Bayesian linear regression with the blm package

Jasper Ginn

The `blm` package contains a simple Gibbs and Metropolis-Hastings sampler to run a Bayesian Linear Regression model. It is written partly in R and partly in Julia. The bridge between these two languages is handled by the JuliaCall library. Julia is a high-performing and high-level language. In the case of an MCMC sampler, we repeatedly sample from conditional posterior distributions, which amplifies the need for speed and efficiency.

In this document, I only focus on the mechanics of the functions and not the interpretation of results.

Setting up blm

To set up the blm package, we load in in R

```
library(blm)
```

This also loads the magrittr package, which is helpful because we can chain commands using the forward-operating pipe command (`%>%`).

Next, we load the Julia environment

```
blm_setup()
```

This can take a couple of seconds.

The data

The data we will use is included in the `blm` library, and contains compensation data on 336 directors in 52 companies and 3 sectors.

```
library(table1)
data("directors")
dir2 <- directors
```

```
# Add labels
label(dir2$Compensation) <- "Compensation ('000 GBR)"
label(dir2$Age) <- "Age (years)"
label(dir2$Male) <- "Gender (male)"
label(dir2$Sector) <- "Sector the company belongs to"
table1(Compensation ~ Age + Male + Sector, data=dir2)
```

	Overall (n=336)
Compensation ('000 GBR)	
Mean (SD)	174 (225)
Median [Min, Max]	151 [8.00, 4000]
Age (years)	
Mean (SD)	64.4 (6.83)
Median [Min, Max]	65.0 [43.0, 85.0]
Gender (male)	
0	60 (17.9%)
1	276 (82.1%)
Sector the company belongs to	
Basic Materials	74 (22.0%)
Financial	123 (36.6%)
Services	139 (41.4%)

From the summary statistics, we gather that the outcome variable is skewed, which is why we will use log (compensation) rather than the untransformed variable. We also center the Age predictor and Gender predictor[1]

```
library(dplyr)
directors <- directors %>%
  mutate(Compensation = log(Compensation),
         Age = Age - mean(Age),
         Male = as.numeric(Male),
         Male = Male - mean(Male))
```

For comparison purposes, we run a linear regression (lm) model using Maximum Likelihood

[1]Even though Male is a categorical variable, centering it reduces autocorrelation.

```r
# Linear model for comparison
ffit <- lm("Compensation ~ Age + Male", data=directors)
summary(ffit)
```

```
##
## Call:
## lm(formula = "Compensation ~ Age + Male", data = directors)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7791 -0.2544  0.0242  0.2583  3.2690
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.991156   0.028107 177.576   <2e-16 ***
## Age         0.008434   0.004315   1.954   0.0515 .
## Male        0.065336   0.076832   0.850   0.3957
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5152 on 333 degrees of freedom
## Multiple R-squared:  0.01787,    Adjusted R-squared:  0.01197
## F-statistic: 3.029 on 2 and 333 DF,  p-value: 0.04969
```

Running a blm model

To fit a blm model, we call the `blm()` function, which is similar to the `lm()` function.

```r
bfit <- blm("Compensation ~ Age + Male",
            data=directors)
```

At this point, we can call `print()` and `summary()` on the data to check the model status.

```r
print(bfit)
```

```
## Bayesian Linear Model (BLM) object:
##
## Data:
##  Predictors: 2
##  Outcome: Compensation
##
## Sampler:
##  Chains: 1
##  Iterations: 10000
##  Thinning: 1
##  Burn: 1000
##
## Priors (Coefficients) :
##       b0   b1   b2
## mu     0    0    0
## tau 1000 1000 1000
##
## Priors (Residuals) :
##       sigma
## rate   0.01
## scale  0.01
```

This gives us information about the data, the sampling options and the priors, which are uninformative at this point.

```r
summary(bfit)
```

```
## Model results for blm object:
##
## Formula: 'Compensation ~ Age + Male'
##
```

```
## Sampled: FALSE
##
## Sampling settings :
##  Obs. Predictors Chains Iterations Thinning Burn
##   336          2      1      10000        1 1000
```

`summary()` gives us information about the model fit, MAP estimates and so on. Given that we have not sampled the posterior, however, these statistics are not yet supplied.

