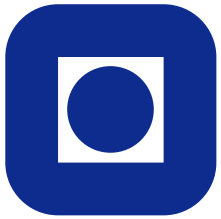


NTNU
Norwegian University of
Science and Technology

R-INLA: An R-package for INLA

Andrea Riebler <andrea.riebler@math.ntnu.no>
Department of Mathematical Sciences, NTNU

November 5-6, 2014

Outline

Structure of an R-INLA program

Simple example

Random effects

Model choice/checking

Useful features

Implementing INLA

All procedures required to perform INLA need to be carefully implemented to achieve a good speed; easier to implement a slow version of INLA.

Implementing INLA

All procedures required to perform INLA need to be carefully implemented to achieve a good speed; easier to implement a slow version of INLA.

— The `GMRFLib`-library

- Basic library written in `C`, user friendly for programmers

Implementing INLA

All procedures required to perform INLA need to be carefully implemented to achieve a good speed; easier to implement a slow version of INLA.

— The `GMRFLib`-library

— The `inla`-program

- Define *latent Gaussian models* and interface with the `GMRFLib`-library
- Avoids the need for `C`-programming
- Models are defined using `.ini`-files
- Requires to write input files in a special format
- `inla`-program write all the results (E/Var/marginals) to files

Implementing INLA

All procedures required to perform INLA need to be carefully implemented to achieve a good speed; easier to implement a slow version of INLA.

— The `GMRFLib`-library

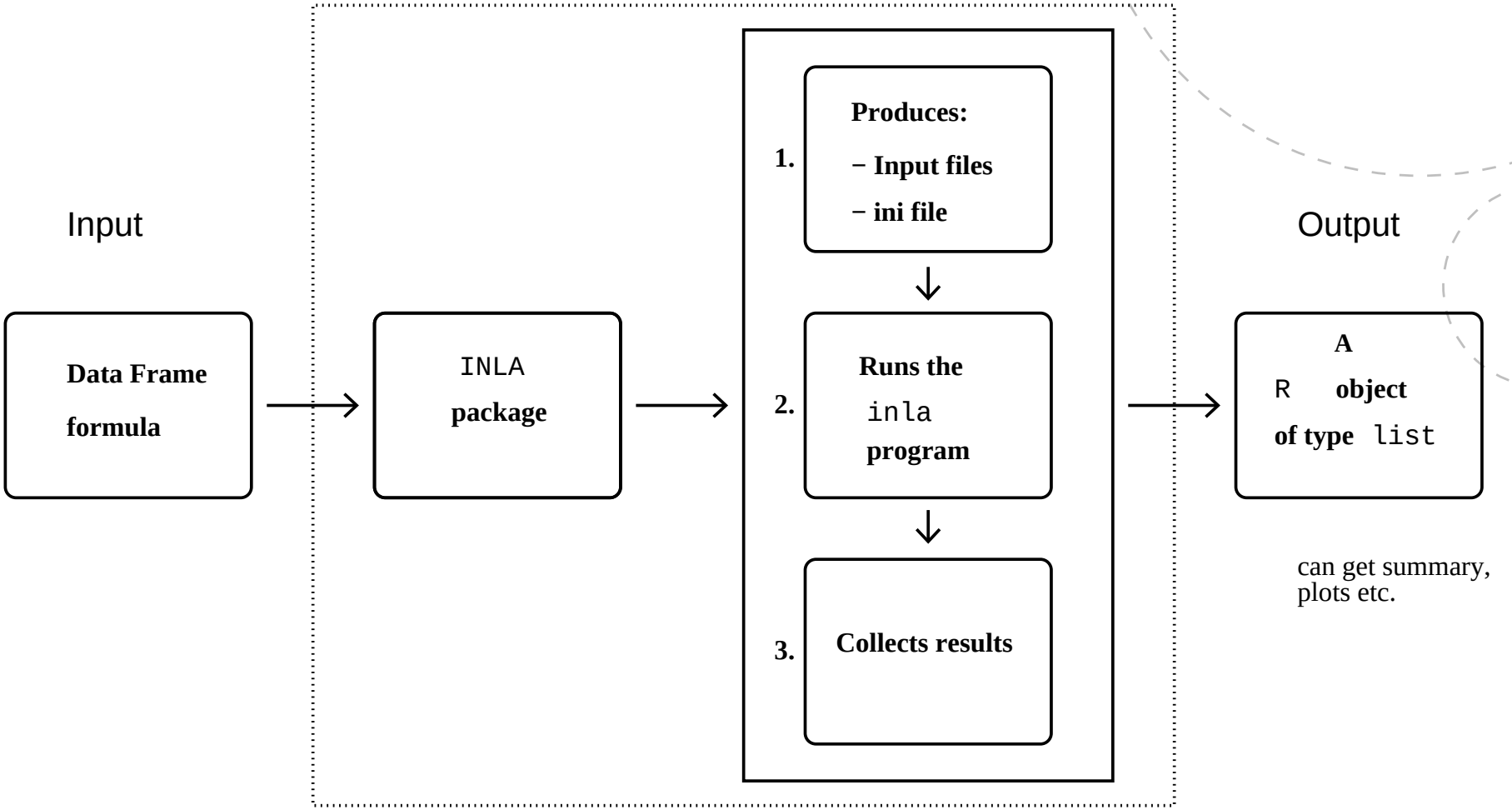
— The `inla`-program

— The `INLA` package for `R`

- `R`-interface to the `inla`-program.
- Convert “formula”-statements into “.ini”-file definitions

The first two are *not* particularly user-friendly. They are used in the background by the `INLA` package.

The INLA package for R



Getting R-INLA

— The web page www.r-inla.org contains source-code, examples, reports and instructions for installing the package

— The package can be installed by

```
> source("http://www.math.ntnu.no/inla/givemeINLA.R")
```

— Later, it can be upgraded with

```
> inla.upgrade(testing=TRUE)
```

— Works on Linux, Windows and Mac

Documentation

See `r-inla.org` for documentation and worked through examples.

[1] Spatial models based on stochastic partial differential equations

Documentation

See `r-inla.org` for documentation and worked through examples.

Tutorials:

- Basic INLA (in preparation and expected by the end of the year)
- SPDE-based models (continuously indexed random effects) [1] in INLA (see `r-inla.org`)

[1] Spatial models based on stochastic partial differential equations

Structure of an R-program using R-INLA

There are essentially four parts to an R-INLA-program:

1. The data organization
2. The `formula`-notation (similar to `lm` and `glm` functions)
3. The call to the `inla`-program
4. The extraction of posterior information

Data organization

The responses and covariates are collected in a list or data frame. Assume response `y`, covariates `x1` and `x2`, and time index `t`. Then they can be organized with

```
# Option 1
data = list(y = y, x1 = x1, x2 = x2, t = t)

# Option 2
data = data.frame(y = y, x1 = x1, x2 = x2, t = t)
```

`formula`: specifying the linear predictor

The model is specified through `formula` similar to `glm`:

```
formula = y ~ x1 + x2 + f(t, ...)
```

- *y* is the name of the response in the `data`
- The fixed effects are given i.i.d. Gaussian priors
- The `f` function specifies random effects (e.g. temporal, spatial, smooth effect of covariates and Besag model)
- Use `-1` if you don't want an automatic intercept

The `inla` function

```
result = inla(
    # Description of linear predictor
    formula,
    # Likelihood
    family = "gaussian",
    # List or data frame with response, covariates, etc.
    data = data,

    ## This is all that is needed for a basic call
    # check what happens
    verbose = TRUE,
    # keep working files
    keep = TRUE,

    # there are also some "control statements"
    # to customize things)
```

Likelihood functions

- "gaussian"
- "T"
- "poisson"
- "nbinomial"
- "binomial"
- "exponential"
- "weibull"
- "gev"
- "coxph"

- See list at http://www.r-inla.org/models/likelihoods or

```
1 names(inla.models()$likelihood)
```

Posterior inference

Main functions:

- `summary(result)`
- `plot(result)`
- `result2 = inla.hyperpar(result)`
- `result2 = inla.cpo(result)`

Example: Simple linear regression

Stage 1: Gaussian likelihood

$$y_i|\eta_i \sim \mathcal{N}(\eta_i, \sigma_o^2)$$

Stage 2: Covariates are connected to likelihood by

$$\eta_i = \beta_0 + \beta_1 x_i$$

Stage 3: σ_o^2: variance of observation noise

Example: Simple linear regression

```
1  # Generate data
2  x = sort(runif(100))
3  y = 1 + 2*x + rnorm(n = 100, sd = 0.1)
4
5  # Run inla
6  formula = y ~ 1 + x
7  result = inla(formula,
8                data = list(x = x, y = y),
9                family = "gaussian")
10
11 # Get summary
12 summary(result)
```

summary(result)

```
Call:
"inla(formula = y ~ x, data = data)"

Time used:
 Pre-processing    Running inla Post-processing           Total
         0.3198          0.0396          0.0352          0.3946

Fixed effects:
              mean     sd 0.025quant 0.5quant 0.975quant   mode kld
(Intercept) 1.0140 0.0195     0.9757   1.0140     1.0523 1.0140   0
x           2.0025 0.0316     1.9405   2.0025     2.0645 2.0025   0

The model has no random effects

Model hyperparameters:
                                          mean   sd     0.025quant 0.5quant
Precision for the Gaussian observations 114.87  16.28  85.66      113.98
                                        0.975quant mode
Precision for the Gaussian observations 149.37     112.42

Expected number of effective parameters(std dev): 2.189(0.0213)
Number of equivalent replicates : 45.69

Marginal Likelihood:  78.36
```

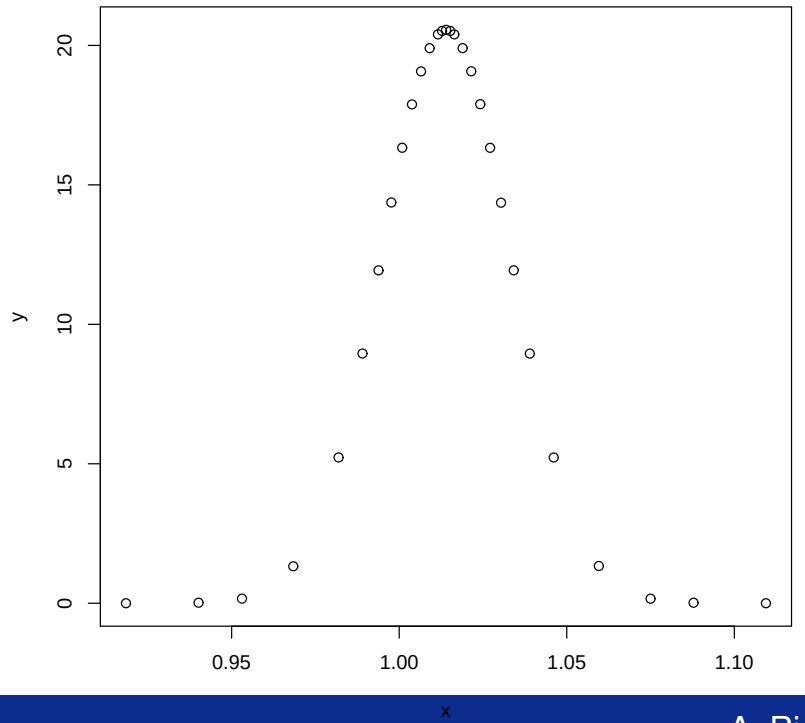
result$summary.fixed

```
                  mean          sd 0.025quant 0.5quant 0.975quant     mode
(Intercept) 1.014012 0.01949959  0.9756744 1.014011   1.052312 1.014012
x           2.002502 0.03155688  1.9404595 2.002501   2.064484 2.002502
                       kld
(Intercept) 1.167936e-12
x           4.459456e-13
```

Marginal posterior densities

The marginal posterior densities are stored as a matrices with *x*- and *y*-values

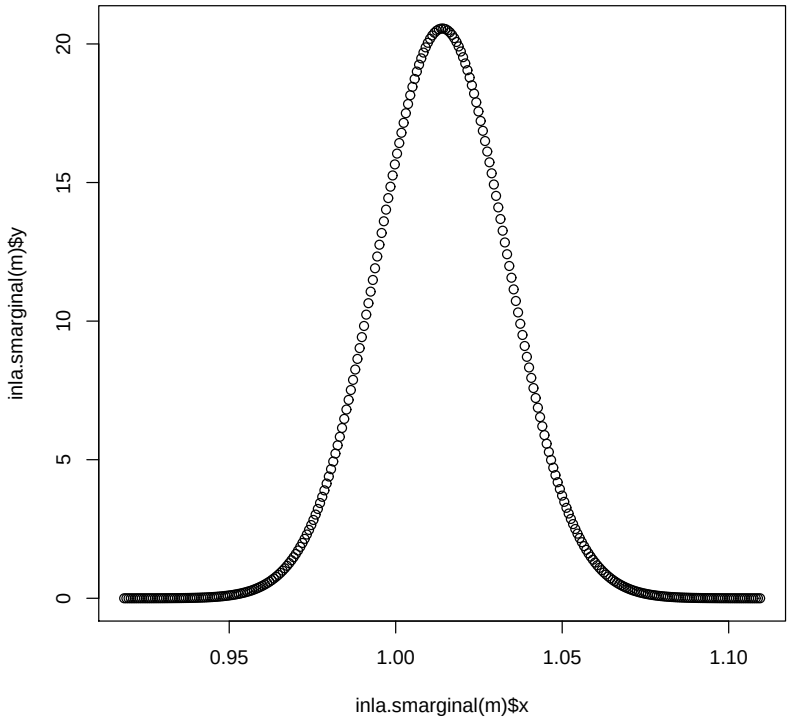
```
1 m = result$marginals.fixed[[1]]
2 plot(m)
```



Marginal posterior densities

The rough shape can be interpolated to higher resolution

```
1 plot(inla.smarginal(m))
```



Marginal posterior densities

```
# Extract quantiles
> inla.qmarginal(0.05, m)
[1] 0.9818604

# Distribution function
> inla.pmarginal(0.975, m)
[1] 0.02314047

# Density function
> inla.dmarginal(1, m)
[1] 15.80794

# Generate realizations
> inla.rmarginal(4, m)
[1] 1.009122 1.013116 1.032004 1.007458
```

Marginal posterior densities

```
# Calculate expected value of x and x^2
> E = inla.emarginal(function(x) c(x,x^2), m)
> E
[1] 1.014012 1.028600

# Calculate sd
> sqrt(E[2]-E[1]^2)
[1] 0.0194985

# Compare to estimate
> round(result$summary.fixed[,1:2], 3)
               mean    sd
(Intercept) 1.014 0.019
x           2.003 0.032
```

Get estimates for variance not precision

Assume that we are interested in posterior mean and standard deviation of $\sigma_0^2 = \frac{1}{\tau_0}$. This can be easily done by selecting the appropriate posterior marginal from the output of the `inla()` function:

```
# get the marginal for the precision
> tau0 = result$marginals.hyperpar$"Precision for the Gaussian
    observations"

# Calculate expected value of 1/x and 1/x^2
> E = inla.emarginal(function(x) c(1/x,(1/x)^2), tau0)

# Calculate sd
> mysd = sqrt(E[2] - E[1]^2)

> print(c(mean=E[1], sd=mysd))
      mean         sd
0.01055980 0.00151231
```

Add random effects

```
f(name, model="...", hyper=...,
                  constr=FALSE, cyclic=FALSE, ...)
```

- `name` – the index of the effect
- `model` – the type of latent model. E.g. `"iid"`, `"rw2"`, `"ar1"`, `"besag"`, and so on
- `hyper` – specify the prior on the hyperparameters
- `constr` – sum-to-zero constraint?
- `cyclic` – are you cyclic?
- ...

Example: Add random effect

Add an AR(1) random effect to the linear predictor.

