PRACTICAL 3

Aim of this practical:

1. Fit temporal models for discrete time points using `inlabru`
2. Forecasting for future observations
3. Set penalized complexity (PC) priors
4. Fit temporal models to non-Gaussian data

we are going to learn:

- How to fit autoregressive and random walk models for time series data
- How to set PC priors in `inlabru`
- Forecast future observations using the posterior predictive density

0 AR(1) models in `inlabru`

In this exercise we will:

- Simulate a time series with autocorrelated errors.
- Fit an AR(1) process with `inlabru`
- Visualize model predictions.
- Forecasting for future observations

Start by loading useful libraries:

```
library(tidyverse)
library(INLA)
library(ggplot2)
library(patchwork)
library(inlabru)
```

Time series analysis is particularly valuable for modelling data with temporal dependence or autocorrelation, where observations taken at nearby time points tend to be more similar than those further apart.

A time series process is a stochastic process $\{X_t \,|\, t \in T\}$, which is a collection of random variables that are ordered in time where T is the *index set* that determines the set of discrete and equally spaced time points at which the process is defined and observations are made.

Autoregressive processes allow us to account for the time dependence by regressing X_t on past values $X_{t-1}, \dots, X_{t-p}$ with associated coefficients ϕ_k for each lag $k = 1, \dots, p$. Thus an **autoregressive process of order** p, denoted AR(p) , is given by:

$$X_t = \phi_1 X_{t-1} + \dots + \phi_p X_{t-p} + \varepsilon_t; \;\; \varepsilon_t \sim \mathcal{N}(0, \sigma_e^2)$$

Consider now an univariate time series y_t which evolves over time according to some autoregressive stochastic process. For example, a time series where the system follows an AR(1) process can be defined as:

$$
\begin{aligned}
y_t &\sim \mathcal{N}(\mu_t, \tau_y^{-1}) \\
\eta_t &= g^{-1}(\mu_t) = \alpha + u_t \\
u_t &= \phi u_{t-1} + \delta_t;\ \delta_t \sim \mathcal{N}(0, \tau_u^{-1});\ t > 1 \\
u_1 &= \mathcal{N}(0, \kappa^{-1}) \\
\kappa &= \tau_u(1-\phi^2)
\end{aligned}
$$

The response y_t is assumed to be normal distributed with mean $\alpha + u_t$ and precision error τ_y (here $g(\cdot)$ is just the identity link function that maps the linear predictor to the mean of the process). Then, the process u_t follows an AR(1) process where u_1 is drawn from a stationary normal distribution such that κ denotes the marginal precision for state u_t

The covariance matrix is then given by:

$$
\Sigma = \frac{\tau_u^{-1}}{1-\phi^2}
\begin{bmatrix}
1 & \phi & \phi^2 & \dots & \phi^{n-1} \\
\phi & 1 & \phi & \dots & \phi^{n-2} \\
\phi^2 & \phi & 1 & \dots & \phi^{n-3} \\
\phi^{n-1} & \phi^{n-2} & \phi^{n-3} & \dots & 1
\end{bmatrix}
$$

Notice that conditionally on u_t, the observed data y_y are independent from$y_{t-1}, y_{t-2}.y_{t-3}, \dots$, also the conditional distribution of u_t is a markov chain such that $\pi(u_t|u_{t-1}, u_{t-2}, u_{t-3}) = \pi(u_t|u_{t-1})$. Thus, each time point is only conditionally dependent on the two closest time points:

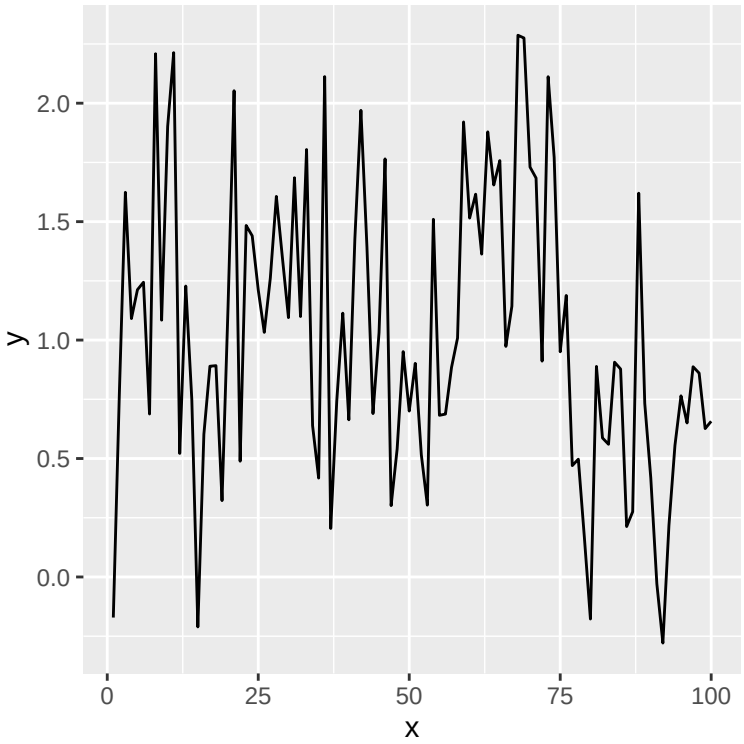
$$
u_t|\mathbf{u}_{-t} \sim \mathcal{N}\left(\frac{\phi}{1-\phi^2}(u_{t-1} + u_{t+1}), \frac{\tau_u^{-1}}{1-\phi^2}\right)
$$

0.1.1 Simulate example data

First, we simulate data from the model:

$$
\begin{aligned}
y_t &= \alpha + u_t + \varepsilon_t;\ \varepsilon_t \sim \mathcal{N}(0, \tau_y^{-1}) \\
u_t &= \phi y_{t-1} + \delta_t;\ \delta_t \sim \mathcal{N}(0, \tau_u^{-1})
\end{aligned}
$$

```
set.seed(123)

phi = 0.8
tau_u = 10
marg.prec = tau_u * (1-phi^2) # ar1 in INLA is parametrized as marginal variance
u_t =  as.vector(arima.sim(list(order = c(1,0,0), ar = phi),
                           n = 100,
                           sd=sqrt(1/tau_u)))
a = 1
tau_e = 5
epsilon_t = rnorm(100, sd = sqrt(1/tau_e))
y = a + u_t + epsilon_t


ts_dat <- data.frame(y =y , x= 1:100)
```



0.1.2 Fitting an AR(1) model with `inlabru`

Model components

First, we define the model components, notice that the latent field is defined by two components: the intercept α and the autoregressive random effects u_t:

```
# Model components
cmp =  ~ -1 + alpha(1) + ut(x,model = "ar1")
```

The we can define the formula for the linear predictor and specify the observational model

```
# Model formula
formula = y ~ alpha + ut
# Observational model
lik =  bru_obs(formula = y ~.,
            family = "gaussian",
            data = ts_dat)
```

Lastly, we fit the model using the `bru` function and compare the model estimates with the true mdeol parameters we simulate our data from.

```
# fit the model
fit.ar1 = bru(cmp, lik)

# compare against the true values

data.frame(
  true = c(a,tau_e,marg.prec,phi),
```

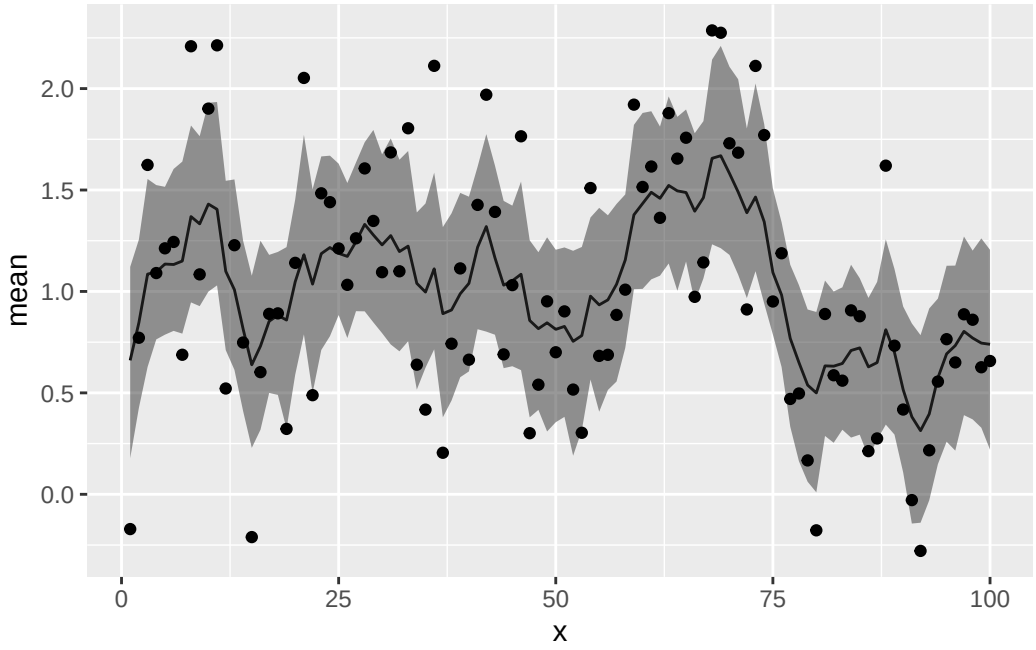
```
  rbind(fit.ar1$summary.fixed[,c(1,3,5)],
        fit.ar1$summary.hyperpar[,c(1,3,5)])
        ) %>% round(2)
```

