

The blmr package

Manfred Schmid

2016-01-25

Note

The blmr package was created as part of a DataScience course at Aarhus University and is NOT intended for use. Expecially correct handling of prior means and data precision is not guaranteed.

Introduction

This vignette describes various functionalities of the blmr package. The blmr package allows for fitting of Bayesian Linear Models to data. The fit is stored in an S3 object of class *blm*. The package includes many generic functions available for Linear Model *lm* objects (coef, resid, ...). The plotting function is designed to visualize the blm model together with the data. At the end I also implemented a few functions for analysis of blm objects.

Core Functions Math

I don't provide example data but rather demonstrate functionality using random data. The basic linear regression model is described by the formula:

$$y = w_0 + w_1 * x + \epsilon$$

However, this can easily be generalized to fit an arbitraty number of additive terms:

$$\mathbf{y} = \mathbf{w}\Phi_{\mathbf{x}} + \epsilon$$

where:

$\Phi_{\mathbf{x}}$ is the design matrix.

\mathbf{w} is the vector of coefficients, ie the weights on the design matrix columns.

ϵ is the vector of errors on the observables.

\mathbf{y} vector of observed values.

Frequentist solution for fitting linear models is to minimize the sum of squared residuals (ie the sum of squared distance between observed and fitted y values). There is a very elegant solution for this problem using matrix algebra (the so-called Moore-Penrose pseudoinverse):

$$\mathbf{w} = (\Phi_{\mathbf{x}}^T \Phi_{\mathbf{x}})^{-1} \Phi_{\mathbf{x}}^T \mathbf{y}$$

For a bayesian approach one needs to consider not the best fit, but a likelihood distribution of coefficients that updates a *prior* distribution of the weights. For simplicity, both prior and likelihood are considered normal distributed.

Prior:

$$\mathbf{w}_0 = N(\boldsymbol{\mu}_0, \sigma^2 \boldsymbol{\Lambda}_0^{-1})$$

Posterior:

$$\mathbf{w}_{\mathbf{x},\mathbf{y}} = N(\boldsymbol{\mu}_{\mathbf{x},\mathbf{y}}, \sigma^2 \boldsymbol{\Lambda}_{\mathbf{x},\mathbf{y}}^{-1})$$

Posterior Distribution of Coefficients

Thomas provided the following equations for updating (slightly changed variable names for consistency):

$$\boldsymbol{\Lambda}_{\mathbf{x},\mathbf{y}} = \beta \Phi_{\mathbf{x}}^T \Phi_{\mathbf{x}} + \boldsymbol{\Lambda}_0$$

$$\mu_{x,y} = \beta \Lambda_{x,y}^{-1} \Phi_x^T y)$$

Wikipedia (https://en.wikipedia.org/wiki/Bayesian_linear_regression) suggests the following equation:

$$\Lambda_{x,y} = \Phi_x^T \Phi_x + \Lambda_0$$

$$\mu_{x,y} = \Lambda_{x,y}^{-1}(\mu_0 \Lambda_0 + \Phi_x^T y)$$

This is somewhat odd. The solution from Wikipedia apparently does not consider error precision in the calculations. That is, the precision of the posterior coefficients would not be influenced by the precision of the data. By contrast, equations of the posterior means by Thomas does not consider the prior means when computing the posterior MAP values, ie this solution is possibly only correct when $\mu_0 = 0$.

I found this complete formulas from free available course material

(http://gandalf.psych.umn.edu/users/schrater/schrater_lab/courses/PattRecog09/BayesRegress.pdf) from Paul Schrater (University of Minnesota). Similar to Thomas, with a term including the mean of the prior added to the equation for the posterior mean.

$$\Lambda_{x,y} = \beta \Phi_x^T \Phi_x + \Lambda_0$$

$$\mu_{x,y} = \Lambda_{x,y}^{-1}(\mu_0 \Lambda_0 + \beta \Phi_x^T y)$$

The solution used in the *blm* function of the *blmr* package implements the equations as given by Paul Schrater.

Distribution of Fitted/Predicted Values

The distribution of predicted/fitted values for y is from Thomas (identical to the one provided by Schrater):

$$p(y_{pred}|model) = N(y|\mu_{x,y}, \frac{1}{\beta} + \Phi_x^T \Lambda_{x,y}^{-1} \Phi_x)$$

Total Probability of a Model Fit

I used a formula from Wikipedia (https://en.wikipedia.org/wiki/Bayesian_linear_regression). This equation is based on the fit and response, and the parameters for the prior and posterior inverse gamma distributions *a* and *b*.

$$p(y|model) = \frac{1}{(2 * \pi i)^{\frac{n}{2}}} * \sqrt{\frac{\det(\Phi_0^T)}{\det(\Phi_{x,y}^T)}} * \frac{b_0^{a_0}}{b_{x,y}^{a_{x,y}}} * \frac{\Gamma a_{x,y}}{\Gamma a_0}$$

where: a_0 : prior value for the inverse gamma distribution parameter *a* (default: $a_0 = 1$).

b_0 : prior value for the inverse gamma distribution parameter *b* (default: $b_0 = 1$).

$a_{x,y} = a_0 + \frac{n}{2}$: posterior value for *a*.

$b_{x,y} = b_0 + \frac{1}{2}(y^T y + \mu_0^T \Lambda_0 \mu_0 + \mu_{x,y}^T \Lambda_{x,y} \mu_{x,y})$: posterior value for *b*.

The above equation leads to very small probabilities, for many models not computable in R. However, it is possible to compute the log-likelihood, and this is computed by default.

$$\ln(p(y|model)) = 1 - \frac{n * \ln(2 * \pi i) + \ln(\det(\Phi_0^T)) - \ln(\det(\Phi_{x,y}^T))}{2} + a_0 * \ln(b_0) - a_{x,y} * \ln(b_{x,y}) + \ln(\Gamma a_{x,y}) - \ln(\Gamma a_0)$$

To install and load the package:

```
#devtools::install_github('manschmi/blmr')
library(blmr)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

Fitting a Bayesian Model

To simulate a simple linear regression problem I use the model data:

```
set.seed(0)
w0 <- 1
w1 <- 2
x <- seq(-100,100,10)
b <- 0.001
e <- rnorm(length(x), mean=0, sd=sqrt(1/b) )
y <- w0 + w1*x + e
```

The blm fit using default settings is straightforward.

```
blm_mod <- blm(y~x)
blm_mod
```

```
## Call:
## blm(formula = y ~ x)
##
## Coefficients:
## (Intercept)          x
##  0.01364827  1.79928333
```

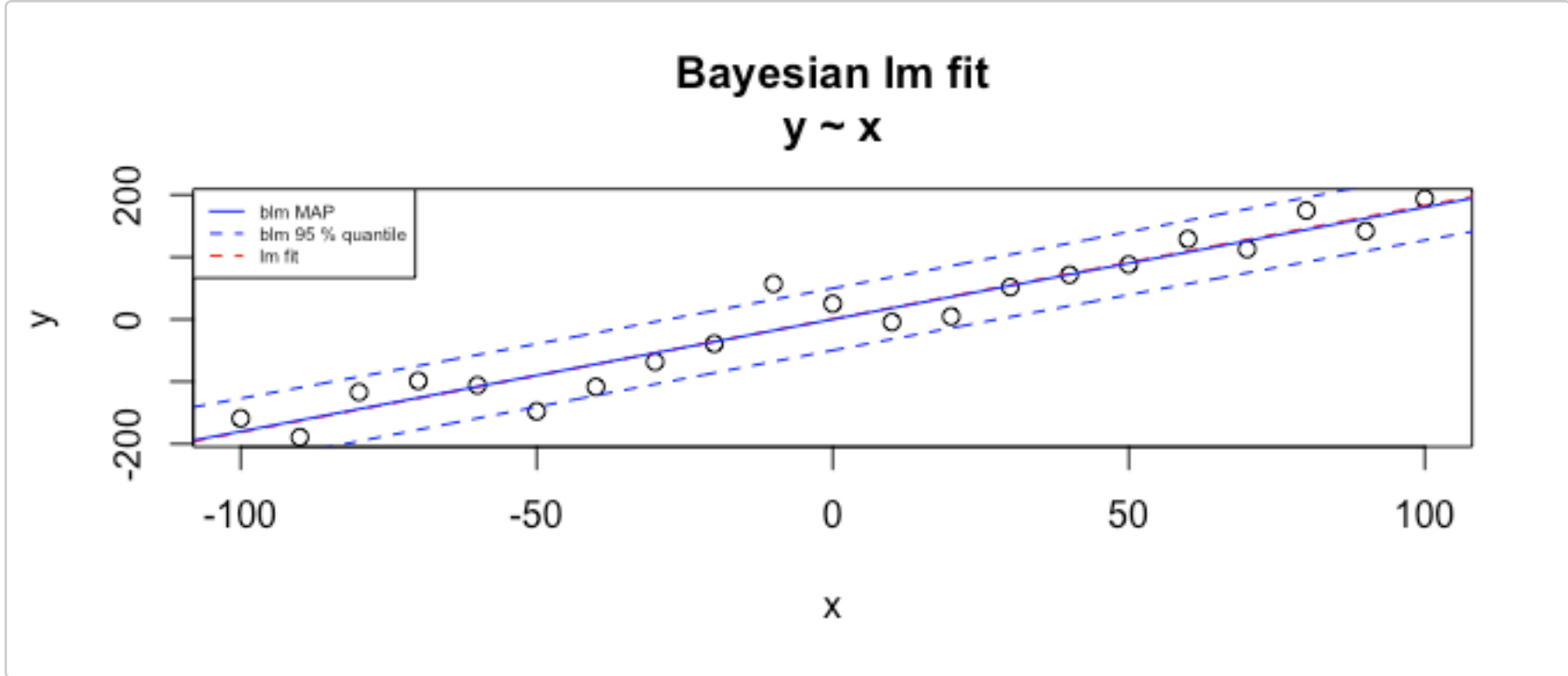
With default setting the result is very similar to a lm fit.

