^{new}, from $\mathcal{N}(\beta, \sigma_\beta^2)$. Then, using the β^{new}, we generate new (λ_0, λ_1) from their joint full consitionals, $f(\lambda_0, \lambda_1|x, t_1, \beta^{new})$.

Similarly to above,

$$\alpha_2(\beta, \lambda_0, \lambda_1, \beta', \lambda_0', \lambda_1') = \min\left\{1, \frac{\pi(\beta', \lambda_0', \lambda_1')q(\beta, \lambda_0, \lambda_1|\beta', \lambda_0', \lambda_1')}{\pi(\beta, \lambda_0, \lambda_1)q(\beta', \lambda_0', \lambda_1'|\beta, \lambda_0, \lambda_1)}\right\}$$

where

$$\pi(\beta, \lambda_0, \lambda_1) = f(\theta'|x)$$

is the limiting distribution of the Markov Chain, as above.

Moreover,

$$q(\beta, \lambda_0, \lambda_1|\beta', \lambda_0', \lambda_1') = f(\beta'|\beta)f(\lambda_0'|x, t_1, \beta')f(\lambda_1'|x, t_1, \beta')$$

where $f(\beta'|\beta) = f(\beta|\beta')$, because we again use the symmetric normal proposal. Thus we get

$$\alpha_2(\beta, \lambda_0, \lambda_1, \beta', \lambda_0', \lambda_1') = \min\left\{1, \frac{e^{-2\lambda_0'(t_1-t_0+1/\beta')-2\lambda_1'(t_2-t_1+1/\beta')-1/\beta'}\lambda_0'^{2y_0'+2}\lambda_1'^{2y_1'+2}\frac{1}{\beta'^5}}{e^{-2\lambda_0(t_1-t_0+1/\beta)-2\lambda_1(t_2-t_1+1/\beta)-1/\beta}\lambda_0^{2y_0+2}\lambda_1^{2y_1+2}\frac{1}{\beta^5}}\right\}$$

```
block_mcmc = function(t_1, lambda_0, lambda_1, beta, x, sigma_t, sigma_beta, n = 10000) {
    set.seed(1890)
    # Initialising a data frame to store the values in
    mcmc_data = data.frame(matrix(nrow = n, ncol = 4))
    colnames(mcmc_data) = c("t_1", "lambda_0", "lambda_1", "beta")
    mcmc_data[1, ] = c(t_1, lambda_0, lambda_1, beta)

    success_count1 = 0
    success_count2 = 0
    for (i in 2:n) {
        ################################ Block 1
        t_new = rnorm(1, mean = t_1, sd = sigma_t)
        # Now we got to find a new t_1, hence
        while (t_new < t_0 | t_new > t_2) {
            t_new = rnorm(1, mean = t_1, sd = sigma_t)
        }
        # We extract the two y-s
```

```
        y_new = get_y(x, t_new)
        y = get_y(x, t_1)

        # We get new lambda_0 and lambda_1 using t_new
        lambda_0_new = rgamma(1, shape = y_new$y_0 + 2, rate = (t_new - t_0 + 1/beta))
        lambda_1_new = rgamma(1, shape = y_new$y_1 + 2, rate = (t_2 - t_new + 1/beta))

        # We now calculate the accpetance probability on log scale We calculate the
        # numerator on log scale
        log_numerator = -2 * lambda_0_new * (t_new - t_0 + 1/beta) - 2 * lambda_1_new *
            (t_2 - t_new + 1/beta) + 2 * (y_new$y_0 + 1) * log(lambda_0_new) + 2 *
            (y_new$y_1 + 1) * log(lambda_1_new)
        # We calculate the denominator on log scale
        log_denominator = -2 * lambda_0 * (t_1 - t_0 + 1/beta) - 2 * lambda_1 * (t_2 -
            t_1 + 1/beta) + 2 * (y$y_0 + 1) * log(lambda_0) + 2 * (y$y_1 + 1) * log(lambda_1)

        log_accept = log_numerator - log_denominator

        # Now we accept or reject
        u = runif(1)
        if (u < exp(log_accept)) {
            # If accepted, update the variables
            t_1 = t_new
            lambda_0 = lambda_0_new
            lambda_1 = lambda_1_new
            # Increase the count of successes for block 1
            success_count1 = success_count1 + 1
        }
        # else{ do nothing }

        ##################################

        # Block 2 We only use one y as t_1 is fixed in this block
        y = get_y(x, t_1)
        # We propose a new beta < 0
        beta_new = rnorm(1, mean = beta, sd = sigma_beta)
        while (beta_new < 0) {
            beta_new = rnorm(1, mean = beta, sd = sigma_beta)
        }

        lambda_0_new = rgamma(1, shape = y$y_0 + 2, rate = (t_1 - t_0 + 1/beta_new))
        lambda_1_new = rgamma(1, shape = y$y_1 + 2, rate = (t_2 - t_1 + 1/beta_new))

        # Now we calculate the acceptance probability
        log_numerator2 = -2 * lambda_0_new * (t_1 - t_0 + 1/beta_new) - 2 * lambda_1_new *
            (t_2 - t_1 + 1/beta_new) - 1/beta_new + 2 * (y$y_0 + 1) * log(lambda_0_new) +
            2 * (y$y_1 + 1) * log(lambda_1_new) - log(beta_new^5)

        log_denominator2 = -2 * lambda_0 * (t_1 - t_0 + 1/beta) - 2 * lambda_1 *
            (t_2 - t_1 + 1/beta) - 1/beta + 2 * (y$y_0 + 1) * log(lambda_0) + 2 *
            (y$y_1 + 1) * log(lambda_1) - log(beta^5)

        log_accept2 = log_numerator2 - log_denominator2
```

```
        if (runif(1) < exp(log_accept2)) {
            # If accepted, update variables
            beta = beta_new
            lambda_0 = lambda_0_new
            lambda_1 = lambda_1_new
            # We increase the success count for block 2
            success_count2 = success_count2 + 1
        }
        # else{ do nothing }

        # Add the last realisation to the data
        mcmc_data[i, ] = c(t_1, lambda_0, lambda_1, beta)
    }
    # return the data and the success rate
    return(list(chain = mcmc_data, success_rate1 = success_count1/n, success_rate2 = success_count2/n))
}
```

