User defined integration points

Haavard Rue (hrue@r-inla.org)

May 29th 2017 (updated April 9th 2019)

Introduction

This short note describe a new option that allow the user to use user-defined integration points (or "design" points), instead of the default ones. The relevant integration in INLA does

$$\int f(x|\theta, y)\ \pi(\theta|y)\ d\theta = f(x|y)$$

where $\pi(\theta|y)$ is the approximated posterior marginal for the hyperparameters, and where $f(x|\theta, y)$ is the approximated marginal for x for that configuration. The output of this integral is the posterior marginal $f(x|y)$. In practice, we use a discrete set of integration points for θ, and corresponding weights w, to get

$$f(x|y) \approx \sum_i f(x|\theta_i, y) w_i \pi(\theta_i|y)$$

for which we require $w_i \geq 0$ and $\sum_i w_i = 1$. Usually, the integration is done in a *standardised scale*,

$$z = A(\theta - \gamma)$$

i.e. with respect to $\pi(z|y)$. Here, γ is the mode of $\pi(\theta|y)$ and the matrix A is the negative square root of the approximated covariance matrix for $\theta|y$ at the mode.

The relevant options are

```
opts = control.inla(int.strategy = "user", int.design = Design)
```

where `Design` is a matrix with the integration points and the integration weights. The jth row of `Design` consists of the values $\theta_j = (\theta_{1j}, \ldots, \theta_{mj})$, and the integration weight for this configuration, w_j. The values are in the θ-scale, meaning that you have to know exactly what you are doing, including knowing the ordering of the hyperparameters.

Another version, is to define the points in the standardised scale z. To do this, use

```
opts = control.inla(int.strategy = "user.std", int.design = Design)
```

instead. The meaning of `Design` is unchanged, except that these can be given in standardised coordinates. This version is more relevant if you want to implement a generic new integration design instead of the ones already provided.

First example

In this artificial example, we want to compute the change of the marginal variance of one component, x_1, of a hidden AR(1) process, with respect to lag one correlation ρ. So we want to compute

$$\frac{\partial \text{SD}(x_1|y)}{\partial \rho}$$

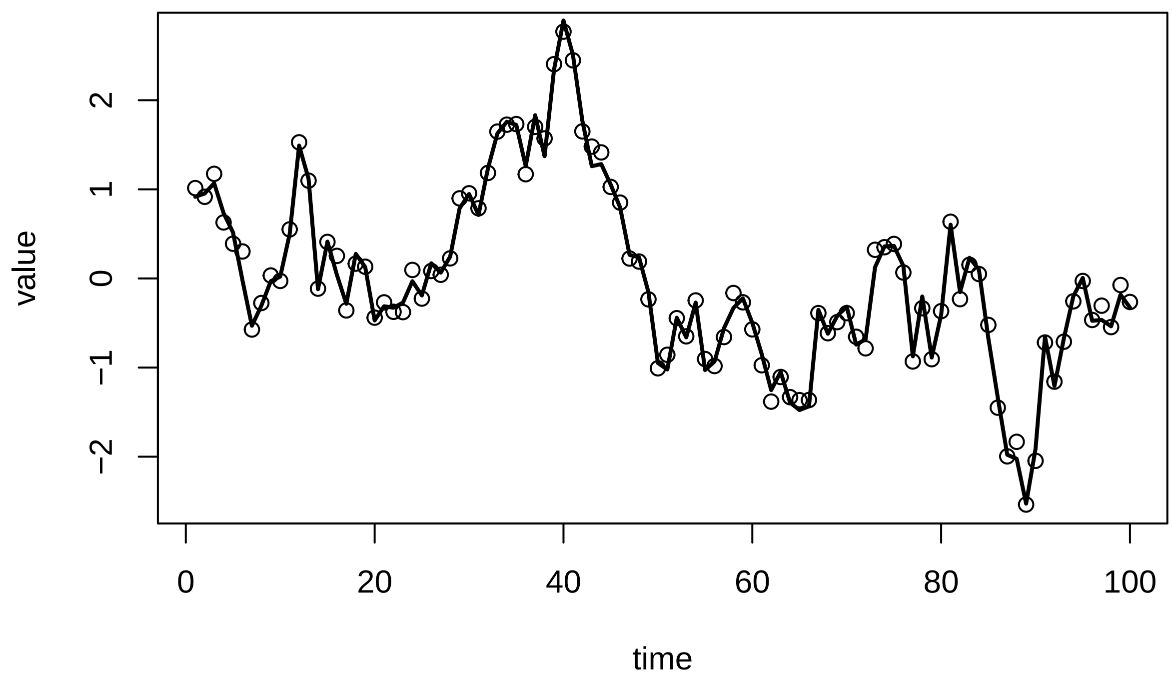
for a fixed value of $\rho = \rho_0$. We have to compute a numerical approximation, using finite difference. While doing this, it is a good idea to keep the design fixed, to avoid introducing an error for changing that part as well.