```
lm_mod <- lm(y~x)
lm_mod
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)          x
##    0.6087    1.8207
```

We can easily visualize this (more on the plotting function later):

```
plot(blm_mod, legend_parm=list(cex=.5))
```



Priors

The normal blm fit is extremely close to the lm fit. This is mostly because an uninformative prior is used. Here the prior would have means 0 for both intercept and slope and variance 1. But what about if you want to specify other priors. The prior (supported by this package) need to be multivariate normal distributions and for the package also contains a S3 class for multivariate normal distribution *mvnd*. Its functionality is very simple and easies demonstrated by example:

```
custom_prior <- mvnd(means = c(1,1), covar = matrix(c(2,.2,.2,2),ncol=2))
custom_prior
```

```
## $means
## [1] 1 1
##
## $covar
##      [,1] [,2]
## [1,] 2.0 0.2
## [2,] 0.2 2.0
##
## attr(,"class")
## [1] "mvnd"
```

```
blm(y~x, prior=custom_prior)
```

```
## Call:
## blm(formula = y ~ x, prior = custom_prior)
##
## Coefficients:
## (Intercept)          x
## 1.061041      1.815816
```

A typical use case would be to use the posterior distribution of an existing blm object as prior for a new model. An this is easy as an *blm* object can be used as prior.

```
blm(y~x, prior=blm_mod)
```

```
## Call:
## blm(formula = y ~ x, prior = blm_mod)
##
## Coefficients:
```

```
## (Intercept)          x
## 0.02669794  1.80991777
```

Precision

When precision of the error β is not provided it is estimated from the data using the deviance of the observed y values from the lm (ie MLE) fit using the following equation:

$$1. \text{MLE coefficients: } \mathbf{w} = (\mathbf{\Phi}_x^T \mathbf{\Phi}_x)^{-1} \mathbf{\Phi}_x^T \mathbf{y}$$

$$2. \text{MLE residuals: } \mathbf{RSS} = \mathbf{y} - \mathbf{w} \mathbf{\Phi}_x$$

.

$$3. \text{precision: } \beta = \left(\frac{\sum \mathbf{RSS}^2}{n - p} \right)^{-1}$$

Pretty straightforward, simply do the MLE fit, get the variance of the residuals using degrees of freedom of the regression model.

As alternative it is also possible to specify beta as argument to the *blm* function.

```
set.seed(0)
w0 <- 1
w1 <- 2
x <- seq(-100,100,10)
b <- 0.001
y <- w0 + w1*x + rnorm(length(x), mean=0, sd=sqrt(1/b) )
mod_with_b <- blm(y~x, beta=b)
mod_with_b
```

```
## Call:
## blm(formula = y ~ x, beta = b)
##
## Coefficients:
## (Intercept)          x
## 0.01252017  1.79733663
```

```
precision(mod_with_b)
```

```
## [1] 0.001
```

Compared to test without providing precision (ie where it is estimated from the data):

```
mod_wout_b <- blm(y~x)
mod_wout_b
```

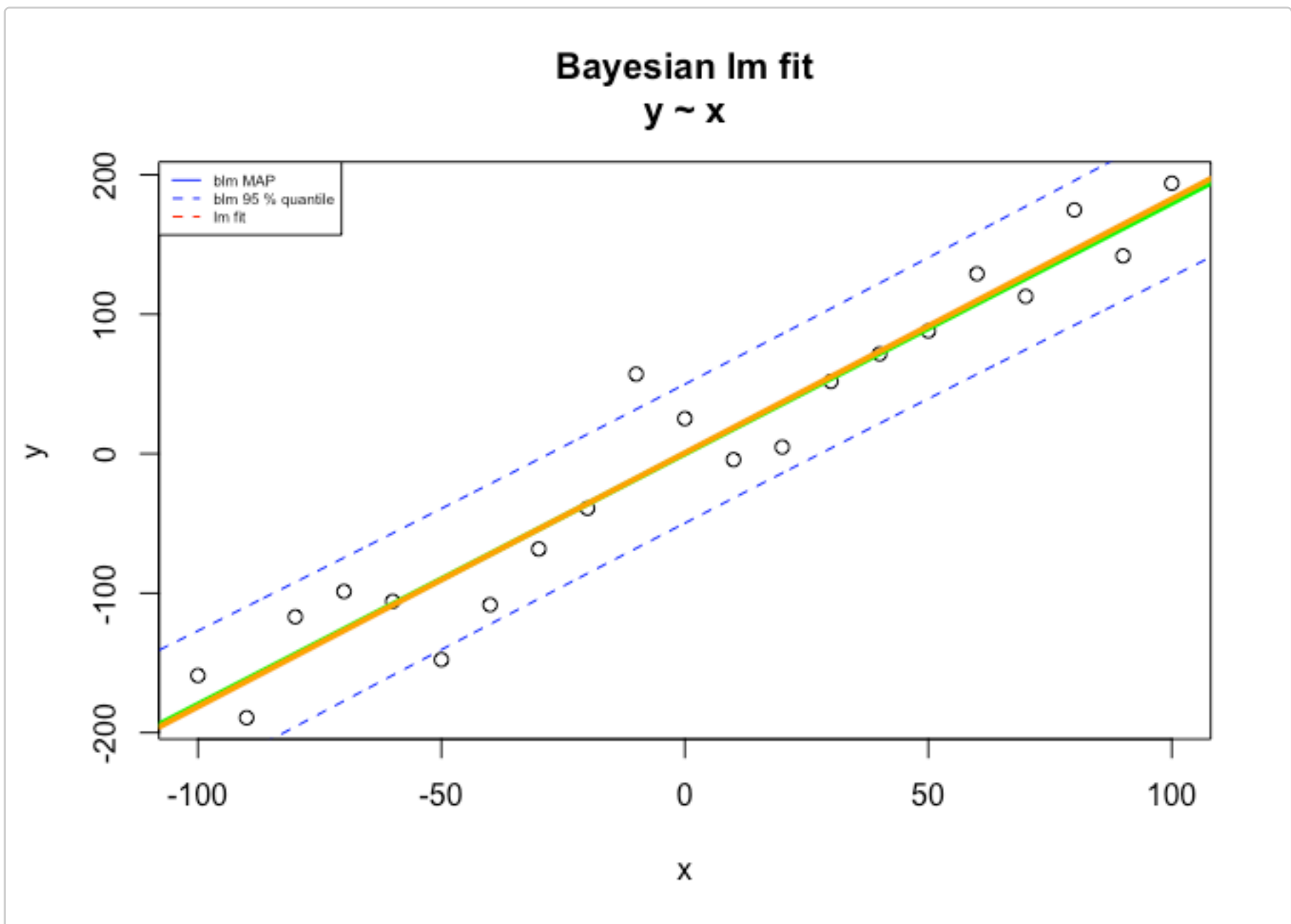
```
## Call:
## blm(formula = y ~ x)
##
## Coefficients:
## (Intercept)          x
## 0.01364827  1.79928333
```

```
precision(mod_wout_b)
```

```
## [1] 0.001092169
```

The MAP and covariance values are very close although not exactly the same. Lets check it out on a plot:

```
plot(blm(y~x), legend_parm=list(cex=.5))
abline(blm(y~x, beta = b), col='green', lwd=3)
abline(lm(y~x), col='orange', lwd=3)
```



The *blm* object

The *blm* object contains the following slots. The most important of these can be accessed via generic or *blm*-specific functions (in paranthesis).

call: the matched call (no fun)

formula: the formula used (formula())

df.residual: the degrees of freedom of the model (df.residual())

frame: the model frame used (model.frame())

matrix: the model matrix used (model.matrix())

beta: the precision of the data (precision())

prior: the prior distribution used (no fun)

posterior: the posterior distribution (coef(), coef(, var=TRUE))

The `coef()` (or `coefficients()`) returns by default only the MAP estimate of the coefficients.

