Bayesian Additive Regression Trees for Multivariate skewed responses

Seungha Um

September 30, 2022

Introduction

This vignette demonstrates how to fit the skewBART model using the `skewBART` package. The small MCMC iterations and tree numbers are used to reduce computation time; in practice, larger values (say, 200 trees and 5000 MCMC iterations) should be used. Our model is an extension of BART to accommodate univariate and multivariate skewed responses.

First, we load the requisite packages:

```
## Load library
library(tidyverse) # Load the tidyverse, mainly for ggplot
#> -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
#> v ggplot2 3.3.6     v purrr   0.3.4
#> v tibble  3.1.8     v dplyr   1.0.10
#> v tidyr   1.2.0     v stringr 1.4.1
#> v readr   2.1.2     v forcats 0.5.2
#> -- Conflicts ------------------------------------------ tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
library(kableExtra) # For fancy tables
#>
#> Attaching package: 'kableExtra'
#>
#> The following object is masked from 'package:dplyr':
#>
#>     group_rows
library(skewBART) # Our package
#> Loading required package: Rcpp
library(zeallot) # For the %<-% operator, used when generating data
library(tictoc)
library(loo)
#> This is loo version 2.5.1
#> - Online documentation and vignettes at mc-stan.org/loo
#> - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar
options(knitr.table.format = 'markdown')
```

Univariate skewed response

We first illustrate on a simple example due to Friedman which takes

$$f(x) = 10\sin(\pi\, x_1\, x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5.$$

We set $Y_i = f(x_i) + \epsilon_i$ with ϵ_i have a skew-normal distribution with scale $\sigma = 3$ and shape $\alpha = 1$.

Setup

The following code creates a function to simulate data from the model.

```
## Create a function for Friedman's example
sim_fried <- function(N, P, alpha, sigma) {
  lambda <- alpha * sigma/sqrt(1+alpha^2)
  tau <- sigma/sqrt(1+alpha^2)
  X <- matrix(runif(N * P), nrow = N)
  mu <- 10 * sin(pi * X[,1] * X[,2]) + 20 * (X[,3] - 0.5)^2 + 10 * X[,4] + 5 * X[,5]
  Z <- abs(rnorm(N, mean=0, sd=1) )
  Y <- mu + lambda * Z + rnorm(N, mean=0, sd=sqrt(tau))
  EY <- mu + lambda * sqrt(2/pi)
  return(list(X = X, Y = Y, EY = EY, mu = mu, Z=Z, tau = tau, lambda = lambda))
}
```

We then create a training set and testing set:

```
## Traning dataset : n = 100 observations, P = 5 covariates, sigma = 1, alpha = 3
set.seed(12345)
c(X,Y,EY,mu,Z,tau,lambda) %<-% sim_fried(100, 5, 1, 3)
c(test_X,test_Y,test_EY,test_mu,test_Z,test_tau,test_lambda)  %<-% sim_fried(50, 5, 1 ,3)
```

Next, we create an object containing the hyperparameters for the model using the `UHypers` function (U is for univariate, as opposed to multivariate). This function uses default hyperparameters unless specific hyperparameters are provided by the user; see `?UHypers` for more details.

```
hypers <- UHypers(X, Y, num_tree = 20)
opts <- UOpts(num_burn = 100, num_save = 500)
```

The function `UOpts` produces a list of the parameters for running the Markov chain for univariate skewBART model. For illustration we use only 250 burn-in and save iterations; the defaults are 2500 for each. `UOpts` can also be used to control some features of the chain, such as whether hyperparameters are updated.

