Master project - backup

Halvard

2024-11-12

```
#Libraries
library(ggplot2) #plotting
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
library(tidyr) #for pivot_longer?
```

```
## Warning: package 'tidyr' was built under R version 4.3.3
```

```
library(ggpubr)
library(INLA) #posterior inference
```

```
## Warning: package 'INLA' was built under R version 4.3.3
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loading required package: sp
```

```
## Warning: package 'sp' was built under R version 4.3.3
```

```
## This is INLA_24.05.01-1 built 2024-05-01 18:49:50 UTC.
##  - See www.r-inla.org/contact-us for how to get help.
##  - List available models/likelihoods/etc with inla.list.models()
##  - Use inla.doc(<NAME>) to access documentation
```

```
library("MASS") #for ginv
```

Somw theory

Random walk 1

The easiest way to simulate a random walk is through the assumption of independent normal distributed increments. We know that '

$$x_{t+1}|x_1, ..., x_t, \sigma = x_{t+1}|x_t, \sigma \sim N(x_t, \sigma^2)$$

Thus, all we need to do is sample from the standard normal, scale them by σ and add them sequentially. Standard to assert that $x_0 = 0$.

Defining a basic plotting function that will come in handy

```
plot_realizations <- function(df, title, xlabel = "t", ylabel = "y", legend = TRUE){
  #Need to give in a dataframe with fitting colnames, used by the legend
  df$t <- 1:nrow(df)  # Create a time index from 1 to n
  df_long <- df %>% pivot_longer(cols = -t, names_to = "variable", values_to = "value")

  ggplot(df_long, aes(x = t, y = value, color = variable)) +
  geom_line() +
  labs(title = title,
       x = xlabel, y = ylabel) +
    if (legend) {theme(legend.title = element_blank())}
    else {theme(legend.position = "none")}
}
```

Now, lets define functionality for the random walk.

```
RW1 <- function(sigma, N){
  # sigma^2 is the variance for the transitions and N is the number of points
  x <- rep(0, N)
  z <- sigma * rnorm(N-1)
  for(j in 2:N){
    x[j] <- x[j-1] + z[j-1]
  }
  return(x)
}

#Also making a normalized RW1, ie. it sums to zero
Norm_RW1 <- function(sigma, N){
  # sigma^2 is the variance for the transitions and N is the number of points
  x <- rep(0, N)
  z <- sigma * rnorm(N-1)
  for(j in 2:N){
    x[j] <- x[j-1] + z[j-1]
  }
  return(x - mean(x)) #makes the mean zero
}

#Parameters for simulation of RW1
n <- 10
N <- 100
sigma <- 1
```

```r
df <- data.frame(matrix(NA, nrow = N, ncol = n))
set.seed(0)
for (i in 1:n){
  df[, i] <- RW1(sigma, N)
}

RW1_plot <- plot_realizations(df, "RW1 with N=100", legend = FALSE)
```

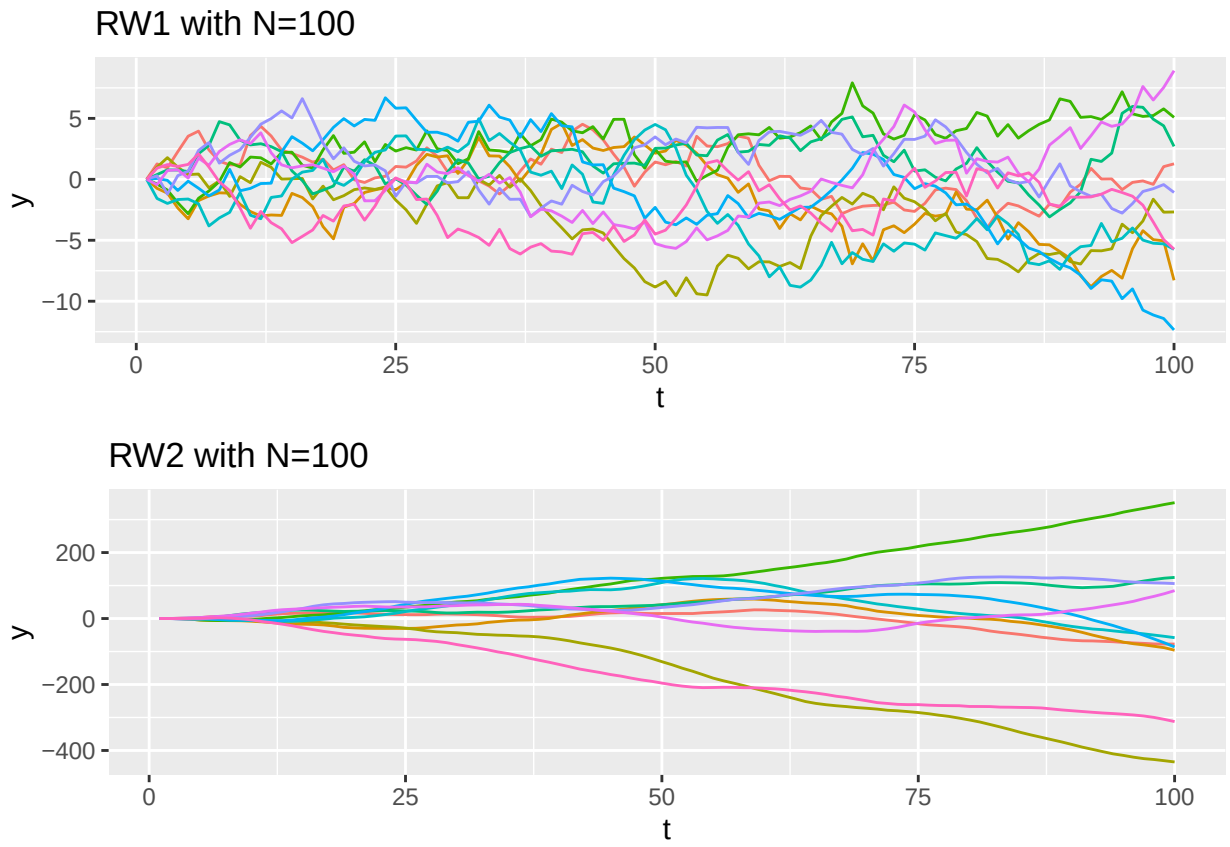
Random walk 2

$$x_t - 2x_{t+1} + x_{t+2} \sim N(0, \sigma^2) x_{t+2} \sim N(2x_{t+1} - x_t, \sigma^2) x_{t+2} = 2x_{t+1} - x_t + \epsilon_t \epsilon_t \sim N(0, \sigma^2)$$

```r
RW2 <- function(sigma, N){
  # sigma^2 is the variance for the transitions and N is the number of points
  x <- rep(0, N)
  z <- sigma * rnorm(N-1)
  for(j in 3:N){
    x[j] <- 2*x[j-1] - x[j-2] + z[j-1]
  }
  return(x)
}

#Parameters for simulation of RW2
n <- 10
N <- 100
sigma <- 1

df2 <- data.frame(matrix(NA, nrow = N, ncol = n))
set.seed(0)
for (i in 1:n){
  df2[, i] <- RW2(sigma, N)
}

# Plot all lines using ggplot
RW2_plot <- plot_realizations(df2, "RW2 with N=100", legend = FALSE)
```

Visualize the plots.

```r
RW_figure <- ggarrange(RW1_plot, RW2_plot, ncol = 1)
RW_figure
```



Simulation study - Gaussian data with or without an offset

We want to conduct a small simulation study to see if the adaptive models improve the standard models in situations with shocks.First we will start with assessing the performance on non.shocked data.

Simulation of non-shocked Gaussian data

We will simulate data with a latent temporal structured random effect as a RW1, denoted $\mathbf{x}$. The total Bayesian hierarchical model can be described as

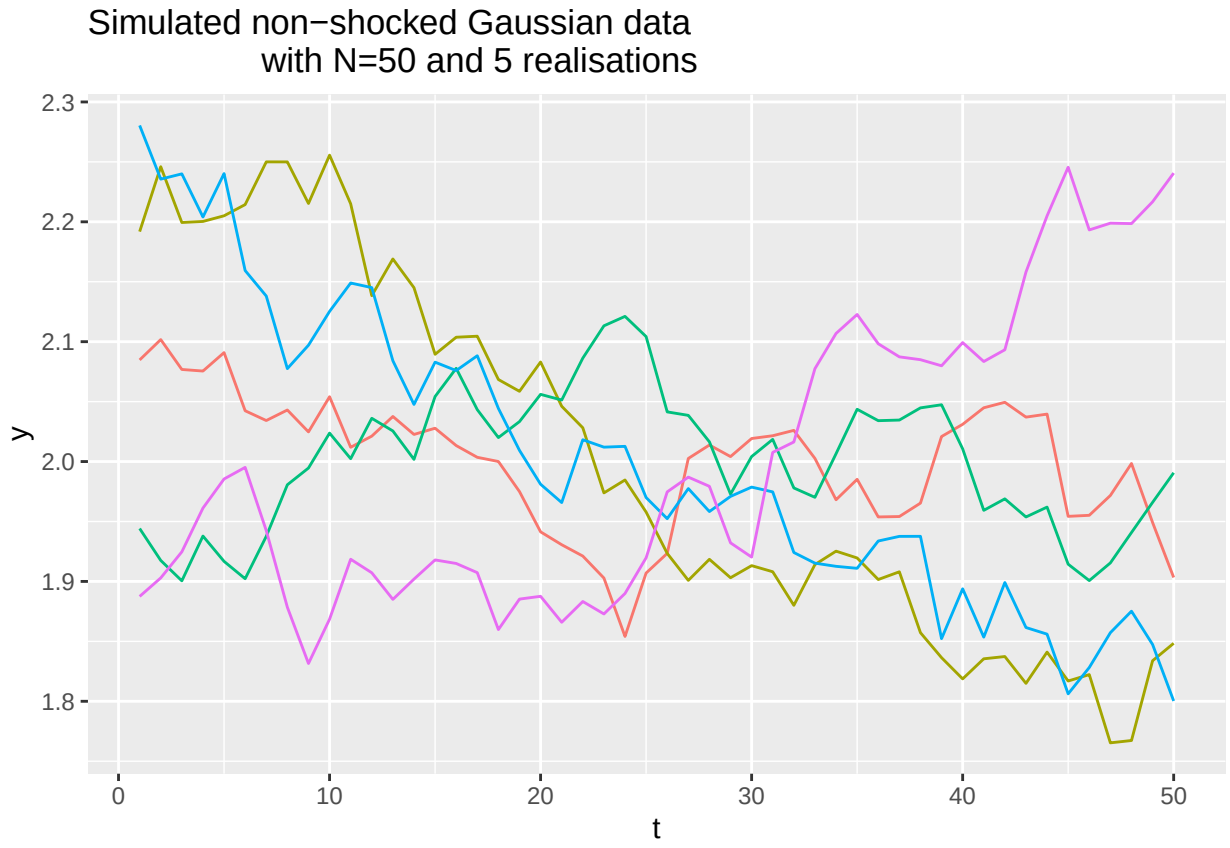
$$y_t|\eta_t \sim N(\eta_t, \sigma_t^2)$$

$$\eta_t = \mu + x_t.$$

For the moment we assume constant σ_t for all timepoints, and choose some fixed σ_r for the random walk. First, lets make the general functions.

```
#function to simulate a realization y
sim_non_shocked_gaussian_data <- function(N, mu, sigma_obs, sigma_rw){
  #N timepoints, mean mu, and standard deviations observations and the RW1
  eta <- mu + Norm_RW1(sigma_rw, N)
  y <- sapply(eta, function(r) rnorm(1, mean = r, sd = sigma_obs))
  return(y)
}
```

```r
sim_non_shocked_gaussian_dataframe <- function(N, mu, sigma_obs, sigma_rw, n, seed = 50){
  set.seed(seed)
  df <- data.frame(matrix(NA, nrow = N, ncol = n))
  for(i in 1:n){
    df[, i] <- sim_non_shocked_gaussian_data(N, mu, sigma_obs, sigma_rw)
  }
  return(df)
}

#The dataframe for all non-shocked Gaussian data
N <- 50
n <- 100
sigma_obs <- 0.001
sigma_rw <- 0.03
mu <- 2
NSG_dataframe <- sim_non_shocked_gaussian_dataframe(N, mu, sigma_obs, sigma_rw, n)
NSG_dataframe$t <- 1:N #needed for random effects later

#Visualizing some simulated data
plot_realizations(NSG_dataframe[, 1:5], "Simulated non-shocked Gaussian data
                  with N=50 and 5 realisations", legend = FALSE)
```



