rbart: Bayesian Tree Modeling

Hugh Chipman, Edward George, Robert McCulloch, and Matthew Pratola

Contents

1 HBART: Bayesian Ensemble Modeling for mean and variance **1**

2 Markov Chain Monte Carlo (MCMC) **2**

3 Simulated Data **2**

4 Run rbart on the simulated data **4**
4.1 `rbart` . . . 4
4.2 `predict.rbart` . . . 5

5 Examine the model inference on the simulated data **6**
5.1 Plot posterior means of $f(x)$ and $s(x)$ . . . 6
5.2 Inference for f . . . 7
5.3 Inference for s . . . 8
5.4 Checking for Fit and Heteroskedasticity: `plotFunctionDraws` . . . 9
5.5 Predictive quantile-quantile plot: `hbartqqplot` . . . 10

6 The used cars data **11**
6.1 Read in the Cars data and run rbart: `rbartModelMatrix` . . . 11
6.2 Checking for Heteroskedasticity . . . 14
6.3 Predictive quantile-quantile plot . . . 15

7 BART from rbart, more `rbart` parameters **15**

8 Checking MCMC burn in **17**

---

1 HBART: Bayesian Ensemble Modeling for mean and variance

The `R` package `rbart` provides Bayesian tree ensemble modeling for predictor dependent mean and variance.

The basic model is:

$$Y = f(x) + s(x)\,Z, \quad Z \sim N(0,1)$$

The functions f and s may depend on a high dimensional x.

The function f is modeled as the sum of many trees and the function s is modeled as the product of trees:

$$f(x) = \sum_{i=1}^{m} f(x; T_i, M_i)$$

$$s(x) = \prod_{i=1}^{m'} s(x; \mathcal{T}_i, S_i)$$

`rbart` is the core function in the package `rbart`.

First we will try `rbart` on simple simulated data with just one variable in x.
With just one variable in x, simple plots suffice to display the mean and variance functions underlying the data.

Then we will try some real data with several variables (the cars data).

We will refer to our model as HBART, for heteroskedastic BART.
BART simply models the errors as iid normal.

---

2 Markov Chain Monte Carlo (MCMC)

`rbart` runs a Markov Chain Monte Carlo (MCMC) algorithm. At each MCMC iteration we have a draw of all the trees making up f and a draw of all the trees making up s.
We can think of ourselves as having draws of the functions f and s at each iteration.

The idea is that after initial burn-in the draws represent the posterior

$$(f, s) \,|\, x, y$$

where (x, y) is the training data and (f, s) are the two functions.

The trees themselves are not very interpretable so typically we will look at draws of f and s evaluated at particular x values.

We look at

$$\{f_d(x)\}, \;\; \{s_d(x)\}, \;\; d = 1, 2, \ldots D$$

where d indexes draws 1 to D.

So, for example, a prediction for y given $x = \tilde{x}$ could be the estimate of the posterior mean $\hat{f}(\tilde{x}) \approx \frac{1}{d} \sum_{i=1}^{D} f_d(\tilde{x})$.

Inference for $f(\tilde{x})$ could be summarized by quantiles of the draws $\{f_d(\tilde{x})\}$. Similarly for $s(\tilde{x})$.

Draws from the predictive distribution of Y given $x = \tilde{x}$ are given by $f_d(\tilde{x}) + s_d(\tilde{x})\, Z_d$ where the $Z_d \sim N(0, 1)$, *i.i.d.*

---

3 Simulated Data

Let's simulate some simple data to illustrate `rbart` with.

We will use just one variable in x and let:

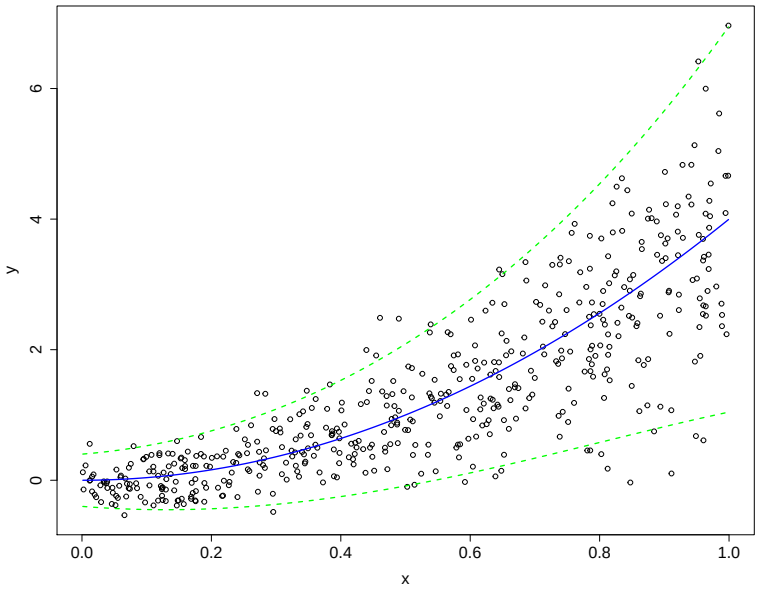
$$f(x) = 4x^2, \;\; s(x) = .2 \exp(2x)$$

