

“Expression”: a do-it-yourself prior

This prior allow the user to define an expression for the log-density of any (univariate) prior $\log \pi(\theta)$, as a function of the corresponding θ (which is in the internal scale; be aware).

The expression is evaluated using the `muparser`-library¹, with some local configuration changes to make it more “R”-like in style. The format is

```
expression: <statement>; <statement>; ...; return(<value>)
```

where “<statement>” is any regular statement (more below) and value returned, “<value>” is the value for the log-density of the prior, evaluated at the current value for θ .

The following expression implements the normal prior (in not a good way...)

```
expression:
  mean = 0; sigma = 1;
  dens = 1/sqrt(2*pi) * 1/sigma * exp(-0.5*(x-mean)^2/sigma^2);
  logdens = log(dens);
  return(logdens)
```

All variables in the expression are initialised with the current value of θ before the expression is evaluated. In this way, the variable `x` in this example will be θ .

Notes

1. `return (x)` (with a space before “`(.)`”) is NOT allowed, it must be `return(x)`.
2. A “`;`” is needed to terminate each expression, a newline DOES NOT terminate an expression.
3. You can use “`_`” in variable-names, like `log_precision = <whatever>;` see the following example.

Known functions

Known functions (besides common math-functions like “`exp(.)`”, “`sin(.)`”, etc...) are

- `gamma(x)` is the Gamma-function and `lgamma(x)` is its log, and the derivatives `digamma(x)` and `trigamma(x)` (see `?gamma` in R).
- `pi` is π
- x^y is expressed as either `x^y` or `pow(x;y)`

Example

```
y = rnorm(1)

## using buildt-in prior
a = 1
b = 0.1
hyper = list(prec = list(prior = "loggamma", param = c(a, b)))
r = inla(y ~ 1, data = data.frame(y),
        control.family = list(hyper = hyper))

## implementing the loggamma-prior using "expression:"
```

¹See <http://muparser.sourceforge.net/> for more documentation

```

loggamma = "expression:
  a = 1;
  b = 0.1;
  precision = exp(log_precision);
  logdens = log(b^a) - lgamma(a)
            + (a-1)*log_precision - b*precision;
  log_jacobian = log_precision;
  return(logdens + log_jacobian);"

hyper.new = list(prec = list(prior = loggamma))
r.new = inla(y ~ 1, data = data.frame(y),
  control.family = list(hyper = hyper.new))

## and we verify that we get the same result...
print(r$summary.hyperpar[1,"mean"])
print(r.new$summary.hyperpar[1,"mean"])

```