We can update sampling settings as follows

```
bfit <- bfit %>%
  # Update sampling settings
  set_sampling_options(., chains = 3, iterations = 15000, thinning = 2, burn = 2000)
```

Priors may be made informative as follows

```
bfit <- bfit %>%
  # Add priors (we're on a log scale)
  set_prior("b1", mu=.01, sd=.2) %>% # 1 % increase / every year ==> 20% spread
  set_prior("b2", mu=.05, sd=.03)
print(bfit)
```

```
## Bayesian Linear Model (BLM) object:
##
## Data:
##  Predictors: 2
##  Outcome: Compensation
##
## Sampler:
##  Chains: 3
##  Iterations: 15000
##  Thinning: 2
##  Burn: 2000
##
```

```
## Priors (Coefficients) :
##       b0   b1   b2
## mu     0 0.01 0.05
## tau 1000 0.20 0.03
##
## Priors (Residuals) :
##       sigma
## rate   0.01
## scale  0.01
```

To add informative hypotheses to the model, we can use `set_hypothesis()`

```
bfit <- bfit %>%
  # H1: Males earn about the same as females
  set_hypothesis("b0 + b2 < b0") %>%
  # H2: Directors only earn more as they get older
  set_hypothesis("b1 > 0")
print(bfit)
```

```
## Bayesian Linear Model (BLM) object:
##
## Data:
##  Predictors: 2
##  Outcome: Compensation
##
## Sampler:
##  Chains: 3
##  Iterations: 15000
##  Thinning: 2
##  Burn: 2000
##
## Priors (Coefficients) :
##       b0   b1   b2
## mu     0 0.01 0.05
```

```
## tau 1000 0.20 0.03
##
## Priors (Residuals) :
##       sigma
## rate   0.01
## scale  0.01
##
## Bayesian Linear Model (BLM) hypothesis:
##
## Hypothesis: b0 + b2 < b0
## Elements: 1
##
## Bayesian Linear Model (BLM) hypothesis:
##
## Hypothesis: b1 > 0
## Elements: 1
```

If we do not specify initial values, then `blm` will draw these from the informative or uninformative priors. Given that we are on a log scale, it would be good to set these close to 0.

```
bfit <- bfit %>%
  # Set initial values
  # If we draw from the priors the starting values will be too large
  # This is an issue for MH because it takes many more iterations to converge
  set_initial_values(chain_1 = list("b" = c(7, -5, 0), "sigma" = 1),
                     chain_2 = list("b" = c(-5, 0, 7), "sigma" = 2))
```

If you want to change the sampler (i.e. use Metropolis-Hastings instead of a Gibbs sampler), then you can use `set_sampler()`. However, this is not recommended as you will only lose efficiency. The MH algorithm was implemented for didactical reasons.

```
bfit <- bfit %>%
  # Change the sampler of Age to metropolis hastings (random walk)
  # Lambda parameter controls the variance of the (normal) proposal distribution
```

```
  set_sampler("b1", type="MH", lambda=0.01)
```

If you want to compute the intercept-only model next to the model you are interested in, you can add the line `compute_null_model()`:

```
bfit <- bfit %>%
  # This has the same parameters as set_sampling_options()
  compute_null_model()
```

You can always use `get_parameter_names()` to obtain a mapping of variables to parameters:

```
bfit %>%
  get_parameter_names()
```

```
## Parameter / variable names for blm object:
##
## Mapping :
##
## Intercept b0
## Age       b1
## Male      b2
```

Sampling from the posterior

If we are happy with these settings, we can sample from the posterior distribution. This process takes the longest during the first time you call the `sample_posterior()` function. This happens because Julia's Just-In-Time (JIT) compiler compiles the code when it is called the first time. In subsequent evaluations of the code, it then uses the already compiled code. To carry out the `blm` sampling plan, you execute:

```
bfit <- bfit %>%
  # Sample the posterior
  sample_posterior()
```

This has several effects:

1. Your sampling plan is now locked, and you can only change the burn-in value. You can delete the posterior as follows:

```
bfit <- bfit %>%
  # Remove posterior samples
  delete_posterior()
```