Stage 1:

$$y_i|\eta_i \sim \mathcal{N}(\eta_i, \sigma_o^2)$$

Stage 2: Covariates and AR(1) component connected to likelihood by

$$\eta_i = \beta_0 + \beta_1 x_i + a_i$$

Stage 3:
- σ_o^2 : variance of observation noise
- ρ : dependence in AR(1) process
- σ^2 : variance of the innovations in AR(1) process

Example: Add random effect

```
# Generate AR(1) sequence
t = 1:100
ar = rep(0,100)
for(i in 2:100)
    ar[i] = 0.8*ar[i-1]+rnorm(n = 1, sd = 0.1)

# Generate data with AR(1) component
x = runif(100)
y = 1 + 2*x + ar + rnorm(n = 100, sd = 0.1)

# Run inla
formula = y ~ 1 + x + f(t, model="ar1")
result = inla(formula,
              data = list(x = x, y = y, t = t),
              family = "gaussian")

# Get summary
summary(result)
```

summary(result)

```
Fixed effects:
              mean     sd 0.025quant 0.5quant 0.975quant   mode kld
(Intercept) 1.0354 0.0624      0.913   1.0344     1.1635 1.0328   0
x           2.0173 0.0459      1.927   2.0173     2.1077 2.0173   0

Random effects:
Name      Model
 t   AR1 model

Model hyperparameters:
                                                mean     sd       0.025quant 0.5quant
Precision for the Gaussian observations 129.8753  49.6529  60.8214    120.5645
Precision for t                          38.3033  13.9965  16.8866     36.4192
Rho for t                                 0.8031   0.0817   0.6028      0.8181
                                                0.975quant mode
Precision for the Gaussian observations 251.9389   104.1904
Precision for t                          70.9695    32.7097
Rho for t                                 0.9185     0.8463
```

The interpretation of NA

R-INLA uses NA differently than other packages

- NA in the response means no likelihood contribution, i.e. response is unobserved
- NA in a fixed effect means no contribution to the linear predictor, i.e. the covariate is set equal to zero
- NA in a random effect f(...) means no contribution to the linear predictor

Prediction

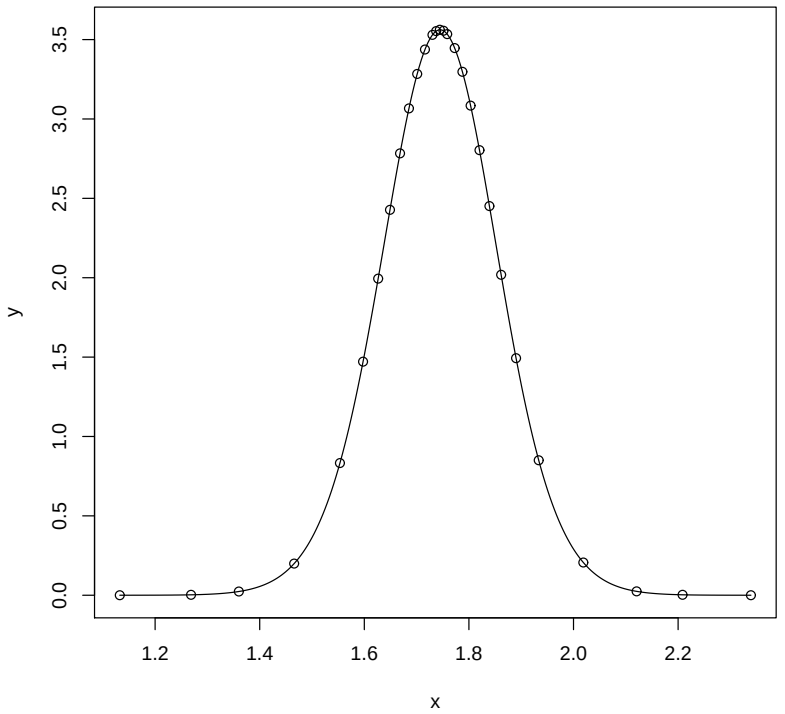
The distribution of the linear predictor at an unobserved location can be computed by specifying the value of the covariate *x* and the desired time index *t* and set *y* to `NA`.

```
# Add new location
x = c(x, 0.3)
t = c(t, 101)
y = c(y, NA)

# Re-compute
result.pred = inla(formula,
                   data = list(x = x, t = t, y = y),
                   family="gaussian",
                   control.predictor = list(compute = TRUE))
```

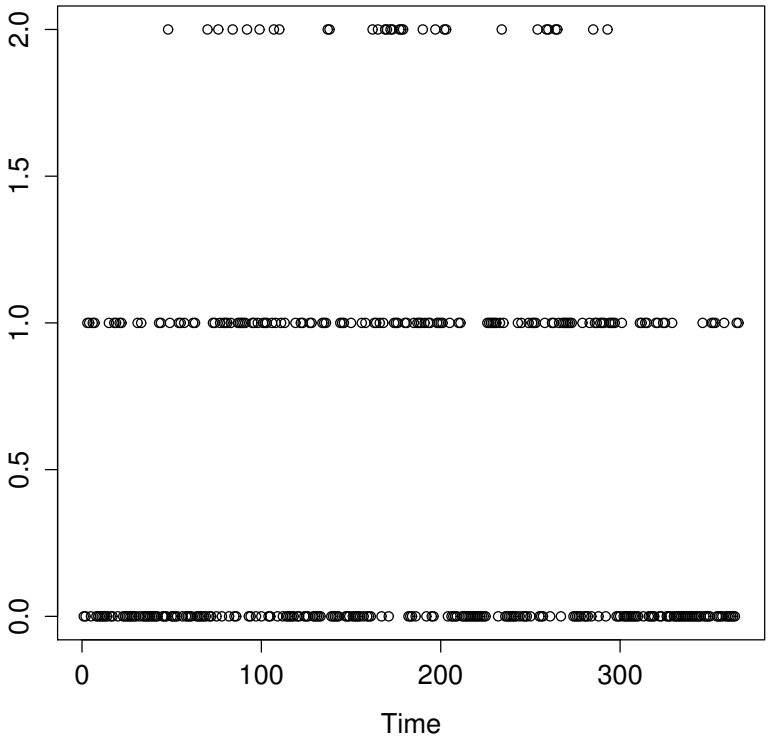
Prediction

```
> m = result.pred$marginals.linear.predictor[[101]]
> round(result.pred$summary.linear.predictor[101,], 3)
                mean    sd 0.025quant 0.5quant 0.975quant  mode
predictor.101 1.744 0.116      1.514    1.744      1.972 1.745
> plot(m)
> lines(inla.smarginal(m))
```



Example: Smoothing binary time series

The data set `Tokyo` is available in the `INLA` package and consists of the number of days in Tokyo with rainfall above 1 mm in 1983–1984.



We want to smooth the time-series.

Observations

Each observation consists of

- t: Day of year; $t \in \{1, 2, \ldots, 366\}$
- n_t: Number of observations for day t in 1983–1984; $n_t \in \{1, 2\}$
- y_t: Number of days with rain out of n_t days for day t; $y_t \in \{0, 1, 2\}$

```
> data(Tokyo)
> head(Tokyo)
> head(Tokyo,4)
  y n time
1 0 2    1
2 0 2    2
3 1 2    3
4 1 2    4
> Tokyo[60,]
   y n time
60 0 1   60
```

Hierarchical model

Stage 1: We have binomial responses with known n_t, but unknown probabilities

$$y_t \sim \text{Binomial}(n_t, p_t)$$

Stage 2: A cyclic second order random walk (CRW2) is connected to the likelihood by

$$p_t = \frac{\exp(\eta_t)}{1 + \exp(\eta_t)} \text{ with linear predictor } \eta_t = \text{CRW2}_t$$

Stage 3: τ: Scale parameter in CRW2 with prior

$$\pi(\tau) \sim \text{Gamma}(1, 5 \cdot 10^{-5})$$

Computations

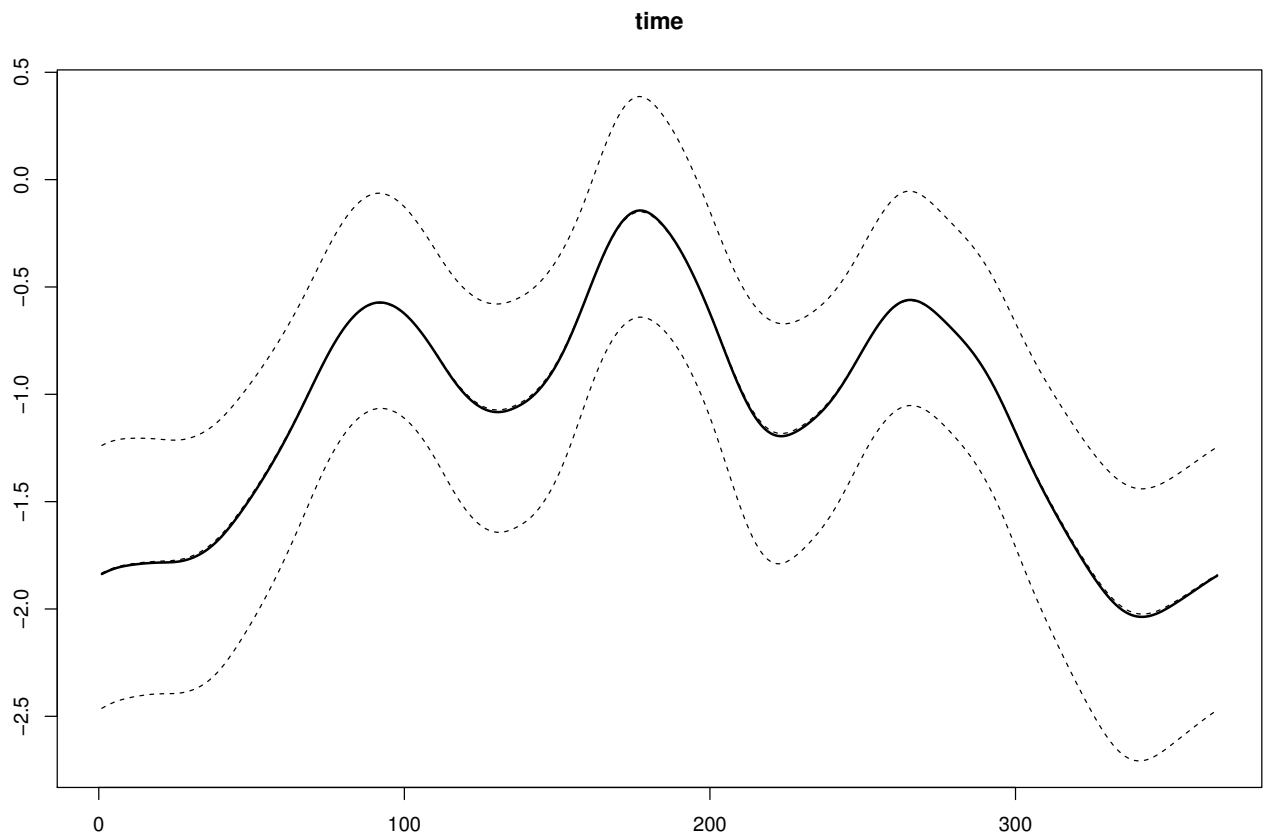
```
# Read data
data(Tokyo)

# Specify linear predictor
formula = y ~ -1 + f(time, model="rw2", cyclic=TRUE)

# Run model
result = inla(formula,
              family = "binomial",
              Ntrials = n,
              data = Tokyo)
```

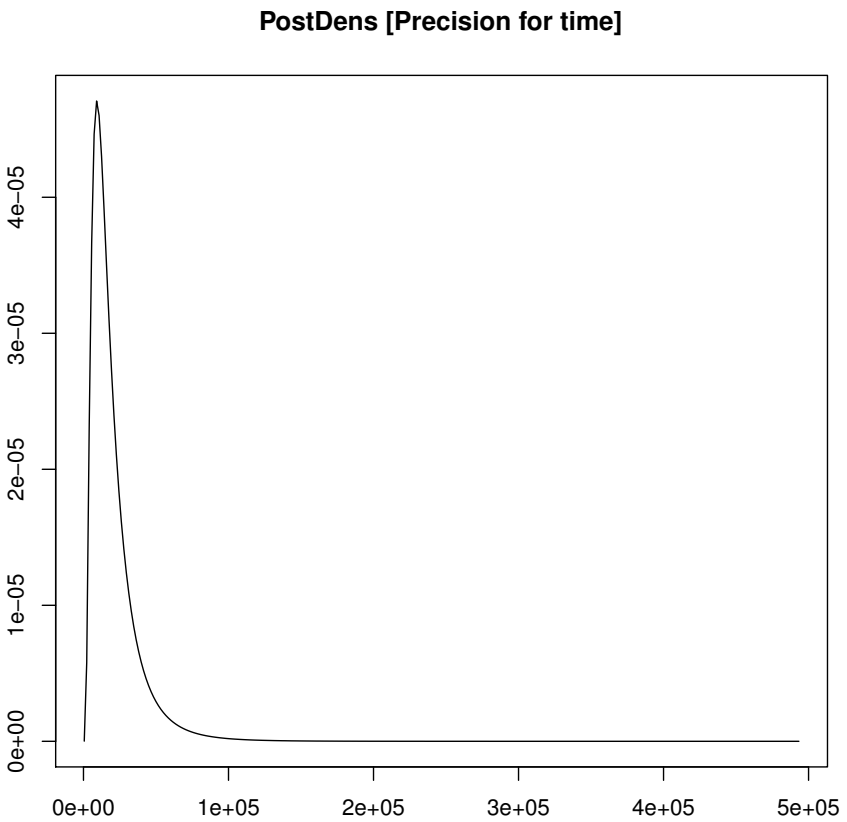
Marginal posterior of CRW2

```
1 plot(result)
```



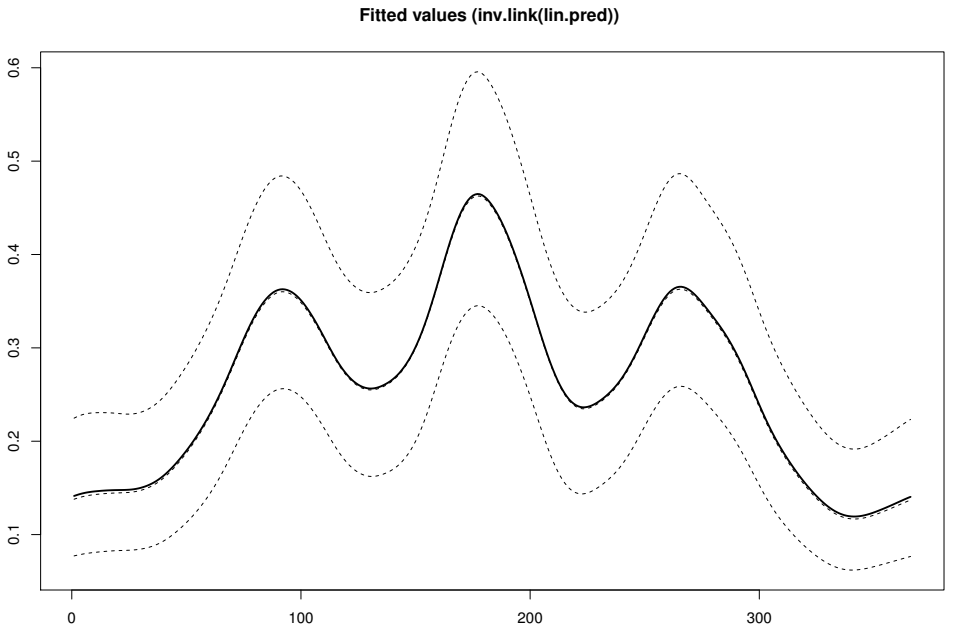
Marginal posterior of scale parameter

```
1 plot(result)
```



Transform to probability

```
result = inla(formula,
              family = "binomial",
              Ntrials = n,
              data = Tokyo,
              control.predictor = list(compute=TRUE))
plot(result)
```



Other choices for `f`-terms

```
names(inla.models()$latent)
```

```
 [1] "linear"       "iid"          "mec"          "meb"          "rgeneric"
 [6] "rw1"          "rw2"          "crw2"         "seasonal"     "besag"
[11] "besag2"       "bym"          "bym2"         "besagproper"  "besagproper2"
[16] "ar1"          "ar"           "ou"           "generic"      "generic0"
[21] "generic1"     "generic2"     "spde"         "spde2"        "iid1d"
[26] "iid2d"        "iid3d"        "iid4d"        "iid5d"        "2diid"
[31] "z"            "rw2d"         "slm"          "matern2d"     "copy"
[36] "clinear"
```

Add weight to components of a random effect

```
formula = y ~ ... + f(idx , weight, model = <MODEL>, ...)
```

extends the usual

$$\eta_i = \ldots + f_{\text{idx}_i}$$

to

$$\eta_i = \ldots + \text{weight}_{\text{idx}_i} f_{\text{idx}_i}$$

Changing the prior: Internal scale

— Hyperparameters are represented internally with more well-behaved transformations, e.g. correlation ρ and precision τ are internally represented as

$$\theta_1 = \log(\tau)$$
$$\theta_2 = \log\left(\frac{1+\rho}{1-\rho}\right)$$

— The prior must be set on the parameter in **internal scale**

— Initial values for the mode-search must be set in **internal scale**

— The functions `to.theta` and `from.theta` can be used to map back and forth.