Simulate the data:

```
# y = f(x) + s(x) Z

#basic parameters
set.seed(27)
n=500 #train (and test) sample sizes
p=1 #just one x

# train data
x = matrix(sort(runif(n*p)),ncol=p) #iid uniform x values
fx = 4*(x[,1]^2) #quadratric function f
sx = .2*exp(2*x[,1]) # exponential function s
y = fx + sx*rnorm(n)

##test data (the p added to the variable names is for predict)
np=1000
xp = matrix(sort(runif(np*p)),ncol=p)
fxp = 4*(xp[,1]^2)
sxp = .2*exp(2*xp[,1])
yp = fxp + sxp*rnorm(n)
```

Now, let's have a look at the simulated data:

```
plot(x,y,ylab="y",cex.axis=1.5,cex.lab=1.5)
lines(x,fx,col="blue",lwd=2)
lines(x,fx+2*sx,col="green",lwd=2,lty=2)
lines(x,fx-2*sx,col="green",lwd=2,lty=2)
```



The solid blue line is x *versus* $f(x)$ and the dashed green lines are x *versus* $f(x) \pm 2s(x)$.

So, we have a nice simple example with a nonlinear mean and heteroskedasticity.

---

4 Run rbart on the simulated data

4.1 rbart

Let's run `rbart` on the simulated data.

We will run `rbart` twice.
First we will use the default settings (`resdef` below).
Second we will change the defaults to take advantage of the fact that we know we are estimating a nice smooth function and to make the function return faster by reducing the number of MCMC iterations. In the second run (`res` below) our choices of `nskip`, `ndpost`, and `nadapt` reduce the number of iterations relative to the default values while the choice of `k`=5 smooths the function. The parameters are explained below.

```
library(rbart)
set.seed(99)

# first run at default settings
resdef = rbart(x,y)

# second run, this setting will give us a smoother function (k=5)
#  and use fewer iterations at each MCMC stage so that it runs faster
res = rbart(x,y,nskip=100,ndpost=400,k=5,numcut=1000,nadapt=200,adaptevery=20,tc=5)
```

The `rbart` MCMC runs in three stages of MCMC iterations.

In the first stage, the parameters determining how the MCMC is run are tuned to improve performance:

- **nadapt** : number of tuning MCMC iterations in the first stage.
- **adaptevery** : every *adaptevery* draws (out of the *nadapt* first stage tuning draws) update the MCMC parameters.

The second and third stage use fixed values of the MCMC parameters so that we have a legitimate Markov Chain. The second stage is a set of "burn-in" draws which are not kept. The final, third set of iterations are the draws we keep to represent our MCMC inference.

- **nskip** : number of burn-in draws in the second stage.
- **ndpost** : number of kept draws in the third stage.

Other arguments used in the second `rbart` call above are:

- **k** : shrinkage parameter for bottom node mean parameters for the trees in the f ensemble sum. Bigger k means more shrinkage so you can get a smoother function.
- **numcut** : number of cutpoints c to be used in the decision rules $x_j < c$ in each tree.
- **tc** : thread count, number of openmp threads to run. To take advantage of multiple cores, your system must support openmp. If unsupported, rbart will run on a single core.

As usual, `rbart` returns a list.

Most of the components the list returned by `rbart` are not very useful by themselves.
We will call a predict method to get the inference conditional on test values for x.