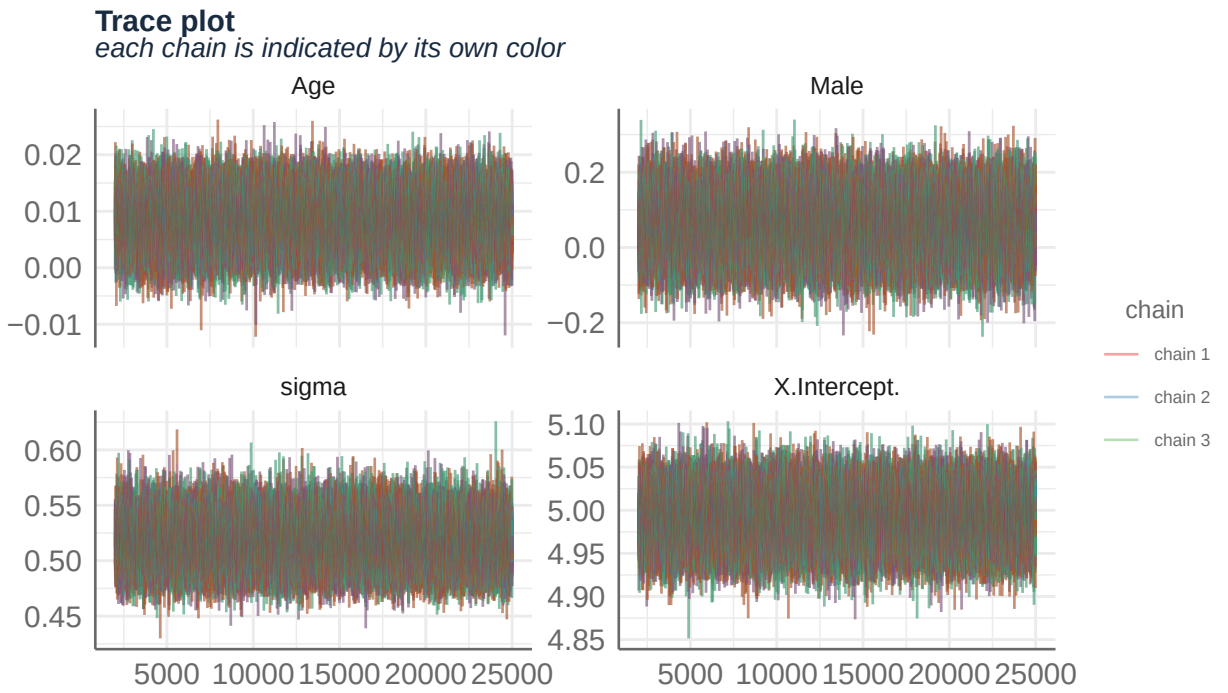
2. The model DIC, hypotheses, intercept-only model, model Bayes Factor and R-squared values are calculated automatically.
3. If you want to draw more samples under the sampling plan, you can do so as follows:

```
bfit <- bfit %>%
  # Draw more samples
  update_posterior(iterations=10000)
```

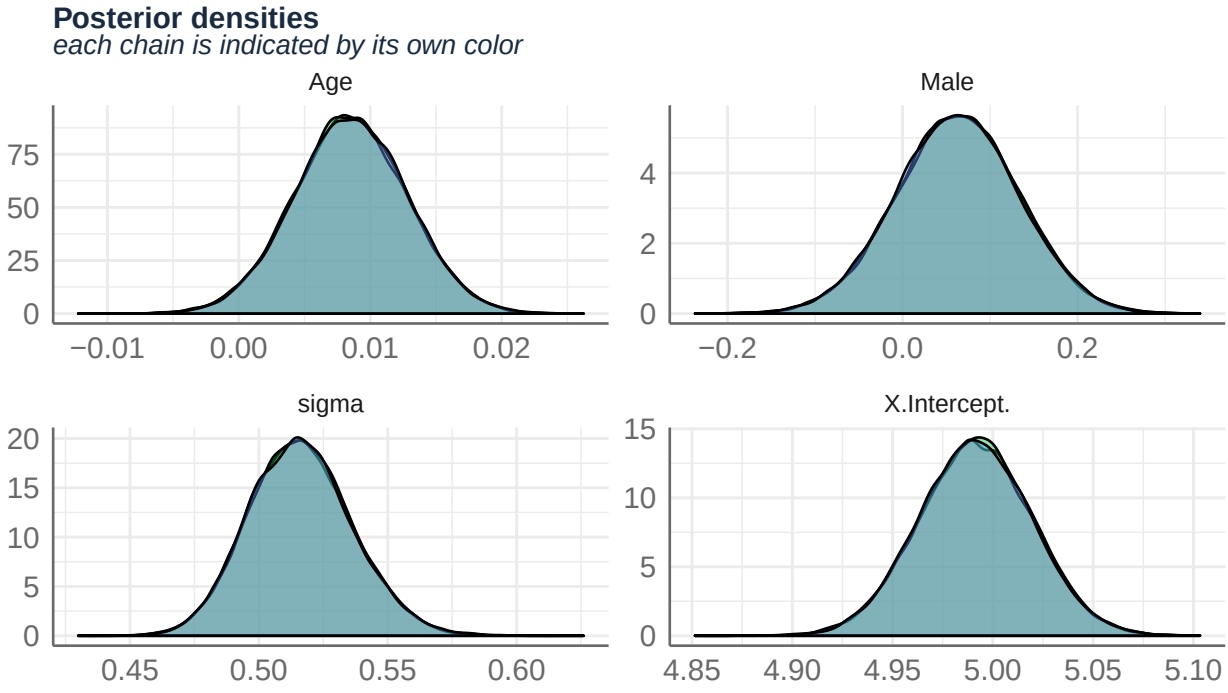
Assessing convergence

To assess convergence, we can check the convergence plots:

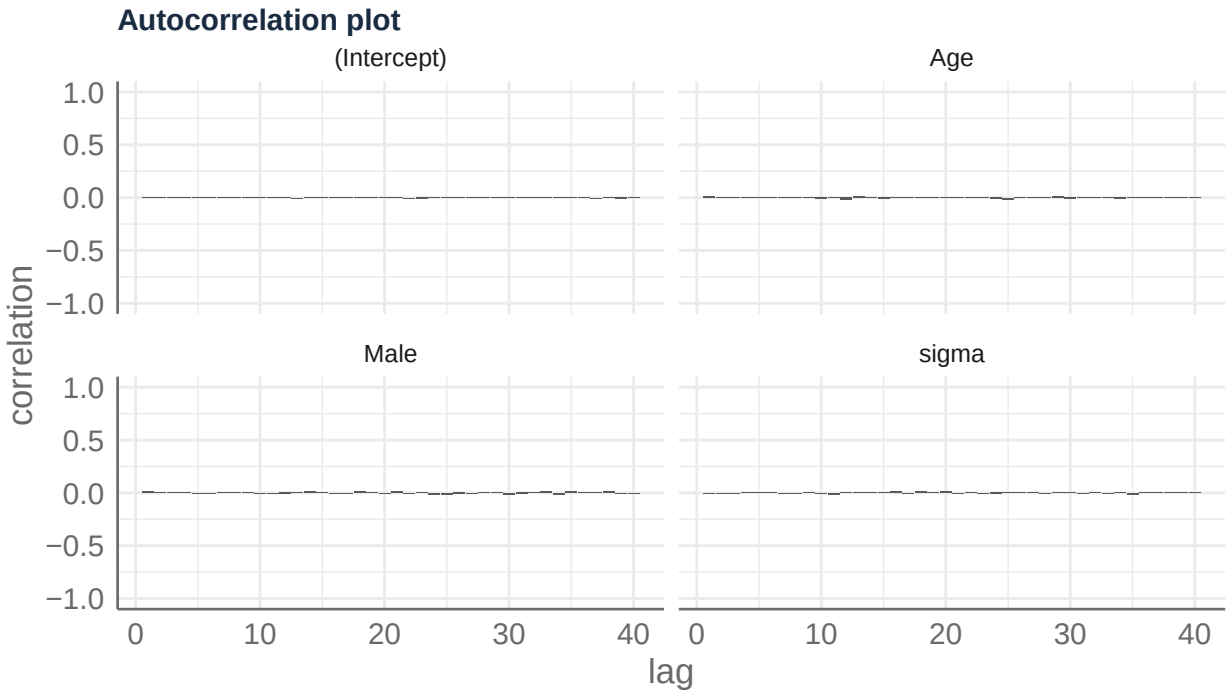
```
bfit %>%
  plot("history")
```




```
plot(bfit, "density")
```



```
plot(bfit, "autocorrelation")
```



If these look fine, you can also view the Gelman-Rubin statistic:

```
bfit %>%
  evaluate_convergence_diagnostics()
```