Changing the prior: Old and new style

There exists two ways to set priors:

— Old style:

```
prior=..., param=..., intial=..., fixed=....
```

— New style:

```
hyper = list(prec = list(initial=2, fixed=TRUE))
```

The old style exists only for backwards-compatibility.

Changing the prior: Code

```
# Old way
formula = y ~ f(idx, model = "iid", prior = "loggamma",
                  param = c(1, 0.1), initial = 4, fixed = FALSE)

# New way
hyper = list(prec = list(prior = "loggamma",
                         param = c(1, 0.1),
                         initial = 4,
                         fixed = FALSE))

formula = y ~ f(idx, model = "iid", hyper = hyper) + ...
```

Changing the prior: Default options

```
# Default options can be seen with
inla.models()$latent$iid$hyper
```

```
theta
      name "log precision"
short.name "prec"
     prior "loggamma"
     param c(1e+00, 5e-05)
   initial 4
     fixed FALSE
  to.theta function(x){log(x)}
from.theta function(x){exp(x)}
```

Changing the prior: Available models

Some of the available choices:

- `"gaussian"`
- `"loggamma"`
- `"flat"`
- `"logtgaussian"`

Also possible to make your own prior (on internal scale) with

- `"expression:"`: `R` expression that calculates log-prior
- `"table:"`: Tabulated values that are interpolated

How to assign your own prior (table)

It is possible to provide a table of suitable support values x (internal scale) and the corresponding log-density values y as string:

$$\text{“}table:\ \ x\_1\ \ldots\ \ x\_n\ y\_1\ \ldots\ \ y\_n\text{”}$$

```
# use suitable support points x
lprec = seq(-10, 10, len=100)

# link the x and corresponding y values into a string which
    begins with "table:""
prior.table = INLA:::inla.paste(c("table:",cbind(lprec,
    prior.function(lprec))))
```

This is consequently assigned as

```
hyper = list(prec = list(prior = prior.table))
```

EPIL example

Seizure counts in a randomised trial of anti-conversant therapy in epilepsy. From WinBUGS manual.

Patient	y1	y2	y3	y4	Trt	Base	Age
1	5	3	3	3	0	11	31
2	3	5	3	3	0	11	30
3	2	4	0	5	0	6	25
....							
59	1	4	3	2	1	12	37

Covariates are treatment (0,1), 8-week baseline seizure counts, and age in years.

Repeated Poisson counts

$$
\begin{aligned}
y_{jk} &\sim \text{Poisson}(\mu_{jk});\ j = 1, \ldots, 59;\ k = 1, \ldots, 4 \\
\log(\mu_{jk}) &= \alpha_0 + \alpha_1 \log(\text{Base}_j/4) + \alpha_2 \text{Trt}_j \\
&\quad + \alpha_3 \text{Trt}_j \log(\text{Base}_j/4) + \alpha_4 \text{Age}_j \\
&\quad + \alpha_5 V4 + \text{Ind}_j + \beta_{jk}
\end{aligned}
$$

$$
\begin{aligned}
\alpha_i &\sim \mathcal{N}(0, \tau_\alpha) && \tau_\alpha \text{ known (0.001)} \\
\text{Ind}_j &\sim \mathcal{N}(0, \tau_{\text{Ind}}) && \tau_{\text{Ind}} \sim \text{Gamma}(1, 0.01) \\
\beta_{jk} &\sim \mathcal{N}(0, \tau_\beta) && \tau_\beta \sim \text{Gamma}(1, 0.01)
\end{aligned}
$$

Here, `V4` is an indicator variable for the 4th visit.

Model specification in INLA

```
> data(Epil)
> head(Epil,n=3)
   y Trt Base Age V4 rand Ind        CTrt      ClBase4   CV4        ClAge
1  5   0   11  31  0    1   1 -0.5254237 -0.75635379 -0.25  0.11420370
2  3   0   11  31  0    2   1 -0.5254237 -0.75635379 -0.25  0.11420370
3  3   0   11  31  0    3   1 -0.5254237 -0.75635379 -0.25  0.11420370
4  3   0   11  31  1    4   1 -0.5254237 -0.75635379  0.75  0.11420370
```

Model specification in INLA

```
> data(Epil)
> head(Epil,n=3)
   y Trt Base Age V4 rand Ind        CTrt      ClBase4   CV4        ClAge
1  5   0   11  31  0    1   1 -0.5254237 -0.75635379 -0.25  0.11420370
2  3   0   11  31  0    2   1 -0.5254237 -0.75635379 -0.25  0.11420370
3  3   0   11  31  0    3   1 -0.5254237 -0.75635379 -0.25  0.11420370
4  3   0   11  31  1    4   1 -0.5254237 -0.75635379  0.75  0.11420370
```

```
> formula = y ~ ClBase4*CTrt + ClAge + CV4 +
    f(Ind, model="iid",
      hyper = list(prec = list(prior = "loggamma",
                               param = c(1,0.01)))) +
    f(rand, model="iid",
      hyper = list(prec = list(prior = "loggamma",
                               param = c(1,0.01))))
```

Model specification in INLA

```
> data(Epil)
> head(Epil,n=3)
   y Trt Base Age V4 rand Ind        CTrt      ClBase4   CV4        ClAge
1  5   0   11  31  0    1   1 -0.5254237 -0.75635379 -0.25  0.11420370
2  3   0   11  31  0    2   1 -0.5254237 -0.75635379 -0.25  0.11420370
3  3   0   11  31  0    3   1 -0.5254237 -0.75635379 -0.25  0.11420370
4  3   0   11  31  1    4   1 -0.5254237 -0.75635379  0.75  0.11420370
```

```
> formula = y ~ ClBase4*CTrt + ClAge + CV4 +
    f(Ind, model="iid",
      hyper = list(prec = list(prior = "loggamma",
                               param = c(1,0.01)))) +
    f(rand, model="iid",
      hyper = list(prec = list(prior = "loggamma",
                               param = c(1,0.01))))
```

```
> result = inla(formula, family="poisson", data = Epil,
     control.fixed = list(prec.intercept = 0.001,
                          prec = 0.001))
```

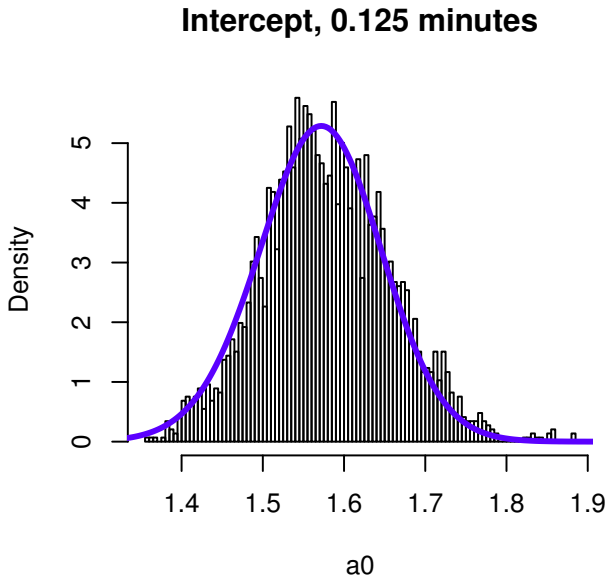
Comparing results with MCMC

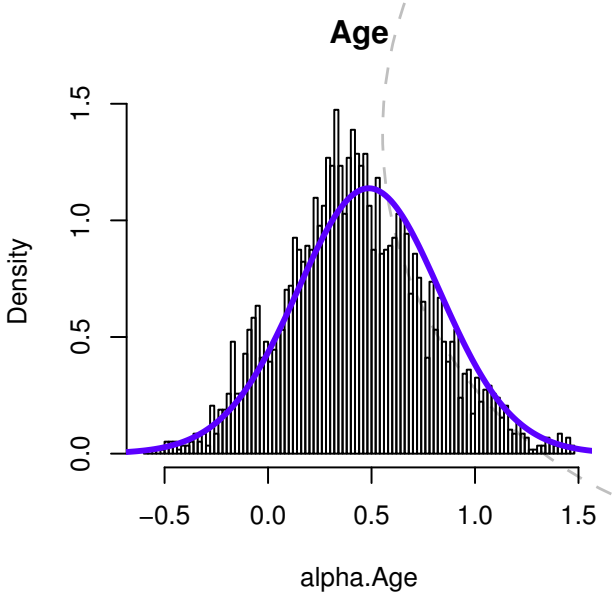
— When comparing the results of R-INLA with MCMC, it is important to use the same model.

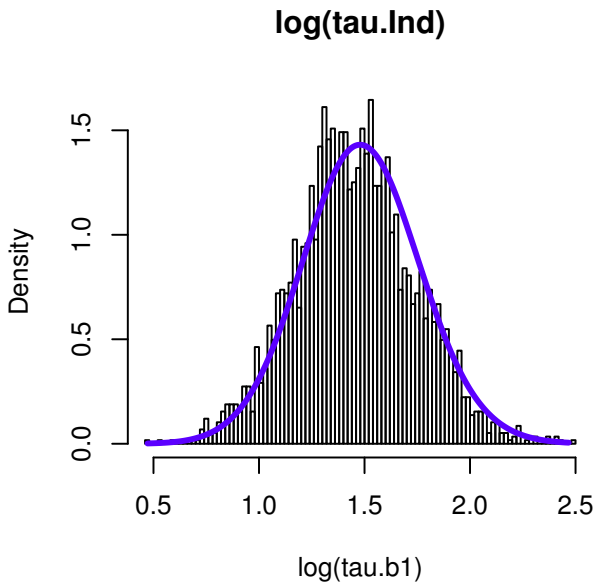
Comparing results with MCMC

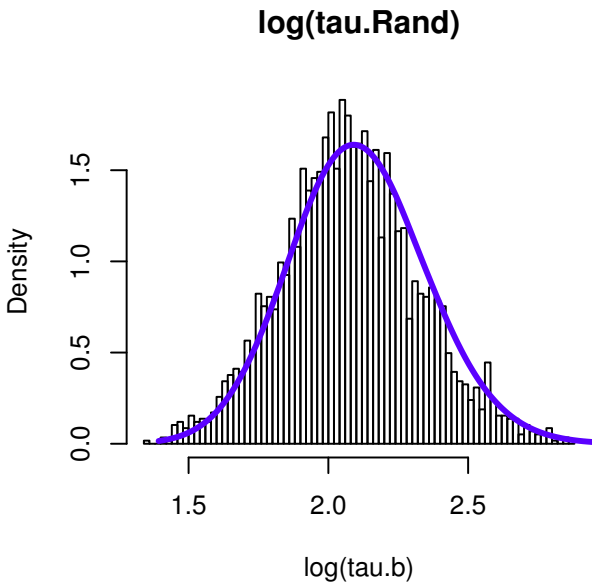
— When comparing the results of R-INLA with MCMC, it is important to use the same model.

— Here we have compared the results with those obtained using JAGS via the rjags package