Fitting the skewBART model

After creating the hyperparameters and options, we fit the data by running

```
## Fit the model
tic("skewBART")
fitted_skewbart <- skewBART(X = X, Y = Y, test_X = test_X,
                            hypers = hypers, opts = opts)
#> Finishing iteration 100 of 600   Finishing iteration 200 of 600  Finishing iteration 300 of 600  Fin
toc()
#> skewBART: 5.294 sec elapsed
```

Alternatively, we can use the function `skewBART_parallel` running multiple `skewBART` threads in parallel as provided by the `parallel` package. The option `cores` specifies the number of cores to employ in multi-threading. The `skewBART_parallel` function is only available when you are running R on Mac OS or Linux or other variants of Unix.

```
tic("skewBART in parallel")
## parallel::mcparallel/mccollect do not exist on windows
if(.Platform$OS.type =='unix'){
fitted_skewbart_parallel <- skewBART_parallel(X = X, Y = Y, test_X = test_X,
                                hypers = hypers, opts = opts, cores = 5)
}
toc()
#> skewBART in parallel: 2.268 sec elapsed
```

Contents of the model fit

The fitted model is a list containing the following elements:

- `y_hat_train`: samples with `num_save` rows containing samples of the estimated mean for each observation in the training set.
- `y_hat_test`: same as `y_hat_train`, but for the testing set.
- `y_hat_train_mean`: the posterior mean of the predicted value of each observation on the training set; this includes both the prediction from the BART model and an offset due to the fact that the skew-normal distribution does not have mean 0.
- `y_hat_test_mean`: same as `y_hat_train_mean` but on the testing set.
- `f_hat_train` and `f_hat_test` are analogous to `y_hat_train` and `y_hat_test`, but just give the value of the BART function - for BART these are the same quantity, but for skewBART we need to account for the fact that the errors do not have mean 0.
- `sigma`: posterior samples of parameter σ in the skew-normal error.
- `tau`: posterior samples of parameter τ in the skew-normal error.
- `alpha`: posterior samples of parameter α in the skew-normal error.
- `lambda`: posterior samples of parameter λ in the skew-normal error.
- `likelihood_mat`: matrix of log-likelihood values to calculate the log pseudo marginal likelihood (LPML)

The parameters (λ, τ) and (α, σ) give equivalent descriptions of the skew-normal distribution, and are related by the fact that $\lambda = \alpha\sigma/\sqrt{1+\alpha^2}$ and $\tau = \sigma/\sqrt{1+\alpha^2}$; the MCMC is run on (λ, τ), but the skew-normal (with location parameter 0) is typically parameterized in terms of (α, σ) as

$$p(y) = \frac{2}{\sigma}\phi\left(\frac{y}{\sigma}\right)\,\Phi\left(\alpha\frac{y}{\sigma}\right).$$