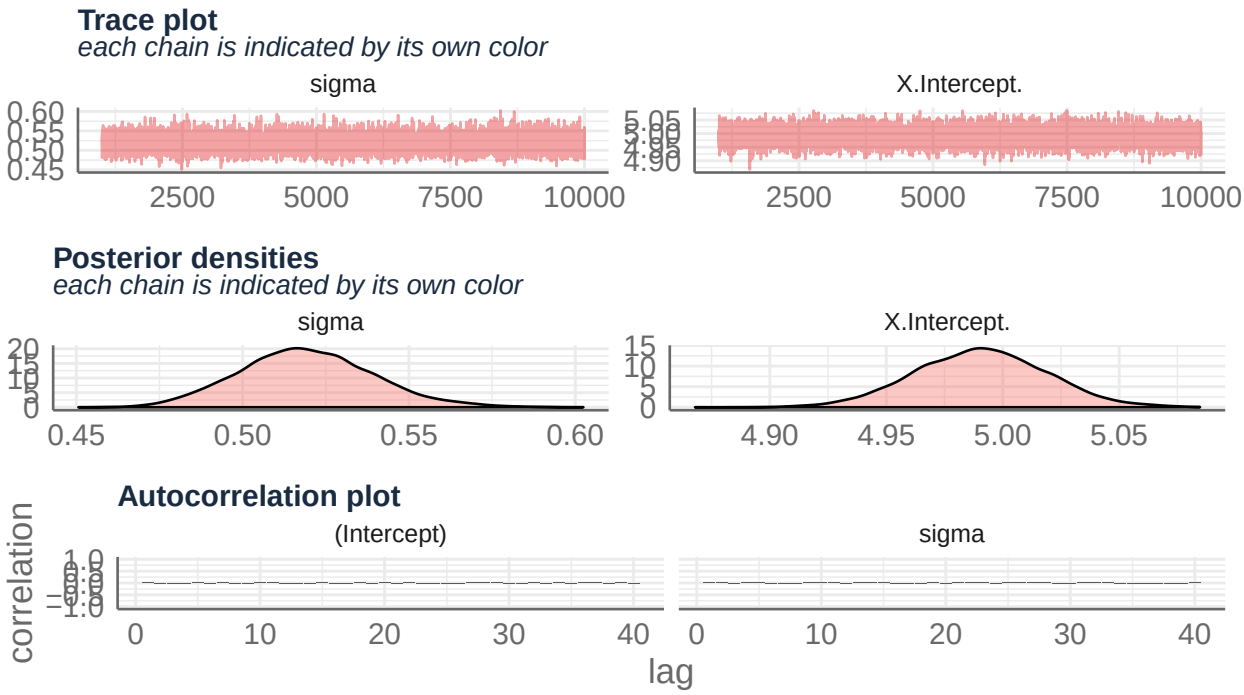
```
## Convergence diagnostics for blm object:
##
## Formula: 'Compensation ~ Age + Male'
##
## Sampled: TRUE
##
## Burn-in diagnostic:
##         b0   b1   b2   b3
## Chain 1 0.00 0.00 0.00 0.00
## Chain 2 0.00 0.00 0.00 0.00
## Chain 1 0.00 0.00 0.00 0.00
```

```
##
## Gelman-Rubin statistic:
##          b0   b1   b2   b3
## Value   1.00 1.00 1.00 1.00
```

The burn-in diagnostic are regression coefficients of the samples on the index, which will deviate if there is a trend in the data (unless they are non-linear and cancel each other out).

If you compute the intercept-only model, you will also want to check whether that model has converged:

```
plot(bfit, "nullmodel")
```



To evaluate the effect of autocorrelation on the effective sample size of the MCMC samples, you can execute:

```
bfit %>%
  evaluate_effective_sample_size()
```

```
## Effective sample size for blm object:
##
```

```
## Effective sample size :
##                 n
## (Intercept) 69000
## Age         65669
## Male        65991
## sigma       69000
```

If you use the MH sampler, you can also check the number of accepted samples like so:

```
bfit %>%
  evaluate_accepted_draws()
```

```
## Accepted draws for blm object:
##
## Accepted draws :
##             chain_1 chain_2 chain_3
## (Intercept)   25000   25000   25000
## Age           25000   25000   25000
## Male          25000   25000   25000
## sigma         25000   25000   25000
```

Evaluating results

Most results are returned when you sample the model. You can view them by executing:

```
summary(bfit)
```

```
## Model results for blm object:
##
## Formula: 'Compensation ~ Age + Male'
##
## Sampled: TRUE
##
## Sampling settings :
```

```
##  Obs. Predictors Chains Iterations Thinning Burn
##   336          2      3      25000        2 2000
##
## ----------------------
##
## Maximum a posteriori (MAP) estimates :
##             Est. (mean)    SD NAIVE MCERR. TS MCERR.
## (Intercept)       4.991 0.028            0         0
## Age               0.008 0.004            0         0
## Male              0.063 0.070            0         0
## sigma             0.516 0.020            0         0
##
## 95% credible interval :
##                 2.5%    25%    50%    75%  97.5%
## (Intercept)  4.9361 4.9723 4.9914 5.0104 5.0464
## Age          0.0001 0.0056 0.0085 0.0114 0.0169
## Male        -0.0751 0.0160 0.0630 0.1100 0.1999
## sigma        0.4786 0.5023 0.5157 0.5294 0.5575
##
## ----------------------
##
## Model R-squared :
##           2.5%    25%    50%   75%  97.5%
## (Model) 0.0021 0.0117 0.0202 0.031 0.0576
##
## Model fit :
##                   LL     DIC Eff. P
## Null model -255.462 512.792  1.868
## User model -252.444 508.948  4.060
##
## Wagemakers' Approx. Bayes Factor :
##      BF
## 1 0.035
```

```
##
## (BF > 1 is evidence for the user-defined model against the intercept-only model)
##
## Hypothesis evaluation :
##                  complexity   fit   BF_c  Pr_a  Pr_b
## H1: b0 + b2 < b0      0.048 0.184  4.504 0.673 0.573
## H2: b1 > 0            0.520 0.976 37.025 0.327 0.279
## Hu:                       .     .      .     . 0.148
```