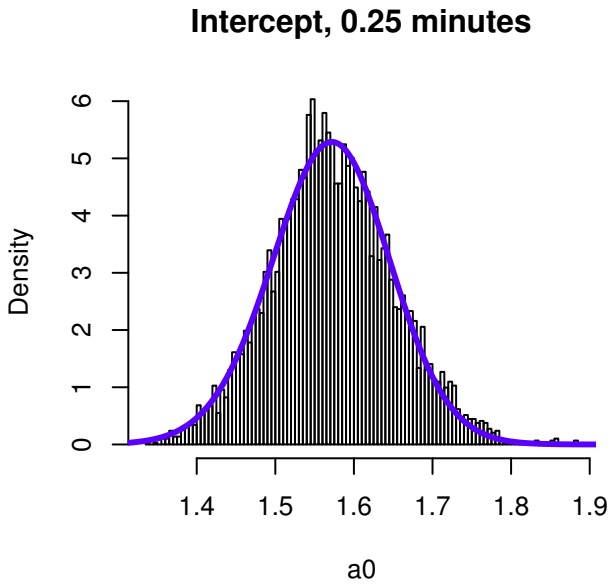


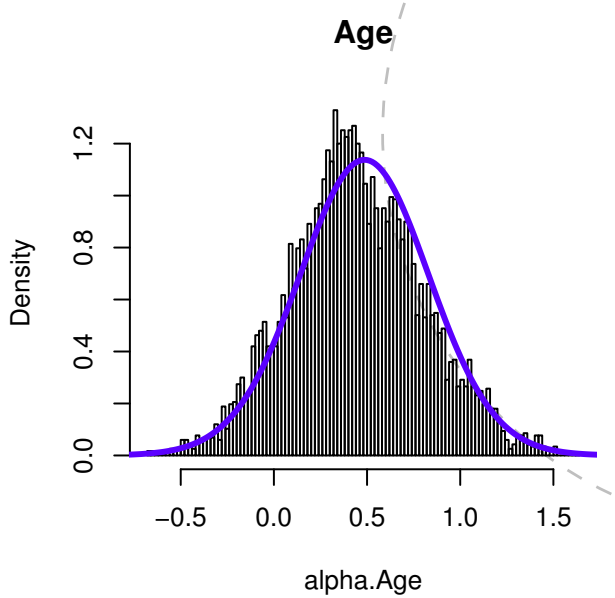


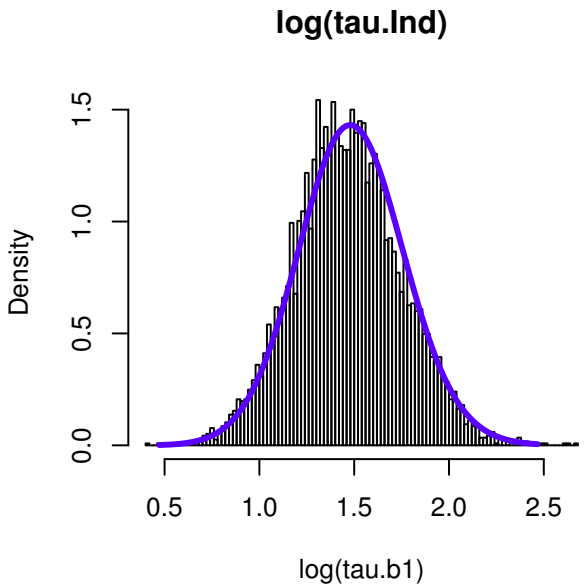


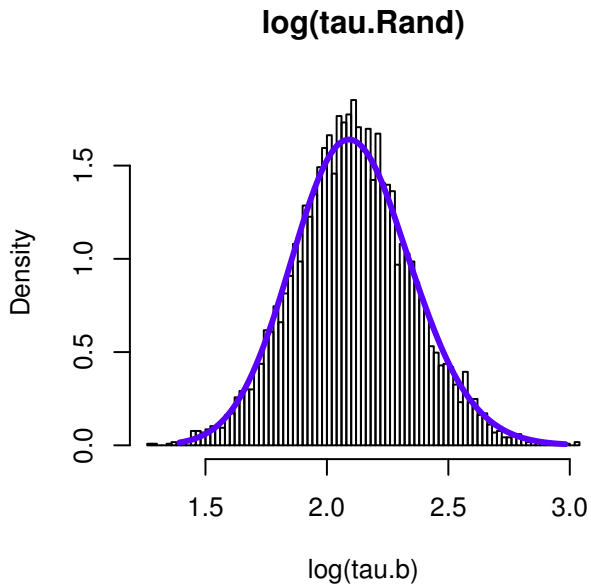


Running time of INLA $<$ 0.5 seconds

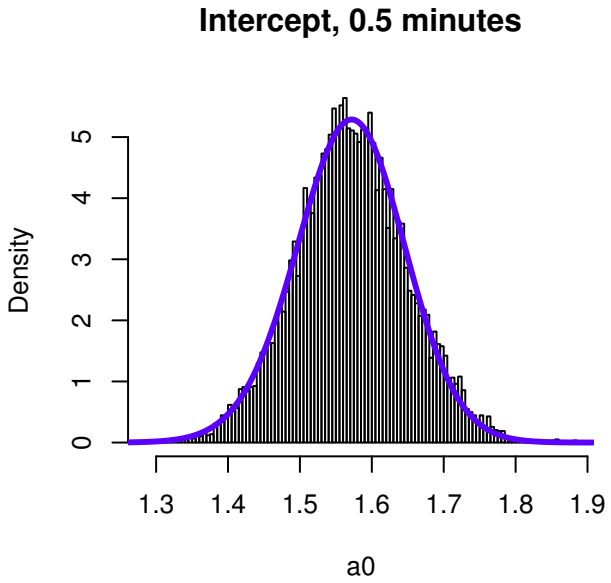


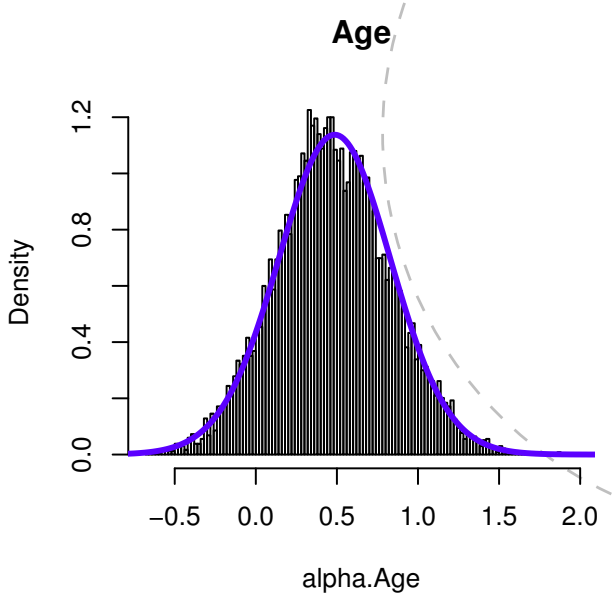


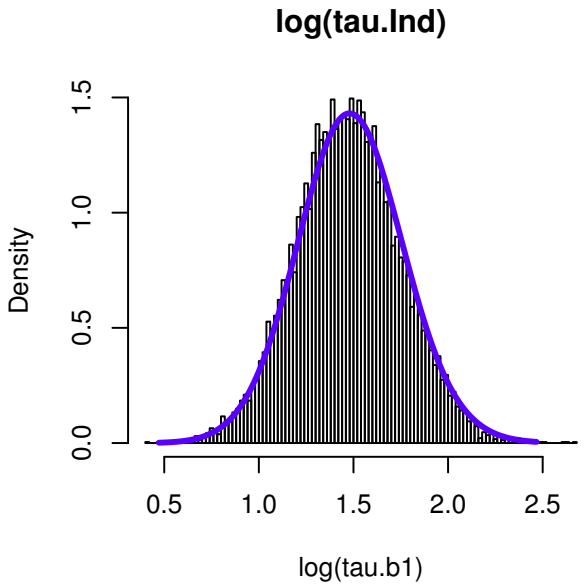


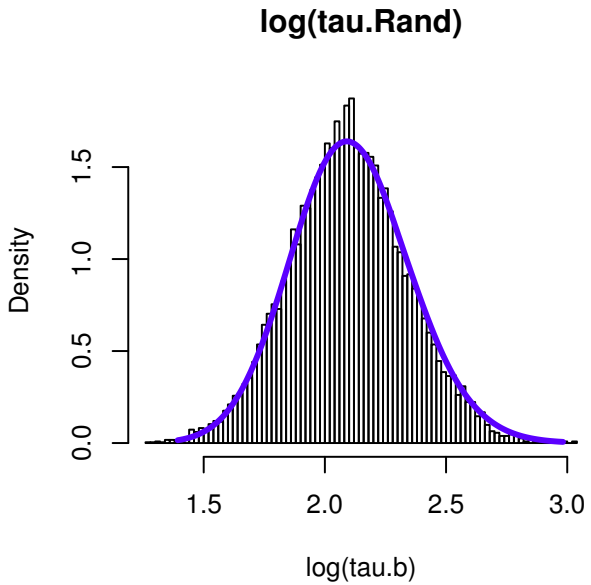


Running time of INLA $<$ 0.5 seconds

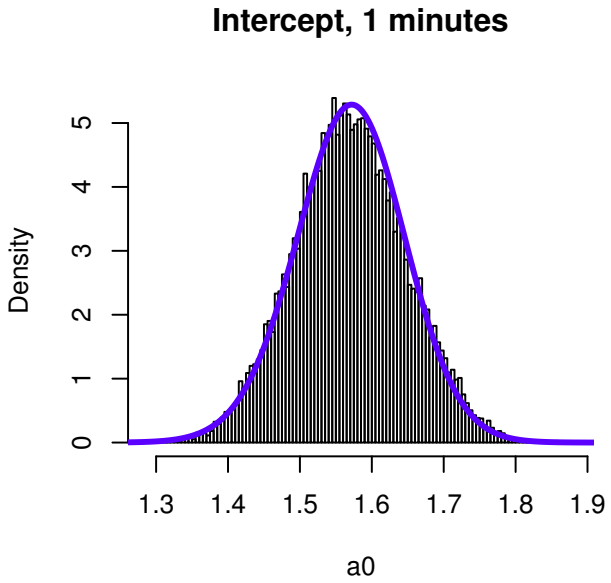


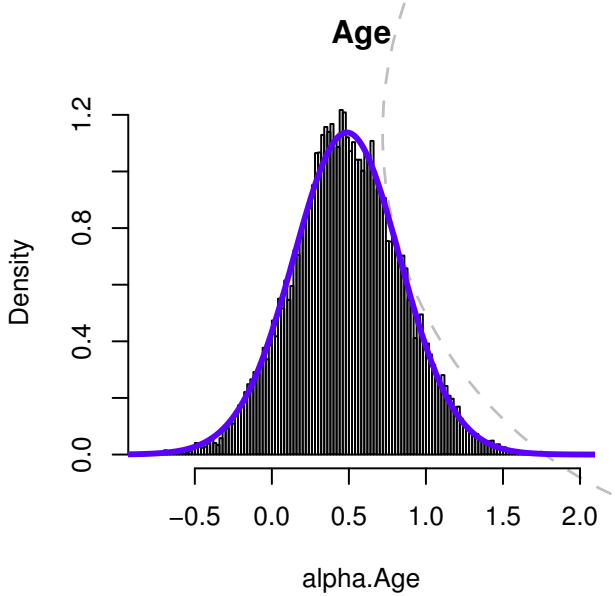


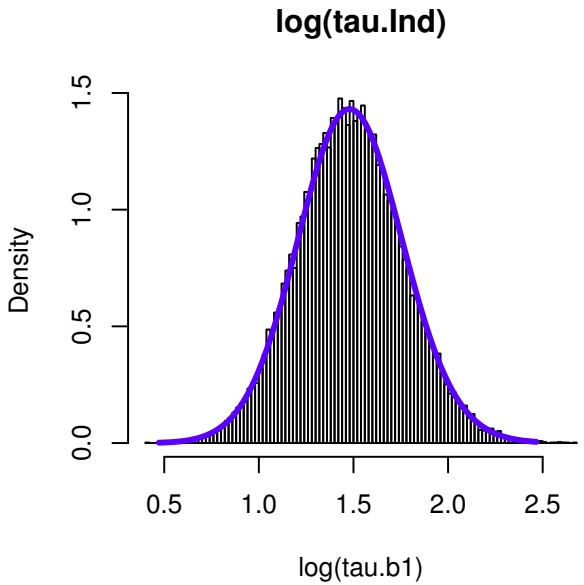


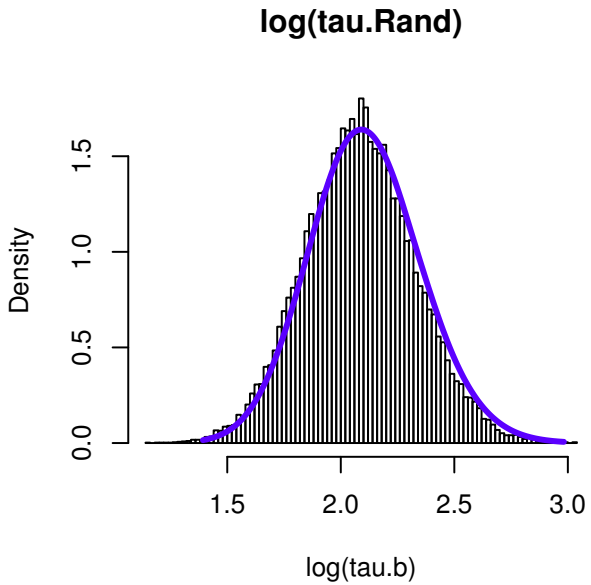


Running time of INLA $<$ 0.5 seconds

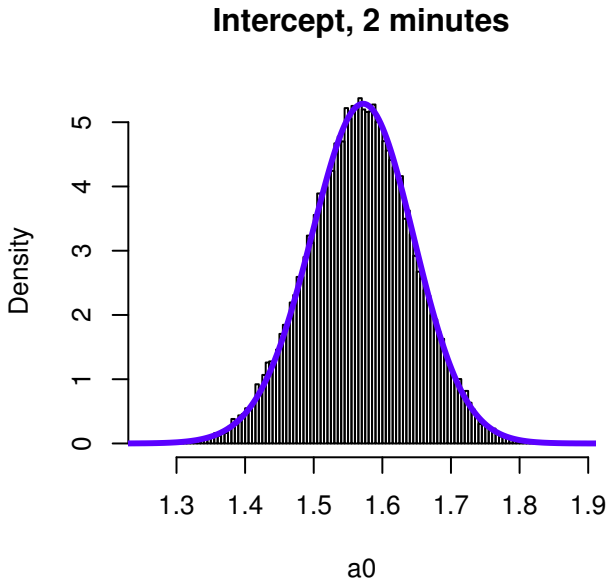


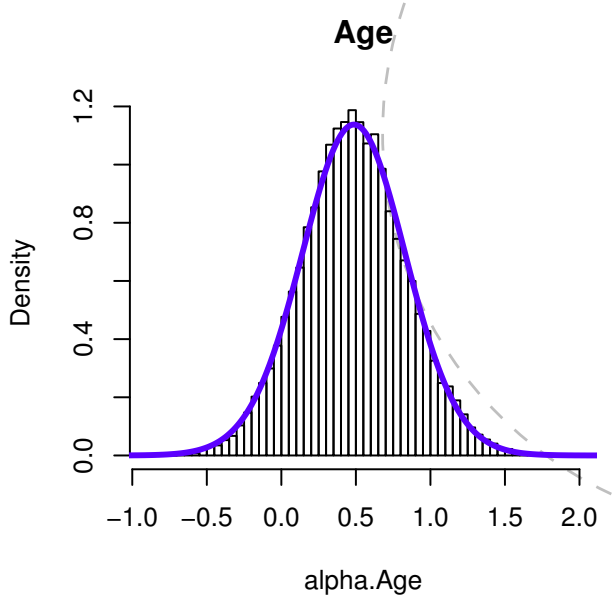


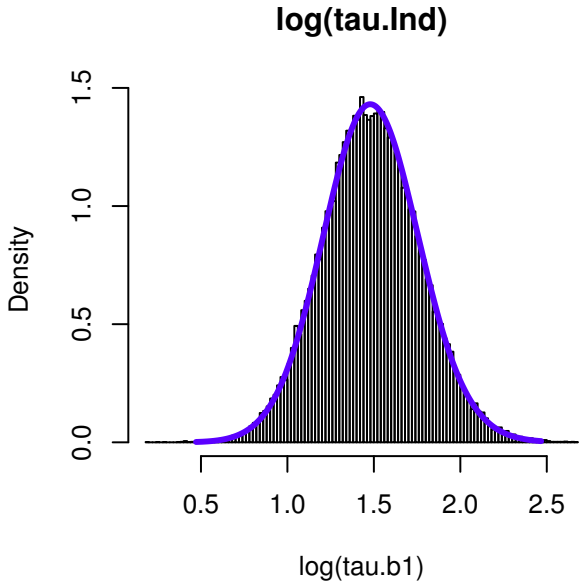


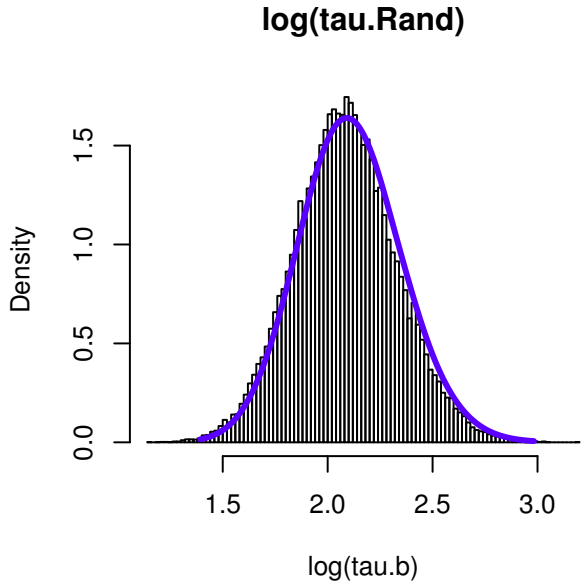


Running time of INLA $<$ 0.5 seconds

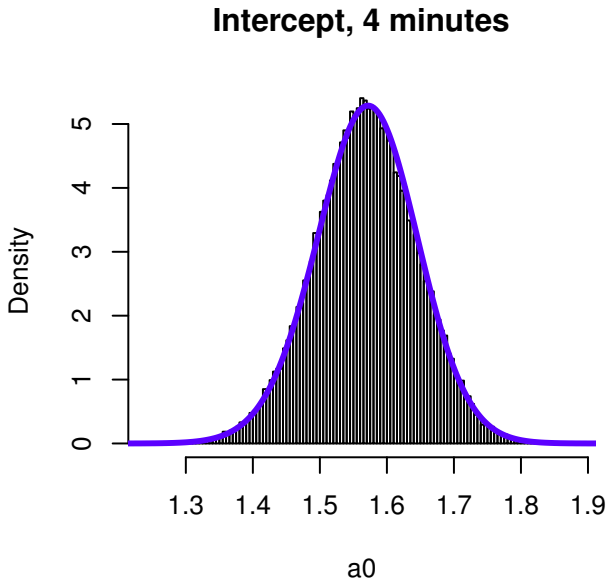


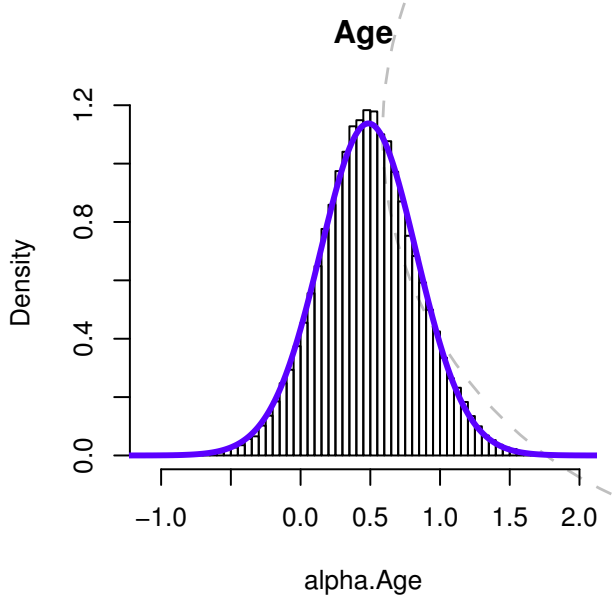


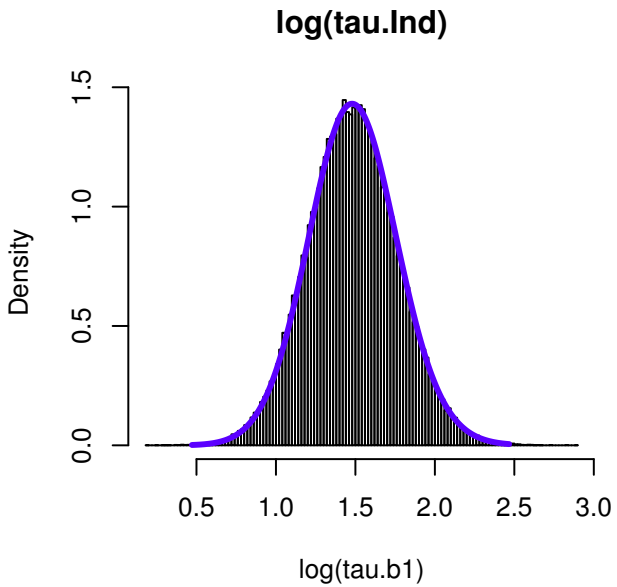


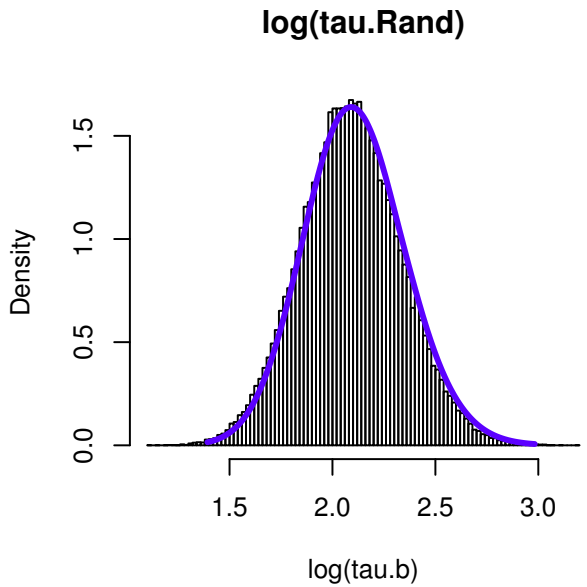


Running time of INLA $<$ 0.5 seconds

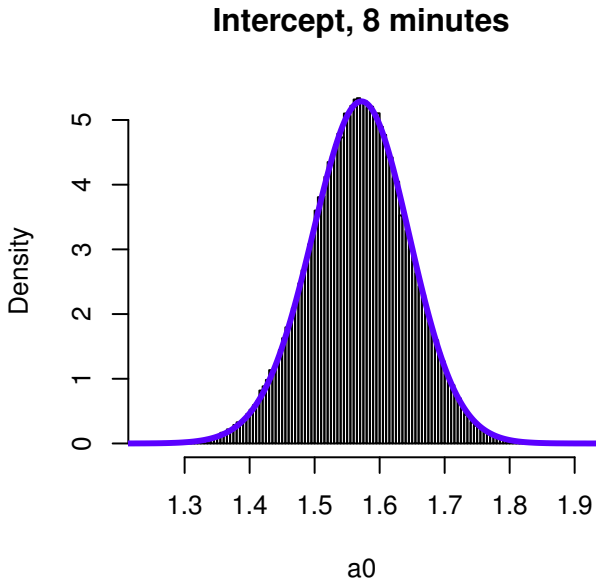


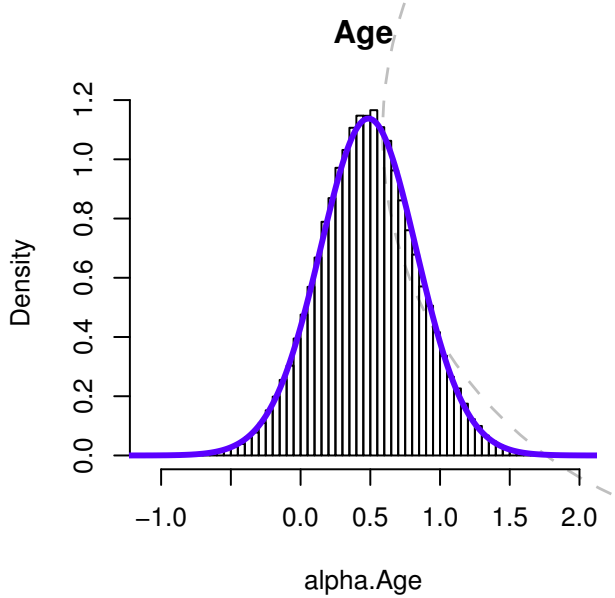


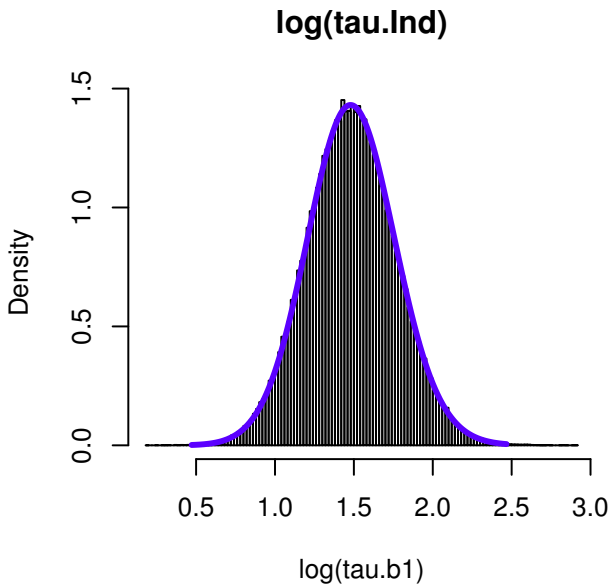


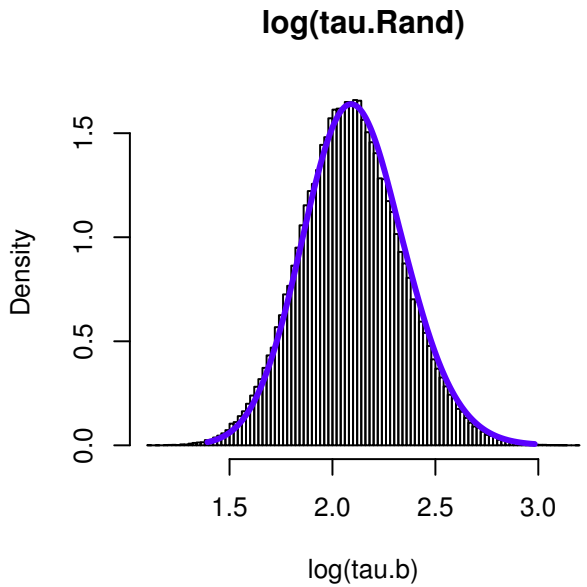


Running time of INLA $<$ 0.5 seconds

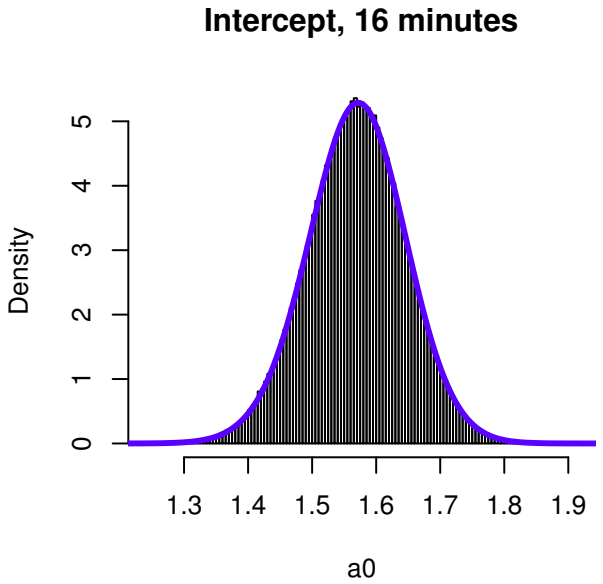


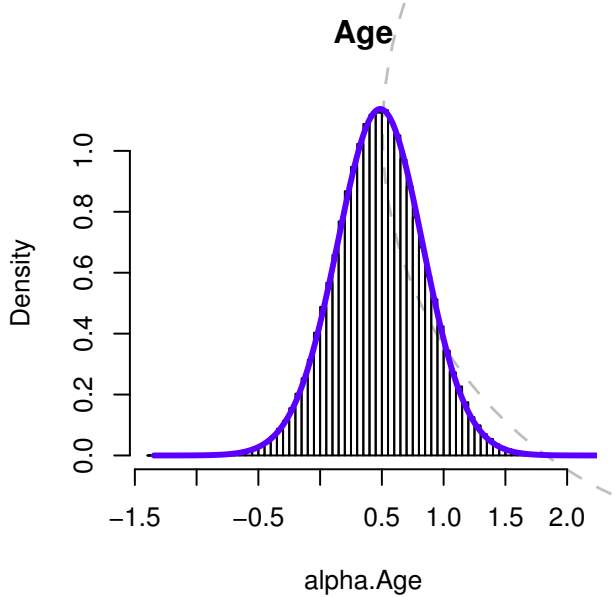


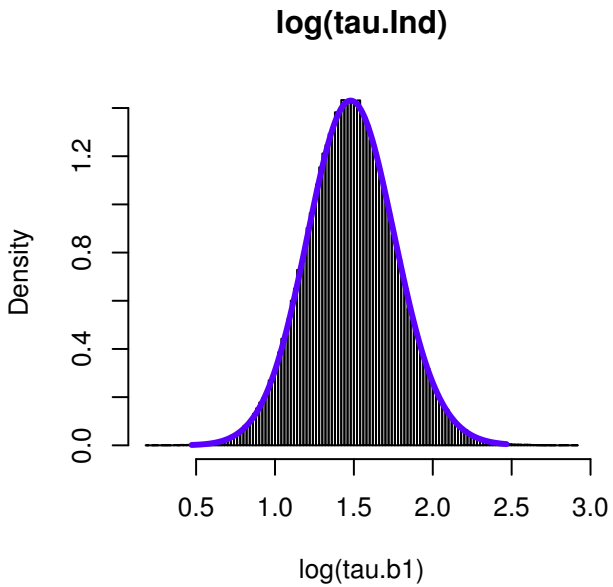


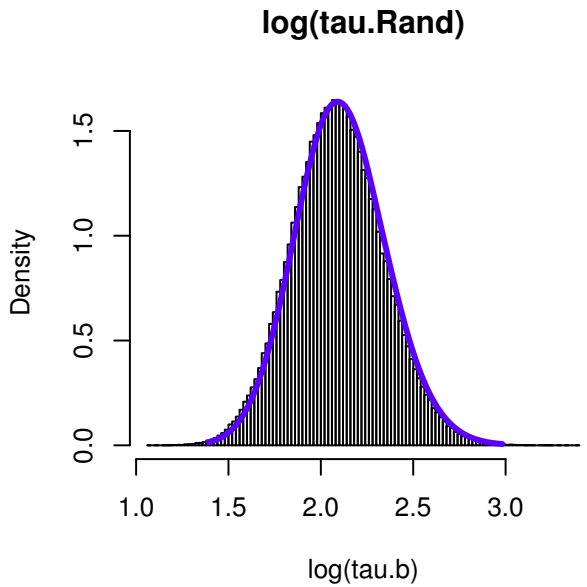


Running time of INLA $<$ 0.5 seconds

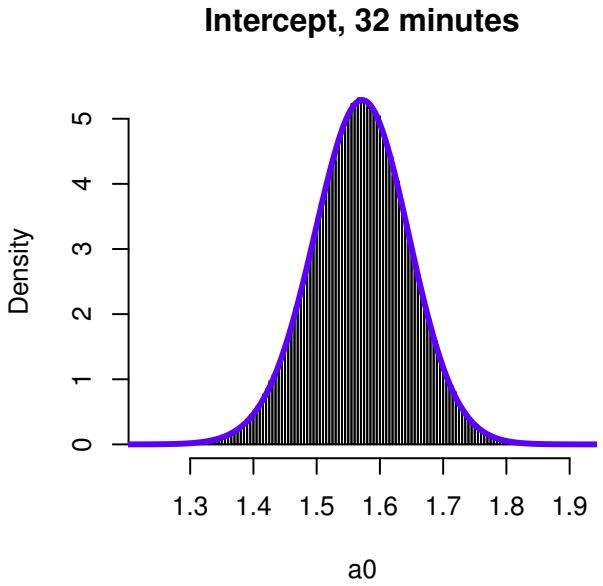


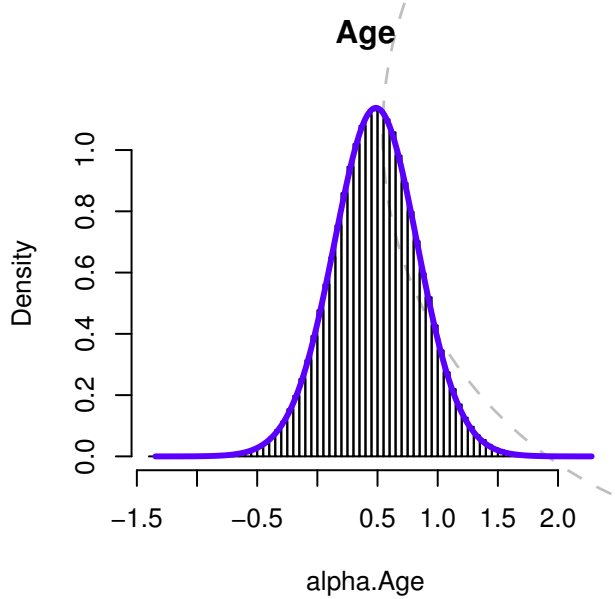


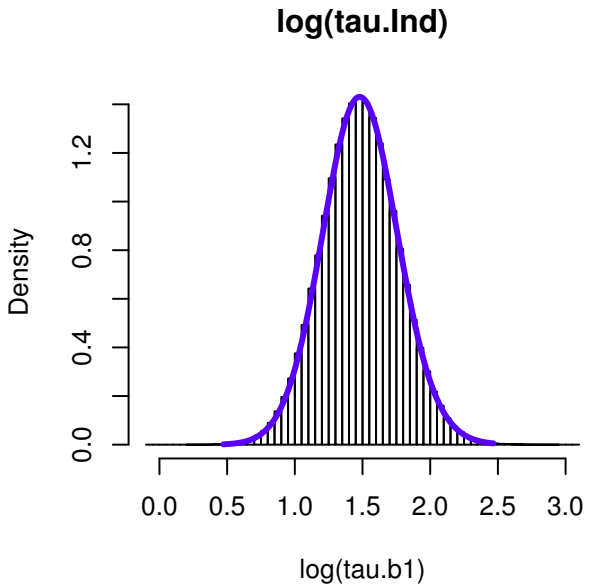


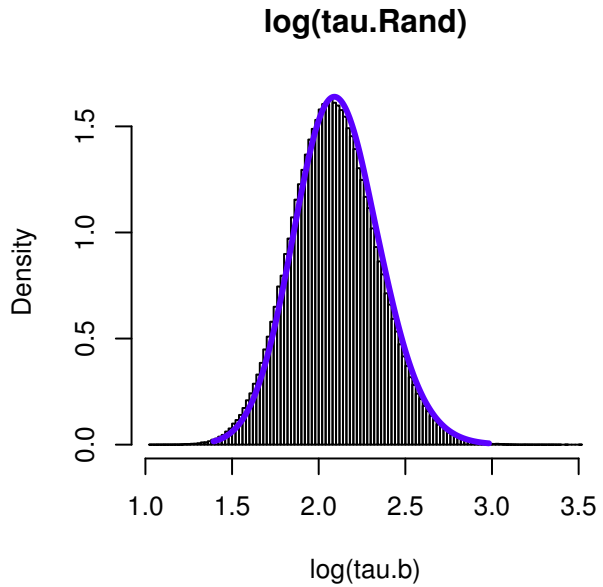


Running time of INLA $<$ 0.5 seconds

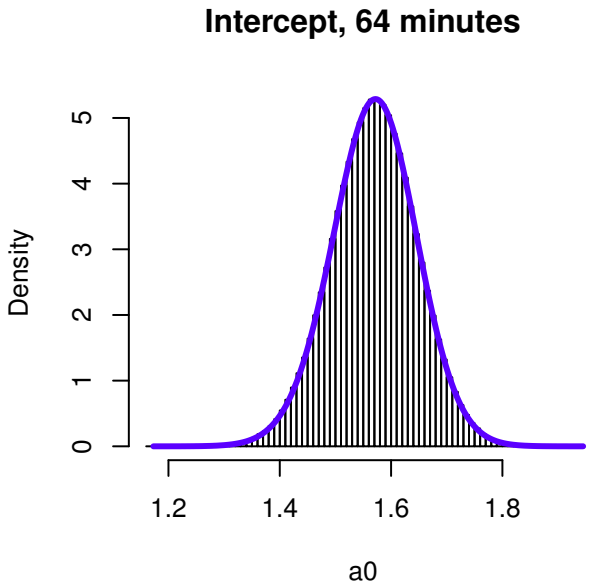


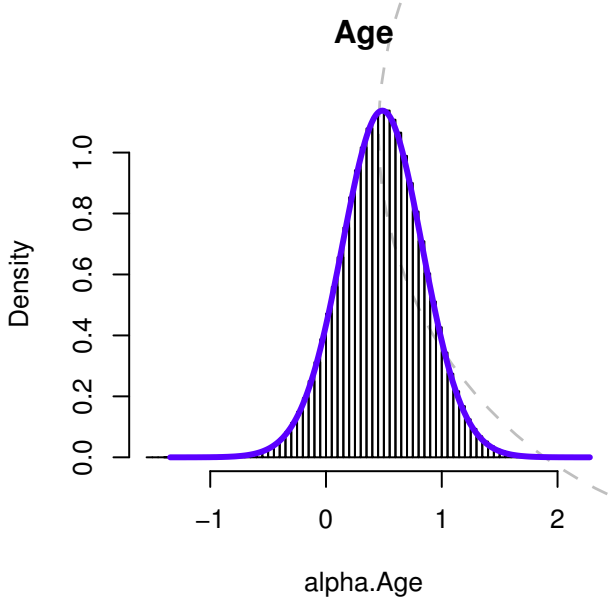


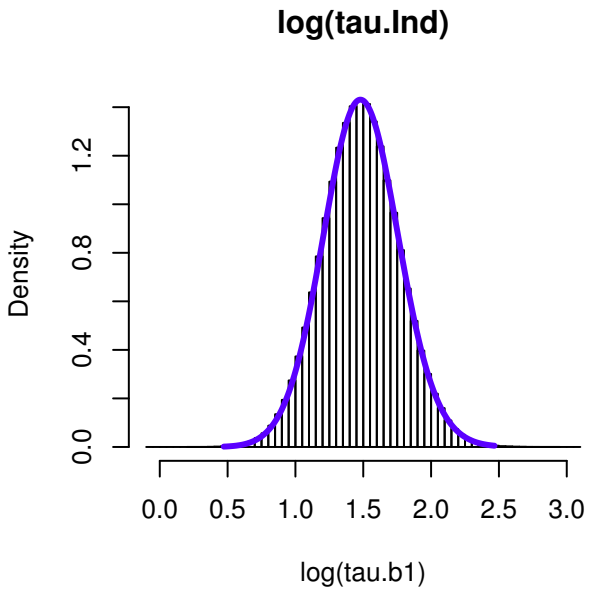


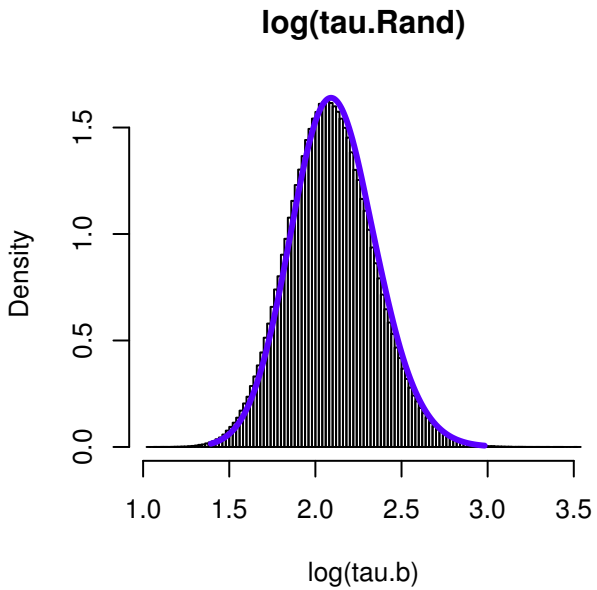


Running time of INLA $<$ 0.5 seconds

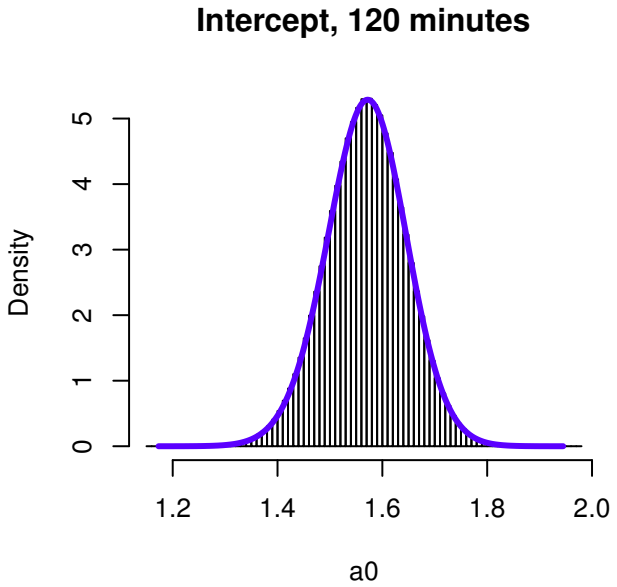


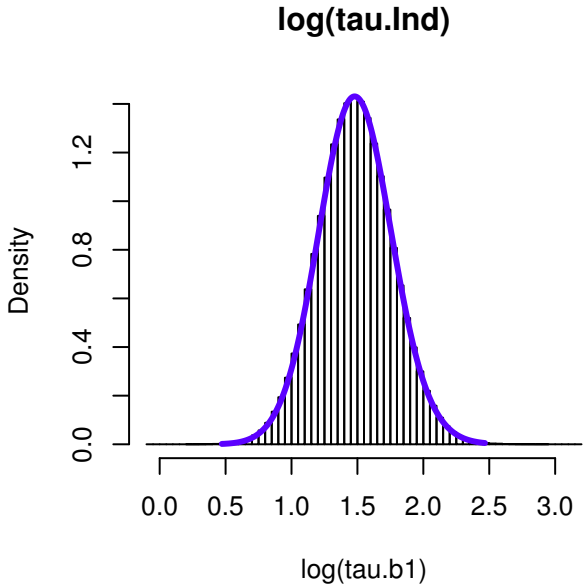


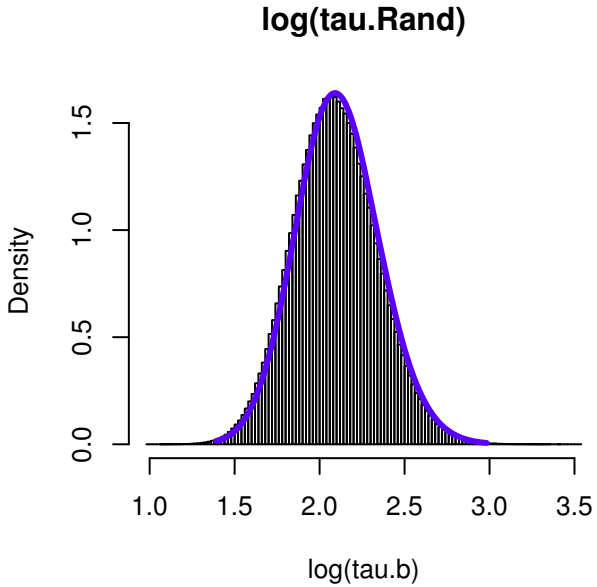




Running time of INLA $<$ 0.5 seconds







Running time of INLA $<$ 0.5 seconds

Control statements

`control.xxx` statements control computations

— `control.fixed`

Control statements

`control.xxx` statements control computations

— `control.fixed`

- `prec`: Default precision for all fixed effects except the intercept.
  `prec.intercept`: Precision for intercept (Default: 0.0)

Control statements

`control.xxx` statements control computations

— `control.fixed`

- `prec`: Default precision for all fixed effects except the intercept.
  `prec.intercept`: Precision for intercept (Default: 0.0)

— `control.predictor`

Control statements

`control.xxx` statements control computations

— `control.fixed`

- `prec`: Default precision for all fixed effects except the intercept.
  `prec.intercept`: Precision for intercept (Default: 0.0)

— `control.predictor`

- `compute`: Compute posterior marginals of linear predictors

Control statements

`control.xxx` statements control computations