	true	mean	X0.025quant	X0.975quant
alpha	1.0	1.00	0.69	1.28
Precision for the Gaussian observations	5.0	4.91	3.11	7.29
Precision for ut	3.6	7.96	2.91	18.09
Rho for ut	0.8	0.80	0.54	0.94

Model predictions

Here, we will predict the mean of our time series along with 95% credible intervals. Note that this interval are for the mean and not for new observations, we will cover forecasting new observations next.

```
pred_ar1 = predict(fit.ar1, ts_dat, ~ alpha + ut)

ggplot(pred_ar1,aes(y=mean,x=x))+
  geom_line()+
    geom_ribbon(aes(x = x, y = mean, ymin = q0.025, ymax = q0.975),
                alpha = 0.5) +
  geom_point(aes(y=y,x=x))
```



0.1.3 Forecasting

A common goal in time series modelling is forecasting into the future. Forecasting can be treated as a missing data problem where future values of the response variable are missing. Let y_m be the missing response, then, by fitting a statistical model to the observed data $\mathbf{y}_{obs}$, we condition on its parameters to obtain the posterior predictive distribution:

$$\pi(y_m \mid \mathbf{y}_{obs}) = \int \pi(y_m, \theta \mid \mathbf{y}_{obs}) d\theta = \int \pi(y_m \mid \mathbf{y}_{obs}, \theta) \pi(\theta \mid \mathbf{y}_{obs}) d\theta$$

This distribution, which integrates over all parameter uncertainty, provides the complete probabilistic forecast for the missing values. `INLA` will automatically compute the predictive distributions for all missing values in the response. To do so, we can augment our data set by including the new time points at which the prediction will be made and setting the response value to `NA` for these new time points:

```
ts.forecast <- rbind(ts_dat,
  data.frame(y = rep(NA, 50), x = 101:150))
```

Next, we fit the `ar1` model to the new dataset so that the predictive distributions are computed:

```
cmp =  ~ -1 + alpha(1) + ut(x,model = "ar1")

pred_lik =  bru_obs(formula = y ~.,
            family = "gaussian",
            data = ts.forecast)

fit.forecast = bru(cmp, pred_lik)
```

Lastly, we can draw samples from the posterior predictive distribution using the `predict` function and visualize our forecast as follows:

```
pred_forecast = predict(fit.forecast, ts.forecast, ~ alpha + ut)

p1= ggplot(pred_forecast,aes(y=mean,x=x))+
  geom_line()+
    geom_ribbon(aes(x = x, y = mean, ymin = q0.025, ymax = q0.975),
                alpha = 0.5) +
  geom_point(data=ts_dat, aes(y=y,x=x))
```

0 Modelling Great Lakes water level

In this exercise we will:

- Fit an AR(1) process with `inlabru` to model lakes water levels
- Change the default priors for the observational error
- Set penalized complexity priors for the correlation and precision parameters of the latent effects.
- Fit a RW(1) model
- Fit a an AR(1) model with group-level correlation

Start by loading useful libraries:

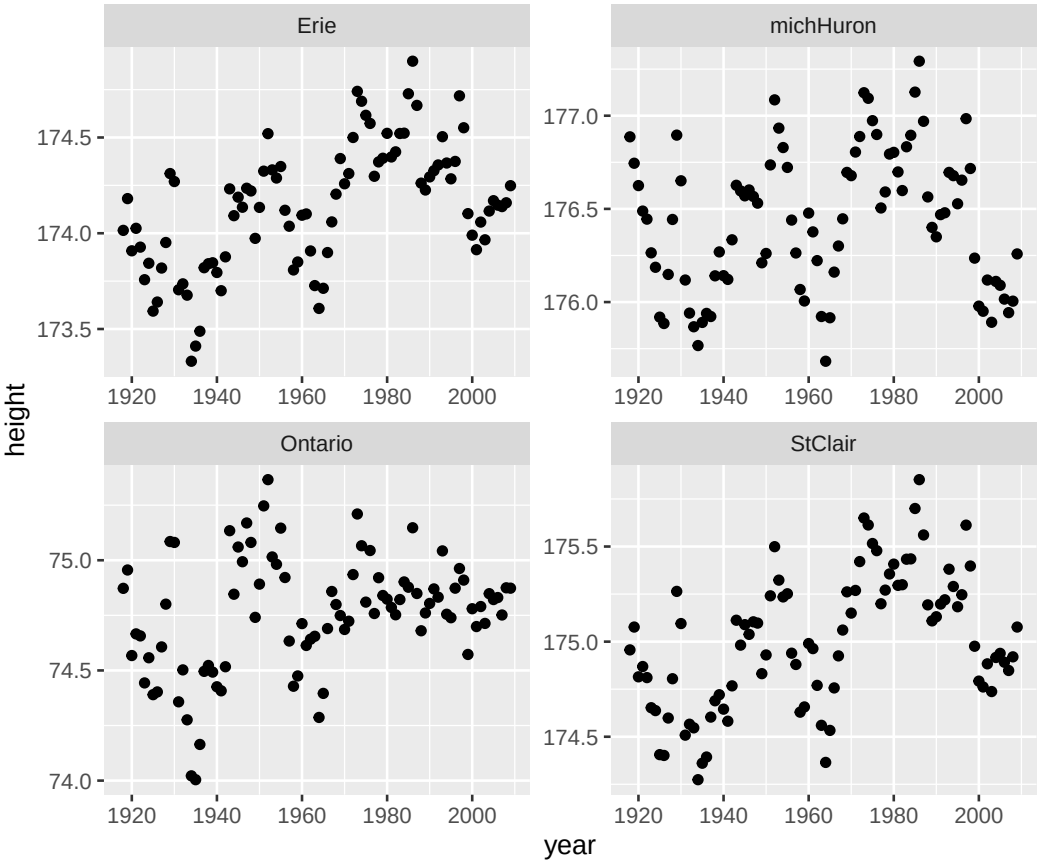
```
library(tidyverse)
library(INLA)
library(ggplot2)
```

```
library(patchwork)
library(inlabru)
library(DAAG)
```

In this exercise we will look at greatLakes dataset from the DAAG package. The data set contains the water level heights for the lakes Erie, Michigan/Huron, Ontario and St Clair from 1918 to 2009.

Lets begin by loading and formatting the data into a tidy format.

```
data("greatLakes")

greatLakes.df = data.frame(as.matrix(greatLakes),
                           year = time(greatLakes)) %>%
  pivot_longer(cols = c("Erie","michHuron","Ontario","StClair"),
               names_to = "Lakes",
               values_to = "height" )
```



0.2.1 Fitting an AR(1) model in `inlabru`

We will focus on the Erin lake for now. Lets begin by fitting an AR(1) model of the form:

$$
\begin{aligned}
\text{height}_t &= \alpha + u_t + \varepsilon_t \,;\; \varepsilon_t \sim \mathcal{N}(0, \tau_e^{-1}) \\
u_t &= \phi u_{t-1} + \delta_t \;\; ; \;\; \delta_t \sim \mathcal{N}(0, \tau_u^{-1}); \;\; t > 1 \\
x_1 &= \mathcal{N}(0, \kappa^{-1}) \\
\kappa &= \tau_u (1 - \phi^2)
\end{aligned}
$$

Where α is the intercept, ϕ is the correlation term, ε is the observational Gaussian error with mean zero and precision τ_e and κ is the marginal precision for the state u_t for $t = 1, \dots, 92$.