Brief testing with INLA on some simulated data

We will fit the simple model in INLA with a latent layer as

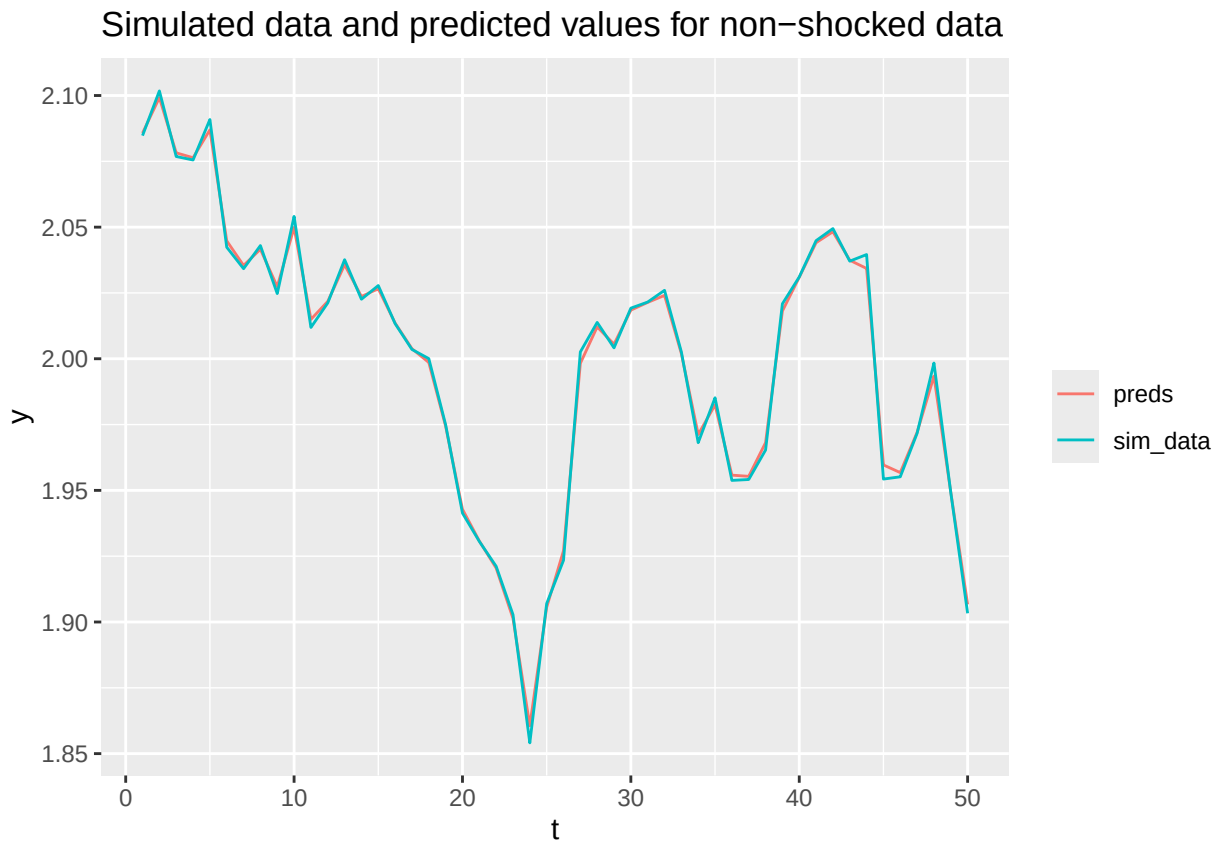
$$\eta_t = \mu + x_t$$

where x_t is a RW1 with some precision τ with a default prior. We use a Gaussian likelihood in the observation layer, again with a default prior for the precision. Same for μ.

```
#Data preperation
NSG_data <- data.frame(matrix(c(NSG_dataframe[, 1], 1:N), nrow = N, ncol = 2))
colnames(NSG_data) <- c("y", "time") #makes the colnames match the formula

#The INLA model
formula <- y ~ f(time, model = "rw1") #intercept is included automatically
res <- inla(formula, family = "gaussian", data = NSG_data)
```

```
## Warning in .recacheSubclasses(def@className, def, env): undefined subclass
## "ndiMatrix" of class "replValueSp"; definition not updated
```

```
#For plotting the data and the predicted values
plot_df <- data.frame(matrix(c(NSG_dataframe[, 1], res$summary.fitted.values$mean), ncol = 2))
colnames(plot_df) <- c("sim_data", "preds") #for legends in the plot

plot_realizations(plot_df, "Simulated data and predicted values for non-shocked data")
```



We see that the model predictions align well with the data it is fit on.

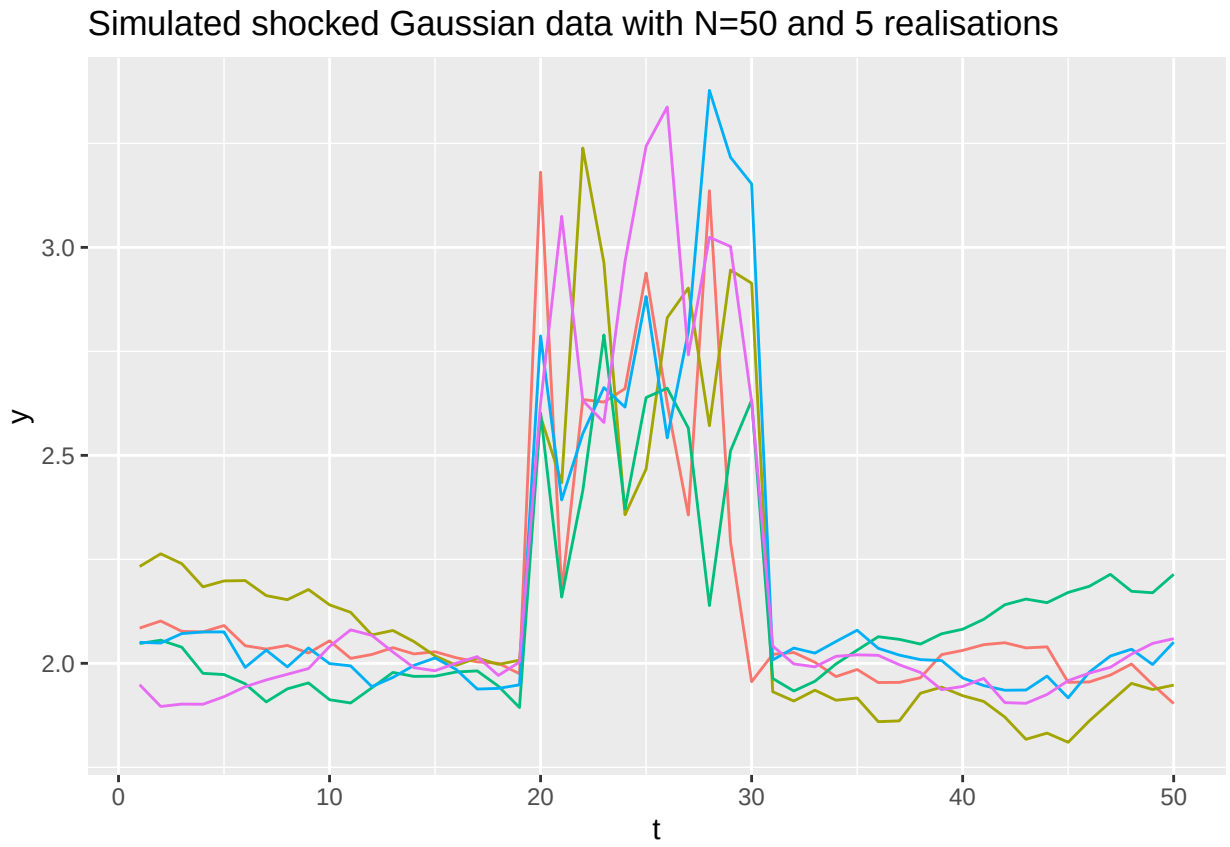
Simulation of shocked Gaussian data

We now want to simulate Gaussian data where we know that there are shocks on certain timepoints. This could be modeled by adding or subtracting a slightly randomized value from the chosen points. Lets say we want a shock from $t = 20$ to $t = 30$, which could be done by adding a $s_t \overset{iid}{\sim} N(0.7, 0.3)$ for instance.

```
#Parameters and constants
N <- 50
n <- 100
sigma_obs <- 0.001
sigma_rw <- 0.03
mu <- 2

#A comparison of non-shocked and shocked simulated data
NSG_data_test <- sim_non_shocked_gaussian_data(N, mu, sigma_obs, sigma_rw)
offset <- c(rep(0, 19),rnorm(11, 0.7, 0.3), rep(0, 20))
SG_data_test <- NSG_data_test + offset

example_dataframe <- data.frame(matrix(c(NSG_data_test, SG_data_test),
                                       nrow = N, ncol = 2))
colnames(example_dataframe) <- c("NS", "S") #Non-shocked and shocked

plot_realizations(example_dataframe, "Comparison of shocked and non-shocked simulated data", "t", "y")
```

Lets make some general functions for simulating shocked Gaussian data.

```r
#function to simulate a shocked realization y
sim_shocked_gaussian_data <- function(N, mu, sigma_obs, sigma_rw, t_start = 20, t_end = 30, mu_offset =
  #N timepoints, mean mu, and standard deviations for observations and the RW1
  #t_start and t_end bound the offset area with specified mean and sd
  eta <- mu + Norm_RW1(sigma_rw, N)
  y <- sapply(eta, function(r) rnorm(1, mean = r, sd = sigma_obs))
  offset <- c(rep(0, t_start - 1),rnorm(t_end - t_start + 1, mu_offset, sigma_offset), rep(0, N - t_end)
  y_offset <- y + offset
  return(y_offset)
}

sim_shocked_gaussian_dataframe <- function(N, mu, sigma_obs, sigma_rw, t_start = 20, t_end = 30, mu_offs
  set.seed(seed)
  df <- data.frame(matrix(NA, nrow = N, ncol = n))
  for(i in 1:n){
    df[, i] <- sim_shocked_gaussian_data(N, mu, sigma_obs, sigma_rw, t_start, t_end, mu_offset, sigma_o
  }
  return(df)
}

#Shocked Gaussian dataframe
SG_dataframe <- sim_shocked_gaussian_dataframe(N, mu, sigma_obs, sigma_rw, n=n)
SG_dataframe$t <- 1:N #needed for random effects later

#Visualizing some simulated data
plot_realizations(SG_dataframe[, 1:5], "Simulated shocked Gaussian data with N=50 and 5 realisations",
```



Implementing the adaptive random walk in INLA

As this is somewhat complicated and new to me, I will start by a slightly easier example, namely the RW1. Can then also compare it to the already defined RW1 in INLA to ensure it works as intended. First some basic theory on defining random effects in INLA from https://becarioprecario.bitbucket.io/inla-gitbook/ch-newmodels.html .