A model that contains all optional elements has the following objects embedded in it:

```
names(bfit)
```

```
## [1] "input"             "sampling_settings" "priors"
## [4] "hypotheses"        "null_model"        "posterior"
## [7] "rsq"               "DIC"               "model_BF"
```

You can use `contains()` to see whether you model contains a specific element[2]:

```
# Use blm::contains() in case of clashes
blm::contains(bfit, "rsq")
```

```
## [1] TRUE
```

If you want to get a summary of the individual elements, you can execute e.g.

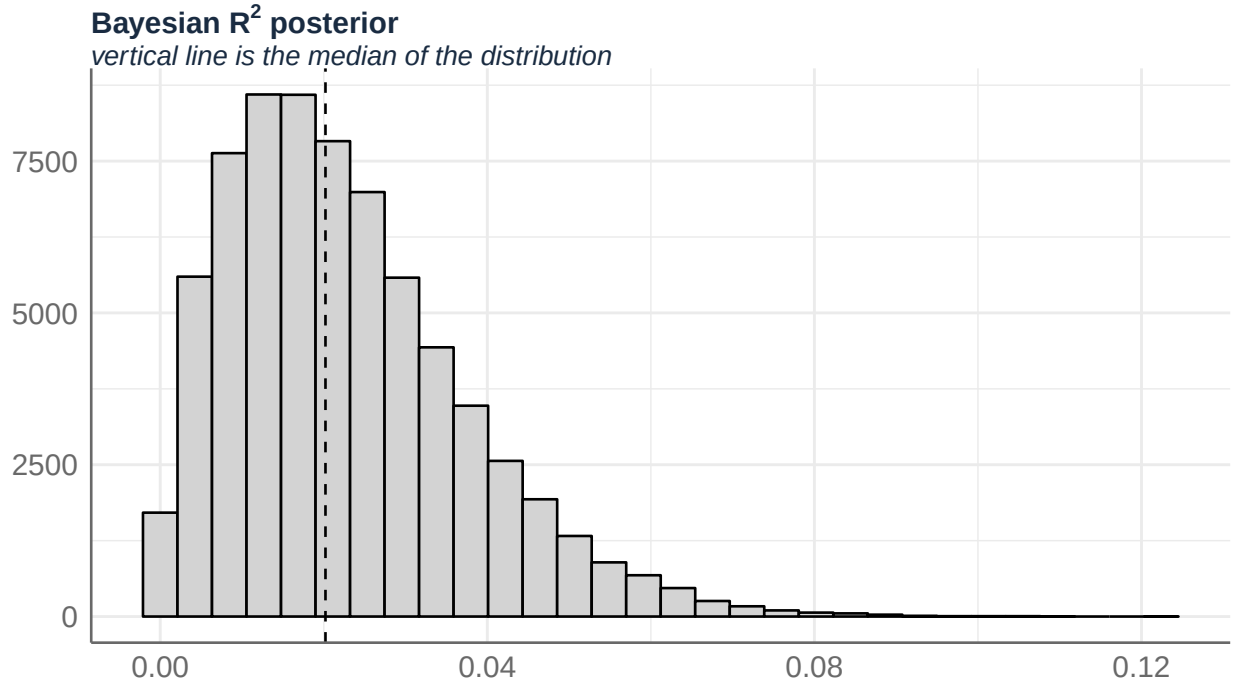
```
# i.e. r-squared
bfit %>%
  get_value("rsq") %>%
  summary()
```

```
## Model R-squared :
##           2.5%    25%    50%   75%  97.5%
## (Model) 0.0021 0.0117 0.0202 0.031 0.0576
```

[2] `contains()` is also an exported function for other packages, so this may clash with e.g. dplyr

You can also view the posterior for the R^2 values:

```
bfit %>%
  get_value("rsq") %>%
  plot()
```



Posterior predictive checks

`blm` contains three posterior predictive checks. The first checks the residuals for skewness, the second for heteroskedasticity and the third for independence.

```
bfit <- bfit %>%
  # Use a fraction of the total posterior
  # (to reduce the size of the output and because it is heavy on this pdf file)
  evaluate_ppc(p=.15)
```