So, for example, the posterior mean of the skewness level α is

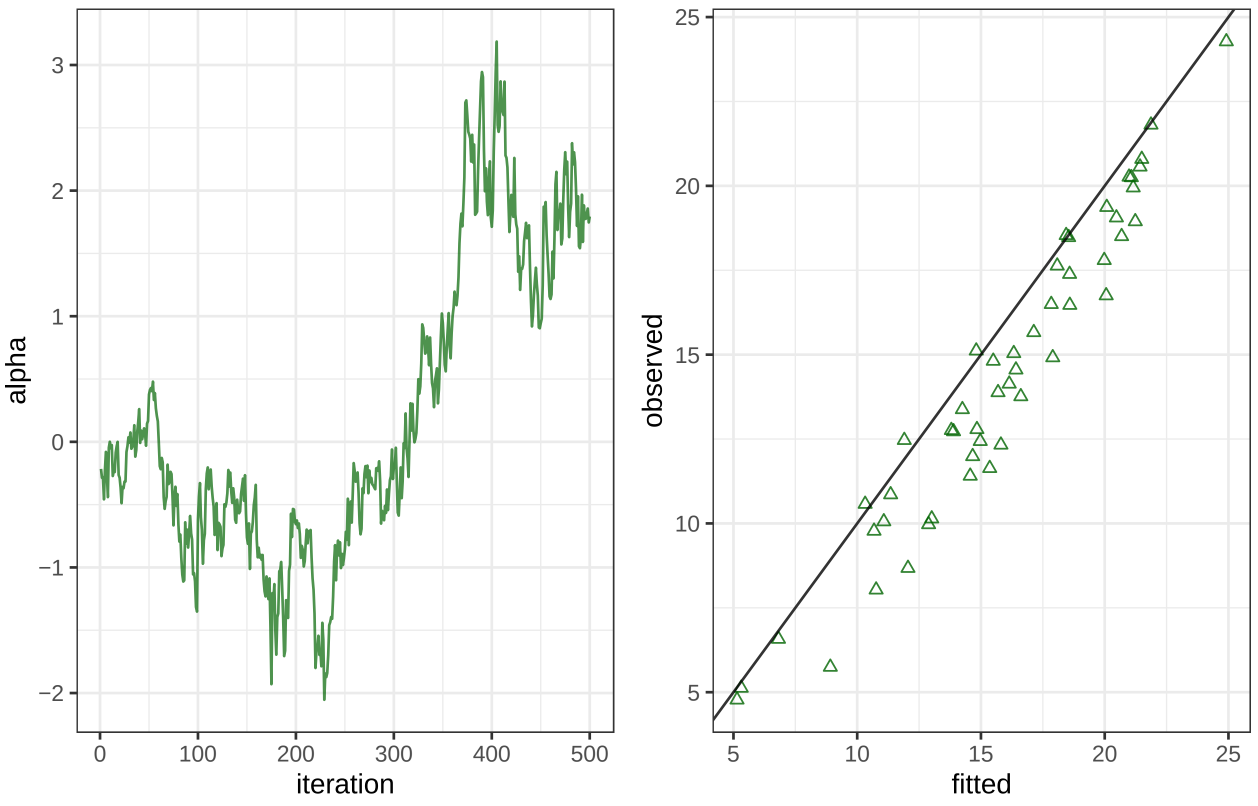
```
mean(fitted_skewbart$alpha)
#> [1] 0.2013032
```

while the posterior mean of the predicted values on the test data is

```
fitted_skewbart$y_hat_test_mean
#>  [1] 20.083390 18.683168 21.972975 14.668508 12.986792  6.921535 16.427648
#>  [8] 14.939619 20.185029 14.767237 20.575030 18.544929 20.785841 16.515828
#> [15] 18.003314 16.243862 17.235935 15.908795 13.905485 21.600218 17.942797
#> [22] 12.004828 18.184419 18.645795 21.086092 18.693026 15.792747 10.866910
#> [29] 16.712765 15.456408 14.351448 21.254320 15.074122 13.112995 13.993953
#> [36] 11.453415 10.422768 15.598937 21.177958  9.016295 25.016375 14.912531
#> [43] 12.153834 21.530420  5.416307 20.162168 11.177045 10.781952 21.338220
#> [50]  5.246441
```

Below, we assess the mixing of the model with a traceplot (left) and plot the fitted versus actual values (right) of $f(x)$.

```r
# create a dataframe for the estimated alpha
df <- data.frame(iteration = 1:length(fitted_skewbart$alpha), alpha=fitted_skewbart$alpha)
p1 <- ggplot(df) + geom_line(aes(iteration, alpha), color = "darkgreen", alpha=0.7) + theme_bw()

# create a dataframe for the fitted values
df_fitted <- data.frame(fitted = fitted_skewbart$f_hat_test_mean, observed = test_mu)
p2 <- ggplot(df_fitted) +
  geom_point(aes(fitted, observed), alpha=0.8, shape=2, color = "darkgreen") +
  geom_abline(intercept = 0, slope = 1, alpha=0.8) +
  theme_bw()
library(ggpubr)
ggarrange(p1, p2)
```