8

We will now explore the properties of the block Metropolitan Hastings algrotihm and compare it to the single-site version implemented in 4)

```
# Initialising values for t, lambda_0 and lambda_1
t_init = (t_0 + t_2)/2
beta = 1

lambda_0 = rgamma(1, shape = get_y(x, t = t_init)$y_0 + 2, rate = (t_init - t_0 +
    1/beta))
lambda_1 = rgamma(1, shape = get_y(x, t = t_init)$y_1 + 2, rate = (t_2 - t_init +
    1/beta))
```

```
sigma_beta = c(1, 5, 10, 30)
sigma_t = c(1, 5, 10, 30)
n = 1000


par(mfrow = c(length(sigma_beta), length(sigma_t)), oma = c(4, 2, 0, 0) + 0.1)
for (sig_beta in sigma_beta) {
    for (sig_t in sigma_t) {
        set.seed(123321)  # For reproducability
        block_realisation = block_mcmc(t_init, lambda_0, lambda_1, beta, x, sigma_t = sig_t,
            sigma_beta = sig_beta, n = n)
        plot(block_realisation$chain$t_1[1:1000], type = "l", main = bquote(sigma[t] ==
            .(sig_t) ~ " and " ~ sigma[beta] == .(sig_beta)), ylab = "Year", xlab = "Iterations")
    }
}
mtext(bquote("Figure 5: Traceplots of " ~ t[1] ~ " up to 1000 iterations with varying " ~
    sigma[t] ~ " and " ~ sigma[beta] ~ " values."), side = 1, outer = TRUE, adj = 1)
```