First we make a subset of the dataset and create a time index T:

```
greatLakes.df$t.idx <- greatLakes.df$year-1917

Erie.df = greatLakes.df %>% filter(Lakes == "Erie")
```

Task

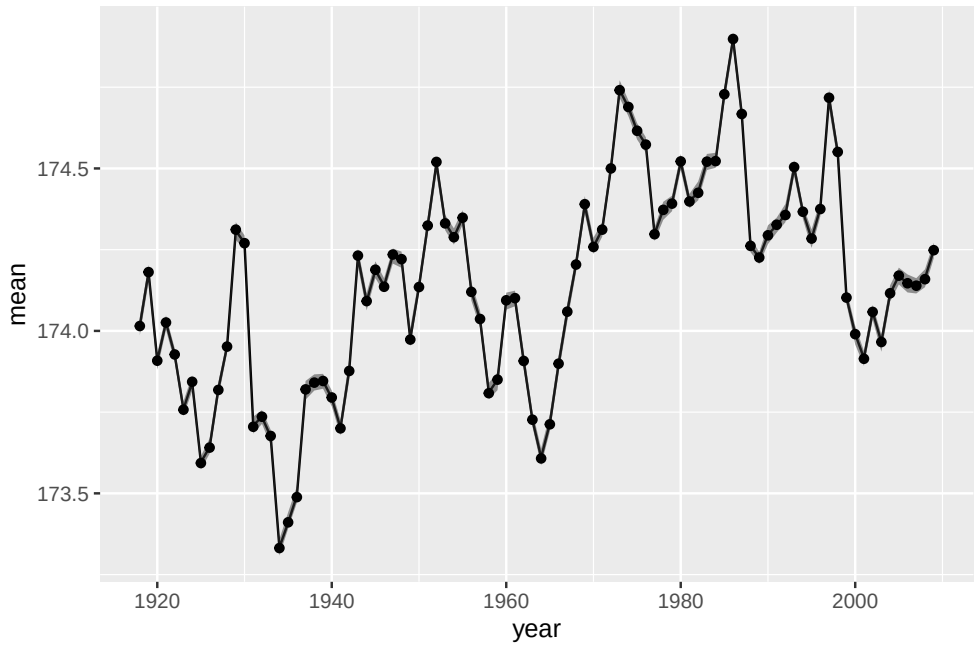
Fit an AR(1) model to the Erie lake data using `inlabru`, then plot the model fitted values showing 95% credible intervals.

Take hint

Remember this is done by (1) defining the model components, (2) the formula and (3) the observational model. Then you can use the `predict` function to compute the predicted values for the mean along with 95% credible intervals.

Click here to see the solution

```
# Model components
cmp =  ~ -1 + alpha(1) + ut(t.idx,model = "ar1")
# Model formula
formula = height ~ alpha + ut


# Observational model
lik =  bru_obs(formula = height   ~.,
           family = "gaussian",
           data = Erie.df )

# fit the model
fit.Erie_ar1 = bru(cmp, lik)

# Model predictions

pred_ar1.Erie = predict(fit.Erie_ar1, Erie.df, ~ alpha + ut)

# plot model fitted values
ggplot(pred_ar1.Erie,aes(y=mean,x=year))+
  geom_line()+
    geom_ribbon(aes(x = year, y = mean, ymin = q0.025, ymax = q0.975),
                alpha = 0.5) +
  geom_point(aes(y=height,x=year))
```



Question

Are there any issues with the fitted model, and if so, how do you think we should address them?
Answer
It is clear that the model overfits the data, leading to poor predictive performance. Thus, we need to introduce some prior information on the what we expect the variation of the process to be.

Priors

Let review INLA' prior parametrization for autoregressive models:

Let $\boldsymbol{\theta} = \{\theta_y, \theta_u, \theta_\phi\}$ be INLA's internal representation of the hyperparameters such that:

$\theta_y = \log(\tau_y^2)$

$\theta_u = \log(\kappa) = \log\left(\tau_u[1-\phi^2]\right)$

$\theta_\phi = \log\left(\frac{1+\phi}{1-\phi}\right)$

The default priors for $\{\theta_y, \theta_u\}$ are log-gamma$(1, 5 \times 10^{-5})$ priors with default initial values set to 4 in each case. Then, Gaussian priors $\alpha \sim \mathcal{N}(0, \tau_y = 0.001)$ and $\theta_\phi \sim \mathcal{N}(0, \tau_y = 0.15)$ are used for the intercept and correlation parameter respectively.