In our manuscript, log pseudo marginal likelihood (LPML) is computed to evaluate the model performance. Using the `loo` package, the LPML can be conveniently computed after obtaining the MCMC samples from the joint posterior.

```r
loo(fitted_skewbart$likelihood_mat, r_eff = NA)
#> Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
#>
#> Computed from 500 by 100 log-likelihood matrix
#>
#>          Estimate   SE
#> elpd_loo   -217.7  7.1
#> p_loo        32.4  4.0
#> looic       435.3 14.2
#> ------
#> Monte Carlo SE of elpd_loo is NA.
#>
#> Pareto k diagnostic values:
#>                          Count Pct.    Min. n_eff
#> (-Inf, 0.5]   (good)     72    72.0%   110
#>  (0.5, 0.7]   (ok)       17    17.0%   63
#>    (0.7, 1]   (bad)       8     8.0%   19
#>    (1, Inf)   (very bad)  3     3.0%   15
#> See help('pareto-k-diagnostic') for details.
```

LPML is obtained by the estimate of expected log pointwise predictive density (elpd_loo). You may have the warning message regarding Pareto k diagnostic values. In the `loo` package, LPML is computed using Pareto smoothed importance sampling and reliability of estimates is evaluated using the shape parameter k. In practice, good performance has been observed for values of k up to 0.7. See help('pareto-k-diagnostic') for details in the `loo` package. Note that this vignette is demonstrated under the small MCMC iterations and tree numbers to reduce computation time. To get k > 0.7, larger values of MCMC iterateions and tree numbers should be used.

Also, the root mean square error can be computed with the posterior mean of the predicted values;

```
rmse <- function(x,y) sqrt(mean((x-y)^2))
rmse(test_Y, fitted_skewbart$y_hat_test_mean)
#> [1] 1.951398
```

Multivariate skewed response

We extend Friedman's example to the bivariate setting by taking

$$Y_{ij} = f(x_i) + \epsilon_{ij}$$

where $\epsilon_i \equiv (\epsilon_{i1}, \epsilon_{i2})$ has a multivariate skew-normal distribution. Equivalently, we assume that ϵ_{ij} can be written as

$$\epsilon_{ij} = \lambda_j \, Z_{ij} + \varepsilon_{ij}$$

where $(\varepsilon_{i1}, \varepsilon_{i2})^\top$ follows a multivariate normal distribution with mean 0 and covariance matrix Σ, while the Z_{ij}'s are independent truncated standard normal random variables. We take $\lambda_1 = 1, \lambda_2 = 3$ and Σ a correlation matrix with correlation $\rho = 0.5$.

Setup

The following code creates a function for generating the data.

```
## Create a function for Friedman example with multivariate framework
sim_data_multi <- function(N, P, lambda, tau, rho) {
  X <- matrix(runif(N * P), nrow = N)
  mu <- 10 * sin(pi * X[,1] * X[,2]) + 20 * (X[,3] - 0.5)^2 + 10 * X[,4] + 5 * X[,5]
  Z <- cbind(lambda[1] * abs(rnorm(N)), lambda[2] * abs(rnorm(N)))
  Sigma <- matrix(c(tau[1], sqrt(tau[1]*tau[2])*rho, sqrt(tau[1]*tau[2])*rho, tau[2]), 2, 2)
  Err <- MASS::mvrnorm(n=N, mu=c(0,0), Sigma = Sigma)
  Y <- cbind(mu, mu) + Z + Err
  EY <- rbind(mu, mu) + lambda * sqrt(2/pi)
  return( list(X = X, Y = Y, EY=EY, mu = mu, lambda = lambda, tau=tau, Z= Z, Sigma = Sigma) )
}
```

We then create the training and testing sets.

```
## Simulate dataset
## Traning dataset : n = 100 observations, P = 5 covariates,
## lambda = (2,3), tau = c(1,1), rho = 0.5.
set.seed(12345)
c(X,Y,EY,mu,lambda,tau,Z,Sigma) %<-% sim_data_multi(100, 5, c(2,3), c(1,1), 0.5)
c(test_X,test_Y,test_EY,test_mu,test_lambda,test_tau,test_Z,test_Sigma) %<-%
  sim_data_multi(50, 5, c(2,3), c(1,1), 0.5)
```