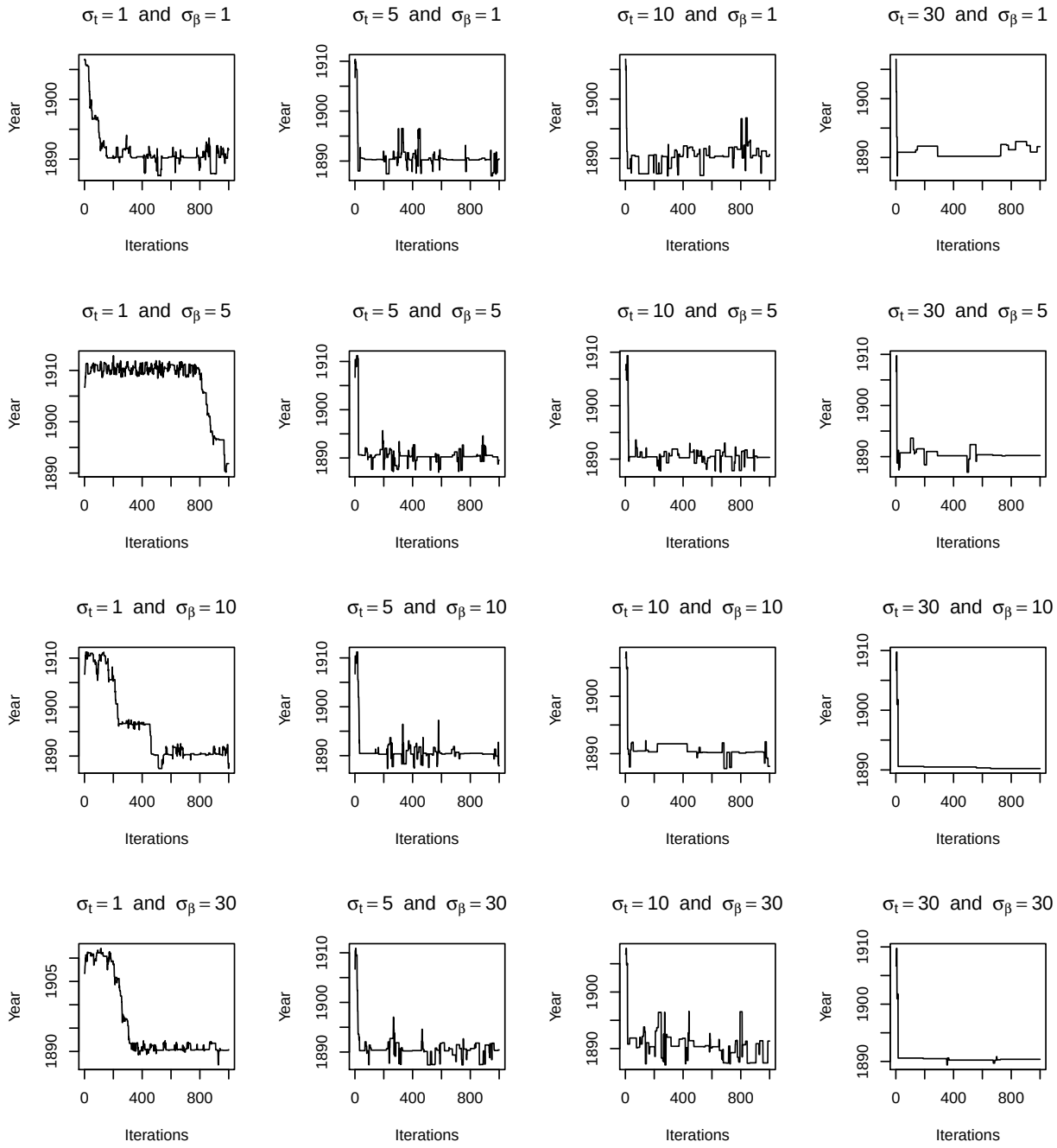


Figure 5: Traceplots of t_1 up to 1000 iterations with varying σ_t and σ_β values.