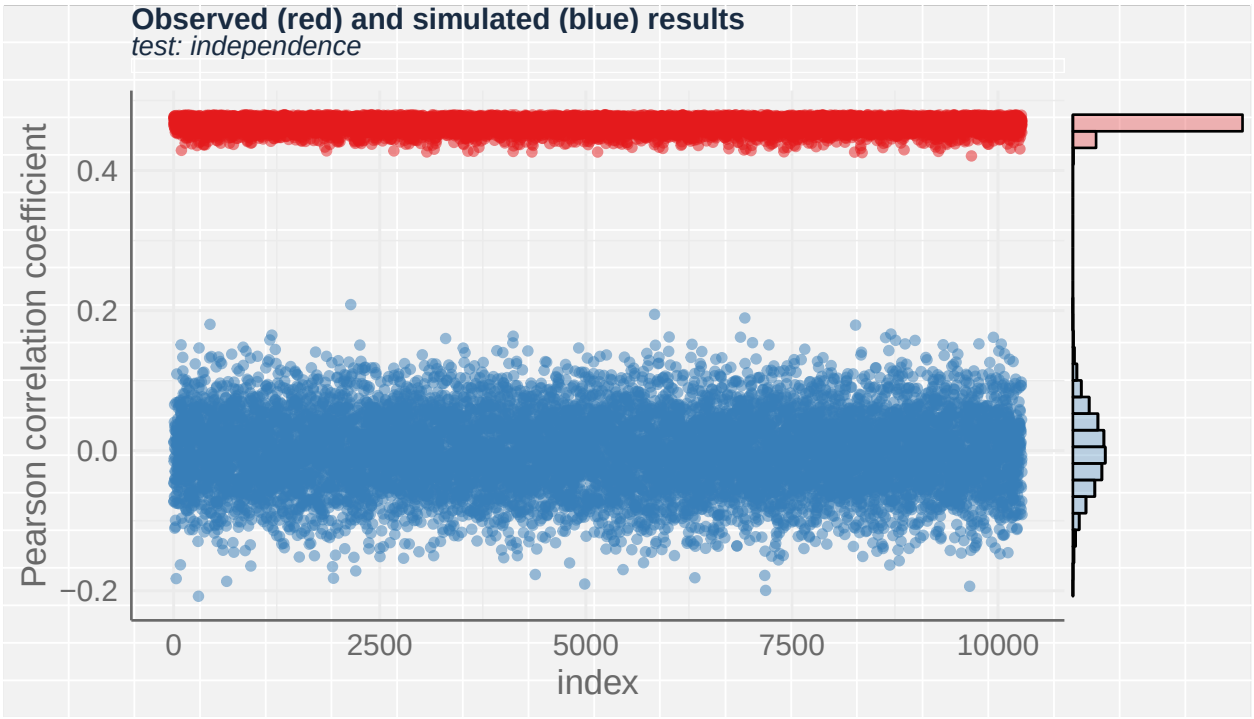
The PPC results are the only results not automatically included in the summary output and the only function in the `evaluate_` family that the user must call himself.

```
bfit %>%
  get_value("ppc") %>%
  summary()
```

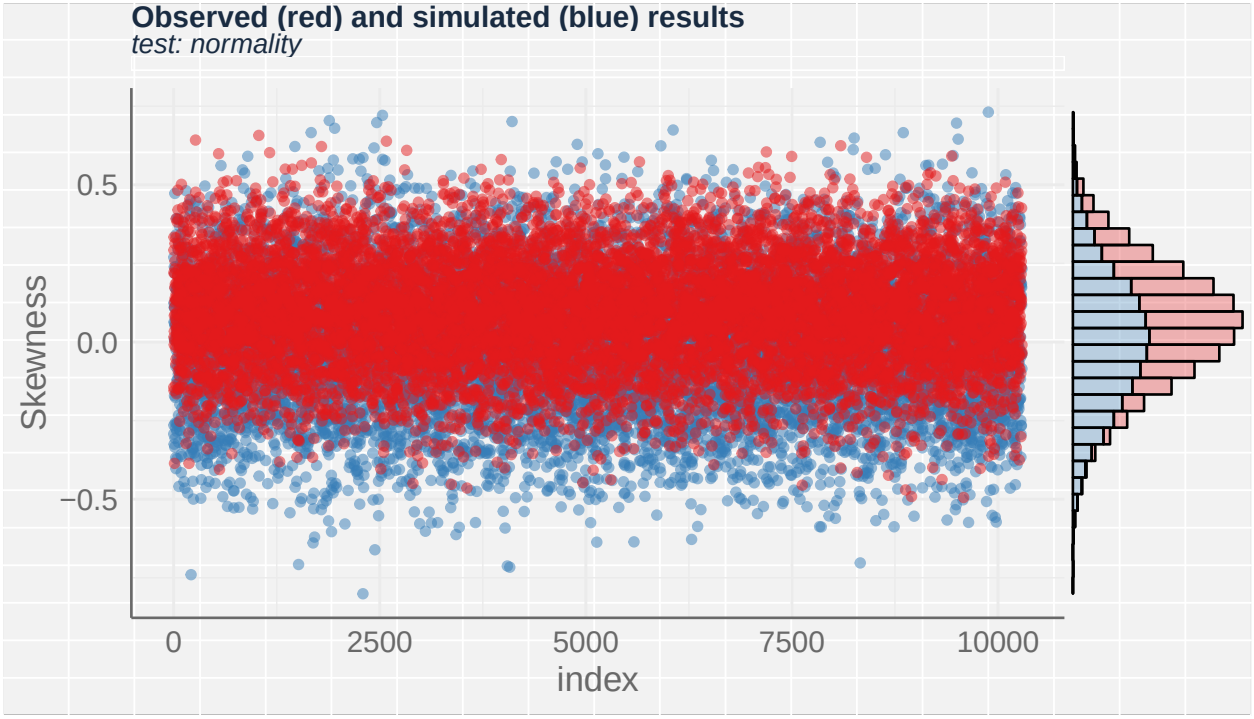
```
## Posterior Predictive Checks (PPC) for blm object:
##
## Bayesian p-value :
##   Normality Heteroskedasticity Independence
## p    0.3597             0.3258            0
```