```
coef(blm_mod)
```

```
## (Intercept)          x
## 0.01364827  1.79928333
```

To get the covariance of the posterior distribution set argument *covar* = *TRUE*:

```
coef(blm_mod, covar=TRUE)
```

```
##           (Intercept)           x
## (Intercept)  0.9775787 0.00000000
## x           0.0000000 0.01175129
```

Extracting residuals, fitted values, ...

Again, this is done using generic functions. So far those are implemented:

resid, residual
deviance
fitted
predict
confint

fitted and residuals

For fitted and residuals it is possible to retrieve the variance of the estimate (same as the variance for the fitted values they are calculated from) together with the MAP estimate using parameter *var = TRUE*.

```
fitted(blm_mod)
```

```
##           1           2           3           4           5
## -179.91468512 -161.92185178 -143.92901844 -125.93618510 -107.94335176
##           6           7           8           9          10
##  -89.95051843  -71.95768509  -53.96485175  -35.97201841  -17.97918507
##           11          12          13          14          15
##   0.01364827   18.00648161   35.99931495   53.99214829   71.98498163
##           16          17          18          19          20
##   89.97781497  107.97064831  125.96348165  143.95631499  161.94914833
##           21
##  179.94198167
```

```
fitted(blm_mod, var=TRUE)
```

```
## $mean
##           1           2           3           4           5
## -179.91468512 -161.92185178 -143.92901844 -125.93618510 -107.94335176
##           6           7           8           9          10
##  -89.95051843  -71.95768509  -53.96485175  -35.97201841  -17.97918507
##           11          12          13          14          15
##   0.01364827   18.00648161   35.99931495   53.99214829   71.98498163
##           16          17          18          19          20
##   89.97781497  107.97064831  125.96348165  143.95631499  161.94914833
##           21
##  179.94198167
##
## $var
##           1           2           3           4           5           6           7
## 1034.0995 1011.7721  991.7949  974.1679  958.8912  945.9648  935.3887
##           8           9          10          11          12          13          14
##  927.1628  921.2871  917.7617  916.5866  917.7617  921.2871  927.1628
##          15          16          17          18          19          20          21
##  935.3887  945.9648  958.8912  974.1679  991.7949 1011.7721 1034.0995
```

```
resid(blm_mod)
```

```
##           1           2           3           4           5           6
## 20.8528063 -27.3945529 26.9809635 27.1739333 2.0554652 -57.7469777
##           7           8           9          10          11          12
## -36.4061828 -14.3550271 -3.2103556 75.0210020 25.1332972 -22.2733726
##          13          14          15          16          17          18
## -31.2914162 -2.1457270 -0.4469945 -1.9909301 21.0053574 -13.1685042
##          19          20          21
## 30.8212007 -20.0835494 13.9660451
```

```
resid(blm_mod, var=TRUE)
```

```
## $mean
##           1           2           3           4           5           6
## 20.8528063 -27.3945529 26.9809635 27.1739333 2.0554652 -57.7469777
##           7           8           9          10          11          12
## -36.4061828 -14.3550271 -3.2103556 75.0210020 25.1332972 -22.2733726
##          13          14          15          16          17          18
## -31.2914162 -2.1457270 -0.4469945 -1.9909301 21.0053574 -13.1685042
##          19          20          21
## 30.8212007 -20.0835494 13.9660451
##
## $var
##           1           2           3           4           5           6           7
## 1034.0995 1011.7721 991.7949 974.1679 958.8912 945.9648 935.3887
##           8           9          10          11          12          13          14
## 927.1628 921.2871 917.7617 916.5866 917.7617 921.2871 927.1628
##          15          16          17          18          19          20          21
## 935.3887 945.9648 958.8912 974.1679 991.7949 1011.7721 1034.0995
```

predict

The implementation of predict is different to provide consistency with *predict.lm* and uses the arguments *se.fit* and *interval*. *se.fit* values are computed as $\sqrt{\sigma^2}$ of the fitted values. Confidence interval are the quantiles from the predicted distribution.

```
predict(blm_mod)
```

```
##           1           2           3           4           5
## -179.91468512 -161.92185178 -143.92901844 -125.93618510 -107.94335176
##           6           7           8           9          10
## -89.95051843 -71.95768509 -53.96485175 -35.97201841 -17.97918507
##          11          12          13          14          15
## 0.01364827 18.00648161 35.99931495 53.99214829 71.98498163
##          16          17          18          19          20
## 89.97781497 107.97064831 125.96348165 143.95631499 161.94914833
##          21
## 179.94198167
```

```
predict(blm_mod, se.fit=TRUE)
```

```
## $fit
##           1           2           3           4           5
## -179.91468512 -161.92185178 -143.92901844 -125.93618510 -107.94335176
##           6           7           8           9          10
## -89.95051843 -71.95768509 -53.96485175 -35.97201841 -17.97918507
```



```
##          11          12          13          14          15
##    0.01364827    18.00648161    35.99931495    53.99214829    71.98498163
##          16          17          18          19          20
##    89.97781497    107.97064831    125.96348165    143.95631499    161.94914833
##          21
##    179.94198167
##
## $se.fit
##          1          2          3          4          5          6          7          8
## 32.15742 31.80836 31.49277 31.21166 30.96597 30.75654 30.58412 30.44935
##          9         10         11         12         13         14         15         16
## 30.35271 30.29458 30.27518 30.29458 30.35271 30.44935 30.58412 30.75654
##         17         18         19         20         21
## 30.96597 31.21166 31.49277 31.80836 32.15742
```

```
predict(blm_mod, se.fit=TRUE, interval='confidence')
```

```
## $fit
##          fit          lwr          upr
## 1  -179.91468512 -127.020440 -232.808930
## 2  -161.92185178 -109.601748 -214.241956
## 3  -143.92901844  -92.128014 -195.730023
## 4  -125.93618510  -74.597568 -177.274803
## 5  -107.94335176  -57.008865 -158.877838
## 6   -89.95051843  -39.360510 -140.540527
## 7   -71.95768509 -21.651277 -122.264093
## 8   -53.96485175  -3.880132 -104.049571
## 9   -35.97201841   13.953750  -85.897786
## 10  -17.97918507   31.850969  -67.809339
## 11    0.01364827   49.811890  -49.784593
## 12   18.00648161   67.836635  -31.823672
## 13   35.99931495   85.925083  -13.926453
## 14   53.99214829  104.076868    3.907429
## 15   71.98498163  122.291389   21.678574
## 16   89.97781497  140.567823   39.387807
## 17  107.97064831  158.905135   57.036162
## 18  125.96348165  177.302099   74.624864
## 19  143.95631499  195.757320   92.155310
## 20  161.94914833  214.269252  109.629045
## 21  179.94198167  232.836227  127.047737
##
## $se.fit
##          1          2          3          4          5          6          7          8
## 32.15742 31.80836 31.49277 31.21166 30.96597 30.75654 30.58412 30.44935
##          9         10         11         12         13         14         15         16
## 30.35271 30.29458 30.27518 30.29458 30.35271 30.44935 30.58412 30.75654
##         17         18         19         20         21
## 30.96597 31.21166 31.49277 31.80836 32.15742
```

This is inconsistent with fitted and resid. However, importantly, this allows native adding of a blm fit to ggplot2 objects just like lm fit (see below under *Plotting*)

confint

Bayesian statistics does not use the term ‘confidence interval’. However, the 95% quantile of the distribution of parameters is pretty obvious choice to be consistent with generic model functions. So this is what is provided here.