Defining new latent random effects in INLA

New latent effects must be specified as GMRFs. So we need μ, Q and θ and its log-priors and initial values. Also need a graph, which I think can just be Q as well, and some log-normalizing constant. As θ is parametrized as $\theta_1 = \log(\tau)$ and $\theta_2 = \text{logit}(\rho)$ where τ is the precision and ρ is the spatial dependence, which I think we define to be 1. The general structure of the inla.rgeneric is shown below:

```
inla.rgeneric.somemodel = function(
  cmd = c("graph", "Q", "mu", "initial", "log.norm.const","log.prior", "quit"),
  theta = NULL)
{
  # for reference and potential storage for objects to
  # cache, this is the environment of this function
  # which holds arguments passed as `...` in
  # `inla.rgeneric.define()`.
  envir = parent.env(environment())
  graph = function(){ <to be completed> }
```

```
  Q = function() { <to be completed> }
  mu = function() { <to be completed> }
  log.norm.const = function() { <to be completed> }
  log.prior = function() { <to be completed> }
  initial = function() { <to be completed> }
  quit = function() { <to be completed> }

  # sometimes this is useful, as argument 'graph' and 'quit'
  # will pass theta=numeric(0) (or NULL in R-3.6...) as
  # the values of theta are NOT
  # required for defining the graph. however, this statement
  # will ensure that theta is always defined.

  if (!length(theta)) theta = initial()
  val = do.call(match.arg(cmd), args = list())
  return (val)
}

#if W is a needed argument
somemodel.model <- inla.rgeneric.define(inla.rgeneric.somemodel, W = W)
```

Implementing RW1 in INLA

In a RW1 we only have one hyperparameter, namely τ. So we get $\theta = \log(\tau)$ and the precision matrix is defined previously. Lets first define a function for the geometric variance, defined in the overleaf document, to scale the precision matrix. Use the function ginv from the library MASS to calculate the generalized inverse.

```
geometric_variance <- function(R) {
  #Input: R is a square structure matrix, often sparse
  N <- dim(R)[1]
  GV <- exp(1 / N * sum(log(diag(ginv(R)))))
  return(GV)
}
```

Then, lets define the inla.rgeneric function with all its necessary subfunctions.

```
inla.rgeneric.RW1.model = function(
  cmd = c("graph", "Q", "mu", "initial", "log.norm.const","log.prior", "quit"),
  theta = NULL)
{
  #Input:
  #N is the number of timepoints
  #R_star is the scaled structure matrix

  envir = parent.env(environment())

  interpret_theta <- function() { return(list(tau = exp(theta[1L])))}

  graph <- function() {return(Q())}

  Q <- function() {
```

```
    p <- interpret_theta()
    Q <- p$tau * R_star
    return(inla.as.sparse(Q))
  }

  mu <- function() {return(numeric(0))}

  initial <- function() {return(4)}#default for precisions: initial = 4

  log.norm.const <- function() {return(numeric(0))} #Inla computes it

  log.prior <- function() {#default: shape = 1, rate = 0.00005 for tau
    p <- interpret_theta()
    prior <- dgamma(p$tau, shape = 1, rate = 0.00005, log = TRUE) + theta[1L]
    return(prior)
  }

  quit <- function() {return(invisible())}

  #to ensure theta is defined
  if (!length(theta)) theta = initial()

  vals <- do.call(match.arg(cmd), args = list())
  return(vals)
}
```

We need to pass the function above the number of timepoints N and the scaled structure matrix R^*. The scaled structure matrix for a RW1 is defined by the fucntion below, followed by some testing.

```
Q <- function(N) {
  # Input: N timepoints
  R <- toeplitz(c(2, -1, rep(0, N - 2)))# 2 on diag and -1 on firstdiags
  R[1, 1] <- R[N, N] <- 1 # 1 for first and last diag element
  gv <- geometric_variance(R)
  R_star <- gv * R
  return(R_star) #returns the scaled structure matrix for a RW1
}

N <- 50 #is defined further up as well
R_star <- Q(N)
RW1_model <- inla.rgeneric.define(inla.rgeneric.RW1.model, N = N, R_star = R_star)
```

The RW1_model above is now a custom latent effect which can be included in INLA formulas to define models. However, it is only defined for $N = 50$, and we need to define separate ones for other N.Now, lets check if it works as intended on some data from earlier. Then we need to define the INLA formula where we add a constraint to enforce a sum to zero constraint as the model includes an intercept.

```
#The INLA formula for a latent model with intercept and user defined RW1
formula_M <- y ~ f(time, model = RW1_model, extraconstr = list(A = matrix(1, nrow = 1, ncol = N), e = 0
res_M <- inla(formula_M, family = "gaussian", data = NSG_data)

#The standard RW1 model from INLA
formula_I <- y ~ f(time, model = "rw1")
```

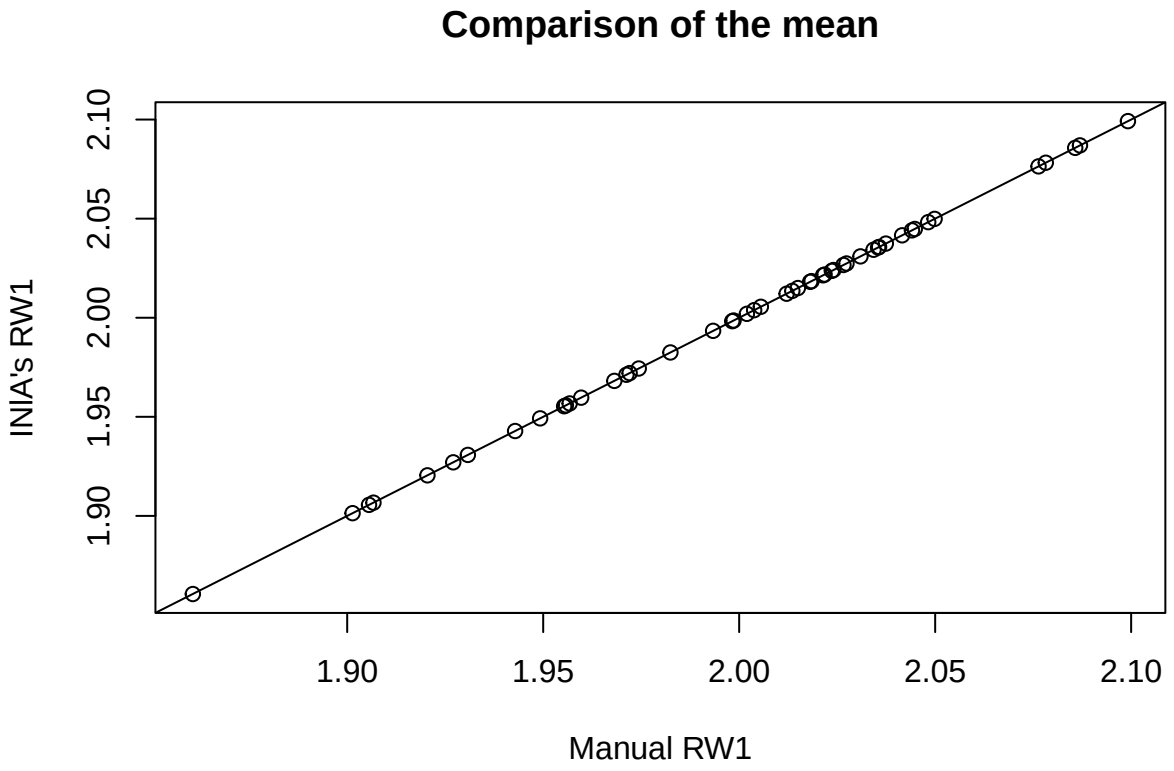
```r
res_I <- inla(formula_I, family = "gaussian", data = NSG_data)

summary(res_M)
```

```
## Time used:
##     Pre = 0.485, Running = 4.68, Post = 0.127, Total = 5.29
## Fixed effects:
##             mean    sd 0.025quant 0.5quant 0.975quant mode kld
## (Intercept)    2 0.001      1.998        2      2.002    2   0
##
## Random effects:
##   Name     Model
##     time RGeneric2
##
## Model hyperparameters:
##                                           mean       sd 0.025quant 0.5quant
## Precision for the Gaussian observations 29206.60 28160.69    3729.20 21002.28
## Theta1 for time                             5.18     0.23       4.72     5.18
##                                         0.975quant     mode
## Precision for the Gaussian observations   1.04e+05 10052.12
## Theta1 for time                           5.63e+00     5.19
##
## Marginal log-Likelihood:  98.00
##  is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

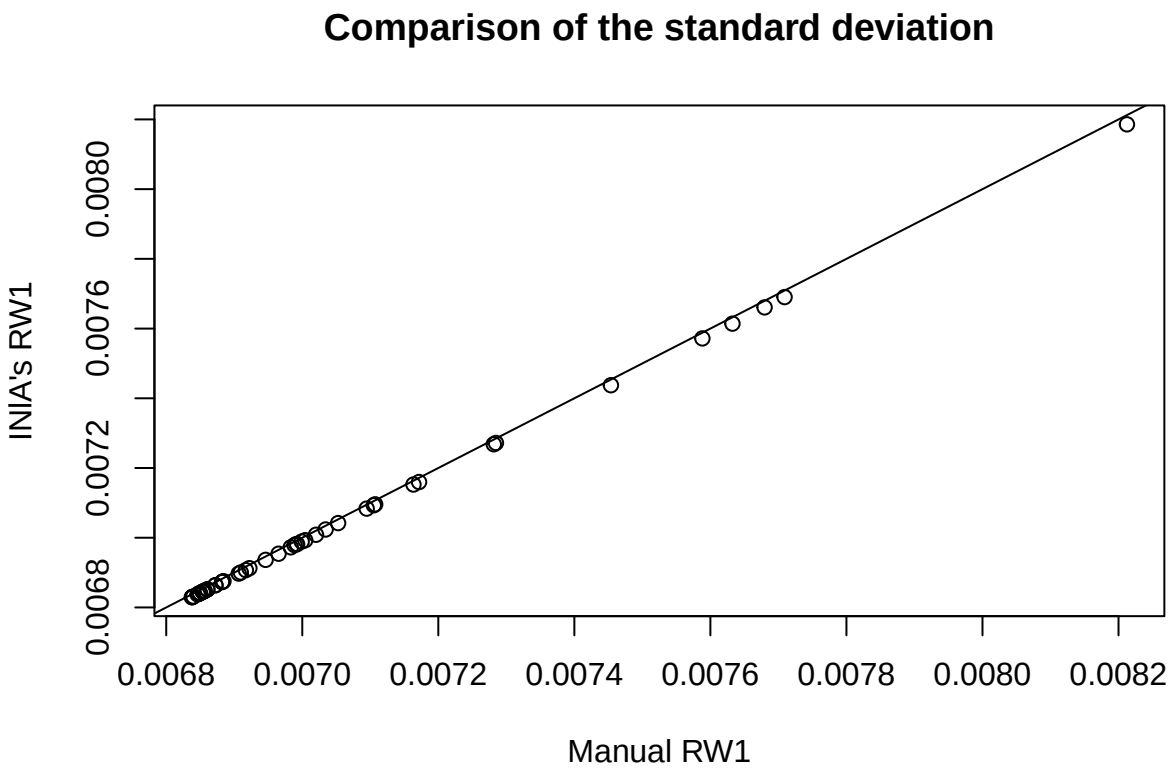
```r
summary(res_I)
```

```
## Time used:
##     Pre = 0.479, Running = 0.551, Post = 0.16, Total = 1.19
## Fixed effects:
##             mean    sd 0.025quant 0.5quant 0.975quant mode kld
## (Intercept)    2 0.001      1.998        2      2.002    2   0
##
## Random effects:
##   Name     Model
##     time RW1 model
##
## Model hyperparameters:
##                                           mean       sd 0.025quant 0.5quant
## Precision for the Gaussian observations 29303.78 28242.40    3751.19 21077.28
## Precision for time                       1373.98   318.72     847.20  1340.31
##                                         0.975quant     mode
## Precision for the Gaussian observations  104070.72 10104.86
## Precision for time                         2094.85  1277.62
##
## Marginal log-Likelihood:  98.01
##  is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

```r
#plots to compare the model output
plot(res_M$summary.fitted.values$mean, res_I$summary.fitted.values$mean, main = "Comparison of the mean
abline(0, 1)
```




```r
plot(res_M$summary.fitted.values$sd, res_I$summary.fitted.values$sd, main = "Comparison of the standard
abline(0, 1)
```



From the summaries it is clear they are very similar. However, not exactly equal, as for instance the mean of the Gaussian precision. This is supported by the plots of the means and standard deviations, which show that they are almost the same, but clearly not identical.