Based on figure 5 it apears like σ_β do not have nearly as much influence on the movement of the chain of t_1 compared to σ_t. This is observable by seeing that the change in behaviour within each row is much more profound than within each column. This is a natural consecuence from the fact that σ_t directly influence the proposals of t_1 in contrast to σ_β which directly influence the proposals of β (see below). Moreover, observe that the burn-in periode is faster as σ_t increase, but the change in burn-in is not changing much when $\sigma_t \geq 5$. Furthermore, when $\sigma_t = 30$ the chain fails to move around, this is likely happen because too large steps are proposed which then fails to be accepted. For instance when $\sigma_t = \sigma_\beta = 30$ the acceptance rate is 0.014 and 0.072 for the first and second block respectivly. The effect of too large steps can also be observed for σ_t equal 10 and even 5, where there are a number of flat regions.

Now we inspect how the chain of βs for different σ_ts and σ_βs.

```r
set.seed(123321)

sigma_beta = c(1, 5, 10, 30)
sigma_t = c(1, 5, 10, 30)
n = 1000

par(mfrow = c(length(sigma_beta), length(sigma_t)), oma = c(4, 2, 0, 0) + 0.1)
for (sig_beta in sigma_beta) {
    for (sig_t in sigma_t) {
        set.seed(123321)
        block_realisation = block_mcmc(t_init, lambda_0, lambda_1, beta, x, sigma_t = sig_t,
            sigma_beta = sig_beta, n = n)
        plot(block_realisation$chain$beta[1:1000], type = "l", main = bquote(sigma[t] ==
            .(sig_t) ~ " and " ~ sigma[beta] == .(sig_beta)), ylab = bquote(beta),
            xlab = "Iterations")
    }
}
mtext(bquote("Figure 6: Traceplots of " ~ beta ~ " with varying " ~ sigma[t] ~ " and " ~
    sigma[beta] ~ " values."), side = 1, outer = TRUE, adj = 1)
```