```
confint(blm_mod)
```

```
##                2.5 %   97.5 %
## (Intercept) -1.924219 1.951515
## x           1.586817 2.011750
```

update

The update function used for the blm package was designed for versatility. It takes a blm object as input, together with a set of parameters (a new formula, prior and/or data) to update to and returns a **new** model with the updated fit. By default, the posterior of the input model will be used as prior for the updated model. Lets update our blm_mod for example:

```
x2 <- rnorm(50)
y2 <- rnorm(50, w1 * x2 + w0, 1/b)
new_mod <- update(blm_mod, data=data.frame(x=x2, y=y2))
new_mod
```

```
## Call:
## blm(formula = y ~ x, beta = 0.00109216923687356, prior = "object$posterior",
##      data = data.frame(x = x2, y = y2))
##
## Coefficients:
## (Intercept)          x
##    2.663505    1.888751
```

As can be seen the posterior of the input model blm_mod is used as prior for the updated model in this case. If we want the updated model to use the same prior, this needs to be specified:

```
update(blm_mod, prior=blm_mod$prior, data=data.frame(x=x2, y=y2))
```

```
## Call:
## blm(formula = y ~ x, beta = 0.00109216923687356, prior = blm_mod$prior,
##      data = data.frame(x = x2, y = y2))
##
## Coefficients:
## (Intercept)          x
##    2.718783    7.379108
```

We can also update the formula. Typically this involves dropping factors.

```
update(blm_mod, y~x+0, prior=blm_mod$prior, data=data.frame(x=x2, y=y2))
```

```
## prior contains more variables than the model : variables not used in the model are ignored in the fit
```

```
## Call:
## blm(formula = y ~ x - 1, beta = 0.00109216923687356, prior = blm_mod$prior,
##      data = data.frame(x = x2, y = y2))
##
## Coefficients:
##          x
## 7.373804
```

As you can see there is a warning that factors are dropped.

And this can be done using R formula update semantics.

```
update(blm_mod, ~.+0, prior=blm_mod$prior, data=data.frame(x=x2, y=y2))
```

```
## prior contains more variables than the model : variables not used in the model are ignored in the fit
```

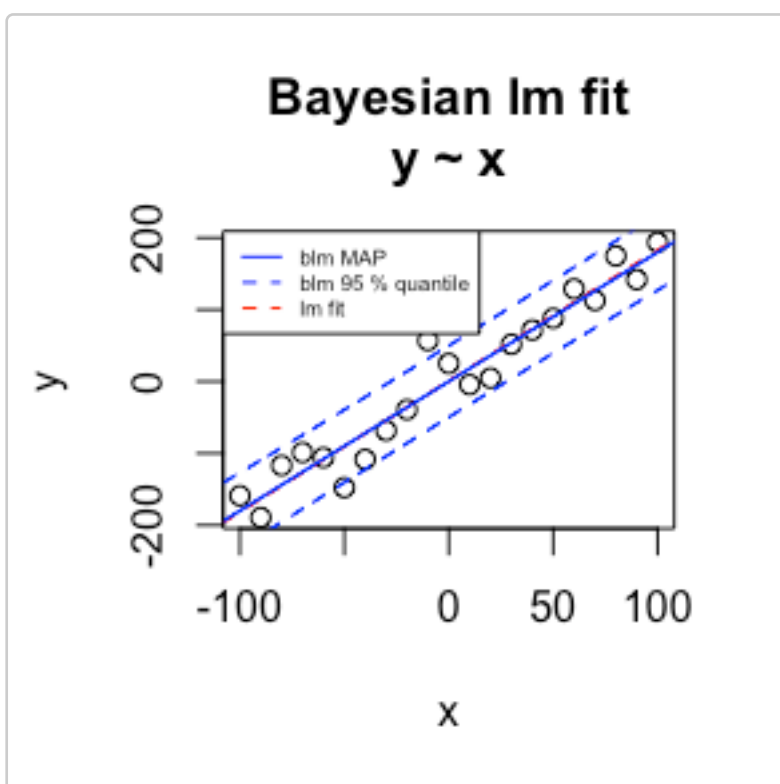
```
## Call:
## blm(formula = y ~ x - 1, beta = 0.00109216923687356, prior = blm_mod$prior,
##      data = data.frame(x = x2, y = y2))
##
## Coefficients:
##          x
## 7.373804
```

Note: one cannot (in the current implementation) update to more complex models.

Plotting

The function `plot.blm` produces a single plot of the data points, together with the blm MAP estimate, the 95% quantile and the lm fits. Note the *legend_parm* passes named arguments to legend. The `cex=.5` is to make prettier plots in the vignette.

```
plot(blm_mod, legend_parm=list(cex=.5))
```

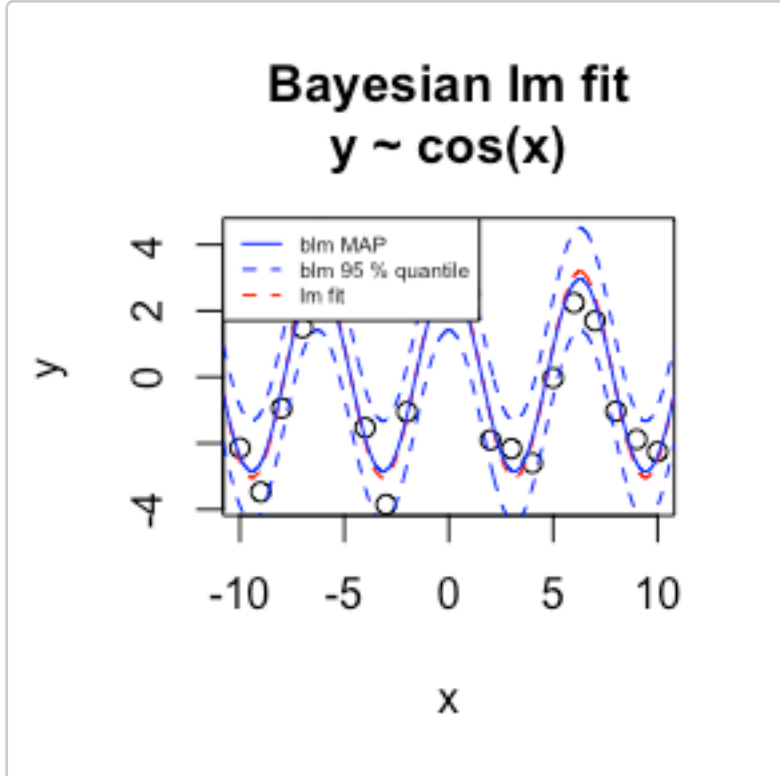


There is a little catch here. The blm object does not store the raw data used to create the model. In a scenario where the x values are not part of the model matrix and frame we need to provide them as arguments to the function:

```
w0 <- .2
w1 <- 3
x <- seq(-10,10,1)
b <- 1.3
y <- w0 + w1*cos(x) + rnorm(length(x), mean=0, sd=sqrt(1/b) )

model <- blm(y ~ cos(x), prior = NULL, beta = b, data = data.frame(x=x, y=y))
#plot(model) fails due to 'lack' of x in the 'model' object

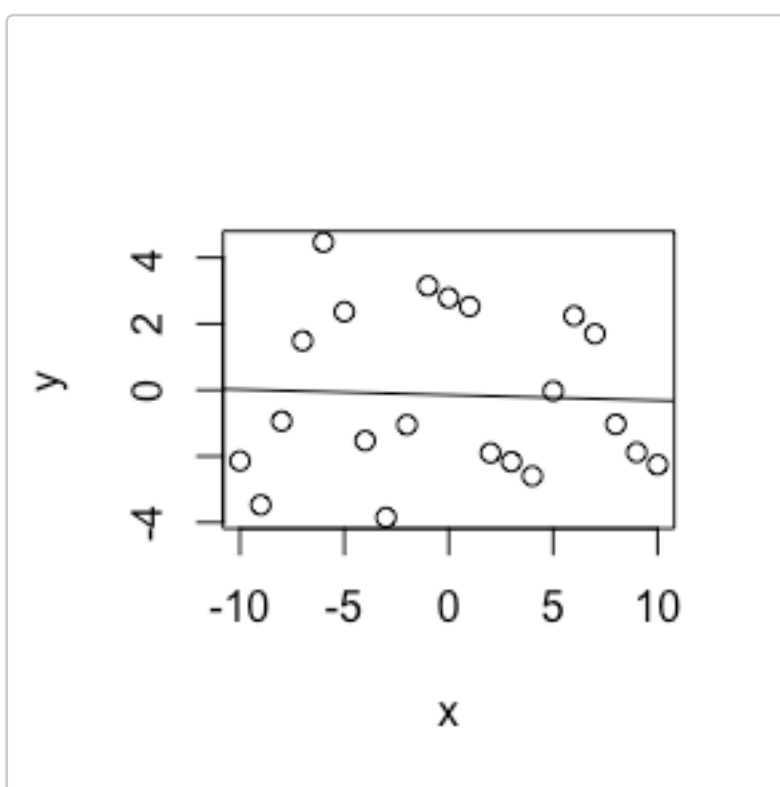
plot(model, explanatory='x', legend_parm=list(cex=.5))
```



Plotting using abline

One can also add the blm fit lines using abline function as for lm fits. Note: this only works for straight lines. It simply extracts the first 2 coefficients from the coef() function.