Let us first setup the experiment

```
n = 100
rho = 0.9
x = scale(arima.sim(n, model = list(ar = rho)))
y = x + rnorm(n, sd = 0.1)
```

this gives the following

```
plot(y, xlab = "time", ylab = "value")
lines(x, lwd=2)
```



To compute the derivative, we do

```
rho.0 = rho
to.theta = inla.models()$latent$ar1$hyper$theta2$to.theta
rho.0.internal = to.theta(rho.0)

r = inla(y ~ -1 + f(time, model="ar1",
    hyper = list(
        theta1 = list(prior = "loggamma",
                      param = c(1,1)),
        theta2 = list(initial = rho.0.internal,
                      fixed=TRUE))),
    control.inla = list(int.strategy = "grid"),
```

```
    data = data.frame(y, time = 1:n))

sd.0 = r$summary.random$time[1,"sd"]
print(sd.0)
```

```
## [1] 0.01625493
```

The ordering of the hyperparamters are as follows,

```
nm = names(r$joint.hyper)
nm = nm[-length(nm)]
print(nm)
```

```
## [1] "Log precision for the Gaussian observations"
## [2] "Log precision for time"
## [3] "Log posterior density"
```

which may sometimes be useful to know about.

Anyway, we will now change ρ a little, while we keep the same integration points,

```
Design = as.matrix(cbind(r$joint.hyper[, seq_along(nm)], 1))
head(Design)
```

```
##      Log precision for the Gaussian observations Log precision for time
## [1,]                                    9.861648             -0.1667940
## [2,]                                    9.010025             -0.1709544
## [3,]                                   10.713271             -0.1626336
## [4,]                                    9.862160             -0.2716433
## [5,]                                    9.861136             -0.0619447
## [6,]                                    9.010537             -0.2758037
##      Log posterior density 1
## [1,]             -72.06833 1
## [2,]             -72.36963 1
## [3,]             -72.50610 1
## [4,]             -72.33779 1
## [5,]             -72.35966 1
## [6,]             -72.66255 1
```

where the last column is the (un-normalised) integration weights. `Design` has dimension 75, 4. We call `inla()` again reusing the previous found mode

```
h.rho = 0.01
rho.1.internal = to.theta(rho.0 + h.rho)
rr = inla(y ~ -1 + f(time, model="ar1",
    hyper = list(
        theta1 = list(prior = "loggamma",
                      param = c(1,1)),
        theta2 = list(initial = rho.1.internal,
                      fixed=TRUE))),
    control.mode = list(result = r, restart=FALSE),
    data = data.frame(y, time = 1:n),
    control.inla = list(
        int.strategy = "user",
        int.design = Design))
sd.1 = rr$summary.random$time[1,"sd"]
print(sd.1)
```

```
## [1] 0.01430746
```

and then our estimate of the derivative is

```
deriv.1 = (sd.1 - sd.0) / h.rho
print(deriv.1)
```

```
## [1] -0.1947477
```

PS: In the logfile of the `inla()`-call, the configurations are shown in the z scale even for `int.strategy="user"`.

