NTNU
Norwegian University of
Science and Technology

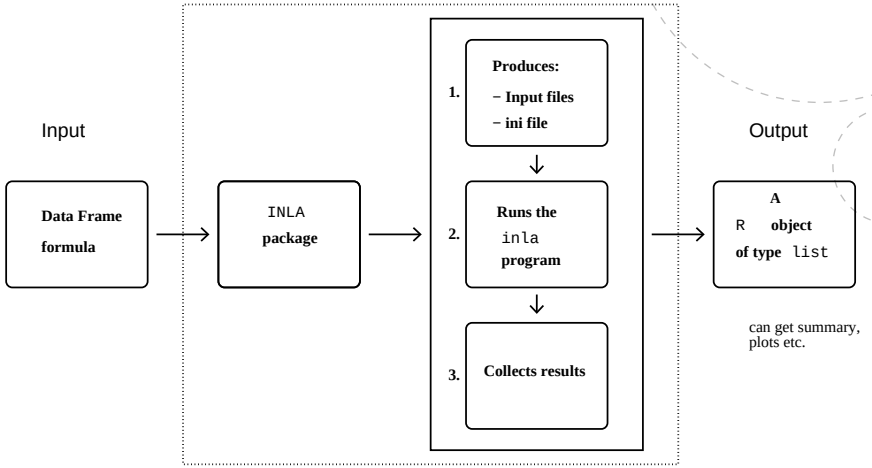
R-INLA: An R-package for INLA

Andrea Riebler <andrea.riebler@math.ntnu.no>

IBS Channel Network Conference 2015
Nijmegen, April 20th, 2015

Outline

The INLA package for R



Getting R-INLA

— The web page www.r-inla.org contains source-code, worked-through examples, reports and instructions for installing the package. INLA tutorial is in preparation.

— The R-package R-INLA works on Linux, Windows and Mac and can be installed by

```
install.packages("INLA",
    repos="http://www.math.ntnu.no/inla/R/testing")
```

Later, it can be upgraded with

```
update.packages(oldPkgs="INLA",
    repos="http://www.math.ntnu.no/inla/R/testing")
```

How to use R-INLA: Ski flying records

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

1. Data organisation: Make an object to store response, covariates, ...

   ```
   data = list(y = y, x = x)
   ```

2. Use the formula-notation to specify the model (similar to lm and glm functions)

   ```
   formula = y~x
   ```

3. Call the inla-program

   ```
   res = inla(formula, data=data, family="gaussian")
   ```

4. Extract posterior information, e.g. for a first overview use

   ```
   summary(res)
   ```


Example: Summary

```
Call:
"inla(formula = formula, family = \"gaussian\", data = data)"

Time used:
 Pre-processing    Running inla Post-processing           Total
         0.0581          0.0161          0.0181          0.0924

Fixed effects:
                mean     sd 0.025quant 0.5quant 0.975quant     mode kld
(Intercept) 137.0288 1.3929   134.2798 137.0288   139.7741 137.0288   0
x             2.1259 0.0526     2.0221   2.1259     2.2295   2.1259   0
...
```


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

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


Posterior inference

Main functions:

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

Example: Simple linear regression

. . . such as our ski flying example.

Stage 1: Gaussian likelihood

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

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

```r
# Generate data
x = sort(runif(100))
y = 1 + 2*x + rnorm(n = 100, sd = 0.1)

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

# Get summary
summary(result)
```


summary(result)

```
Call:
c("inla(formula = formula, family = \"gaussian\", data = list(x = x, ",  "    y = y))")

Time used:
 Pre-processing    Running inla Post-processing           Total
         0.0571          0.0188          0.0166          0.0925

Fixed effects:
             mean     sd 0.025quant 0.5quant 0.975quant   mode kld
(Intercept) 1.0204 0.0201     0.9808   1.0204     1.0599 1.0204   0
x           1.9818 0.0328     1.9173   1.9818     2.0462 1.9818   0

The model has no random effects

Model hyperparameters:
                                          mean    sd 0.025quant 0.5quant
Precision for the Gaussian observations 111.15 15.75      82.88   110.29
                                        0.975quant   mode
Precision for the Gaussian observations     144.53 108.78

Expected number of effective parameters(std dev): 2.183(0.0206)
Number of equivalent replicates : 45.82

Marginal log-Likelihood:  76.74
```