Implementing the adaptive RW1

We now make the previous model more flexible by allowing for two different precisions in the random walk. The new precision is used for transitions involving conflict years which we define.So, we start by computing the scaled structure matrices R1_star and R2_star which we need as inputs for the adaptive random walk.

```
Scaled_structure_matrices_for_ARW1 <- function(N, conflict_years) {
  #Input:
  #N timepoints
  #conflict_years is a list with the conflict years

  R1 <- matrix(0, nrow = N, ncol = N) #should be N = 50, non-conflict
  R2 <- matrix(0, nrow = N, ncol = N) #should be N = 50, conflict
  for( i in 1:(N - 1)){
    if(i %in% conflict_years | (i + 1) %in% conflict_years) {
      R2[c(i, i+1), c(i, i+1)] <- R2[c(i, i+1), c(i, i+1)] + c(1, -1, -1, 1)
    }
    else {
      R1[c(i, i+1), c(i, i+1)] <- R1[c(i, i+1), c(i, i+1)] + c(1, -1, -1, 1)
    }
  }
  gv <- geometric_variance(R1 + R2) #scaling
```

```
  return(list(R1 = R1*gv, R2 = R2*gv))
}

#testing
R_star_list <- Scaled_structure_matrices_for_ARW1(7, c(3, 4, 5))
R_star_list$R1
```

```
##           [,1]      [,2] [,3] [,4] [,5]      [,6]      [,7]
## [1,]  1.039708 -1.039708    0    0    0  0.000000  0.000000
## [2,] -1.039708  1.039708    0    0    0  0.000000  0.000000
## [3,]  0.000000  0.000000    0    0    0  0.000000  0.000000
## [4,]  0.000000  0.000000    0    0    0  0.000000  0.000000
## [5,]  0.000000  0.000000    0    0    0  0.000000  0.000000
## [6,]  0.000000  0.000000    0    0    0  1.039708 -1.039708
## [7,]  0.000000  0.000000    0    0    0 -1.039708  1.039708
```

```
R <- R_star_list$R1 + R_star_list$R2
R
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,]  1.039708 -1.039708  0.000000  0.000000  0.000000  0.000000  0.000000
## [2,] -1.039708  2.079416 -1.039708  0.000000  0.000000  0.000000  0.000000
## [3,]  0.000000 -1.039708  2.079416 -1.039708  0.000000  0.000000  0.000000
## [4,]  0.000000  0.000000 -1.039708  2.079416 -1.039708  0.000000  0.000000
## [5,]  0.000000  0.000000  0.000000 -1.039708  2.079416 -1.039708  0.000000
## [6,]  0.000000  0.000000  0.000000  0.000000 -1.039708  2.079416 -1.039708
## [7,]  0.000000  0.000000  0.000000  0.000000  0.000000 -1.039708  1.039708
```

Now, lets define the adaptive RW1.

```
inla.rgeneric.AdaptiveRW1.model = function(
  cmd = c("graph", "Q", "mu", "initial", "log.norm.const","log.prior", "quit"),
  theta = NULL)
{
  #Input:
  #N is the number of timepoints
  #R_star_list contains R1_star and R2_star, the scaled structure matrices

  envir = parent.env(environment())

  interpret_theta <- function() { return(list(tau1 = exp(theta[1L]),
                                              tau2 = exp(theta[2L])))}

  graph <- function() {return(Q())}

  Q <- function() {
    p <- interpret_theta()
    Q <- R_star_list$R1 * p$tau1 + R_star_list$R2 * p$tau2
    return(inla.as.sparse(Q)) #sparse representation
  }

  mu <- function() {return(numeric(0))}
```

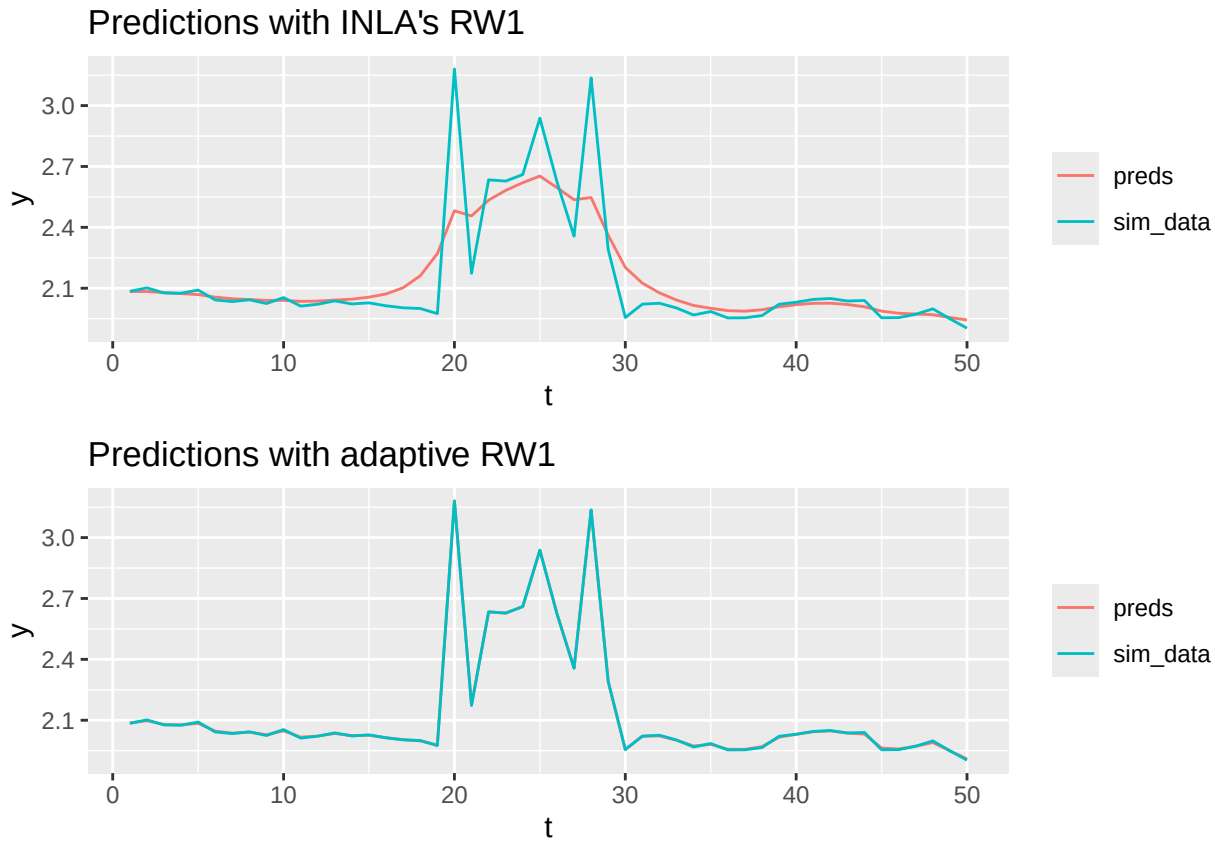
```
  initial <- function() {return(c(4, 4))}#Default initial for precisions

  log.norm.const <- function() {return(numeric(0))}

  log.prior <- function() {#default: shape = 1, rate = 0.00005
    p <- interpret_theta()
    prior <- dgamma(p$tau1, shape = 1, rate = 0.00005, log = TRUE) + theta[1L]+
             dgamma(p$tau2, shape = 1, rate = 0.00005, log = TRUE) + theta[2L]
    return(prior)
  } #the theta terms come from the transformation of variables to the log scale

  quit <- function() {return(invisible())}

  #to ensure theta is defined
  if (!length(theta)) theta = initial()

  vals <- do.call(match.arg(cmd), args = list())
  return(vals)
}

#Computing the scaled R's and defining the ARW1 model
N <- 50 #is defined further up as well
conf_years <- 20:30 #as in the generated data further up
R_star_list <- Scaled_structure_matrices_for_ARW1(N, conf_years)
ARW1_model <- inla.rgeneric.define(inla.rgeneric.AdaptiveRW1.model,
                                   N = N, R_star_list = R_star_list)
```

Lets do some testing for shocked data.

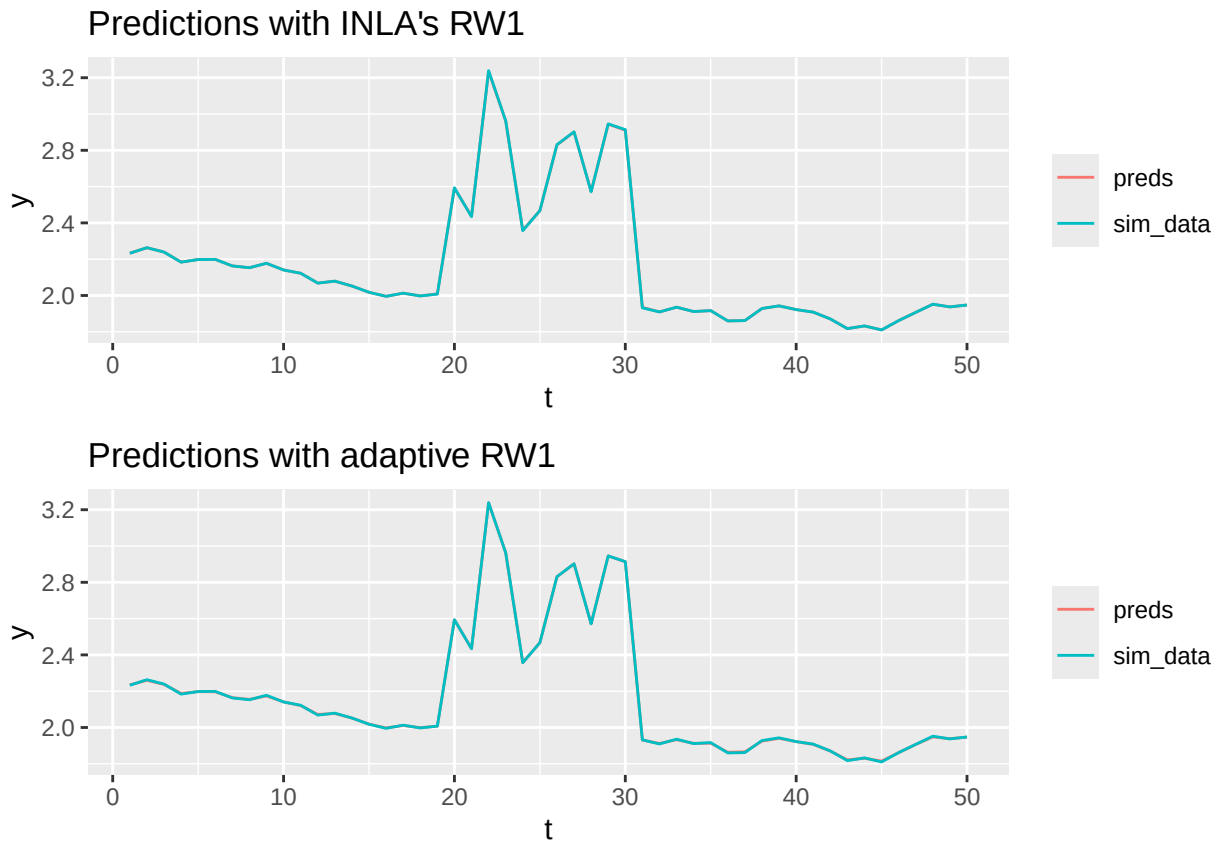
```
#The INLA formula for a latent model with intercept and a RW1
formula_I <- y ~ f(time, model = "rw1")

#The INLA formula for an adaptive RW1
formula_ARW1 <- y ~ f(time, model = ARW1_model,
                  extraconstr = list(A = matrix(1, nrow = 1, ncol = N), e = 0))

figure_list <- list()
for( i in 1:5) {
  test_data <- data.frame(matrix(c(SG_dataframe[, i], 1:N), nrow = N, ncol = 2))
  colnames(test_data) <- c("y", "time") #makes the colnames match the formula

  res_I <- inla(formula_I, family = "gaussian", data = test_data)
  plot_df_I <- data.frame(matrix(c(SG_dataframe[, i], res_I$summary.fitted.values$mean), ncol = 2))
  colnames(plot_df_I) <- c("sim_data", "preds")

  plot_I <- plot_realizations(plot_df_I, "Predictions with INLA's RW1")

  #The adaptive RW1
  res_ARW1 <- inla(formula_ARW1, family = "gaussian", data = test_data)

  plot_df_ARW1 <- data.frame(matrix(c(SG_dataframe[, i], res_ARW1$summary.fitted.values$mean), ncol = 2
  colnames(plot_df_ARW1) <- c("sim_data", "preds")
```

```
  plot_ARW1 <- plot_realizations(plot_df_ARW1, "Predictions with adaptive RW1")

  figure_list[[i]] <- ggarrange(plot_I, plot_ARW1, ncol = 1)
}
figure_list
```