Second example

There is also another (experimental) option, that is

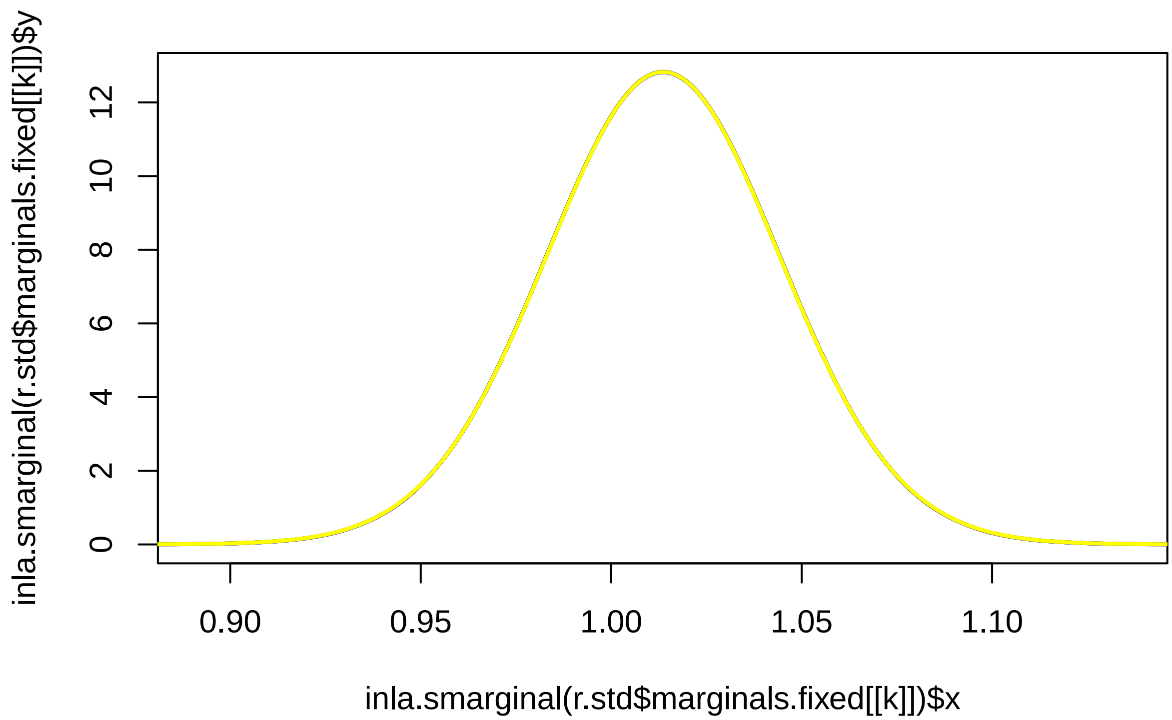
```
control.inla = list(int.stategy = "user.expert")
```