— `control.fixed`

- `prec`: Default precision for all fixed effects except the intercept.
  `prec.intercept`: Precision for intercept (Default: 0.0)

— `control.predictor`

- `compute`: Compute posterior marginals of linear predictors
- `A`: Apply linear transformation to linear predictor; we will see this one later

Control statements

`control.xxx` statements control computations

— `control.fixed`
  - `prec`: Default precision for all fixed effects except the intercept. `prec.intercept`: Precision for intercept (Default: 0.0)

— `control.predictor`
  - `compute`: Compute posterior marginals of linear predictors
  - `A`: Apply linear transformation to linear predictor; we will see this one later

— `control.compute`

Control statements

`control.xxx` statements control computations

— `control.fixed`
  - `prec`: Default precision for all fixed effects except the intercept.
    `prec.intercept`: Precision for intercept (Default: 0.0)

— `control.predictor`
  - `compute`: Compute posterior marginals of linear predictors
  - `A`: Apply linear transformation to linear predictor; we will see this one later

— `control.compute`
  - `dic`, `mlik`, `cpo`: Compute measures of fit?

Control statements

`control.xxx` statements control computations

— `control.fixed`
  - `prec`: Default precision for all fixed effects except the intercept. `prec.intercept`: Precision for intercept (Default: 0.0)

— `control.predictor`
  - `compute`: Compute posterior marginals of linear predictors
  - `A`: Apply linear transformation to linear predictor; we will see this one later

— `control.compute`
  - `dic`, `mlik`, `cpo`: Compute measures of fit?
  - `config`: Save internal GMRF approximations? (needed to use `inla.posterior.sample()`)

Control statements

`control.xxx` statements control computations

— `control.fixed`
  - `prec`: Default precision for all fixed effects except the intercept. `prec.intercept`: Precision for intercept (Default: 0.0)

— `control.predictor`
  - `compute`: Compute posterior marginals of linear predictors
  - `A`: Apply linear transformation to linear predictor; we will see this one later

— `control.compute`
  - `dic`, `mlik`, `cpo`: Compute measures of fit?
  - `config`: Save internal GMRF approximations? (needed to use `inla.posterior.sample()`)

— `control.inla`

Control statements

`control.xxx` statements control computations

— `control.fixed`
  - `prec`: Default precision for all fixed effects except the intercept. `prec.intercept`: Precision for intercept (Default: 0.0)

— `control.predictor`
  - `compute`: Compute posterior marginals of linear predictors
  - `A`: Apply linear transformation to linear predictor; we will see this one later

— `control.compute`
  - `dic`, `mlik`, `cpo`: Compute measures of fit?
  - `config`: Save internal GMRF approximations? (needed to use `inla.posterior.sample()`)

— `control.inla`
  - `strategy` and `int.strategy` contain useful advanced features

Control statements