```
plot(y~x)
abline(blm(y~x))
```

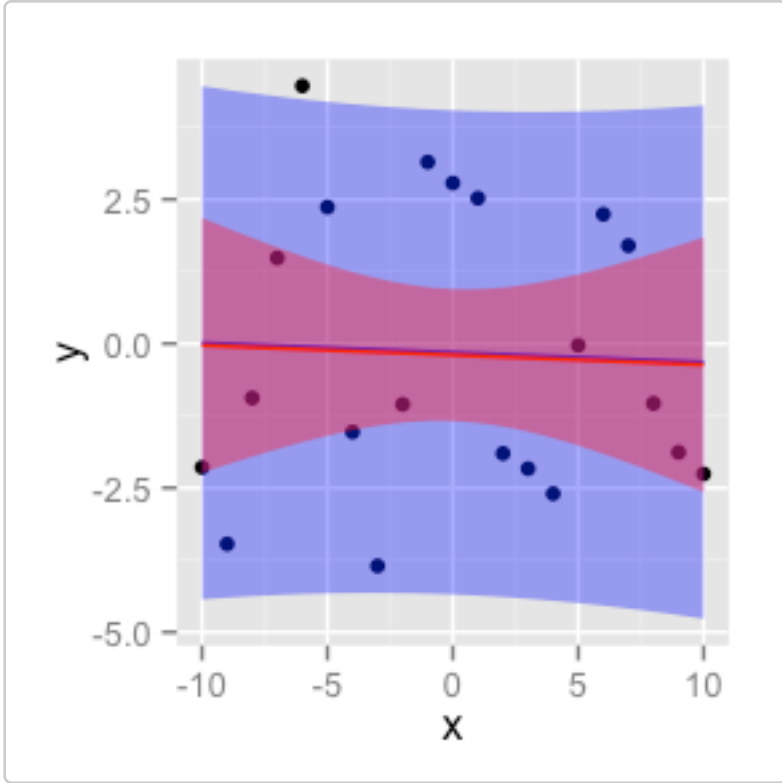


Plotting with ggplot2

One can also add the blm fit lines using stat_smooth for ggplot2 plots. Note: this only works for straight lines. This simply builds the model using default settings and calls the fitted function with se.fit = TRUE. The standard error area is derived from the distribution of the fitted values.

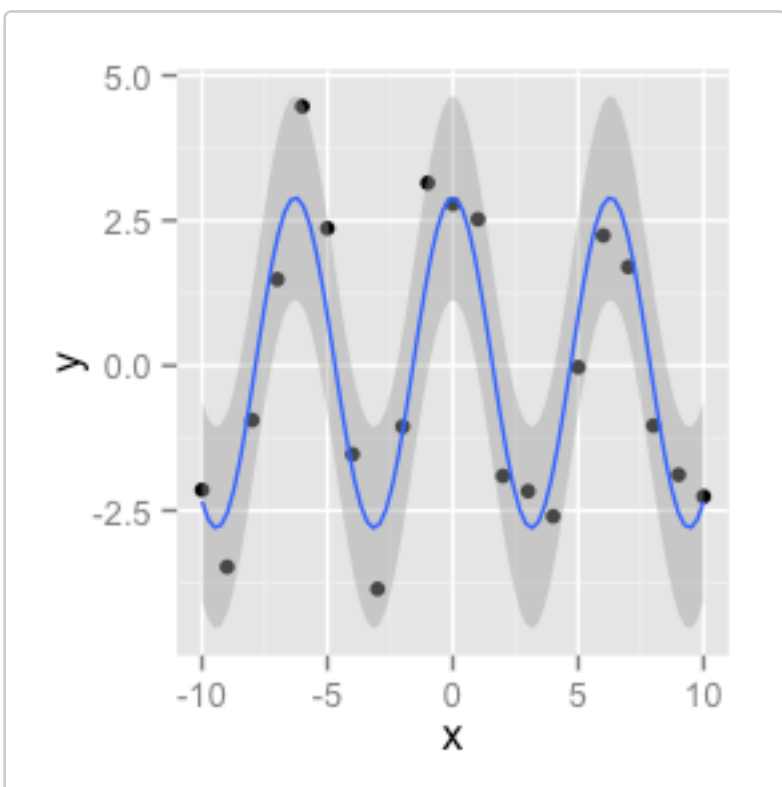
```
d <- data.frame(x=x, y=y)

ggplot(d, aes(x=x, y=y)) +
  geom_point() +
  stat_smooth(method='blm', fill='blue', colour='blue') +
  stat_smooth(method='lm', fill='red', colour='red')
```



Of course, one can pass additional arguments to `blm` from within `stat_smooth`:

```
ggplot(d, aes(x=x, y=y)) +
  geom_point() +
  stat_smooth(method='blm', formula=y~cos(x), beta=1)
```

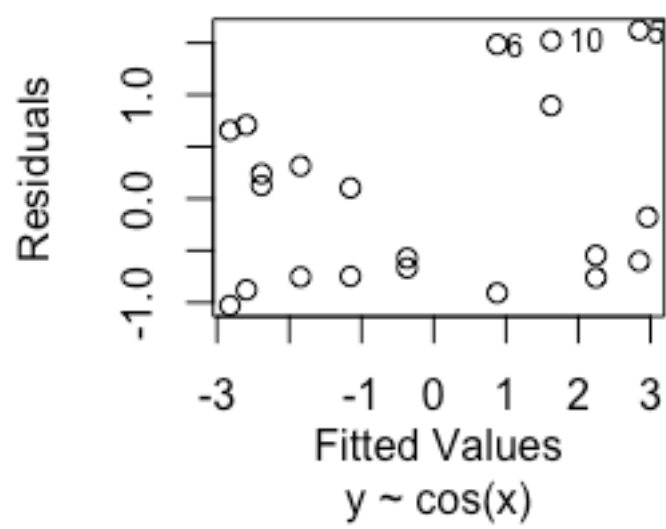


Diagnsotic Plots

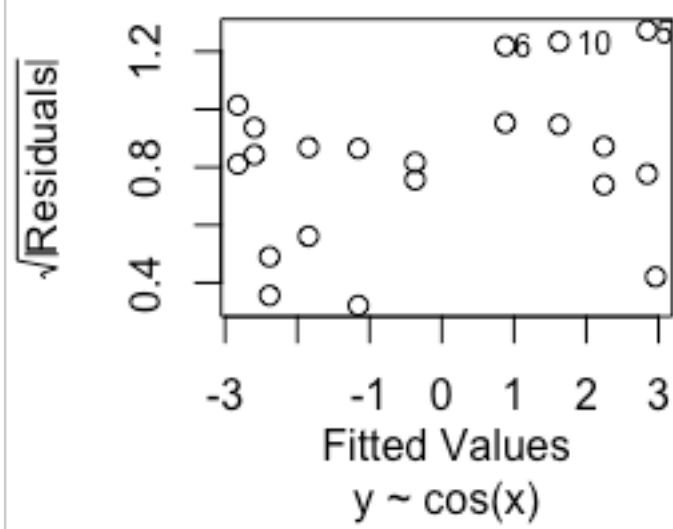
There is also support for some diagnostic plots, akin to `plot.lm`:

```
diagnostic_plots(model)
```

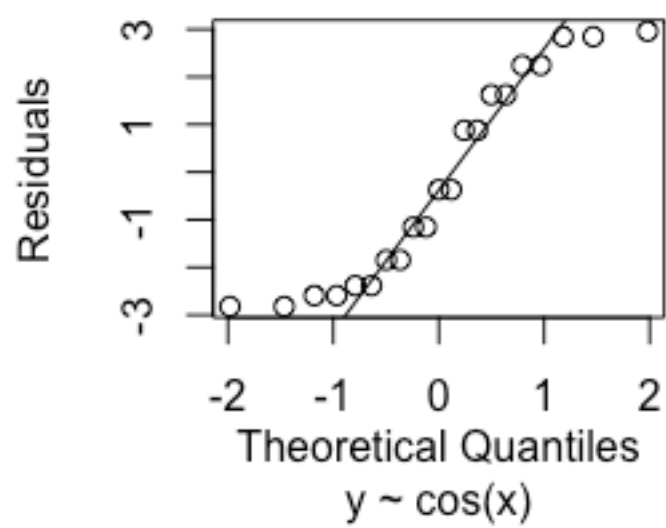
Residuals vs Fitted



Scale-Location

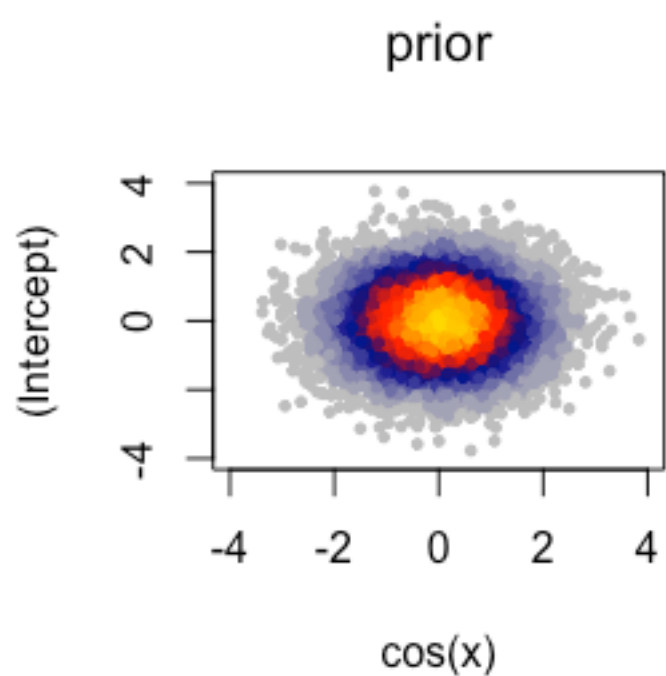


Normal Q-Q

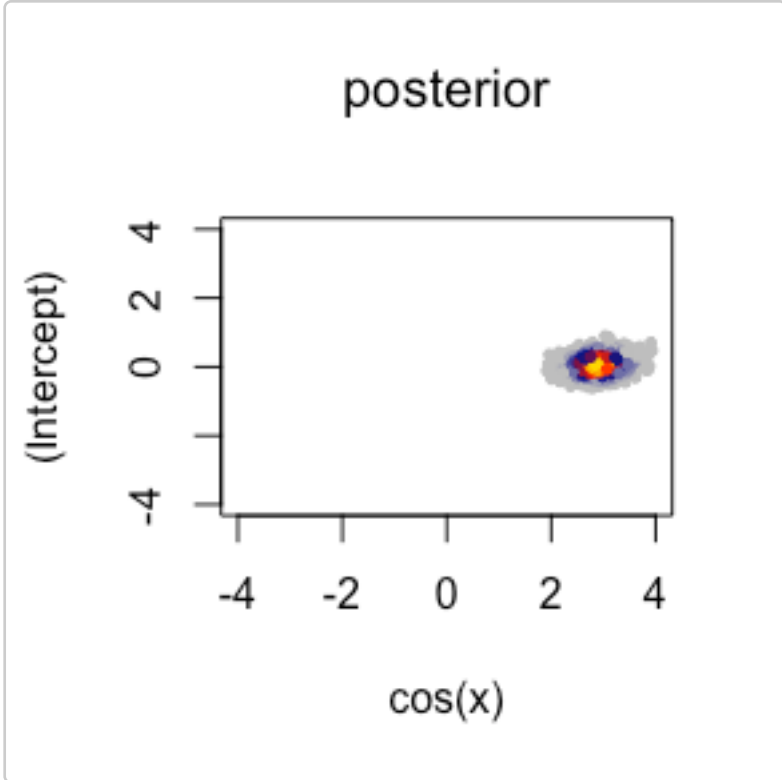


To visualize the distribution of the coefficients I also implemented a kernel density plot for *mvnd* objects:

```
kernel_density(model$prior, xlim=c(-4,4), ylim=c(-4,4), main = 'prior')
```



```
kernel_density(model$posterior, xlim=c(-4,4), ylim=c(-4,4), main = 'posterior')
```



Complex Models

Additional terms can be used as already seen above.

A model with a cosine term.

```
w0 <- .2
w1 <- 3
w2 <- 10
x <- seq(-100,100,1)
b <- 1.3
y <- w0 + w1*x + w2*cos(x) + rnorm(length(x), mean=0, sd=sqrt(1/b) )

mod <- blm(y ~ x + cos(x), prior = NULL, beta = b, data = data.frame(x=x, y=y))
summary(mod)
```

```
## Call:
## blm(formula = y ~ x + cos(x), prior = NULL, beta = b, data = data.frame(x = x,
##   y = y))
##
## ----
## Posterior Coefficients:
##
## Estimate:
## (Intercept)          x      cos(x)
##   0.3096717   2.9993262   9.9821406
##
## Covariance:
##           (Intercept)          x      cos(x)
## (Intercept) 3.812429e-03 0.000000e+00 2.431988e-06
## x           0.000000e+00 1.136737e-06 0.000000e+00
## cos(x)      2.431988e-06 0.000000e+00 7.598659e-03
##
## ----
## Prior Coefficients:
##
## Estimate:
## (Intercept)          x      cos(x)
##           0           0           0
##
## Covariance:
```

```
##           (Intercept) x cos(x)
## (Intercept)          1 0      0
## x                   0 1      0
## cos(x)               0 0      1
```

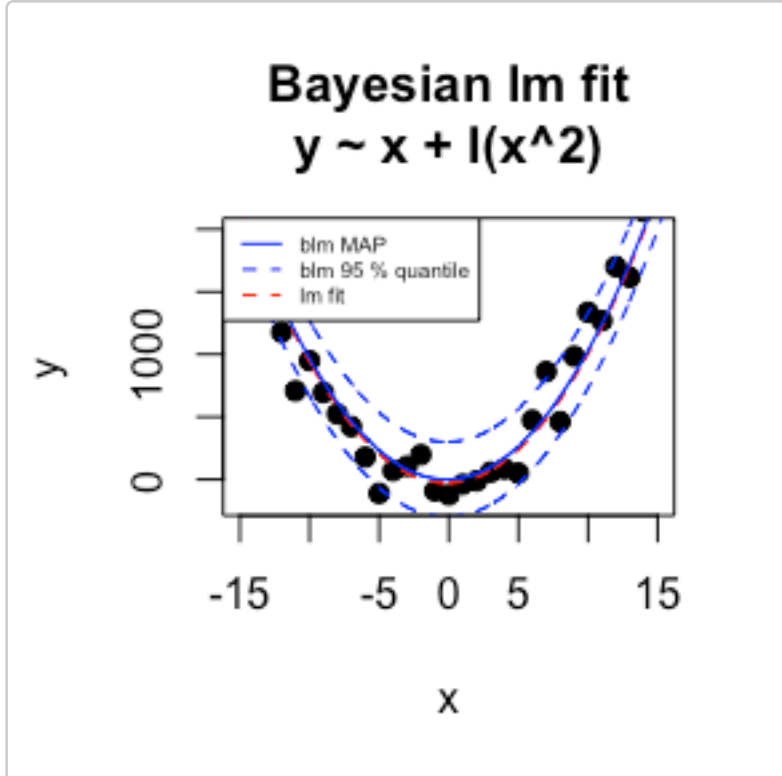
A model with a polynomial term.