The functions `Hypers` and `Opts` are the multivariate extensions of `UHypers` and `UOpts`. We use these to set the hyperparameters and MCMC options.

```
## Create a list of the hyperparameters of the model.
hypers <- Hypers(X = X, Y = Y, num_tree = 20)
opts <- Opts(num_burn = 50, num_save = 100)
```

Fitting the multi-skewBART model

The model is fit using the `MultiskewBART` function:

```
tic("MultiskewBART")
fitted_Multiskewbart <- MultiskewBART(X = X, Y = Y, test_X = test_X, hypers=hypers, opts=opts)
#> Finishing iteration 100 of 150
toc()
#> MultiskewBART: 15.293 sec elapsed
```

Alternatively, we can use the function `MultiskewBART_parallel` running multiple `MultiskewBART` threads in parallel as provided by the `parallel`. The option `cores` specifies the number of cores to employ in multi-threading. The `MultiskewBART_parallel` function is only available when you are running R on Mac OS or Linux or other variants of Unix.

```
tic("MultiskewBART in parallel")
## parallel::mcparallel/mccollect do not exist on windows
if(.Platform$OS.type =='unix'){
fitted_Multi_parallel <- MultiskewBART_parallel(X = X, Y = Y, test_X = test_X,
                                                hypers = hypers, opts = opts,
                                                cores = 5)
}
toc()
#> MultiskewBART in parallel: 7.504 sec elapsed
```

Contents of the Model Fit

The model fit contains quantities analogous to a univariate skewBART fit, but now the dimension of the objects is different. For example. `y_hat_train` is now an $N \times 2\times$ `num_save` array, Σ is a $2 \times 2\times$ `num_save` array, and so forth. We can get the posterior mean of λ_1 and λ_2 by computing

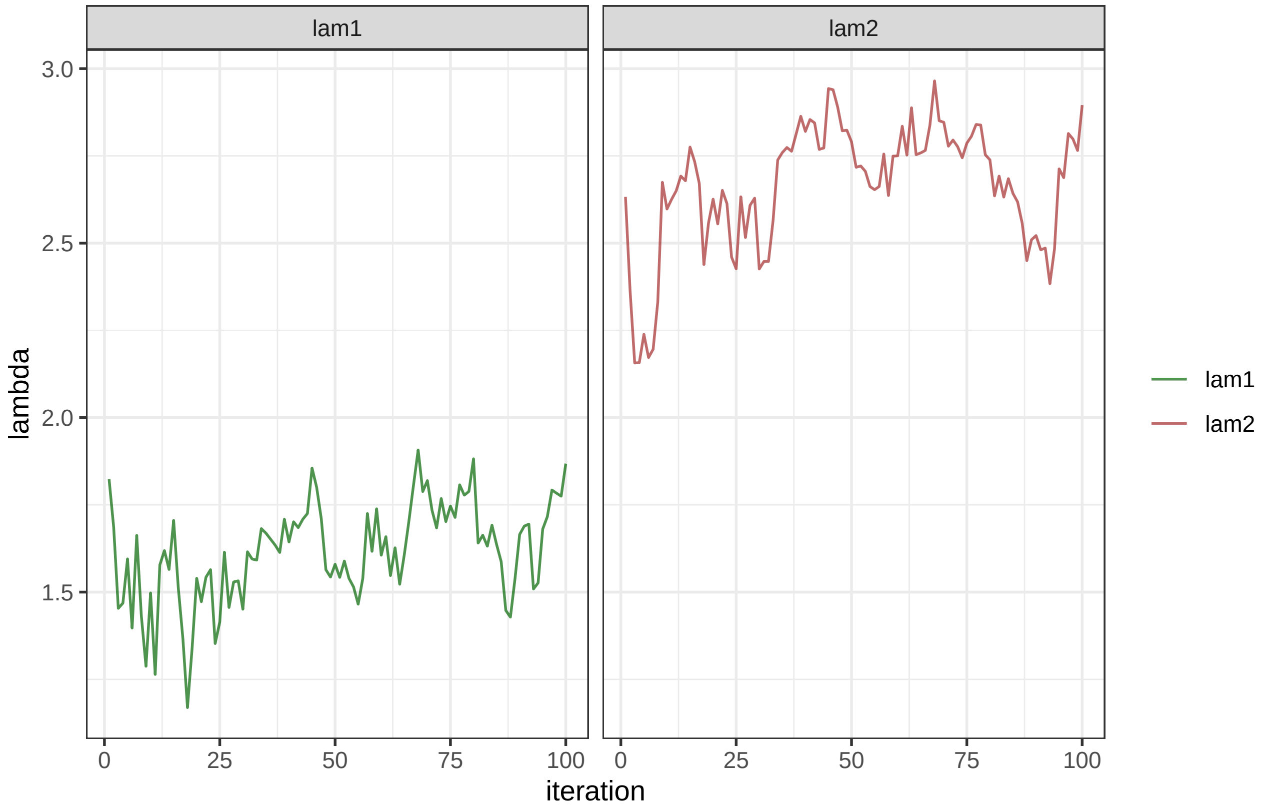
```
mean(fitted_Multiskewbart$lambda[,1])
#> [1] 1.616041
mean(fitted_Multiskewbart$lambda[,2])
#> [1] 2.664963
```