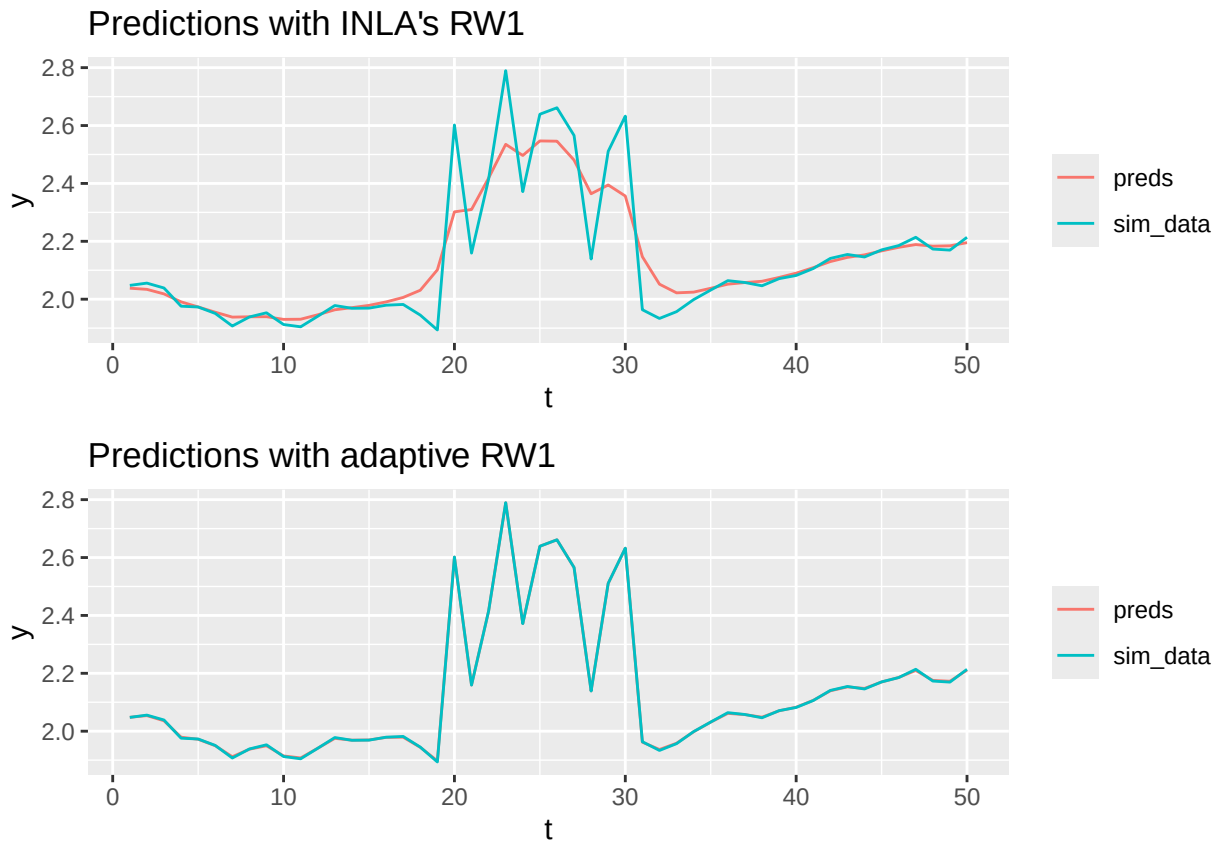
```
## [[1]]
```



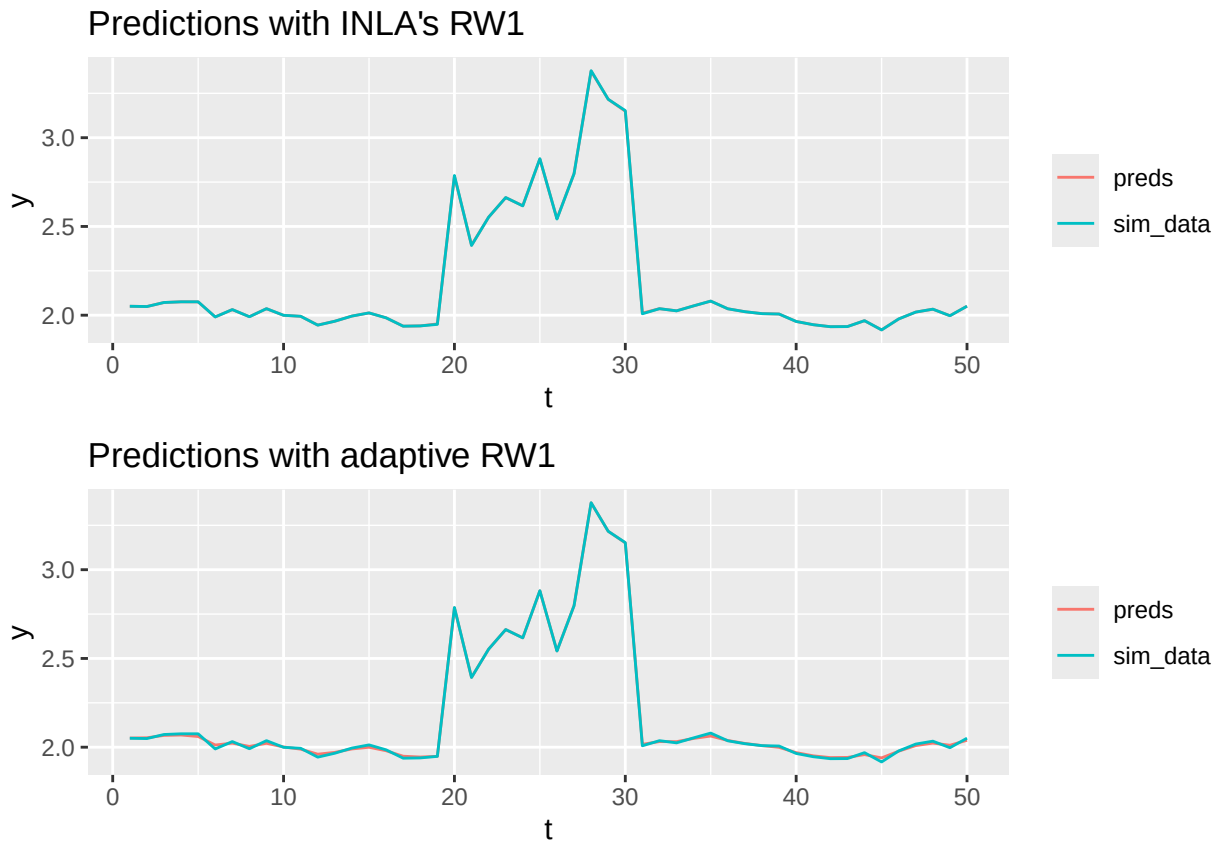

```
##
## [[2]]
```


Predictions with INLA's RW1
3.2
2.8
2.4
2.0
y
0
10
20
30
40
50
t
preds
sim_data
Predictions with adaptive RW1
3.2
2.8
2.4
2.0
y
0
10
20
30
40
50
t
preds
sim_data

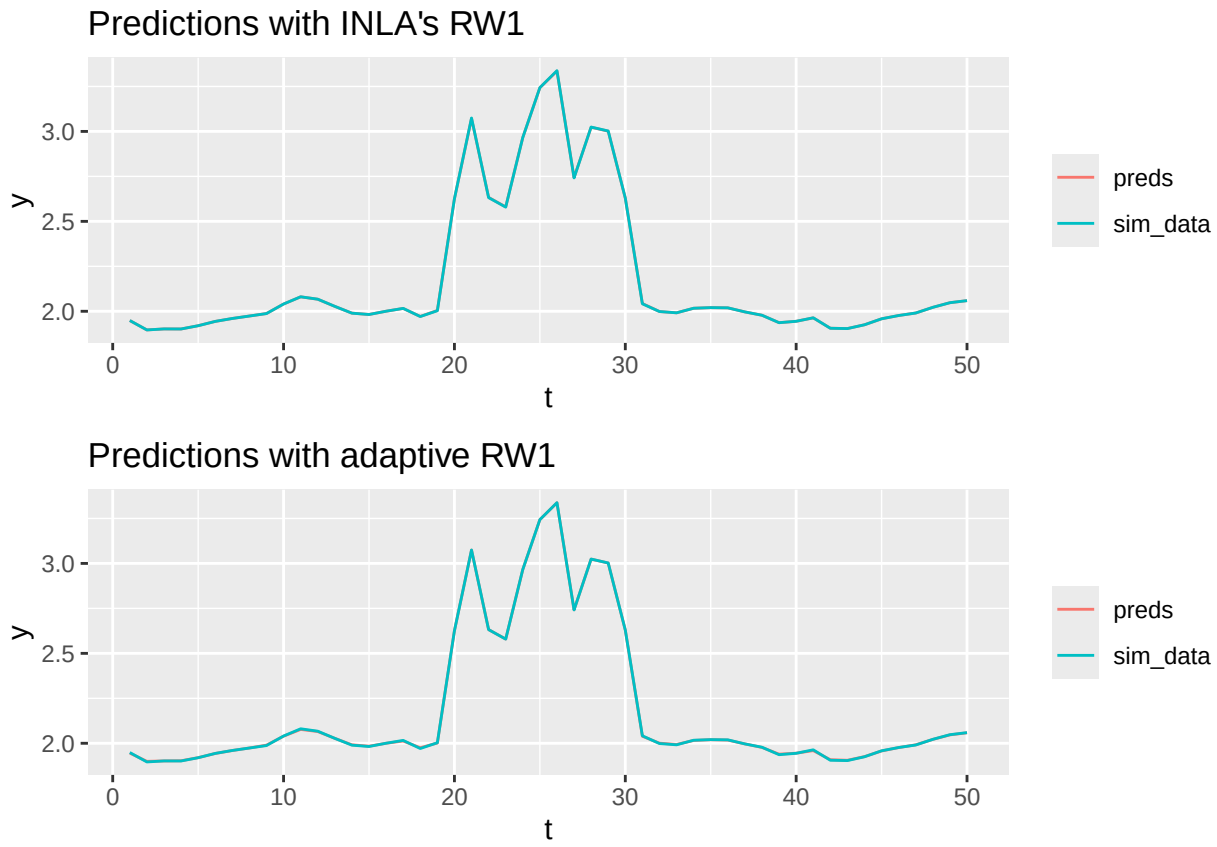
```
##
## [[3]]
```


Predictions with INLA's RW1
2.8
2.6
2.4
2.2
2.0
y
0
10
20
30
40
50
t
preds
sim_data
Predictions with adaptive RW1
2.8
2.6
2.4
2.2
2.0
y
0
10
20
30
40
50
t
preds
sim_data

```
##
## [[4]]
```