`control.xxx` statements control computations

- `control.fixed`
  - `prec`: Default precision for all fixed effects except the intercept. `prec.intercept`: Precision for intercept (Default: 0.0)
- `control.predictor`
  - `compute`: Compute posterior marginals of linear predictors
  - `A`: Apply linear transformation to linear predictor; we will see this one later
- `control.compute`
  - `dic`, `mlik`, `cpo`: Compute measures of fit?
  - `config`: Save internal GMRF approximations? (needed to use `inla.posterior.sample()`)
- `control.inla`
  - `strategy` and `int.strategy` contain useful advanced features
- There are various others as well; see help.

Model choice

There is a need to compare and choose between various models.
This is a difficult problem, but R-INLA has some options available:

- Marginal likelihood $\Rightarrow$ Bayes factors
- Deviance information criterion (DIC)

There are also some predictive checks for the model:

- Conditional predictive ordinate (CPO)
- Probability integral transform (PIT)

Marginal likelihood

```
result = inla(formula,
              data = data,
              control.compute=list(mlik=TRUE))

# See result
result$mlik
```

— Calculates $\log(\pi(\boldsymbol{y}))$

— Can calculate Bayes factors through differences in value

— **NB**: Problematic for intrinsic models

Deviance information criterion

```
result = inla(formula,
              data = data,
              control.compute=list(dic=TRUE))

# See result
result$dic$dic
```

DIC is a measure of complexity and fit. It is used to compare complex hierarchical models and is defined as:

$$\text{DIC} = \overline{D} + p_D$$

where $\overline{D}$ is the posterior mean of the deviance and p_D is the effective number of parameters. Smaller values of the DIC indicate a better trade-off between complexity and fit of the model.

Conditional predictive ordinate

```
result = inla(formula,
              data = data,
              control.compute=list(cpo=TRUE))

# See result
result$cpo$cpo
```

- Measures fit through the predictive density $\pi(y_i^{obs} \mid \boldsymbol{y}_{-i})$
- Basically, Bayesian hold-one out
- Easy to compute in the INLA-approach

Probability integral transform

```
result = inla(formula,
              data = data,
              control.compute=list(cpo=TRUE))

# See result
result$cpo$pit
```

— Given by

$$\text{Prob}(Y_i \leq y_i^{obs} \mid \boldsymbol{y}_{-i})$$

— Detects outliers

— Should look out for unusually small or large values

— PIT histograms should be uniform

(maybe use transformation for count data,
see Czado et al. (2009) “Predictive Model Assessment for Count Data”)

Useful features

There are several features that can be used to extend the standard models in R-INLA

- Replicate and group
- Multiple likelihoods
- Copy
- Linear transformation of linear predictor (**A** matrix)
- Linear combinations
- Values
- Remote computing

Feature: replicate

"replicate" generates iid replicates from the same f()-model with the same hyperparameters.