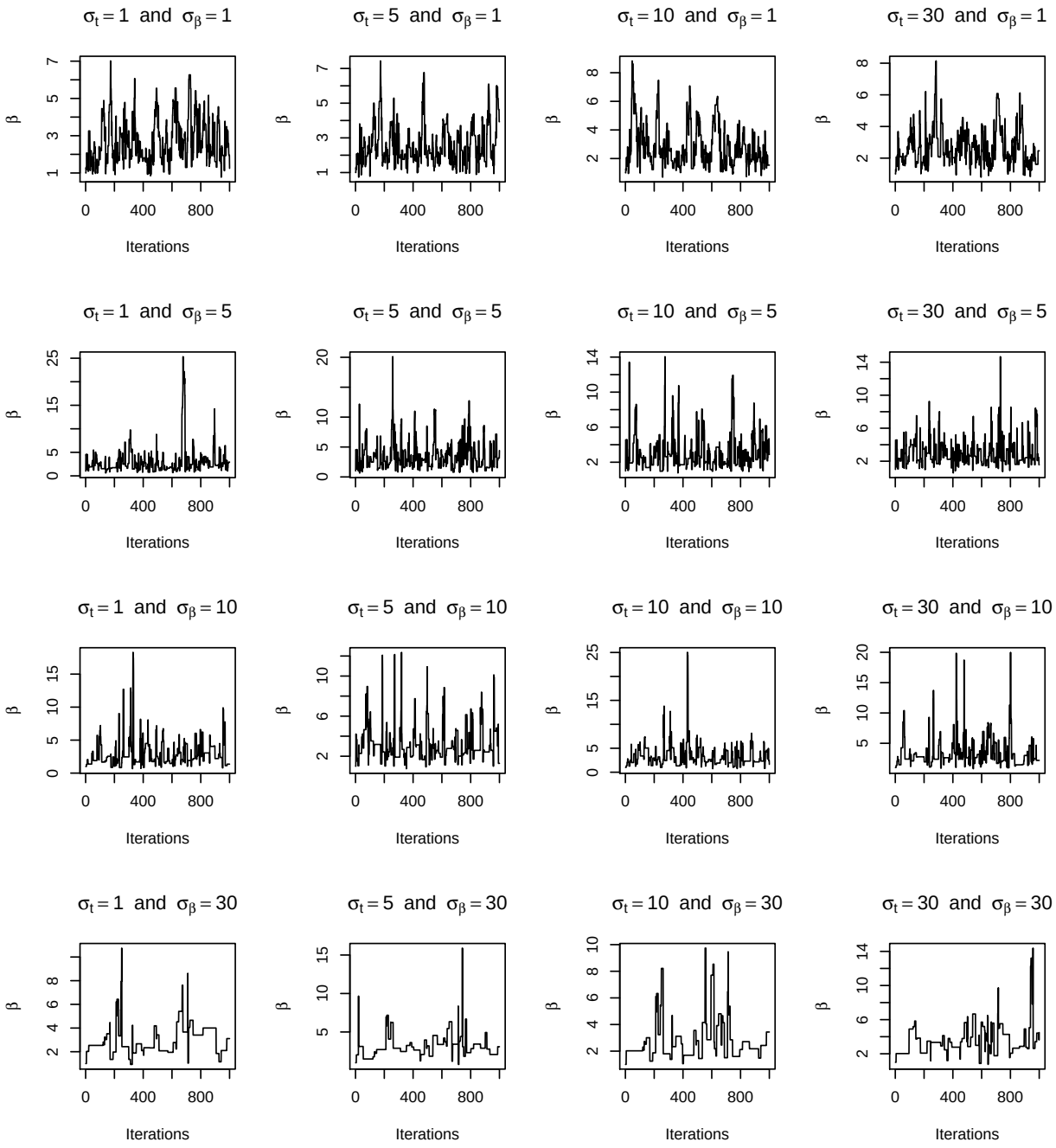


Figure 6: Traceplots of β with varying σ_t and σ_β values.

From figure 6 we obsere some of the similar effects of changing σ_β in relation to β as we did for σ_t in relation to t_1 above. In the first row $\sigma_\beta = 1$ and a state appear to be more dependant on previouse states than for $\sigma_\beta \geq 5$, as one would expect. We can also see that when σ_β is 10 or 30, the algorithm accepts fewer proposals which is apparent from the the increased number and length of horisontal lines.

Below we compare the single-site with $\sigma_t = 10$ and the block implementation with $\sigma_t = 1$ and $\sigma_\beta = 5$.

```
# We start by testing one realisation
sigma_t = 1
sigma_beta = 5
n = 2e+05
```

```
set.seed(123321)

block_realisation = block_mcmc(t_init, lambda_0, lambda_1, beta, x, sigma_t = sigma_t,
    sigma_beta = sigma_beta, n = n)
# We use the third realisation of the single-site from 5 & 6 which is equivalent
# to running the following

# single_site_mcmc1 = single_site_mcmc(t_init, lambda_0, lambda_1, beta, x, sigma
# = 10)
```

```
# We store the acceptance rates to use them in the text
accept_1 = block_realisation$success_rate1
accept_2 = block_realisation$success_rate2
accept_single_site = mcmc_relisation3$success_rate

par(mfrow = c(1, 2), oma = c(1, 1, 0, 0))
plot(block_realisation$chain$t_1[1:1000], type = "l", main = "Block", ylab = bquote("Year/" ~
    t[1]), xlab = "Iterations")
plot(mcmc_relisation3$chain$t_1[1:1000], type = "l", main = "Single site", ylab = bquote("Year/" ~
    t[1]), xlab = "Iterations")
mtext(bquote("Figure 7: \nBlock and single-site with " ~ sigma[t] == .(sigma_t) ~
    " and " ~ sigma[beta] == .(sigma_beta) ~ " for the block and " ~ sigma[t] ==
    .(10) ~ " for the single site."), side = 1, outer = TRUE, adj = 1)
```