Note

Specifically for AR(1) correlation parameter ϕ, INLA uses the following logit transformation on θ_ϕ:

$$\phi = \frac{2\exp(\theta_\phi)}{1+\exp(\theta_\phi)} - 1.$$

Setting priors and PC-priors

Lets now set a Gamma prior with parameters 1 and 1, so that the precision of the Gaussian osbervational error is centered at 1 with a variance of 1. Additionally we will set Penalized Complexity (PC) priors according to the following probability statements:

- $P(\sigma > 1) = 0.01$
- $P(\phi > 0.5) = 0.3$

Notice that the PC prior for the precision τ_u is defined on the standard deviation $\sigma_u = \tau_u^{-1/2}$

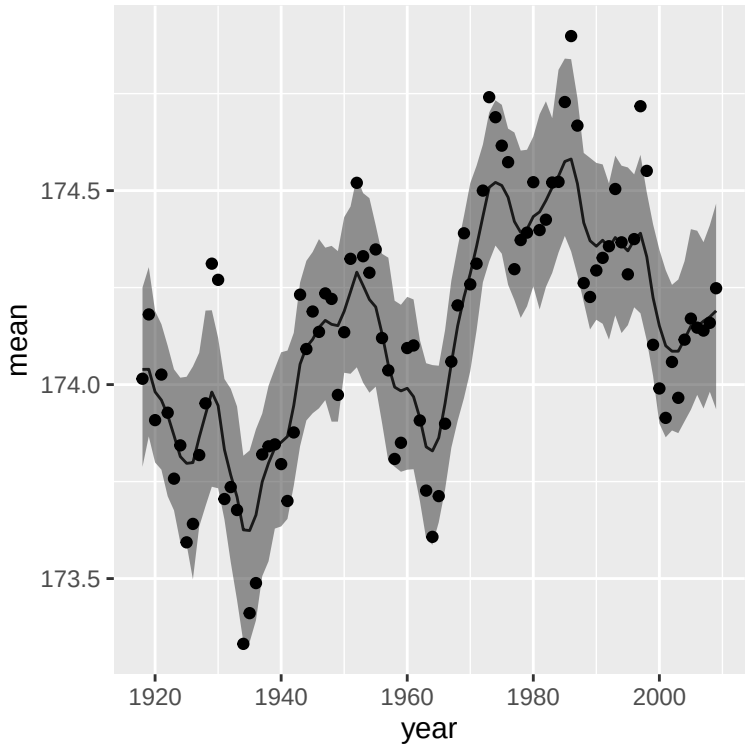
```
pc_prior <- list(theta = list(prior = "pc.prec", param = c(1, 0.01)),
                 rho = list(prior = "pc.cor0", param = c(0.5, 0.3)))

prec.tau_e <- list(prec = list(prior = "loggamma",   # prior name
                               param = c(1, 1))) # prior values

# Model components
cmp =  ~ -1 + alpha(1) + ut(t.idx, model = "ar1",  hyper = pc_prior)
# Model formula
formula = height ~ alpha + ut


# Observational model
lik =  bru_obs(formula = height  ~.,
            family = "gaussian",
            data = Erie.df,
            control.family = list(hyper = prec.tau_e))

# fit the model
fit.Erie_ar1 = bru(cmp, lik)
```

Question

What is the posterior mean for the correlation parameter ρ? _____

Task

Plot the fitted values of the model, has the overfitting problem being alleviated?
Click here to see the solution



0.2.2 Fitting a RW(1) model

Now we fit a random walk of order 1 to the Erie lake data:

$$
\begin{aligned}
y_t &= \alpha + u_t + \varepsilon_t,\ t = 1, \dots, 92 \\
\varepsilon_t &\sim \mathcal{N}(0, \tau_e) \\
u_t - u_{t-1} &\sim \mathcal{N}(0, \tau_u),\ t = 2, \dots, 92
\end{aligned}
$$

Firs we define model priors:

```
pc_prior <- list(theta = list(prior = "pc.prec", param = c(1, 0.01)))

prec.tau_e <- list(prec = list(prior = "loggamma",   # prior name
                               param = c(1, 1))) # prior values
```

Now we define model components:

```
cmp_rw =  ~ -1 + alpha(1) +
  ut(t.idx ,
     constr=FALSE,
     model = "rw1",
     hyper=pc_prior,
     scale.model = TRUE)
```

Notice that we have set `scale.model = TRUE` to scale the latent effects. This is particularly important when Intrinsic Gaussian Markov random fields (IGMRFs) are used as priors (e.g., random walk models or some spatial models) for the latent effects. By defining `scale.model = TRUE`, the `rw1`-model is scaled to have a generalized variance equal to one. By scaling scaling the models we ensure that a fixed hyperprior for the precision

parameter has a similar interpretation for different types of IGMRFs, making precision estimates comparable between different models. Scaling also allows estimates to be less sensitive to re-scaling covariates in the linear predictor and makes the precision invariant to changes in the shape and size of the latent effect (see Sørbye (2014) for further details)
.