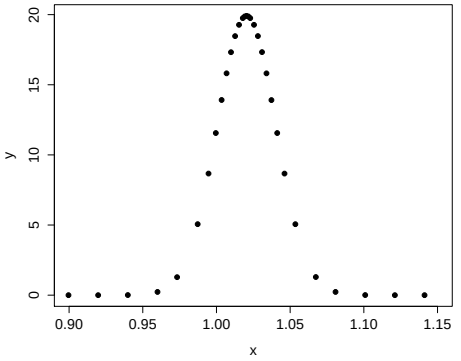

result$summary.fixed

```
                mean         sd 0.025quant 0.5quant 0.975quant     mode
(Intercept) 1.020390 0.02013356  0.9808057 1.020389   1.059935 1.020390
x           1.981786 0.03277825  1.9173423 1.981786   2.046167 1.981787
                     kld
(Intercept) 1.095541e-12
x           9.299953e-13
```


Marginal posterior densities

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

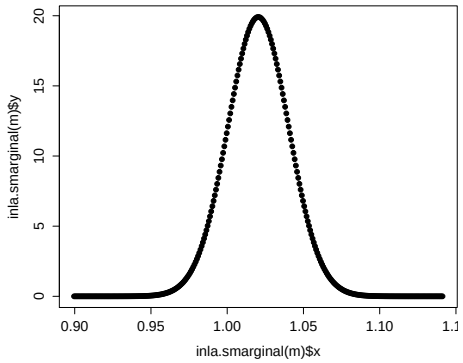
```r
m = result$marginals.fixed[[1]]
plot(m)
```



Marginal posterior densities

The rough shape can be interpolated to higher resolution

```
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
```


Organisation of the returned `inla`-object

```
> names(result)
 [1] "names.fixed"                 "summary.fixed"
 [3] "marginals.fixed"             "summary.lincomb"
 [5] "marginals.lincomb"           "size.lincomb"
 [7] "summary.lincomb.derived"     "marginals.lincomb.derived"
 [9] "size.lincomb.derived"        "mlik"
[11] "cpo"                         "po"
[13] "waic"                        "model.random"
[15] "summary.random"              "marginals.random"
[17] "size.random"                 "summary.linear.predictor"
[19] "marginals.linear.predictor"  "summary.fitted.values"
[21] "marginals.fitted.values"     "size.linear.predictor"
[23] "summary.hyperpar"            "marginals.hyperpar"
...
```

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 (each f-function needs its own!)
- `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

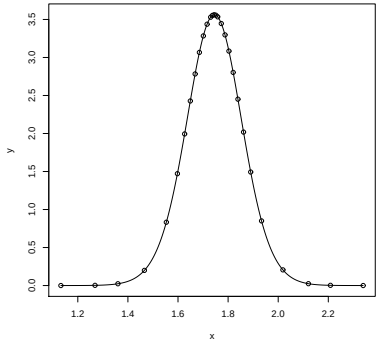
The distribution of the linear predictor at an unobserved location can be computed by specifying the value of the covariate *x* and the desired time index *t* and set *y* to NA.

```
# 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))
```



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"     "generic3"     "spde"         "spde2"
[26] "spde3"        "iid1d"        "iid2d"        "iid3d"        "iid4d"
[31] "iid5d"        "2diid"        "z"            "rw2d"         "rw2diid"
[36] "slm"          "matern2d"     "copy"         "clinear"      "sigm"
[41] "revsigm"
```


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: Code

```
1 hyper = list(prec = list(prior = "loggamma",
2                           param = c(1, 0.1),
3                           initial = 4,
4                           fixed = FALSE))
5
6 formula = y ~ f(idx, model = "iid", hyper = hyper) + ...
```

Changing the prior: Default options

```
# For the iid model, default options can be seen with
inla.doc("iid")
# or
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"

You can get information about the paramterisation of the loggamma prior, say, using

```
inla.doc("loggamma")
```

