“Expression”: a do-it-yourself prior

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

The expression is evaluated using the `muparser`[1], 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>`” is the value for the log-density of the prior, evaluated at the “free parameter(s)”.

The following expression implements the normal prior

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

Since `x` is a variable that is not assigned any value (its a “free parameter”), it must the argument (i.e. θ) to this function. If there are more than one free parameter, then all of them are assigned to θ before the expression is evaluated.

Note

1. `return (x)` (with a space before “(.)”) is not allowed, it must be `return(x)`.
2. You need a “;” to terminate each expression, a newline DOES NOT terminate an expression!
3. You can use a “`_`” in variable-names, like `log_precision = ...`; 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 (see `?gamma` in `R`).
- `pi` is π
- x^y is expressed as `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.data = list(hyper = hyper))

## implementing the loggamma-prior using "expression:"
loggamma = "expression:
            a = 1;
```

[1]See `http://muparser.sourceforge.net/` for more documentation

```
            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.data = 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"])
```

Notes

None.