We can now fit the model with the updated components and plot the predicted values

```
# Model formula
formula = height ~ alpha + ut
# Observational model
lik =  bru_obs(formula = height  ~.,
            family = "gaussian",
            data = Erie.df,
            control.family = list(hyper = prec.tau_e))
# fit the model
fit.Erie_rw1 = bru(cmp_rw, lik)
# Model predictions
pred_rw1.Erie = predict(fit.Erie_rw1, Erie.df, ~ alpha + ut)
```

Question

Take a look at the model summaries using the `summary` function, do you see anything odd?

Answer

The intercept has zero mean and a very large variance. This is because we have not imposed a sum-to-zero constraint on the model random effects (`constr=FALSE`). Without this constraint, intrinsic models are non-identifiable. The intercept and the random effects are confounded, For example, you could add a constant value to every random effect and subtract it from the intercept without changing the model's pre-

dictions. Take for instance for any constant c, the following models are identical:
$y_t = \alpha + u_t + \varepsilon_t \; y_t = (\alpha - c) + (u_t + c) + \varepsilon_t$
Thus, you need to set `constr=FALSE` so that $\sum_t u_t = 0$ to ensure identifiability of α

Task

Fit an RW(1) model to the Erie data but now set `constr=TRUE` to impose a sum-to-zero constraint on the random effect. Then compare your results with the unconstrained model.
Click here to see the solution

```
# Model components
cmp_rw =  ~ -1 + alpha(1) +
  ut(t.idx ,
     constr=TRUE,
     model = "rw1",
     hyper=pc_prior,
     scale.model = TRUE)

fit.Erie_rw1_constr = bru(cmp_rw, lik)

fit.Erie_rw1_constr$summary.fixed
```

```
          mean        sd 0.025quant 0.5quant 0.975quant     mode          kld
alpha 174.1381 0.0237476   174.0914 174.1381   174.1848 174.1381 1.935519e-08
```

0.2.3 Group-level effects

Now we will model the height water levels for all four lakes by grouping the random effects. This will allow a within-lakes correlation to be included. In the next example, we allow for correlated effects using an `ar1` model for the years and `iid` random effects on the lakes. First we create a lakes id and set the priors for our model:

```
greatLakes.df$lake_id <- as.numeric(as.factor(greatLakes.df$Lakes))

pc_prior <- list(theta = list(prior = "pc.prec", param = c(1, 0.01)),
                 rho = list(prior = "pc.cor0", param = c(0.5, 0.3)))

prec.tau_e <- list(prec = list(prior = "loggamma",   # prior name
                               param = c(1, 10))) # prior values
```

Now we define the model components. The lakes IDs that define the group are passed with parameter `group` argument and the `iid` model and other parameters are passed through the `control.group` parameter.

```
# Model components
cmp =  ~ -1 + alpha(1) + ut(year,model = "ar1",
                            hyper = pc_prior,
                            group =lake_id,
                            control.group =
```

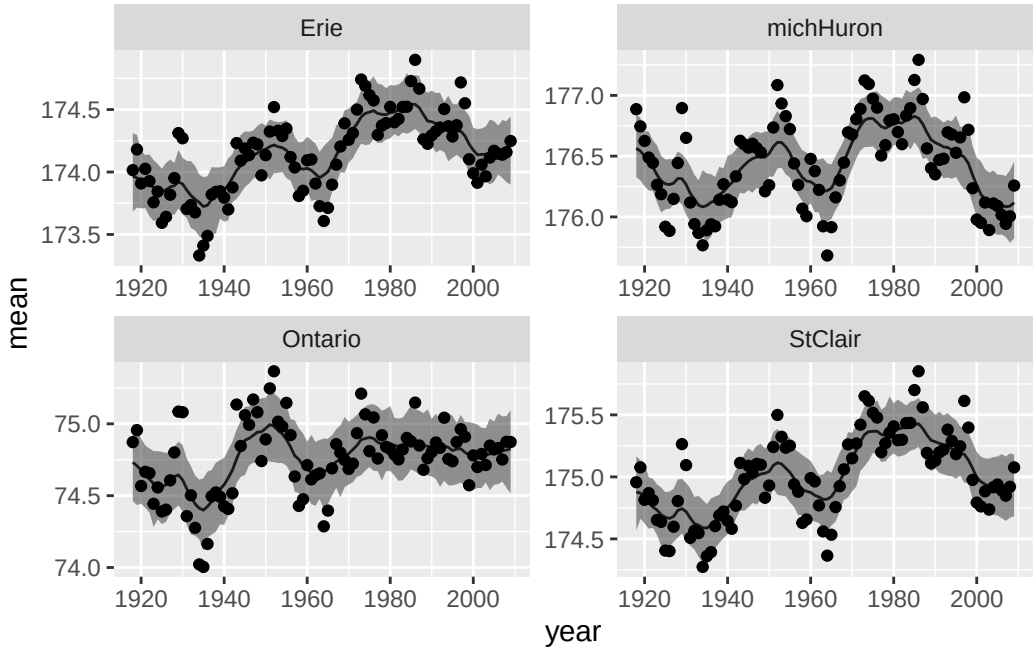
```
                                list(model = "iid",
                                     scale.model = TRUE))
```