It is also possible to use your own prior (on internal scale) with

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

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

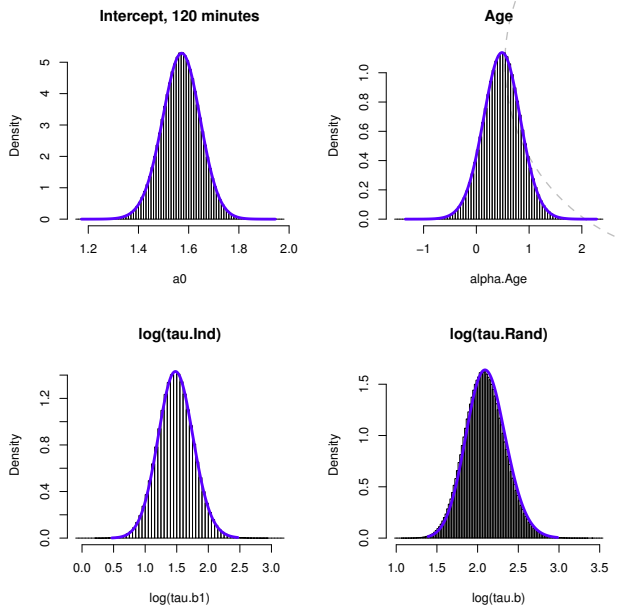
```
> data(Epil)
> head(Epil,n=3)
   y Trt Base Age V4 rand Ind       CTrt      ClBase4   CV4       ClAge
1  5   0   11  31  0    1   1 -0.5254237 -0.75635379 -0.25  0.11420370
2  3   0   11  31  0    2   1 -0.5254237 -0.75635379 -0.25  0.11420370
3  3   0   11  31  0    3   1 -0.5254237 -0.75635379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

```
> 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. That means, same data, same priors, same constraints on parameters, intercept included or not, ...
- Here we have compared the results with those obtained using JAGS via the rjags package



Running time of INLA $<$ 0.5 seconds

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.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
— Remote computing

Feature: replicate

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

If $\boldsymbol{x} \mid \theta \sim$ 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 θ
Arguments

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

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

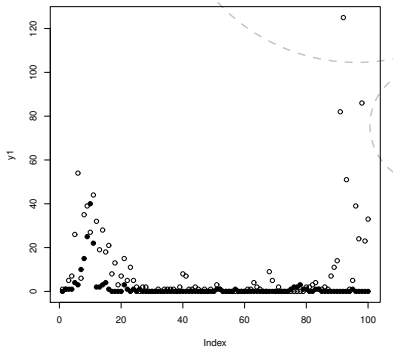
Example: replicate

```
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)
```

```
## 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
```

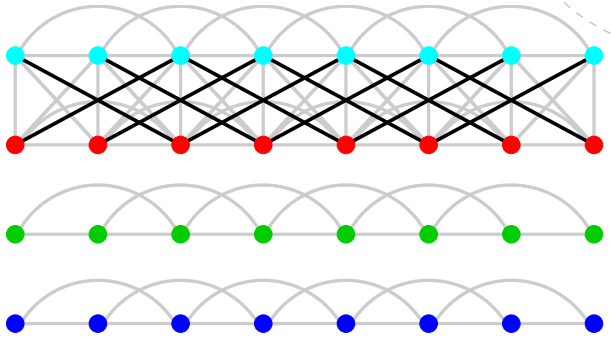
```
formula = y ~ f(i, model="ar1", replicate=r) + intercept -1
result = inla(formula, family = "poisson",
              data = data.frame(y, i, r, intercept))
```


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 RW2

Illustration for $R = 4$:



(For example and code, see Riebler, A., Held, L., & Rue, H. (2012). The Annals of Applied Statistics, 6(1), 304-333.)

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)
- "Marked" point processes
- 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

$$\eta_i = u_i + u_{i+1} + \dots$$

Sometimes this is necessary

Feature: copy

The linear predictor

$$\eta_i = u_i + u_{i+1} + \dots$$

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

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

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.

Feature: A-matrix in the linear predictor

Can turn

$$\boldsymbol{\eta} = \dots$$

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

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). This is the default.

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`.

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:

```
data(Seeds)
formula = r ~ x1*x2+f(plate,model="iid")
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:

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

inla.qstat()

result= inla.qget(result, remove=FALSE)
# after logging out
result= inla.qget(inla.qstat()[[1]]$id, remove=FALSE)
```

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

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)
  (Simpson, et al. (2014), arXiv:1403.4630)
- New features for detecting prior sensitivity (coming soon!)

  (Roos, et al. (2013). arXiv:1312.4797)
- 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), BayesianPspline (Cajo ter Braak) ...

Thank you for your attention!

If you have any doubts or questions, please write me:
`andrea.riebler@math.ntnu.no`