If $\boldsymbol{x} \mid \boldsymbol{\theta} \sim$ AR(1), then nrep=3, makes

$$\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$$

with mutually independent $\boldsymbol{x}_i$'s from AR(1) with the same $\boldsymbol{\theta}$
Arguments

```
f(..., replicate = r [, nrep = nr ])
```

where replicate are integers $1, 2, \ldots,$ etc

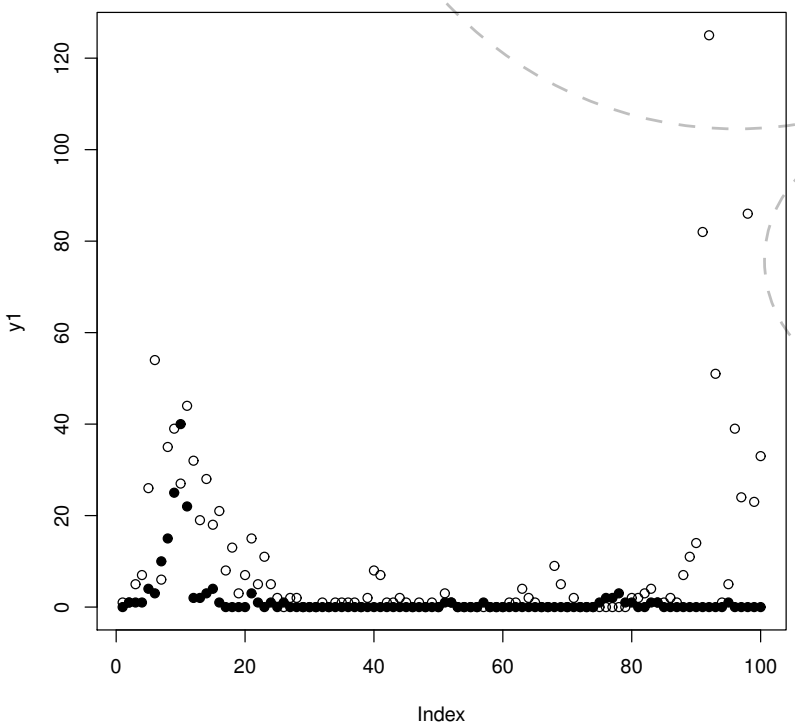
Example: replicate

```
n=100

# x1 and x2 are the same ar1 process with
# different intercepts
x1 = arima.sim(n, model=list(ar=0.9))+1
x2 = arima.sim(n, model=list(ar=0.9))-1

y1 = rpois(n,exp(x1))
y2 = rpois(n,exp(x2))

plot(y1)
points(y2, pch=19}
```



Code to run INLA

```
y = c(y1,y2)
i = rep(1:n,2)
r = rep(1:2,each=n)
intercept = as.factor(r)
```

Code to run INLA

```
y = c(y1,y2)
i = rep(1:n,2)
r = rep(1:2,each=n)
intercept = as.factor(r)
```

```
## showing these for n=10...
> i
 [1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10
> r
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
> intercept
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels: 1 2
```

Code to run INLA

```
y = c(y1,y2)
i = rep(1:n,2)
r = rep(1:2,each=n)
intercept = as.factor(r)
```

```
## showing these for n=10...
> i
 [1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10
> r
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
> intercept
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels: 1 2
```

```
formula = y ~ f(i, model="ar1", replicate=r) + intercept -1
result = inla(formula, family = "poisson",
              data = data.frame(y, i, r, intercept))
```

Feature: group

- Similar concept as replicate, but with a dependence structure on the replicates. E.g. rw1, rw2, ar1, exchangeable
- Implemented as a Kronecker product (often space and time)
- It's possible to use both replicate and group! This will be replications of the grouped model
- Used with `f(..., group = g [, ngroup = ng])`

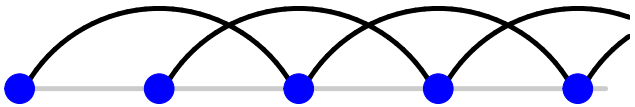
Example: correlated smoothing priors

Consider an RW2, $\alpha_i \sim \mathcal{N}(2\alpha_{i-1} + \alpha_{i-2}, \kappa^{-1})$, and let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_I)^T$, so that:

$$f(\boldsymbol{\alpha}|\kappa) \propto \kappa^{\frac{I-2}{2}} \exp\left(-\frac{\kappa}{2}\sum_{i=3}^{I}(\alpha_i - 2\alpha_{i-1} + \alpha_{i-2})^2\right) = \kappa^{\frac{I-2}{2}} \exp\left(-\frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{Q} \boldsymbol{\alpha}\right)$$

$$\boldsymbol{Q} = \kappa \begin{pmatrix} 1 & -2 & 1 & & & & \\ -2 & 5 & -4 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -4 & 5 & -2 \\ & & & & 1 & -2 & 1 \end{pmatrix}$$

Graph:



Correlated random walk of second order

Assume we have R random walks of second order with $\boldsymbol{\alpha}_i = (\alpha_{i,1}, \ldots, \alpha_{i,R})^T$ and where $\tilde{\boldsymbol{\alpha}} = (\alpha_{1,1}, \ldots, \alpha_{I,1}, \ldots, \alpha_{1,R}, \ldots, \alpha_{I,R})^T$ denotes the stacked vector. Then, the joint distribution is given by:

$$f(\tilde{\boldsymbol{\alpha}}|\boldsymbol{\Sigma}, \kappa) \propto |\kappa \boldsymbol{\Sigma}^{-1}|^{(I-2)/2} \exp\left(-\frac{1}{2}\tilde{\boldsymbol{\alpha}}^T \left\{\boldsymbol{\Sigma}^{-1} \otimes \boldsymbol{Q}\right\} \tilde{\boldsymbol{\alpha}}\right),$$

where $\otimes$ denotes the Kronecker product and $\boldsymbol{\Sigma}$ represents the correlation structure, for example an exchangeable correlation structure. (Caution: There is only one precision parameter!)

Exchangeable correlation structure

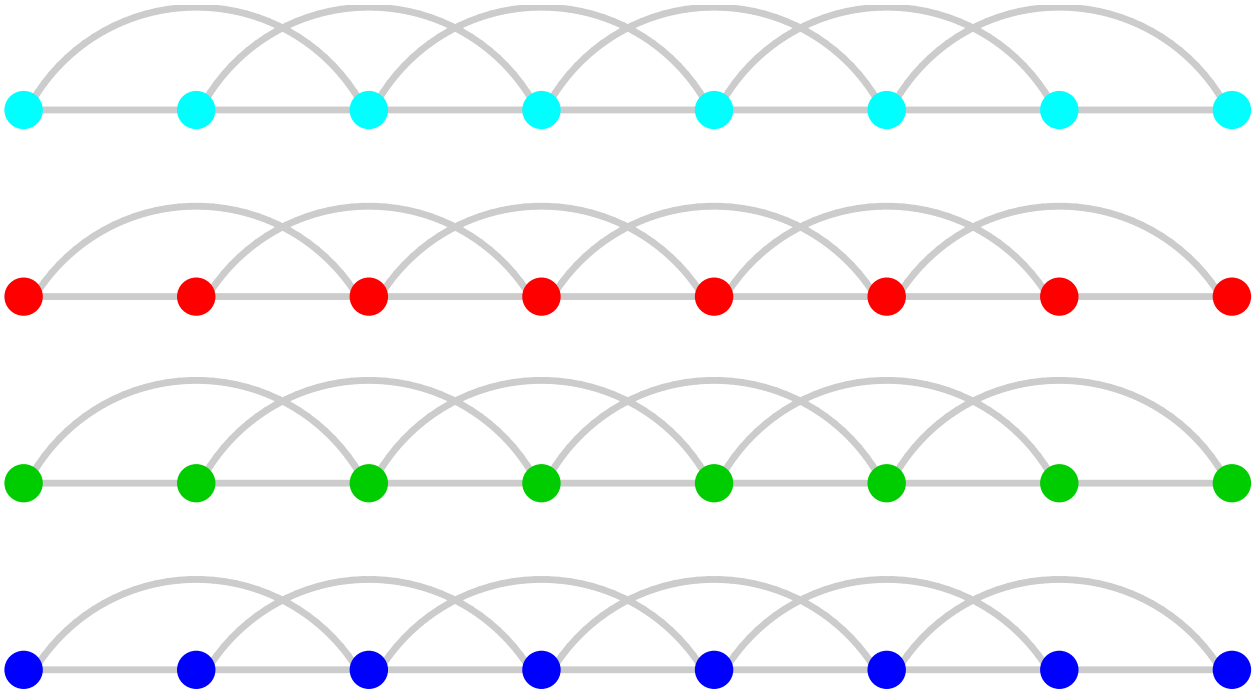
Here, $\mathbf{\Sigma}$ is a $R \times R$ correlation matrix with $R > 1$, $|\rho| \neq 1$:

$$\mathbf{\Sigma} = \begin{pmatrix} 1 & \rho & \cdots & \rho \\ \rho & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \rho \\ \rho & \cdots & \rho & 1 \end{pmatrix}$$

The model specification is analogous to `replicate`, but using the argument `group` instead. The correlation structure is defined within `control.group` in the corresponding `f`-function.

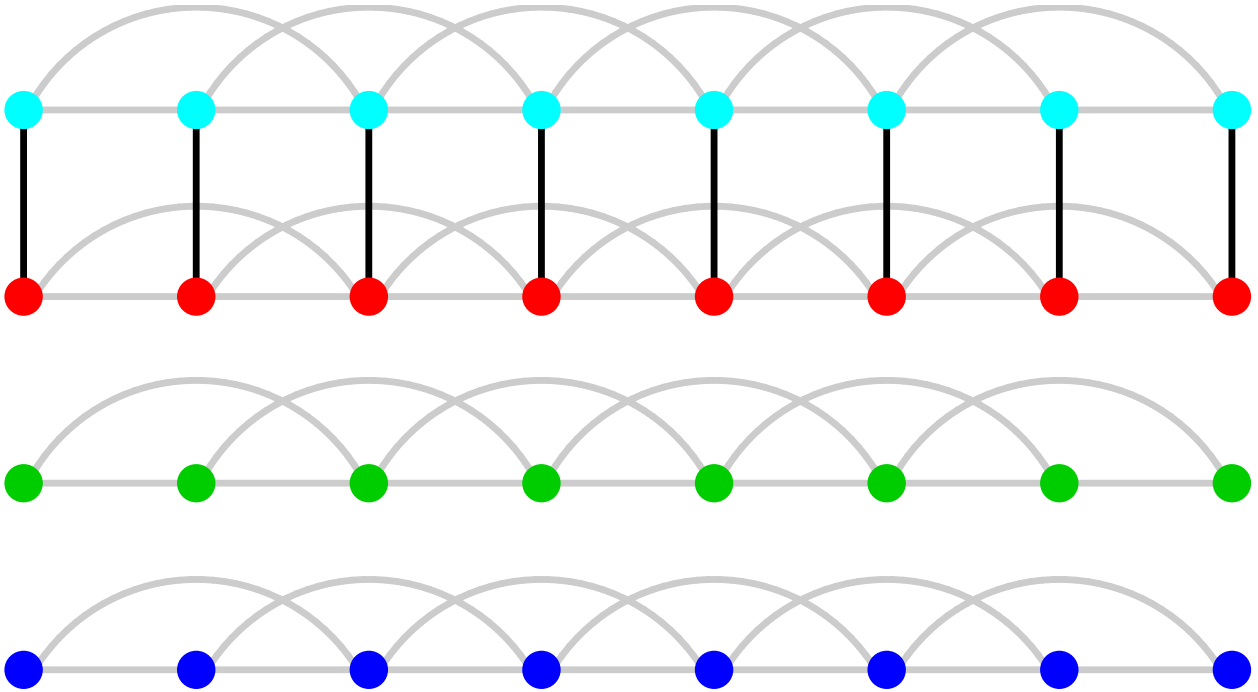
Graphical illustration

Illustration for $R = 4$:



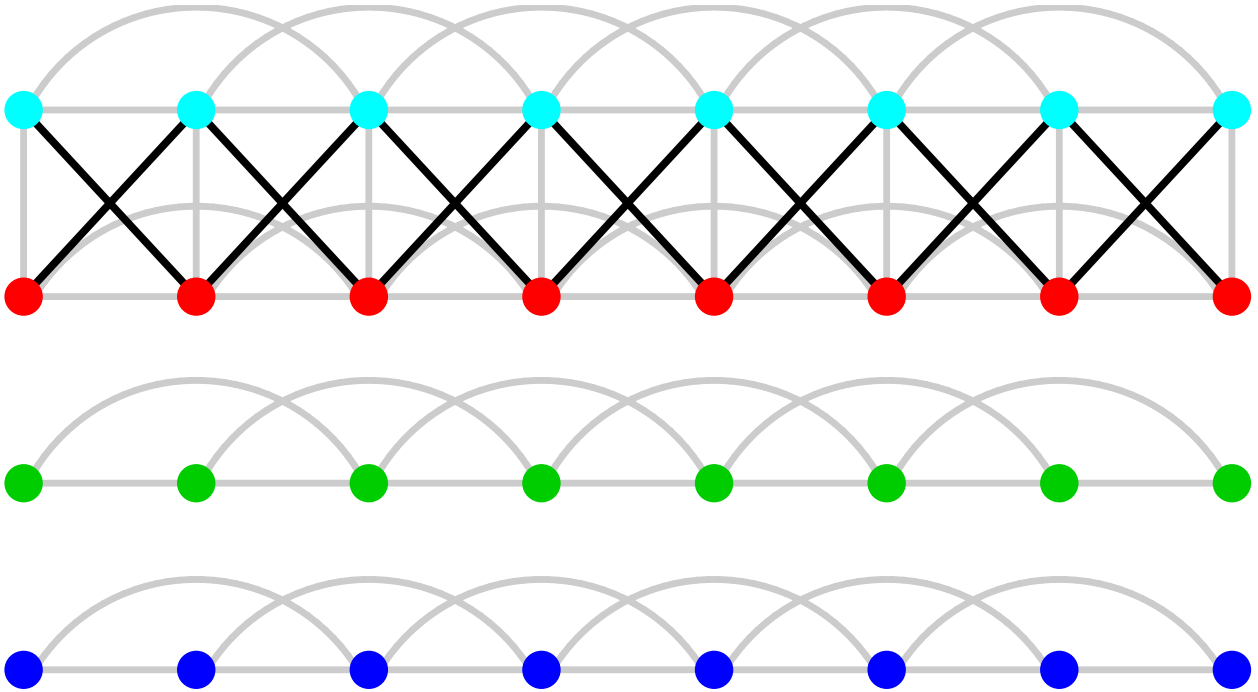
Graphical illustration

Illustration for $R = 4$:



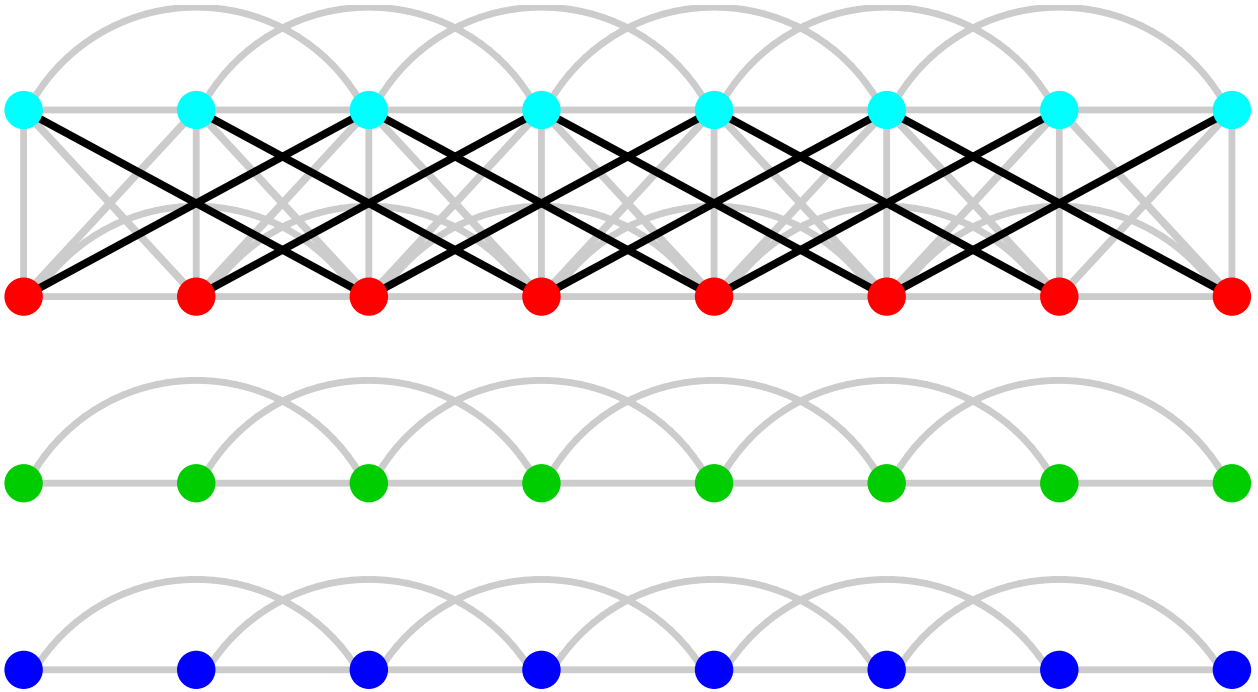
Graphical illustration

Illustration for $R = 4$:



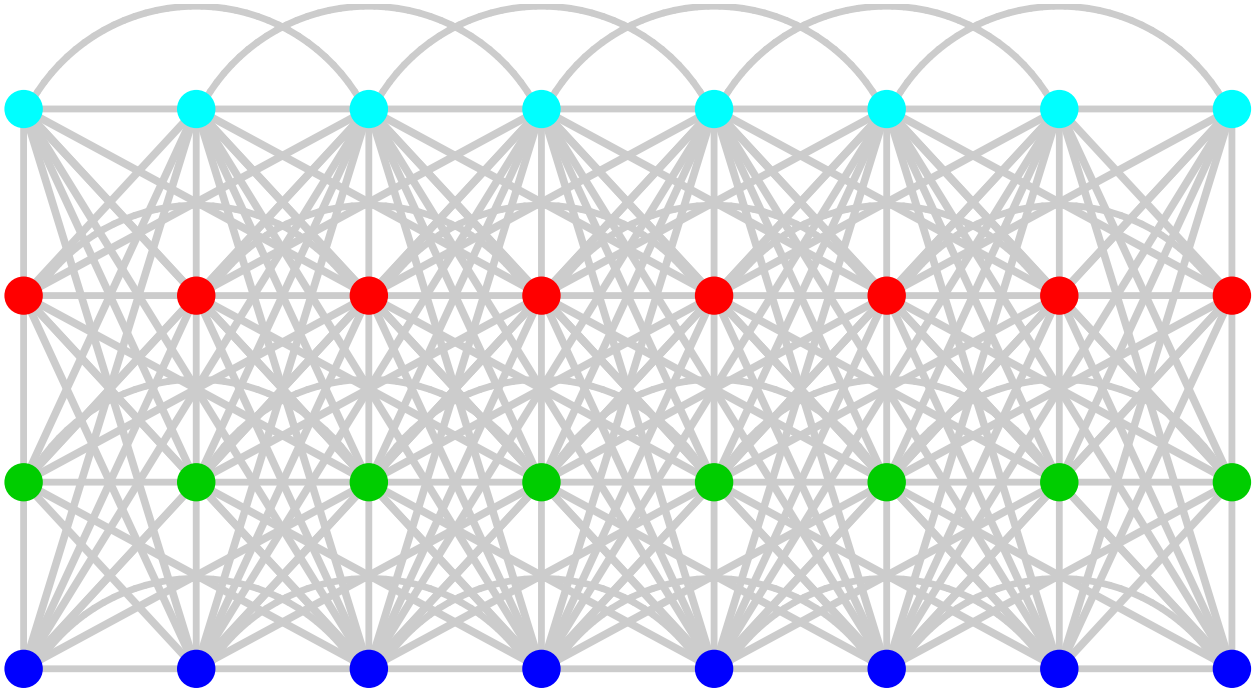
Graphical illustration

Illustration for $R = 4$:



Graphical illustration

Illustration for $R = 4$:



Feature: multiple likelihoods

In many situations, you need to combine data from different sources and need to be able to handle multiple likelihoods.

Examples:

- Joint modelling of longitudinal and event time data (Guo and Carlin, 2004)
- Preferential sampling (Diggle et al, 2010)
- “Marked” point processes
- Animal breeding modelling with multiple traits
- Combining data from multiple experiments

Feature: multiple likelihoods

— We can have different likelihoods, either through different families or the same family with different parameters. E.g. joint modelling of two species (probably) needs different parameters for the distributions of the responses.

— The response `y` is a matrix (or list) where a different "family" is applied to each column

— Each "family" introduces new hyperparameters

Different noise in different groups

```
# Y given by                    # x is given by
          [,1]    [,2]          [1] 1 2 3 4
[1,] 1.00711      NA
[2,] 2.00157      NA
[3,]       NA 3.1127
[4,]       NA 4.0666

# Run model
result = inla(formula = Y ~ 1 + x,
              family = c("gaussian", "gaussian"),
              data = list(Y = Y, x = x))
```

Feature: copy

Allows different elements of the same `f(...)` to be in the the same linear predictor.

Without copy we can not (directly) specify the model

$$\eta_i = u_i + u_{i-1} + \ldots$$

Sometimes this is necessary

Feature: copy

The linear predictor

$$\eta_i = u_i + u_{i-1} + \ldots$$

can be coded as

```
formula = y ~ f(i, model = "iid")
              + f(i.plus, copy="i") + ...
```

— The copy-feature, creates internally an additional sub-model which is ϵ-close to the target

— Many copies allowed, and copies of copies

Feature: copy

It is also possible to include scaled copies

$$\eta_i = u_i + \beta u_{i-1} + \ldots$$

through

```
formula = y ~ f(i, model="iid") +
              f(i.plus, copy="i",
                hyper = list(beta=list(fixed=FALSE)))
              + ...
```

This introduces another hyperparameter in the model.

A-matrix in the linear predictor

Can turn

$$\boldsymbol{\eta} = \ldots$$

into

$$\boldsymbol{\eta}^* = \mathbf{A}\eta$$

with likelihood

$$y_i \sim \pi(y_i|\eta_i^*, \boldsymbol{\theta})$$

```
A = matrix(...)
A = sparseMatrix(...)
result = inla(formula, ...,
              control.predictor = list(A = A))
```

A-matrix in the linear predictor

- Can simplify model formulations
- Achieves to some degree the same as the “copy” feature
- Useful for “P-splines” models and similar “basis-function”-based models

Example

The linear model can be represented by

$$\begin{bmatrix} \eta_1^* \\ \eta_2^* \\ \vdots \\ \eta_n^* \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \text{intercept} \\ \beta \end{bmatrix}.$$

```
1 # Alternative construction
2 A = cbind(rep(1,n), x)
3 x1 = c(1,0); x2 = c(0,1)
4
5 # Run model
6 result = inla(formula = y ~ x1 + x2 - 1,
7                data = list(y = y, x1 = x1, x2 = x2),
8                control.predictor = list(A = A))
```

Feature: Linear combinations

Possible to extract extra information from the model through linear combinations of the latent field, say

$$\boldsymbol{v} = \mathbf{B}\boldsymbol{x}$$

for a $k \times n$ matrix $\mathbf{B}$.

Feature: Linear combinations

Possible to extract extra information from the model through linear combinations of the latent field, say

$$\boldsymbol{v} = \mathbf{B}\boldsymbol{x}$$

for a $k \times n$ matrix $\mathbf{B}$.

Two different approaches:

1. Most "correct" is to do the computations on the enlarged field $\tilde{\boldsymbol{x}} = (\boldsymbol{x}; \boldsymbol{v})$. But this often leads to a more dense precision matrix.
2. The second option is to compute these "offline" (using an approximate version).

Feature: Linear combinations

```
n = 100
x = rnorm(n)
z = rnorm(n)
idx = 1:n
eta = 5 + x + z + rnorm(n)
formula = y ~ 1 + x + z + f(idx, model="iid")
y = rpois(n, lambda = exp(eta))

# Define linear combinations
# Get alpha_x - alpha_z
lc1 = inla.make.lincomb(x=1, z=-1)
names(lc1) = "lc1"

# Get an average over all random effects
lc2 = inla.make.lincomb(idx=rep(1/n, n))
names(lc2) = "lc2"

# Run inla
r = inla(formula, "poisson", data = data.frame(y, x, z, idx),
         lincomb = c(lc1, lc2), control.inla = list(
                   lincomb.derived.correlation.matrix = TRUE))
```

Feature: Linear combinations

Results are saved as

```
> r$summary.lincomb.derived
     ID        mean          sd  0.025quant     0.5quant 0.975quant        mode kld
lc1   1 0.224634269 0.15659950 -0.08306153 0.224559169  0.5324171 0.224423925   0
lc2   2 0.005198819 0.09720661 -0.18594403 0.005198578  0.1961374 0.005206346   0
```

and

- r$marginals.lincomb.derived,
- r$misc$lincomb.derived.correlation.matrix,
- r$misc$lincomb.derived.covariance.matrix.

Values

The `f(...)` models have an argument called `values`

- Not much used
- Defines "time-points" where the model is defined

  `f(x, model = <...>, values = v])`
- Each x_i must be contained in `v`
- by default `v = unique(sort(x))` or `v = 1:n`

Example: RW2

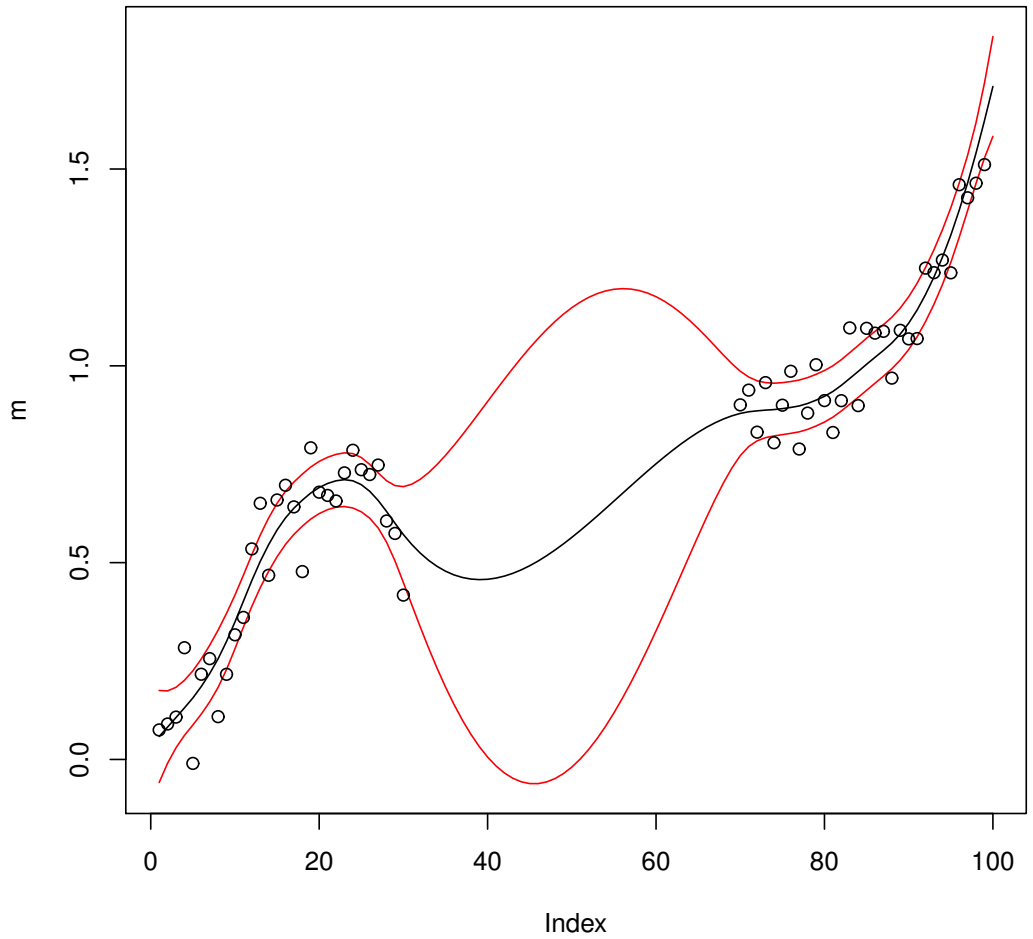
Default behavior for the RW2 model is to include only the specified time steps. If the process is only partially observed, it might be necessary to include unobserved intermediate steps.

```
# Generate data set
y = cumsum(cumsum(rnorm(100, sd=0.01)))+rnorm(100, sd = 0.1)
y = y[c(1:30, 70:100)]

# Make time vector *and* value vector
t = c(1:30, 70:100)
v = 1:100

# Run inla
result = inla(formula = y ~ f(t, model="rw2", values=v, constr=
    FALSE)-1, data = list(y = y, t = t, v = v))
```

Posterior of RW2



Feature: remote computing

Very useful for large models. The R-session runs locally, but the computations are done on a remote (Linux/Mac) server

```
inla(..., inla.call="remote")
```

using ssh.

(Initial set up required, see `inla.remote` and FAQ entry on this issue on r-inla.org)

Example:

```
1 data(Seeds)
2 formula = r ~ x1*x2+f(plate,model="iid")
3 result = inla(formula,data=Seeds,family="binomial",Ntrials=n,
      inla.call="remote")
```

Submit and retrieve jobs

Commands:

```
inla.qget(id, remove = TRUE)
inla.qdel(id)
inla.qstat(id)
inla.qnuke()
```

Example:

```
1 data(Seeds)
2 formula = r ~ x1*x2+f(plate,model="iid")
3 result = inla(formula,data=Seeds,family="binomial",Ntrials=n,
      inla.call="submit")
4
5 inla.qstat()
6
7 result= inla.qget(result, remove=FALSE)
8 # after logging out
9 result= inla.qget(inla.qstat()[[1]]$id, remove=FALSE)
```

Making the INLA program work for you

The R-INLA package works very well for simple problems . . .

- For more complex problems, you may need to help it along.
- Do the parameter estimates make sense?
- Can your model be simplified through algebra (using an **A** matrix etc)?
- Is there a more stable parameterisation?

Tip 1: Set verbose=TRUE

When you are solving a big problem, it's good to set the `verbose` flag to `TRUE`.

- It tells you how big the problem is (nice)
- It keeps track of the optimiser steps (useful!)
- If something breaks, it tells you where (often)

Alternatively, you can look at the logfile

```
1 result$logfile
```

Tip 2: Ask for help

Visit `r-inla.org` and look for similar cases

Private help: Email `help@r-inla.org`

Public help: Public discussion forum at
`r-inla-discussion-group@googlegroups.com`

— Tell us what the problem is (what is in it, how big is it etc)
— Tell us about the error message
— Send us the verbose output

INLA is still an active project

This is an exciting time for the project

- A big push into including more general likelihoods and larger problems
- New way of specifying priors for hyperparameters (PC-priors) (more tomorrow)
- New features for detecting prior sensitivity (coming soon!)
- Extensions beyond the basic latent Gaussian framework
- Packages that extend INLA: AnimalINLA (Anna Marie Holand), ShrinkBayes (Mark van de Wiel), excursions (David Bolin), BAPC (Andrea Riebler), ...