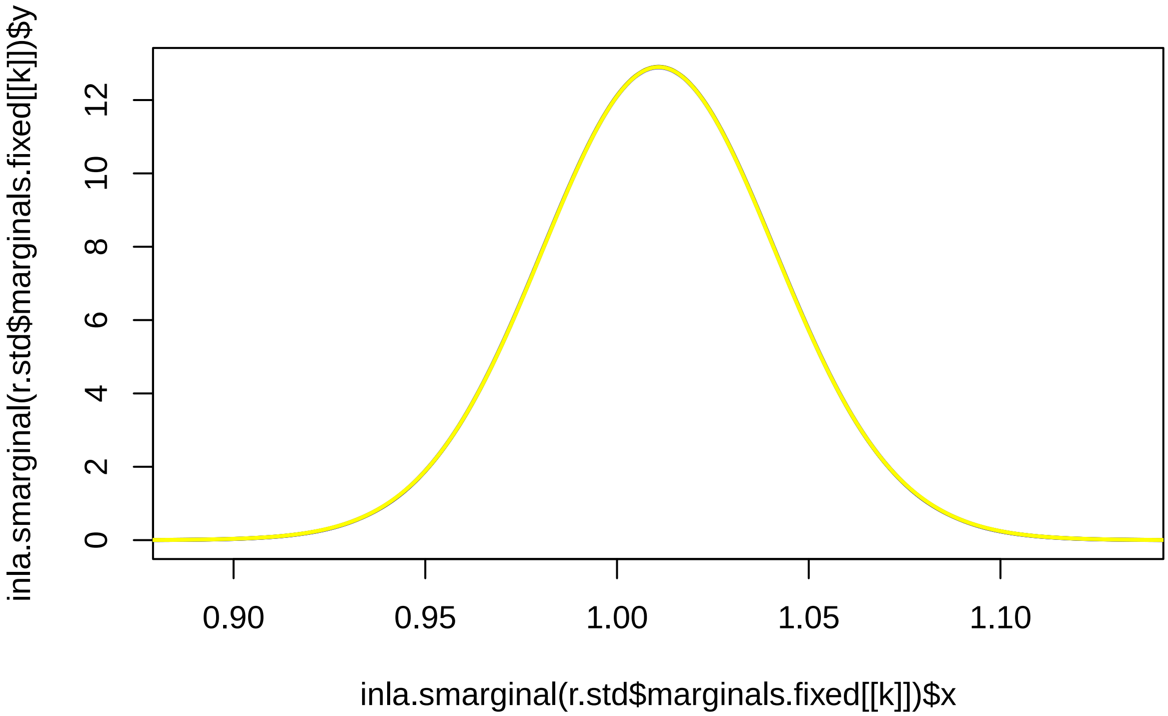
for which the weights **includes** $\pi(\theta_i|y)$. The following example show how to use it, fitting the same model in three different ways.

```
n = 50
x = rnorm(n)
y = 1 + x + rnorm(n, sd = 0.2)
param = c(1, 0.04)
dz = 0.1
r.std = inla(y ~ 1 + x, data = data.frame(y, x),
             control.inla = list(int.strategy = "grid",
                                 dz = dz,
                                 diff.logdens = 8),
             control.family = list(
                 hyper = list(
                     prec = list(
                         prior = "loggamma",
                         param = param))))

s = r.std$internal.summary.hyperpar[1,"sd"]
m = r.std$internal.summary.hyperpar[1,"mean"]
theta = m + s*seq(-4, 4, by = dz)
weight = dnorm(theta,  mean = m, sd = s)

r = rep(list(list()), length(theta))
for(k in seq_along(r)) {
    r[[k]] = inla(y ~ 1 + x,
                  control.family = list(
                      hyper = list(
                          prec = list(
                              initial = theta[k],
                              fixed=TRUE))),
                  data = data.frame(y, x))
}
r.merge = inla.merge(r, prob = weight)
```

```
## Warning in inla.merge(r, prob = weight): This function is experimental.

## Warning in inla.merge(r, prob = weight): Merging 'cpo' and 'pit'-results are/can
## be, approximate only
```

```r
r.design = inla(y ~ 1 + x,
                data = data.frame(y, x),
                control.family = list(
                    hyper = list(
                        prec = list(
                            ## the prior here does not really matter, as we will override
                            ## it with the user.expert in any case.
                            prior = "pc.prec",
                            param = c(1, 0.01)))),
                control.inla = list(int.strategy = "user.expert",
                                    int.design = cbind(theta, weight)))
for(k in 1:2) {
    plot(inla.smarginal(r.std$marginals.fixed[[k]]),
         lwd = 2, lty = 1, type = "l",
         xlim = inla.qmarginal(c(0.0001, 0.9999), r.std$marginals.fixed[[k]]))
    lines(inla.smarginal(r.design$marginals.fixed[[k]]),
          lwd = 2, col = "blue", lty = 1)
    lines(inla.smarginal(r.merge$marginals.fixed[[k]]),
          lwd = 2, col = "yellow", lty = 1)
}
```





inla.smarginal(r.std$marginals.fixed[[k]])$y
0
2
4
6
8
10
12
0.90
0.95
1.00
1.05
1.10
inla.smarginal(r.std$marginals.fixed[[k]])$x