while the posterior mean of the function fit to the testing data is given by

```r
head(fitted_Multiskewbart$f_hat_test_mean)
#>           [,1]     [,2]
#> [1,]  7.457679  6.17290
#> [2,] 23.134296 22.90197
#> [3,] 12.588296 10.90806
#> [4,] 19.997914 19.38693
#> [5,] 23.041064 22.38228
#> [6,] 19.309341 17.95018
```

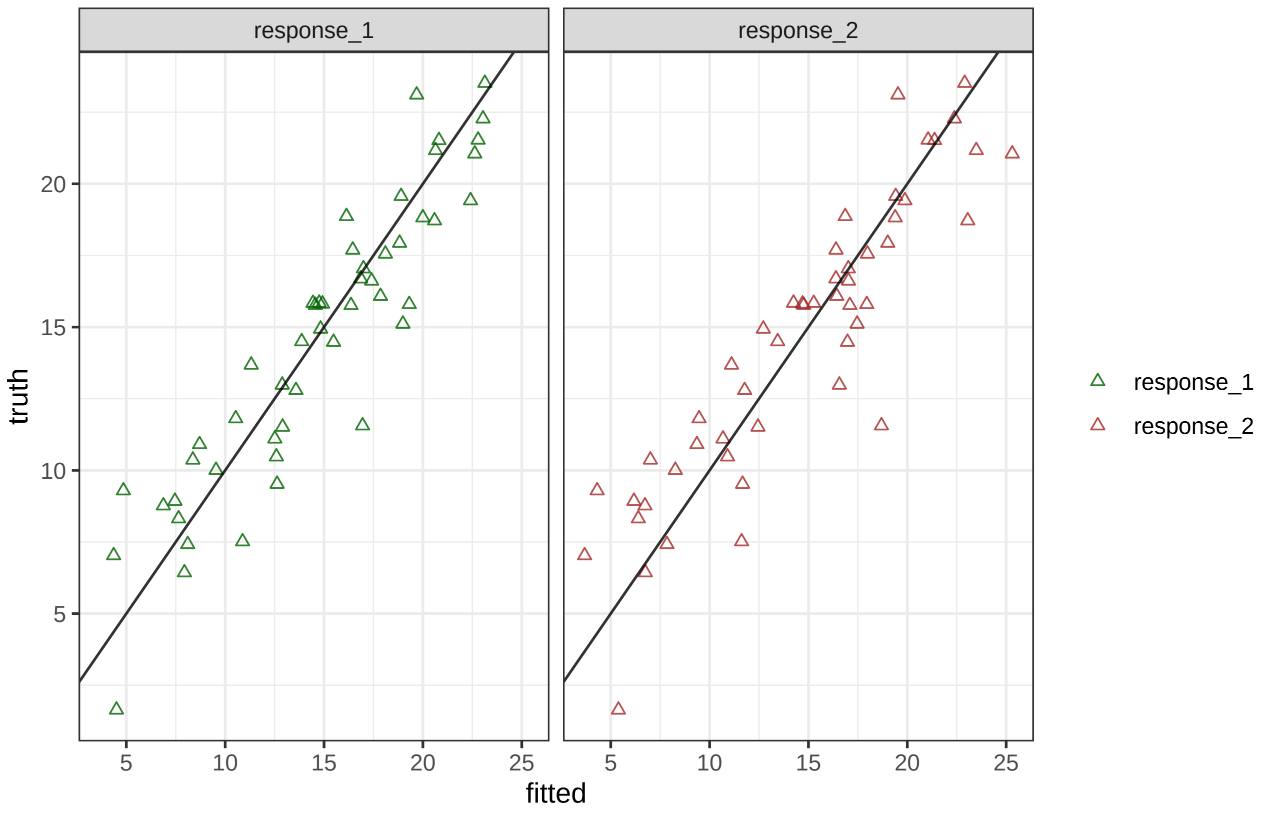
As before, we can examine the mixing of the skewness parameters (λ_1, λ_2) by running

```r
# create a dataframe for estimated skewness levels (lambda)
df <- data.frame(iteration = 1:length(fitted_Multiskewbart$lambda[,1]),
                 lambda = c(fitted_Multiskewbart$lambda[,1], fitted_Multiskewbart$lambda[,2]),
                 grp = rep(c("lam1","lam2"), each=length(fitted_Multiskewbart$lambda[,1])))

ggplot(df) + geom_line(aes(iteration, lambda, colour=grp), alpha=0.7) +
  theme_bw() + facet_wrap(~grp) +
  scale_colour_manual(values=c(lam1 = "darkgreen", lam2= "brown")) +
  theme(legend.title=element_blank())
```