We can also plot these results:

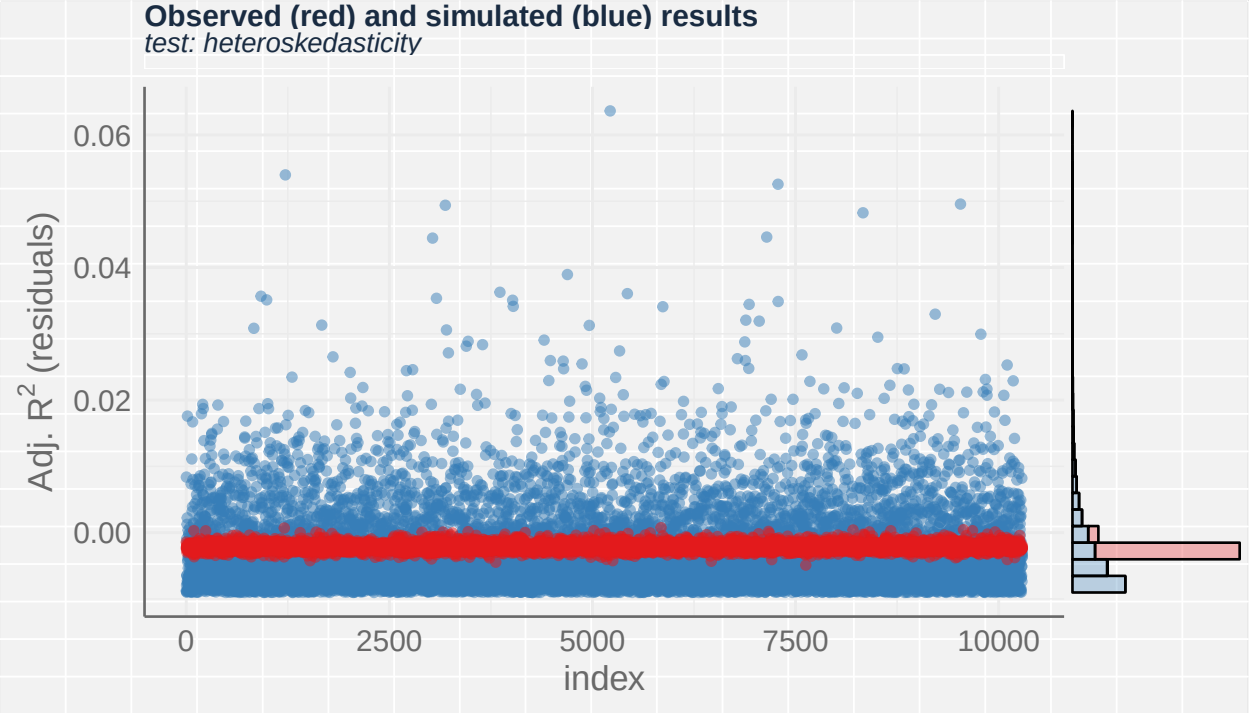
```
bfit %>%
  get_value("ppc") %>%
  plot("independence") %>%
  plot()
```



```
bfit %>%
  get_value("ppc") %>%
  plot("normality") %>%
  plot()
```




```
bfit %>%
  get_value("ppc") %>%
  plot("heteroskedasticity") %>%
  plot()
```



Misc

The `blm` library contains a ggplot2 theme called `theme_blm()`