Predictions with INLA's RW1
y
3.0
2.5
2.0
0
10
20
30
40
50
t
preds
sim_data
Predictions with adaptive RW1
y
3.0
2.5
2.0
0
10
20
30
40
50
t
preds
sim_data

```
##
## [[5]]
```



We observe that the standard RW1 struggles a lot with the first and third realization, and the rest of the plots seem nice.

Model evaluation

Model criteria

We will evaluate the models with root mean square error (RMSE) and average proper logarithmic scoring (LS).

Root mean square eroor A common model criteria is the RMSE. We define a function to calculate this below. The lower RMSE the better.

```
RMSE <- function(data, preds){
  return(sqrt( sum((data - preds)**2) / length(data))) #definition of RMSE
}
```

Average proper logarithmic scoring rule From a paper by Gneiting and Raftery (2007).

$$\text{LS}(p, \omega) = \log p(\omega)$$

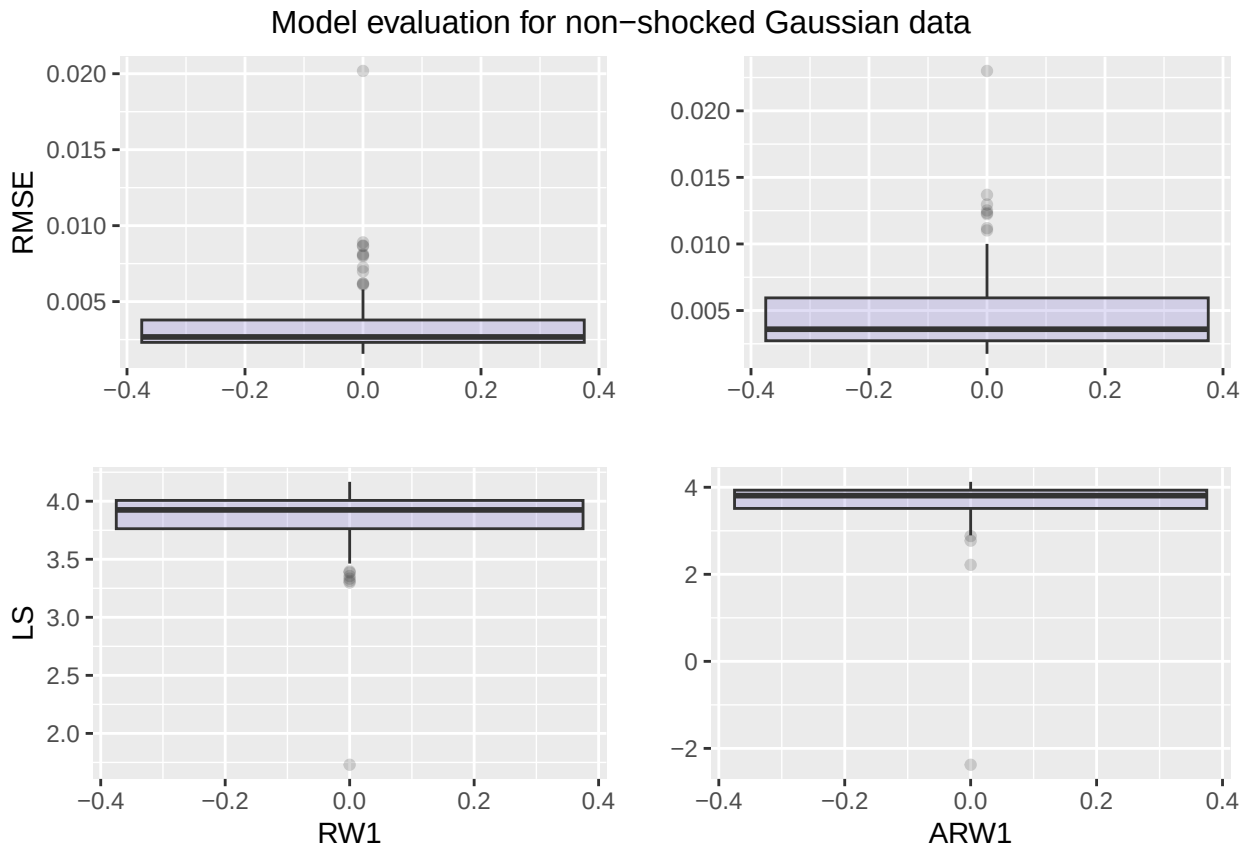
where p is the predicted distribution of a point, which we get from INLA, and ω is the observed value, which is the specific datapoint. We then take the average of the score for all the data points. The higher average proper LS the better.

```r
average_proper_LS <- function(res, data){
  #res is an inla object from calling a model on data, a vector of datapoints
  mean <- res$summary.fitted.values$mean
  sd <- res$summary.fitted.values$sd
  p <- dnorm(data, mean = mean, sd = sd)
  return(mean(log(p)))
}
```

Model evaluation for non-shocked data

Lets do some rigorous testing for the entire dataframes form earlier and evaluate them by the chosen model criterias, RMSE and LS. First lets compare the models for non-shocked Gaussian data.

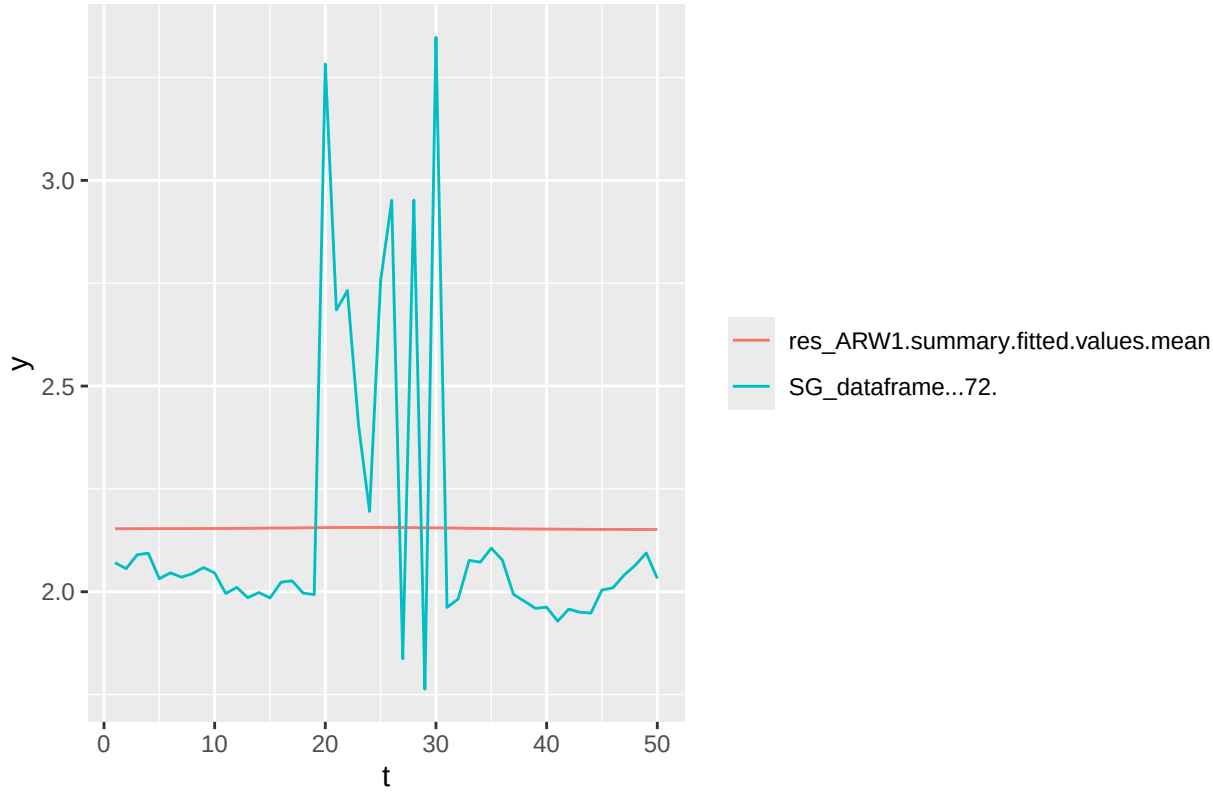
```r
Model_eval_NSG <- data.frame(matrix(NA, nrow = n, ncol = 4))
colnames(Model_eval_NSG) <- c("RMSE_RW1", "LS_RW1", "RMSE_ARW1", "LS_ARW1")

formula_RW1 <- y ~ f(time, model = "rw1")
for(i in 1:n){#iterate over each simulated realization
  test_data <- NSG_dataframe[, c(i, n + 1)] #gets the i-th realization and time
  colnames(test_data) <- c("y", "time") #makes the colnames match the formula

  res_RW1 <- inla(formula_RW1, family = "gaussian", data = test_data)
  LS_RW1 <- average_proper_LS(res_RW1, NSG_dataframe[, i])
  RMSE_RW1 <- RMSE(NSG_dataframe[, i], res_RW1$summary.fitted.values$mean )

  res_ARW1 <- inla(formula_ARW1, family = "gaussian", data = test_data)
  LS_ARW1 <- average_proper_LS(res_ARW1, NSG_dataframe[, i])
  RMSE_ARW1 <- RMSE(NSG_dataframe[, i], res_ARW1$summary.fitted.values$mean )

  Model_eval_NSG[i, ] <- c(RMSE_RW1, LS_RW1, RMSE_ARW1, LS_ARW1)
}

#boxplots of the differences
#boxplot(Model_eval_NSG["RMSE_RW1"] - Model_eval_NSG["RMSE_ARW1"],ylab = "RMSE")
#boxplot(Model_eval_NSG["LS_ARW1"] - Model_eval_NSG["LS_RW1"], ylab = "LS")

myboxplot <- function(data, xlabel = "", ylabel = "") {
  df <- data.frame(d = data[,1])
  BPlot <- ggplot(df, aes(y = d)) +
       geom_boxplot(fill = "slateblue", alpha = 0.2) +
    labs( x = xlabel, y = ylabel)
  return(BPlot)
}

#Plotting
plot_RMSE_RW1_NS <- myboxplot(Model_eval_NSG["RMSE_RW1"], "", "RMSE")
plot_LS_RW1_NS <- myboxplot(Model_eval_NSG["LS_RW1"], "RW1", "LS")
plot_RMSE_ARW1_NS <- myboxplot(Model_eval_NSG["RMSE_ARW1"], "", "")
plot_LS_ARW1_NS <- myboxplot(Model_eval_NSG["LS_ARW1"], "ARW1", "")

plot_eval_NS <- ggarrange(plot_RMSE_RW1_NS, plot_RMSE_ARW1_NS,plot_LS_RW1_NS, plot_LS_ARW1_NS, ncol = 2
annotate_figure(plot_eval_NS, top = text_grob("Model evaluation for non-shocked Gaussian data")) #can a
```