```
w0 <- .2
w1 <- 3
w2 <- 10
x <- seq(-100,100,1)
b <- 0.00003
y <- w0 + w1*x + w2*x^2 + rnorm(length(x), mean=0, sd=sqrt(1/b) )

mod <- blm(y ~ x + I(x^2), prior = NULL, beta = b, data = data.frame(x=x, y=y))
summary(mod)
```

```
## Call:
## blm(formula = y ~ x + I(x^2), prior = NULL, beta = b, data = data.frame(x = x,
##   y = y))
##
## ----
## Posterior Coefficients:
##
## Estimate:
## (Intercept)          x          I(x^2)
## -0.07460305  3.06188624  9.99908252
##
## Covariance:
##           (Intercept)          x          I(x^2)
## (Intercept)  0.9973272461  0.000000000 -1.645795e-04
## x            0.0000000000  0.04694615  0.000000e+00
## I(x^2)       -0.0001645795  0.000000000  8.155853e-06
##
## ----
## Prior Coefficients:
##
## Estimate:
## (Intercept)          x          I(x^2)
##           0           0           0
##
## Covariance:
##           (Intercept) x I(x^2)
## (Intercept)          1 0      0
## x                   0 1      0
## I(x^2)              0 0      1
```

```
plot(mod, pch=19, xlim=c(-15,15), ylim=c(-200,2000), legend_parm=list(cex=.5))
```

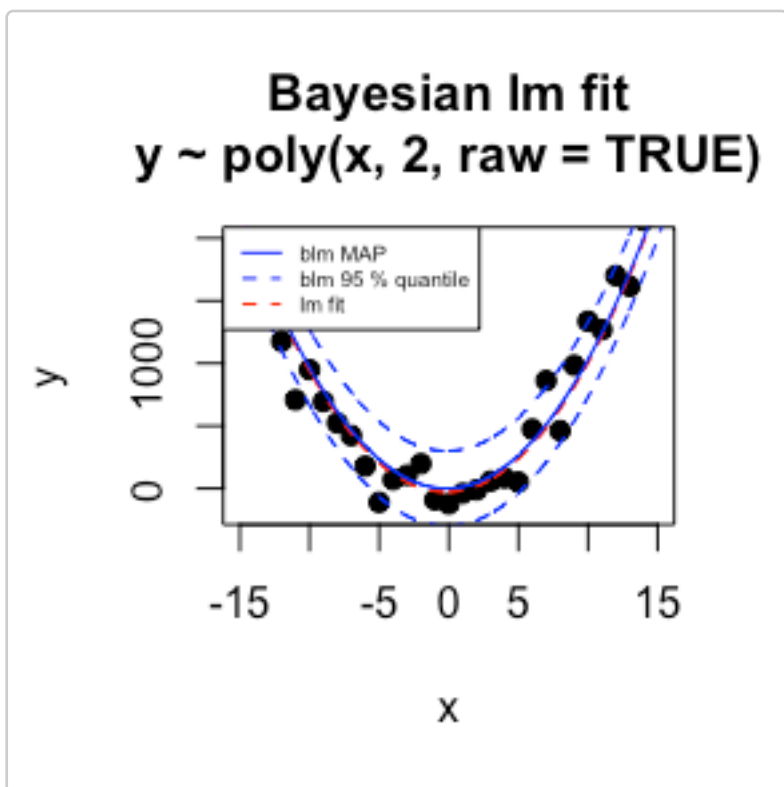



or also:

```
mod <- blm(y ~ poly(x,2, raw=TRUE), prior = NULL, beta = b, data = data.frame(x=x, y=y))
summary(mod)
```

```
## Call:
## blm(formula = y ~ poly(x, 2, raw = TRUE), prior = NULL, beta = b,
##      data = data.frame(x = x, y = y))
##
## ----
## Posterior Coefficients:
##
## Estimate:
##      (Intercept) poly(x, 2, raw = TRUE)1 poly(x, 2, raw = TRUE)2
##      -0.07460305      3.06188624      9.99908252
##
## Covariance:
##      (Intercept) poly(x, 2, raw = TRUE)1
## (Intercept)      0.9973272461      0.00000000
## poly(x, 2, raw = TRUE)1 0.0000000000      0.04694615
## poly(x, 2, raw = TRUE)2 -0.0001645795      0.00000000
##      poly(x, 2, raw = TRUE)2
## (Intercept)      -1.645795e-04
## poly(x, 2, raw = TRUE)1      0.000000e+00
## poly(x, 2, raw = TRUE)2      8.155853e-06
##
## ----
## Prior Coefficients:
##
## Estimate:
##      (Intercept) poly(x, 2, raw = TRUE)1 poly(x, 2, raw = TRUE)2
##      0      0      0
##
## Covariance:
##      (Intercept) poly(x, 2, raw = TRUE)1
## (Intercept)      1      0
## poly(x, 2, raw = TRUE)1      0      1
## poly(x, 2, raw = TRUE)2      0      0
##      poly(x, 2, raw = TRUE)2
## (Intercept)      0
## poly(x, 2, raw = TRUE)1      0
## poly(x, 2, raw = TRUE)2      1
```

```
plot(mod, explanatory='x',pch=19, xlim=c(-15,15), ylim=c(-200,2000), legend_parm=list(cex=.5))
```



'''

Model Analysis and Comparison

The parts below are of preliminary experimental nature and there are absolutely no guarantees for correct functioning.

Bayes Information Criterion (BIC)

Note, this does nothing special on a *blm* compared to an *lm* object. It simply computes:

$$BIC = \log \left(\frac{\sum RSS}{n} \right) * k * \log(n)$$

Where: *n* is the number of data points, *k* is the number of parameters and *RSS* are the squared residuals of the fit.

Bayes Factor

Bayes factor applies a likelihood test comparing the total probability of 2 models. The equation for the likelihood for a *blm* model fit is given in the introduction.

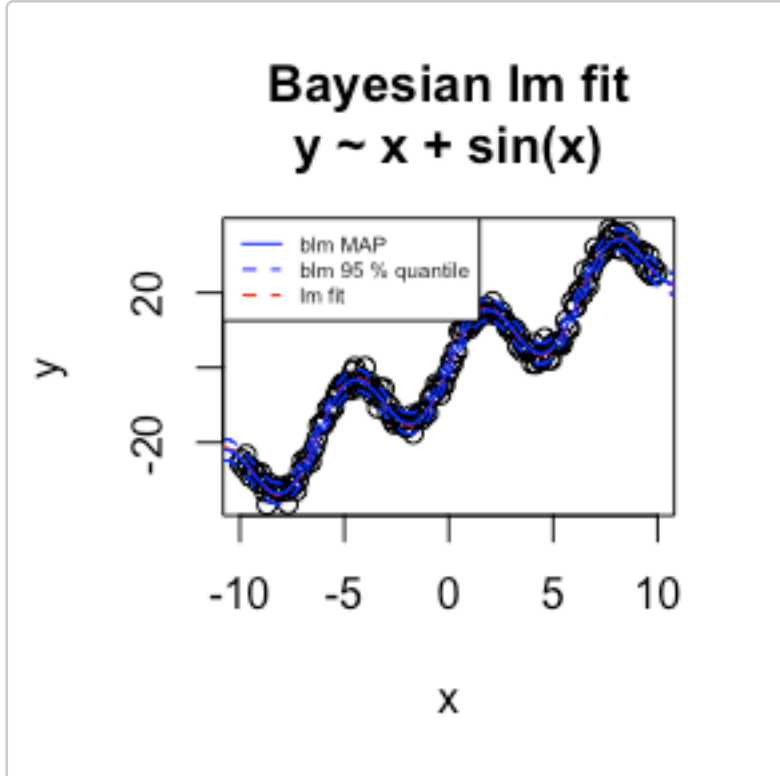
```
set.seed(1)
x <- seq(-10,10,.1)
b <- 0.3

w0 <- 0.2 ; w1 <- 3 ; w2 <- 10

y <- rnorm(201, mean = w0 + w1 * x + w2 *sin(x), sd = sqrt(1/b))
mod1 <- blm(y ~ x + sin(x))
bic(mod1)
```

```
## [1] 224.351
```

```
plot(mod1, xlim=c(-10,10), legend_parm=list(cex=.5))
```



compute this fit to another mod removing the sinus term, clearly less well fitting

```
mod2 <- blm(y ~ x)

bic(mod2)
```

```
## [1] 805.5729
```

The BIC for the second, much less well-fitting model is much higher and thus the BIC indicates a better fit for *mod1*. See *?bic* for more information on how to interpret the values.

Bayes factor comparing the 2 models

```
bayes_factor(mod1, mod2)
```

```
## [1] 97.31074
```

Very strong support. See *?bayes.factor* for more information on how to interpret the value.

Anova F-test comparing the 2 models