We fit the model in a similar fashion as we did before:

```
# Model formula
formula = height ~ alpha + ut


# Observational model
lik =  bru_obs(formula = height  ~.,
               family = "gaussian",
               data = greatLakes.df,
               control.family = list(hyper = prec.tau_e))

# fit the model
fit.all_lakes_ar1 = bru(cmp, lik)

# Model predictions
pred_ar1.all = predict(fit.all_lakes_ar1, greatLakes.df, ~ alpha + ut)
```

Lastly we can visualize group-level model predictions as follows:

```
ggplot(pred_ar1.all,aes(y=mean,x=year))+
  geom_line()+
    geom_ribbon(aes(x = year, y = mean, ymin = q0.025, ymax = q0.975),
                alpha = 0.5) +
  geom_point(aes(y=height,x=year)) + facet_wrap(~Lakes,scales = "free")
```



0 Non-Gaussian data

In the next example we will use the `Toyo` data set to illustrate how temporal models can be fit to non-Gaussian data.

The `Tokyo` data set available in `INLA` contains the recorded days of rain above 1 mm in Tokyo for 2 years, 1983:84. The data set contains the following variables:

- `y` : number of days with rain
- `n` : total number of days
- `time` : day of the year

```
library(INLA)
library(inlabru)
library(ggplot2)
library(tidyr)


data("Tokyo")
```

A possible observational model for these data is

$$
\begin{aligned}
y_t|\eta_t &\sim \text{Bin}(n_t, p_t) \\
\eta_t &= \text{logit}(p_t), \qquad i = 1, \dots, 366
\end{aligned}
$$

$$
n_t = \begin{cases} 1, & \text{for 29 February} \\ 2, & \text{other days} \end{cases}
$$

$$
y_t = \begin{cases} \{0, 1\}, & \text{for 29 February} \\ \{0, 1, 2\}, & \text{other days} \end{cases}
$$

Then, the latent field is given by

$$
\eta_t = \beta_0 + f(\text{time}_t)
$$

- Where the probability of rain depends on on the day of the year t
- β_0 is an intercept
- $f(\text{time}_t)$ is a temporal model, e.g., a RW2 model (this is just a smoother).

The smoothness is controlled by a hyperparameter τ_f . Thus, we assign a prior to τ_f to finalize the model.

We can fit the model as follows:

```
# define model component
cmp =  ~ -1 + beta0(1) + time_effect(time, model = "rw2", cyclic = TRUE)

# define model predictor
eta = y ~ beta0 + time_effect

# build the observation model
```

```
lik = bru_obs(formula = eta,
              family = "binomial",
              Ntrials = n,
              data = Tokyo)

# fit the model
fit = bru(cmp, lik)
```

Notice that we have set `cyclic = TRUE` as this is a cyclic effect. Finally, we can produce model predictions in a similar fashion as we did before:

```
pTokyo = predict(fit, Tokyo, ~ plogis(beta0 + time_effect))

ggplot(data=pTokyo , aes(x= time, y= y) ) +
  geom_point() +
  ylab("") + xlab("") +
  # Custom the Y scales:
  scale_y_continuous(
    # Features of the first axis
    name = "",
    # Add a second axis and specify its features
    sec.axis = sec_axis( transform=~./2, name="Probability")
  )  + geom_line(aes(y=mean*2,x=time)) +
  geom_ribbon(aes( ymin = q0.025*2,
                               ymax = q0.975*2), alpha = 0.5)
```