Model evaluation for shocked data

```
Model_eval_SG <- data.frame(matrix(NA, nrow = n, ncol = 4))
colnames(Model_eval_SG) <- c("RMSE_RW1", "LS_RW1", "RMSE_ARW1", "LS_ARW1")

for(i in 1:n){#iterate over each simulated realization
  test_data <- SG_dataframe[, c(i, n + 1)] #gets the i-th realization and time
  colnames(test_data) <- c("y", "time") #makes the colnames match the formula

  res_RW1 <- inla(formula_RW1, family = "gaussian", data = test_data)
  LS_RW1 <- average_proper_LS(res_RW1, SG_dataframe[, i])
  RMSE_RW1 <- RMSE(SG_dataframe[, i], res_RW1$summary.fitted.values$mean )

  res_ARW1 <- inla(formula_ARW1, family = "gaussian", data = test_data)
  LS_ARW1 <- average_proper_LS(res_ARW1, SG_dataframe[, i])
  RMSE_ARW1 <- RMSE(SG_dataframe[, i], res_ARW1$summary.fitted.values$mean )

  Model_eval_SG[i, ] <- c(RMSE_RW1, LS_RW1, RMSE_ARW1, LS_ARW1)
}
#boxplots of the differences
#boxplot(Model_eval_SG["RMSE_RW1"] - Model_eval_SG["RMSE_ARW1"], ylab = "RMSE")
#boxplot(Model_eval_SG["LS_ARW1"] - Model_eval_SG["LS_RW1"], ylab = "LS")

#check behavior for an outlier realization
```

```r
test_data <- SG_dataframe[, c(72, n + 1)] #gets the i-th realization and time
colnames(test_data) <- c("y", "time")
res_ARW1 <- inla(formula_ARW1, family = "gaussian", data = test_data)

plot_realizations(data.frame(res_ARW1$summary.fitted.values$mean, SG_dataframe[, 72]), "")
```

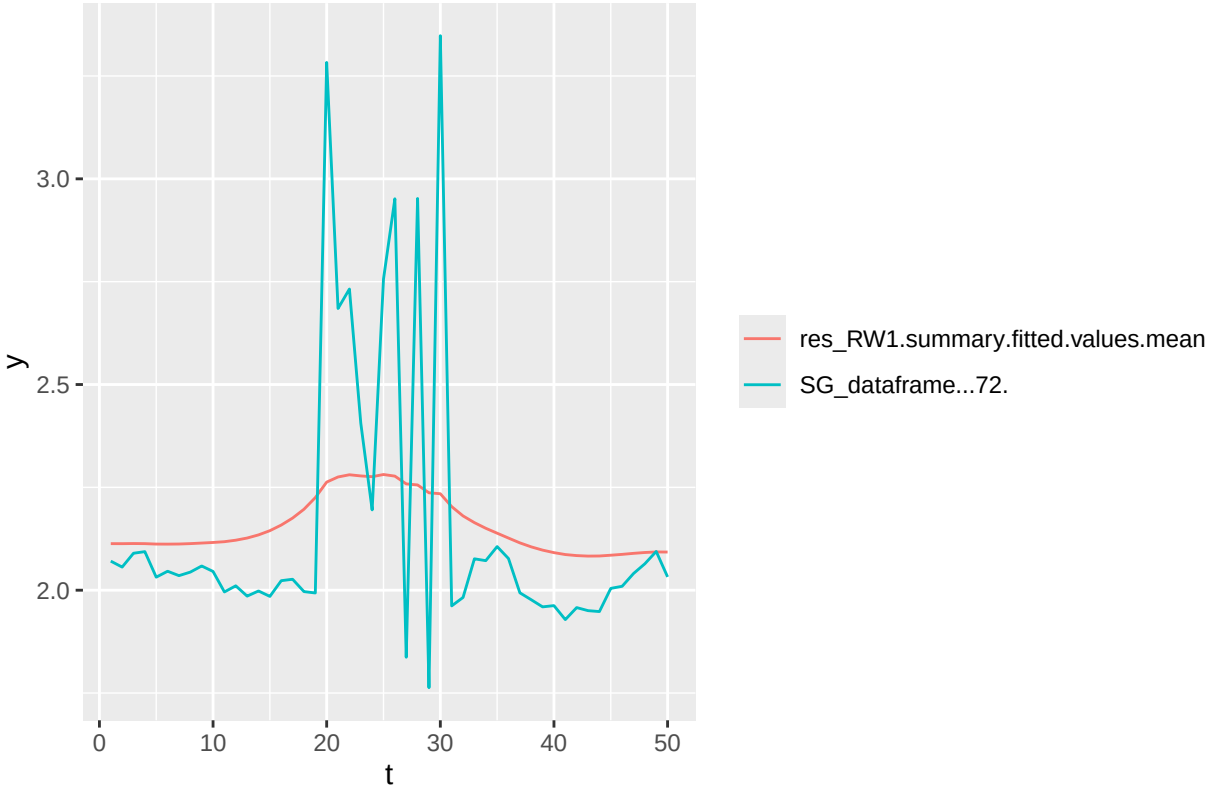



```r
summary(res_ARW1)
```

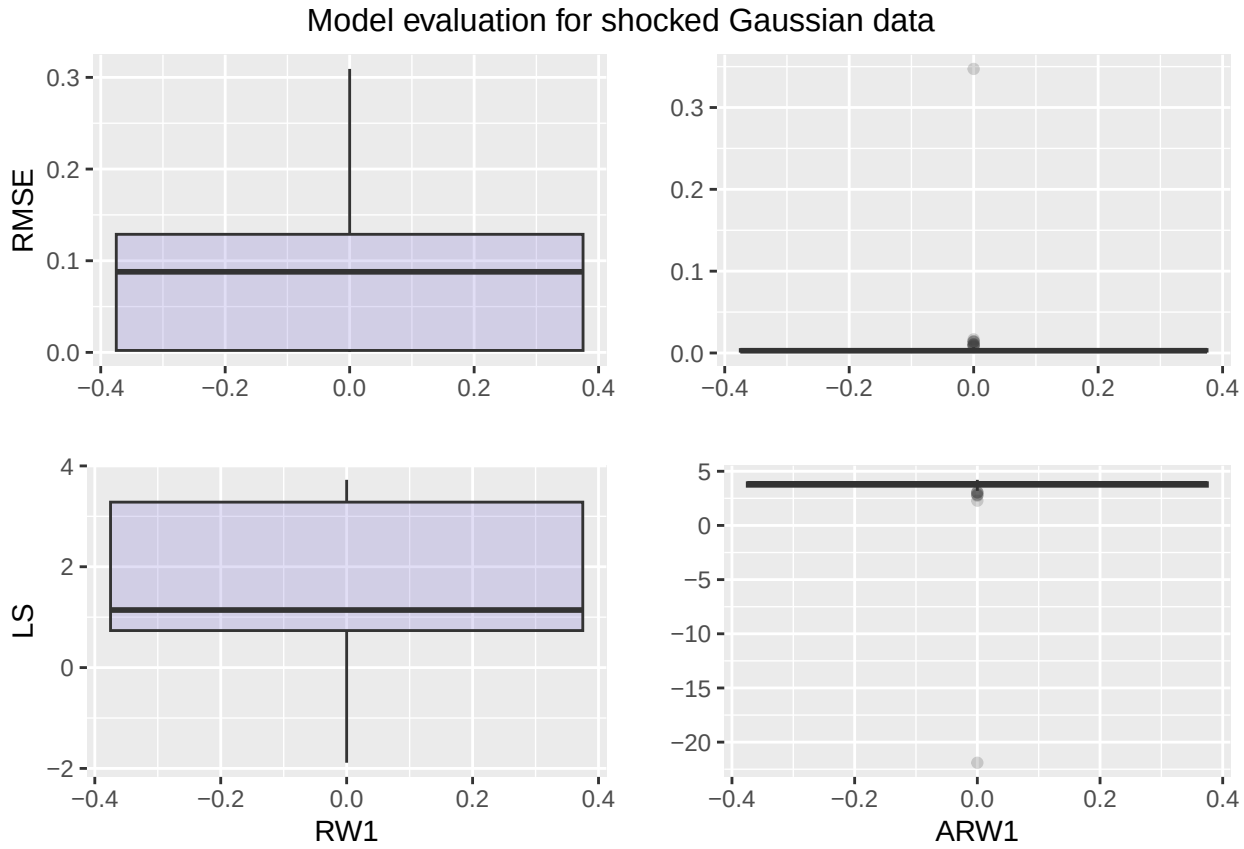
```
## Time used:
##     Pre = 0.488, Running = 5.32, Post = 0.114, Total = 5.92
## Fixed effects:
##              mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## (Intercept) 2.154 0.049      2.057    2.154      2.251 2.154   0
##
## Random effects:
##   Name     Model
##     time RGeneric2
##
## Model hyperparameters:
##                                       mean   sd 0.025quant 0.5quant
## Precision for the Gaussian observations 8.48 1.69       5.58     8.33
## Theta1 for time                         9.49 1.06       7.22     9.55
## Theta2 for time                         9.51 1.04       7.28     9.57
##                                       0.975quant mode
## Precision for the Gaussian observations      12.20 8.08
```

```
## Theta1 for time                                11.35 9.86
## Theta2 for time                                11.36 9.87
##
## Marginal log-Likelihood:  -28.88
##  is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

```
res_RW1 <- inla(formula_RW1, family = "gaussian", data = test_data)
plot_realizations(data.frame(res_RW1$summary.fitted.values$mean, SG_dataframe[, 72]), "")
```




```
#Plotting
plot_RMSE_RW1_S <- myboxplot(Model_eval_SG["RMSE_RW1"], "", "RMSE")
plot_LS_RW1_S <- myboxplot(Model_eval_SG["LS_RW1"], "RW1", "LS")
plot_RMSE_ARW1_S <- myboxplot(Model_eval_SG["RMSE_ARW1"], "", "")
plot_LS_ARW1_S <- myboxplot(Model_eval_SG["LS_ARW1"], "ARW1", "")

plot_eval_NS <- ggarrange(plot_RMSE_RW1_S, plot_RMSE_ARW1_S,plot_LS_RW1_S, plot_LS_ARW1_S, ncol = 2, nr
annotate_figure(plot_eval_NS, top = text_grob("Model evaluation for shocked Gaussian data")) #can add m
```



SOME REFLECTION ON THE RESULTS AND WHCIH METHOD WAS BEST IN EACH CASE

Recreating the simulations study by Wakefield and Aleshin-Guendel

check: maybe weird axis on line 604, definetly weird legend labels, also at 609

inla.doc("rw1") - all info om RW1, for eksempel prior og initial values inla.scale.model - optional argument to scale or not scale a latent effect NB: INLA klikker hvis man bruker funksjoner inne i den, for eksmepel geometric_variance, ellers må de defineres det inne potensielt?

To do: Write about INLA - not started Use set.seed() to make results reproducible - continuous need for this dont need to set the seed for INLA, should be deterministic.

Briefly describe the methods used in the overleaf, both for data and models.

Sim-studie: Har undersøkt en RW1 med Gaussian offset for noen datapunkter. Reprodusere Wakefield: 3 ulike trends og tre ulike varianser, pluss forskjell i conf og non-conf tau eller ikke, så 18 scenarior og sjekke LS og RMSE. Andrea foreslo også harmonisk med noise pluss offset.

Code for ARW1 from github Alekshin Guendel