and plot of the fitted vs actual values by running

```r
df_fitted <- data.frame(fitted = c(fitted_Multiskewbart$f_hat_test_mean),
                        truth = rep(test_mu, 2),
                        grp = rep(c("response_1", "response_2"),each = length(test_mu)))

ggplot(df_fitted) + geom_point(aes(fitted, truth, colour=grp), alpha=0.8, shape=2) +
  geom_abline(intercept = 0, slope = 1, alpha=0.8) +
  scale_colour_manual(values=c(response_1 = "darkgreen", response_2= "brown")) +
  theme_bw() + facet_wrap(~grp) + theme(legend.title=element_blank())
```



The log pseudo marginal likelihood (LPML) can be computed using `likelihood_mat`.

```
loo(fitted_Multiskewbart$likelihood_mat, r_eff = NA)
#> Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
#>
#> Computed from 100 by 100 log-likelihood matrix
#>
#>          Estimate   SE
#> elpd_loo   -419.7  9.1
#> p_loo        34.0  3.6
#> looic       839.4 18.3
#> ------
#> Monte Carlo SE of elpd_loo is NA.
#>
#> Pareto k diagnostic values:
#>                          Count Pct.    Min. n_eff
#> (-Inf, 0.5]   (good)     71    71.0%   21
#>  (0.5, 0.7]   (ok)       18    18.0%   27
#>    (0.7, 1]   (bad)      10    10.0%   7
#>    (1, Inf)   (very bad)  1     1.0%   19
#> See help('pareto-k-diagnostic') for details.
```

Also, the root mean square error can be computed with the posterior mean of the predicted values;

```
rmse <- function(x,y) sqrt(mean((x-y)^2))
rmse(test_Y, fitted_Multiskewbart$y_hat_test_mean)
#> [1] 2.604758
```

One may wish to use the multi-BART, the multivariate version of the standard BART model. In this case, set the option `do_skew` to FALSE.

```r
fitted_Multiskewbart <- MultiskewBART(X = X, Y = Y, test_X = test_X,
                                      do_skew = FALSE, hypers=hypers, opts=opts)
```