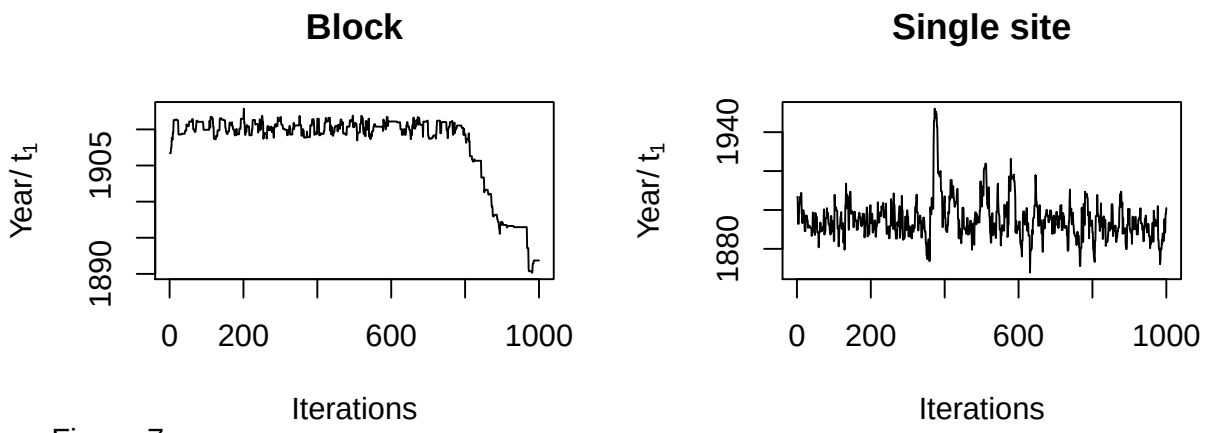


Figure 7:
Block and single−site with $\sigma_t = 1$ and $\sigma_\beta = 5$ for the block and $\sigma_t = 10$ for the single site.

We see that the block MH algorithm converges in around 800 steps, but the single-site algorithm needs less than 100 steps. We also see that the the single-site algorithm apears to vary in each step, in comparison to the block implementation which has slightly more flat regions. This can also be seen from that the acceptance rate for block 1 is 0.22534 and 0.2598 for block 2, but for the single-site the acceptance rate is 0.56855 (Note: We found $\sigma_t = 10$ to be suitable for the single site algorithm despite the slightly high acceptance rate).

We compare the distribution of t_1 from the block and single site algorithms.

```
# We set the burn-in = 1000 for the block algorithm
burn_in = 1000
# burn in single-site = 200
```

```
burn_in_single = 200

par(mfrow = c(1, 2), oma = c(1, 1, 0, 0))
hist(block_realisation$chain$t_1[burn_in:n], main = "Block", xlab = bquote("Year/" ~
    t[1]), freq = FALSE, col = "dark green")
hist(mcmc_relisation3$chain$t_1[burn_in_single:n], main = "Single site", xlab = bquote("Year/" ~
    t[1]), freq = FALSE, col = "dark green", breaks = 20)
mtext(bquote("Figure 8: \nBlock and single-site with " ~ sigma[t] == .(sigma_t) ~
    " and " ~ sigma[beta] == .(sigma_beta) ~ " for the block and " ~ sigma[t] ==
    .(10) ~ " for the single site."), side = 1, outer = TRUE, adj = 1)
```