```
#The rgeneric inla function for user defined random effects, i e. a ARW1
inla.rgeneric.bym2.model = function(
```

```
    cmd = c("graph", "Q", "mu", "initial", "log.norm.const", "log.prior",
            "quit"),
    theta = NULL)
{
    # Assume we are passed the following inputs
    # n: the number of subdivisions (i.e. time points or regions)
    # Q_star: the scaled structure matrix for the structured component
    # gamma_tilde: the inverse eigenvalues of Q_star
    # U_prec, alpha_prec: U and alpha parameters for precision PC prior
    # U_phi, alpha_phi: U and alpha parameters for mixing parameter PC prior

    # for reference and potential storage for objects to
    # cache, this is the environment of this function
    # which holds arguments passed as `...` in
    # `inla.rgeneric.define()`.
    envir = parent.env(environment())

    interpret.theta = function() {
        return(list(prec = exp(theta[1L]),
                    phi = 1 / (1 + exp(-theta[2L]))))
    }

    graph = function(){ return (Q()) }

    Q = function() {
        p = interpret.theta()
        D = (1 / (1 - p$phi)) * diag(n)
        QQ = rbind(cbind(p$prec * D, -sqrt(p$phi * p$prec) * D),
                   cbind(-sqrt(p$phi * p$prec) * D, Q_star + p$phi * D))
        return (inla.as.sparse(QQ))
    }

    mu = function() { return(numeric(0)) }

    log.norm.const = function() { return (numeric(0)) }

    log.prior = function() {
        p = interpret.theta()

        # Construct prior for the precision parameter
        lambda_prec = -log(alpha_prec) / U_prec
        prec_prior = log(lambda_prec) - log(2) - theta[1L] / 2 -
            lambda_prec / sqrt(p$prec)

        # Construct prior for the mixing parameter phi
        dU = sqrt(U_phi * sum(gamma_tilde - 1) -
                      sum(log(1 + U_phi * (gamma_tilde - 1))))
        lambda_phi = -log(1 - alpha_phi) / dU
        d2 = p$phi * sum(gamma_tilde - 1) -
            sum(log(1 + p$phi * (gamma_tilde - 1)))
        phi_prior = log(lambda_phi) + 2 * theta[2L] -
            2 * log1p(exp(theta[2L])) - log(2) - log(d2) / 2 -
            lambda_phi * sqrt(d2) +
```

```
            log(abs(sum( (gamma_tilde - 1) ^ 2 /
                                (1 + exp(theta[2L]) * gamma_tilde))))

        return(prec_prior + phi_prior)

    }

    initial = function() { return(c(4, 0)) }

    quit = function() { return (invisible()) }

    if (!length(theta)) theta = initial()
    val = do.call(match.arg(cmd), args = list())
    return (val)
}

# ------ The following is my work trying to understand the code -------
?do.call
?match.arg
#I believe the match.arg(cmd) are all the values we are interested in
# which were defined as functions with no arguments further up
# or by arguments passed globally to the function as specified at the top,
# so, val = c("graph","Q","mu","initial","log.norm.const","log.prior","quit")
#I assume these quantities are needed later?
?inla.rgeneric.bym2.model
```

```
#might need these, dont know
library(SUMMER)
library(readstata13)
library(dplyr)

vignette()

#### Get smoothed direct estimates w/ time fixed effect, adaptive bym2 ####
# Create structure matrices
conflict_years <- 1993:1999 - 1984
conflict_years_long <- rep(0, num_years)
conflict_years_long[conflict_years] <- 1

R_conflict <- matrix(0, num_years, num_years)
R_nonconflict <- matrix(0, num_years, num_years)
for(i in 1:num_years){
  if(i == 1){
    if(conflict_years_long[i] || conflict_years_long[i + 1]){
      R_conflict[i, i] <- 1
    }
    else{
      R_nonconflict[i, i] <- 1
    }
  }
  else if(i == num_years){
    if(conflict_years_long[i] || conflict_years_long[i - 1]){
      R_conflict[i, i] <- 1
```

```
    }
    else{
      R_nonconflict[i, i] <- 1
    }
  }
  else{
    if(conflict_years_long[i]|| (conflict_years_long[i - 1] &&
                                 conflict_years_long[i + 1])){
      R_conflict[i, i] <- 2

    }
    else if(conflict_years_long[i - 1] || conflict_years_long[i + 1]){
      R_conflict[i, i] <- 1
      R_nonconflict[i, i] <- 1
    }
    else{
      R_nonconflict[i, i] <- 2
    }
  }

  for(j in 1:num_years){
    if(abs(i - j) == 1){
      if(conflict_years_long[i] || conflict_years_long[j]){
        R_conflict[i, j] <- -1
      }
      else{
        R_nonconflict[i, j] <- -1
      }
    }
  }
}

R_1 <- R_nonconflict
R_2 <- R_conflict
scaled_Q <- INLA:::inla.scale.model.bym.internal(R_1 + R_2,
                                                 adjust.for.con.comp = TRUE)$Q
gv <- scaled_Q[1, 1] / (R_1 + R_2)[1, 1]
R_1_star <- gv * R_1
R_2_star <- gv * R_2
vals <- (1:num_years)[-num_years]
R_1_star_hat <- R_1_star[vals, vals]
R_2_star_hat <- R_2_star[vals, vals]
eps <- eigen(solve(R_1_star_hat + R_2_star_hat) %*% R_2_star_hat)$values
gamma_tilde <- c(1 / eigen(R_1_star + R_2_star)$values[1:(num_years - 1)], 0)
save(R_1_star, R_2_star, eps, gamma_tilde,
     file = "../Data/generated_data/structure_matrices.RData")

# Specify PC prior hyperparameters
pc.u.theta <- 0.75
pc.alpha.theta <- 0.75

# Fit model
adaptive_bym2_model <-
```

```
  INLA::inla.rgeneric.define(model = inla.rgeneric.adaptive.bym2.model,
                             n = num_years, R_1_star = R_1_star,
                             R_2_star = R_2_star, gamma_tilde = gamma_tilde,
                             eps = eps, U_prec = pc.u, alpha_prec = pc.alpha,
                             U_phi = pc.u.phi, alpha_phi = pc.alpha.phi,
                             U_theta = pc.u.theta,
                             alpha_theta = pc.alpha.theta)
constr <- list(A = matrix(c(rep(0, num_years), rep(1, num_years)),
                          nrow = 1, ncol = 2 * num_years), e = 0)
mod <- logit.est ~ time  +
  f(region.struct, model = adaptive_bym2_model,
    diagonal = 1e-06, extraconstr = constr, n = 2 * num_years) +
  f(survey.id,  model = "iid", hyper = hyperpc1)
options <- list(dic = TRUE, mlik = TRUE, cpo = TRUE,
                openmp.strategy = "default", return.marginals.predictor = TRUE)
control.inla <- list(strategy = "adaptive", int.strategy = "auto")
fit_adaptive_linear <-
  INLA::inla(mod, family = "gaussian", control.compute = options,
             data = dat,
             control.predictor = list(compute = TRUE),
             control.family =
               list(hyper =  list(prec = list(initial = log(1),
                                              fixed = TRUE))),
             scale = dat$logit.prec,
             control.inla = control.inla, verbose = FALSE)

out_adaptive_linear <- dat %>% select(region, years) %>%
  mutate(median = NA, lower = NA, upper = NA, logit.median = NA,
         logit.lower = NA, logit.upper = NA)
for (i in 1:nrow(dat)) {
  tmp.logit <-
    INLA::inla.rmarginal(1e+05,
                         fit_adaptive_linear$marginals.fitted.values[[i]])
  tmp <- expit(tmp.logit)
  out_adaptive_linear$median[i] <- median(tmp)
  out_adaptive_linear$lower[i] <- quantile(tmp, probs = 0.05)
  out_adaptive_linear$upper[i] <- quantile(tmp, probs = 0.95)
  out_adaptive_linear$logit.median[i] <- median(tmp.logit)
  out_adaptive_linear$logit.lower[i] <- quantile(tmp.logit, probs = 0.05)
  out_adaptive_linear$logit.upper[i] <- quantile(tmp.logit, probs = 0.95)
}

out_adaptive_bym2 <- out_adaptive_linear %>%
  filter(is.na(years))

smoothed_direct_adaptive_bym2$fit <- fit_adaptive_linear
smoothed_direct_adaptive_bym2$model <- mod

out_combined <- rbind(cbind(out_bym2, prior = "bym2"),
                      cbind(out_adaptive_bym2, prior = "adaptive bym2"))


# ------ The following is my work trying to understand the code -------
```

```
#first part is computing the structure and precision matrices, and scaled
# A lot of parameters and priors etc, then we define the ARW1 and include
# it in the formula for the inla call. Also a sum to zero constraint maybe?
# After the model call I am not sure whats happening.
```

Simulation of non-shocked Poisson data

We will simulate data with a latent temporal structured random effect as a RW1. The total Bayesian hierarchical model can be described as

$$Y|\lambda \sim Poisson(E\lambda)\log\lambda_t = \mu + x_t$$

Where $\mathbf{x} \sim RW1(\tau)$ where we fix E, μ and τ for the simulations. When fitting models we will need to assign priors to them.

```
#Simulating non-shocked data

#Parameters
E = 100
mu = 1
sigma = 0.2
T = 100 #Number of time points

#A single simulation
x <- RW1(sigma, T)
rates <- E * exp(mu + x) #rates is E*lambda or E*exp(mu + x)
y <- sapply(rates, function(r) rpois(1, r)) #samples from the Poisson

plot(1:100, x)

sim_non_shocked_data <- function(E, mu, sigma, T){
  x <- RW1(sigma, T)
  rates <- E * exp(mu + x) #rates is E*lambda or E*exp(mu + x)
  y <- sapply(rates, function(r) rpois(1, r)) #samples from the Poisson
  return(y) #The observed data
}

sim_non_shocked_dataframe <- function(E, mu, sigma, T, n, seed = 44){
  set.seed(seed)
  df <- data.frame(matrix(NA, nrow = T, ncol = n))
  for(i in 1:n){df[, i] <- sim_non_shocked_data(E, mu, sigma, T) }
  return(df)
}

test <- sim_non_shocked_dataframe(100, 4, 0.001, 100, 5)

test$t <- 1:nrow(test)  # Create a time index from 1 to n

# Reshape the dataframe to long format
test_long <- test %>% pivot_longer(cols = starts_with("X"), names_to = "variable", values_to = "value")

# Plot all lines using ggplot
ggplot(test_long, aes(x = t, y = value, color = variable)) +
```

```
  geom_line() +
  labs(title = "Simulated non-shocked data with N=100 and 5 realisations",
       x = "Time", y = "y") + theme(legend.position = "none")
```

The code above seems to work fine, but should possibly tune some of the parameters, or maybe the plot of the ys should look insane.

Now the next step is to fit some models with INLA, we will start with the direct smoothed model. ## Fitting the model with INLA

```
#need to input the data as a dataframe with a y column and time column
formula <- y ~ f(time, model = "rw1") #intercept is included automatically

test_data <- data.frame(matrix(c(test[, 2], 1:100, rep(E, 100)), nrow = 100, ncol = 3)) # should make t
colnames(test_data) <- c("y", "time", "E")

res <- inla(formula, E = E, family = "poisson", data = test_data)
plot(res)
summary(res)
?inla
```

Seems like it works quite well, the posterior distributions align rather well with the true parameters for mean and precision.