```
anova(mod1, mod2)
```

```
## Analysis of Variance Table:
##
## Model1: blm(formula = y ~ x + sin(x))
## Model2: blm(formula = y ~ x)
##
##           1           2
## Res.Df    198.0000   199.0000
## RSS       566.9807 10491.9246
## Df         NA        1.0000
## Sum.of.Sq    NA    9924.9439
## F          NA     189.1921
## pF          NA      0.0000
```

Strong support for mod1 over mod2.

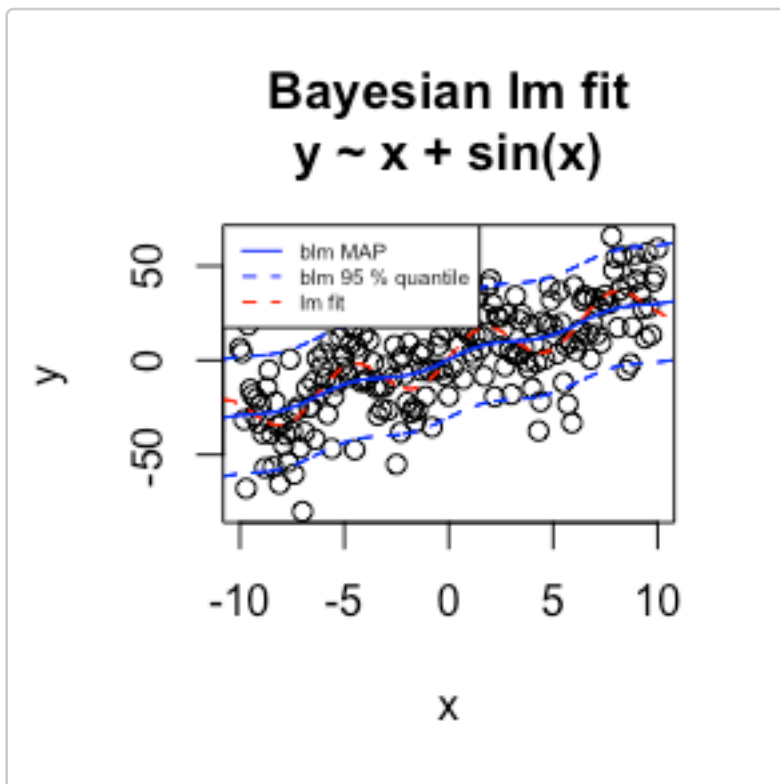
Compare less separated models

```
b <- 0.003
y <- rnorm(201, mean = w0 + w1 * x + w2 *sin(x), sd = sqrt(1/b))
mod1 <- blm(y ~ x + sin(x))
```

```
bic(mod1)
```

```
## [1] 1209.717
```

```
plot(mod1, xlim=c(-10,10), legend_parm=list(cex=.5))
```



```
mod2 <- blm(y ~ x)  
bic(mod2)
```

```
## [1] 1215.66
```

... bic indicates still some positive support for complex mod1, but not very strong.

Bayes factor comparing the 2 models

```
bayes_factor(mod1, mod2)
```

```
## [1] 1.24006
```

No significant support for mod1 over mod2.

Anova F-test comparing the 2 models

```
anova(mod1, mod2)
```

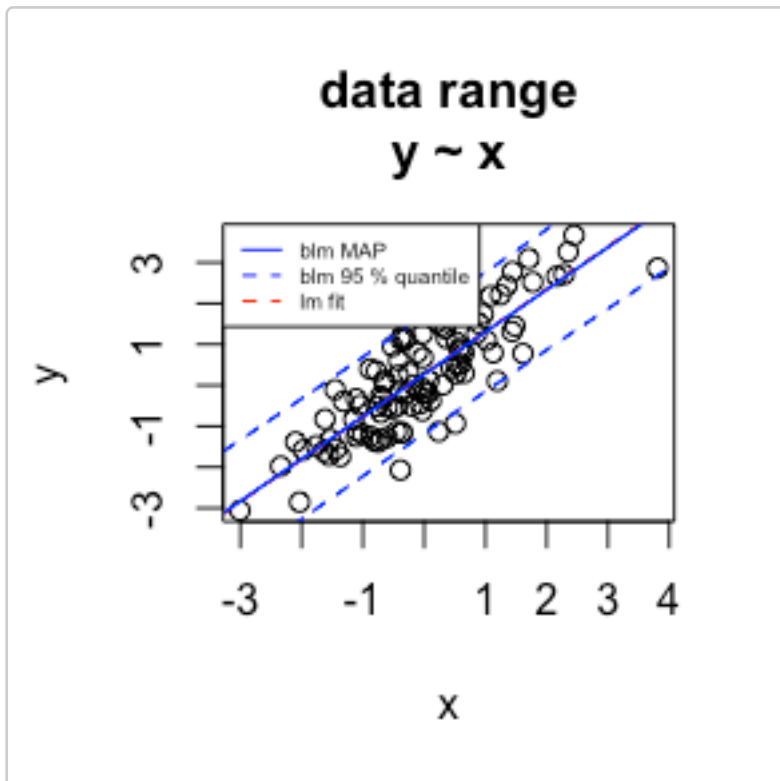
```
## Analysis of Variance Table:  
##  
## Model1: blm(formula = y ~ x + sin(x))  
## Model2: blm(formula = y ~ x)  
##  
##           1           2  
## Res.Df    198.0   199.00000  
## RSS      76316.5 80708.17808  
## Df         NA     1.00000  
## Sum.of.Sq    NA  4391.67976  
## F          NA    10.88286  
## pF         NA     0.00000
```

Still strong support for mod1 over mod2.

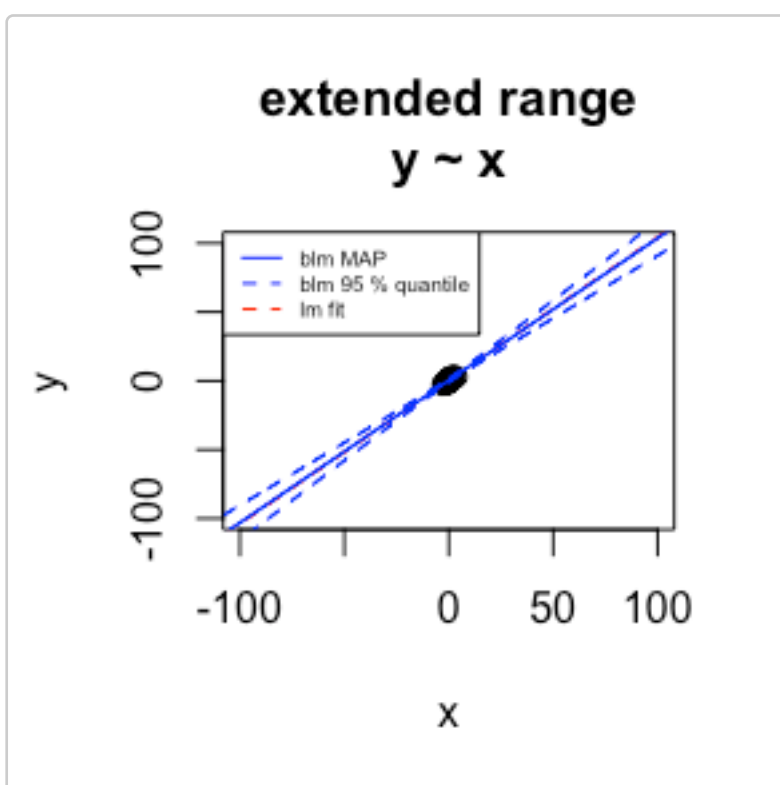
Some Examples that Illustrate the Behaviour of Bayesian Linear Models

Variance of Fitted Values increases with distance to the data

```
w0 <- 0.3 ; w1 <- 1.1 ; b <- 1.3
x <- rnorm(100)
y <- rnorm(100, w1 * x + w0, 1/b)
mod <- blm(y~x, beta=b, data=data.frame(x=x, y=y))
plot(mod, caption='data range', legend_parm=list(cex=.5))
```



```
plot(mod, xlim=c(-100,100), ylim=c(-100,100), caption='extended range', legend_parm=list(cex=.5))
```



updating the model with itself improves the fit

As criteria I measure the Mahalanobis distance between the coefficients of a model to the posterior distribution of another one.

```
mod2 <- update(mod)
mod3 <- update(mod2)
mod4 <- update(mod3)

mahal(mod4$posterior, coef(mod4))
```

```
## [1] 0
```

```
mahal(mod4$posterior, coef(mod3))
```

```
## [1] 0.0001984845
```

```
mahal(mod4$posterior, coef(mod2))
```

```
## [1] 0.001782608
```

```
mahal(mod4$posterior, coef(mod))
```

```
## [1] 0.01594281
```

This should also be reflected in decreasing deviance.

```
deviance(mod4)
```

```
## [1] 55.62441
```

```
deviance(mod3)
```

```
## [1] 55.62468
```

```
deviance(mod2)
```

```
## [1] 55.62544
```

```
deviance(mod)
```

```
## [1] 55.62953
```