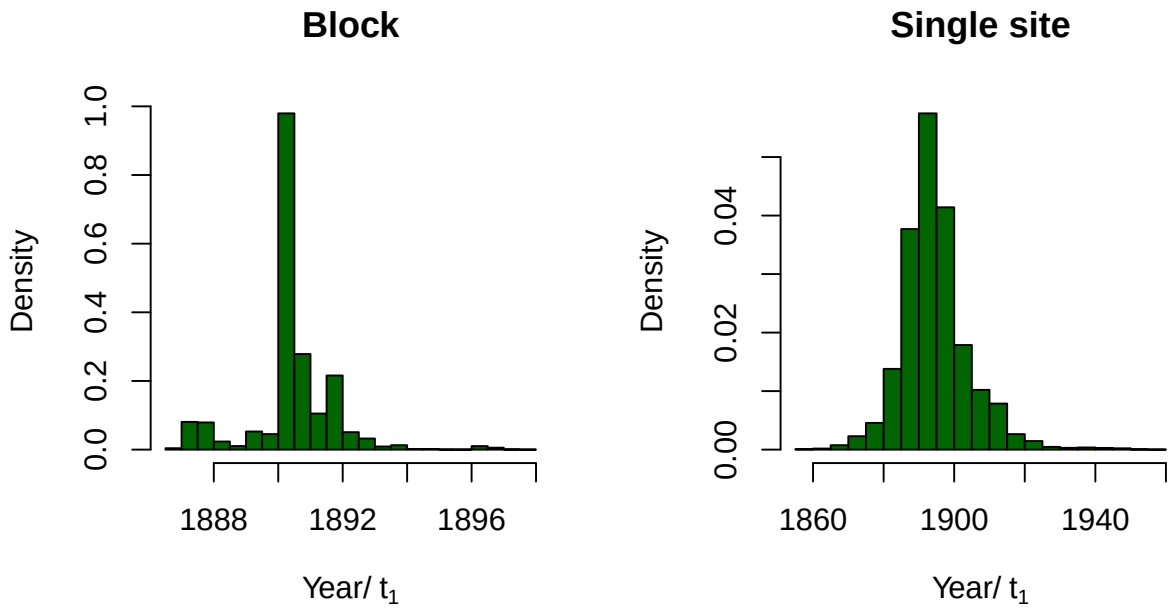


Figure 8:
Block and single−site with $\sigma_t = 1$ and $\sigma_\beta = 5$ for the block and $\sigma_t = 10$ for the single site.

Here we see that the single site algorithm varies much more than the block algorithm (note the difference on the x-axes).

```
block_data = block_realisation$chain[burn_in:n, ]
# We extract the chain for each parameter from the realisation
t_1 = block_data$t_1
lambda_0 = block_data$lambda_0
lambda_1 = block_data$lambda_1
beta = block_data$beta
# Covariance between lambda_0 and lambda_1
cov_of_lambdas = cov(lambda_0, lambda_1)

# Now we plot the histogrmas and the means
par(mfrow = c(2, 2), oma = c(1, 1, 1, 1))
# t_1
hist(t_1, col = "dark green", freq = FALSE, main = bquote("Histogram of " ~ t[1]),
    xlab = bquote(t[1]), breaks = 20)
abline(v = mean(t_1), col = "red")
# lambda_0
hist(lambda_0, col = "dark green", freq = FALSE, main = bquote("Histogram of " ~
```

```
    lambda[0]), xlab = bquote(lambda[0]), breaks = 40)
abline(v = mean(lambda_0), col = "red")
# lambda_1
hist(lambda_1, col = "dark green", freq = FALSE, main = bquote("Histogram of " ~
    lambda[1]), xlab = bquote(lambda[1]), breaks = 30)
abline(v = mean(lambda_1), col = "red")
# beta
hist(beta, col = "dark green", freq = FALSE, main = bquote("Histogram of " ~ beta),
    xlab = bquote(beta), breaks = 80, xlim = c(0, 15))
abline(v = mean(beta), col = "red")
mtext(bquote("Figure 9: Distribution of parameters for the block implementation with " ~
    sigma[t] == .(sigma_t) ~ " and " ~ sigma[beta] == .(sigma_beta) ~ "."), side = 1,
    outer = TRUE, adj = 1)
```

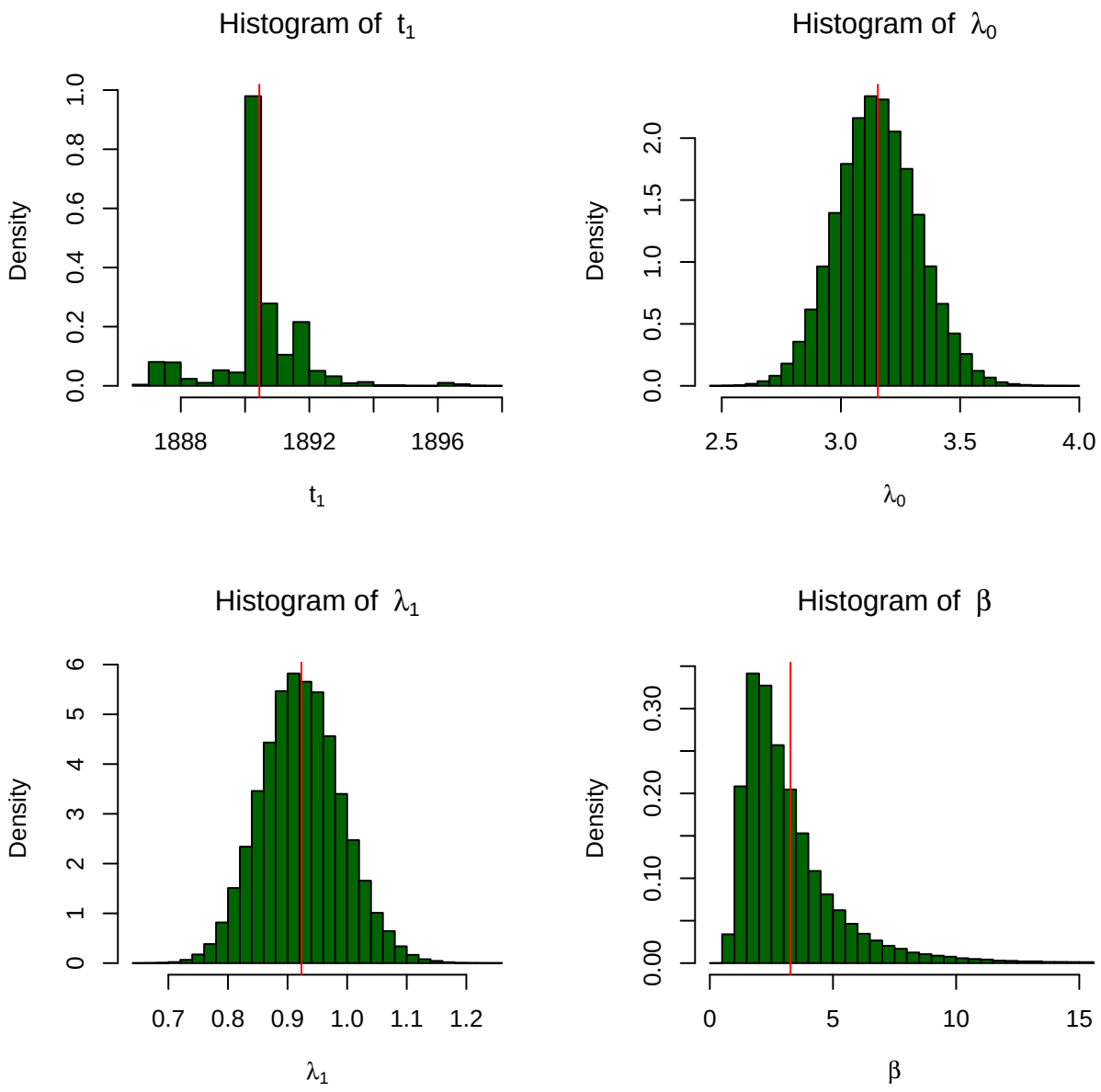


Figure 9: Distribution of parameters for the block implementation with $\sigma_t = 1$ and $\sigma_\beta = 5$.

```
cat(mean(t_1), mean(lambda_0), mean(lambda_1), mean(beta))
```

```
## 1890.444 3.15493 0.9233137 3.275799
```

The histograms illustate the distribution of the parameters and the red line is the mean of each distribution respectivly. Where the mean of $t_1 = 1890.444$, $\lambda_0 = 3.155$, $\lambda_1 = 0.923$ and $\beta = 3.276$. Lastly we find that the covariance between λ_0 and λ_1 is 5.75×10^{-4}.