Here is the structure of the list returned by `rbart`:

```
str(res)
## List of 16
##  $ ots    :<externalptr>
##  $ oid    :<externalptr>
##  $ ovar   :<externalptr>
```

```
##  $ oc     :<externalptr>
##  $ otheta :<externalptr>
##  $ sts    :<externalptr>
##  $ sid    :<externalptr>
##  $ svar   :<externalptr>
##  $ sc     :<externalptr>
##  $ stheta :<externalptr>
##  $ x.train: num [1:500, 1] 0.00136 0.00245 0.00555 0.01194 0.01198 ...
##  $ y.train: num [1:500] 0.12261 -0.14139 0.22727 -0.00631 0.55786 ...
##  $ ntree  : num 200
##  $ ntreeh : num 40
##  $ ndpost : num 400
##  $ xicuts :List of 1
##   ..$ : num [1:1000] 0.00235 0.00335 0.00435 0.00534 0.00634 ...
##   ..- attr(*, "class")= chr "BARTcutinfo"
##  - attr(*, "class")= chr "rbart"
```

All of the "external pointers" reference information that will be used by the predict function.

Notable and understandable components of the list returned by `rbart` are:

- **ndpost** : number of kept posterior MCMC draws.
- **ntree** : the number of trees used in the ensemble sum for f.
- **ntreeh** : the number of trees used in the ensemble product for s.
- **xicuts** : these are the cutpoints c used in constructing the decision rules of the form $x < c$ in each individual tree.

Since we only have one x variable, xicuts is a list of length 1, whose only element gives the cutpoints used with our single x.
With p variables in x, *xicuts* would be a list of length p, where the j^{th} list element gives the cutpoints used with x_j. The user can input a list of cutpoints in the call to `rbart`. The default behaviour uses **numcut** cutpoints equally spaced over the range of an x_j.

4.2 predict.rbart

Now we call the predict function using the lists `resdef` and `res` returned from `rbart` and the test data x_p.

```
resdefp = predict(resdef,x.test=xp) #get prediction for test x in xp, using resdef
resp = predict(res,x.test=xp) #get prediction for test x in xp, using res
```

Here is the structure of the list returned by predict.rbart:

```
names(resp)
```

```
##  [1] "mdraws"  "sdraws"  "mmean"   "smean"   "msd"     "ssd"     "m.5"
##  [8] "m.lower" "m.upper" "s.5"     "s.lower" "s.upper" "q.lower" "q.upper"
```

```
str(resp)
```

```
## List of 14
##  $ mdraws : num [1:400, 1:1000] 0.1473 -0.0137 0.0515 0.0391 -0.0159 ...
##  $ sdraws : num [1:400, 1:1000] 0.259 0.333 0.335 0.3 0.303 ...
##  $ mmean  : num [1:1000] 0.0299 0.0299 0.0299 0.0299 0.0299 ...
##  $ smean  : num [1:1000] 0.312 0.312 0.312 0.312 0.312 ...
##  $ msd    : num [1:1000] 0.0979 0.0979 0.0979 0.0979 0.0979 ...
##  $ ssd    : num [1:1000] 0.0358 0.0358 0.0358 0.0358 0.0358 ...
##  $ m.5    : num [1:1000] 0.0311 0.0311 0.0311 0.0311 0.0311 ...
```

```
##  $ m.lower: num [1:1000] -0.156 -0.156 -0.156 -0.156 -0.156 ...
##  $ m.upper: num [1:1000] 0.226 0.226 0.226 0.226 0.226 ...
##  $ s.5    : num [1:1000] 0.309 0.309 0.309 0.309 0.309 ...
##  $ s.lower: num [1:1000] 0.253 0.253 0.253 0.253 0.253 ...
##  $ s.upper: num [1:1000] 0.385 0.385 0.385 0.385 0.385 ...
##  $ q.lower: num 0.025
##  $ q.upper: num 0.975
```

```
summary(resp$mmean-apply(resp$mdraws,2,mean))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0       0       0       0       0
```

Let nd be the number of kept draws (ndpost) and np be the number of test observations.

The two key components of resp are:

- **mdraws** : nd x np matrix. (d, j) element if the d^{th} draw of the function f evaluated at the j^{th} test x.
- **sdraws** : nd x np matrix. (d, j) element if the d^{th} draw of the function s evaluated at the j^{th} test x.

The rest of the components are convenience summaries of the np columns of mdraws and sdraws.
For example we see above the mmean is just the average over rows (MCMC draws) of the f evaluation on the test x.

- **mmean** and **smean** : posterior means of $f(x)$ and $s(x)$ for test x.
- **msd** and **ssd** : posterior standard deviations of $f(x)$ and $s(x)$ for test x.
- **q.lower**, **q.upper**: upper and lower quantiles used in summaries.
- **m.lower**, **m.5**, **m.upper** : q.lower, q.upper and .5 quantiles of mdraws columns.
- **s.lower**, **s.5**, **s.upper** : q.lower, q.upper and .5 quantiles of sdraws columns.

---

5 Examine the model inference on the simulated data

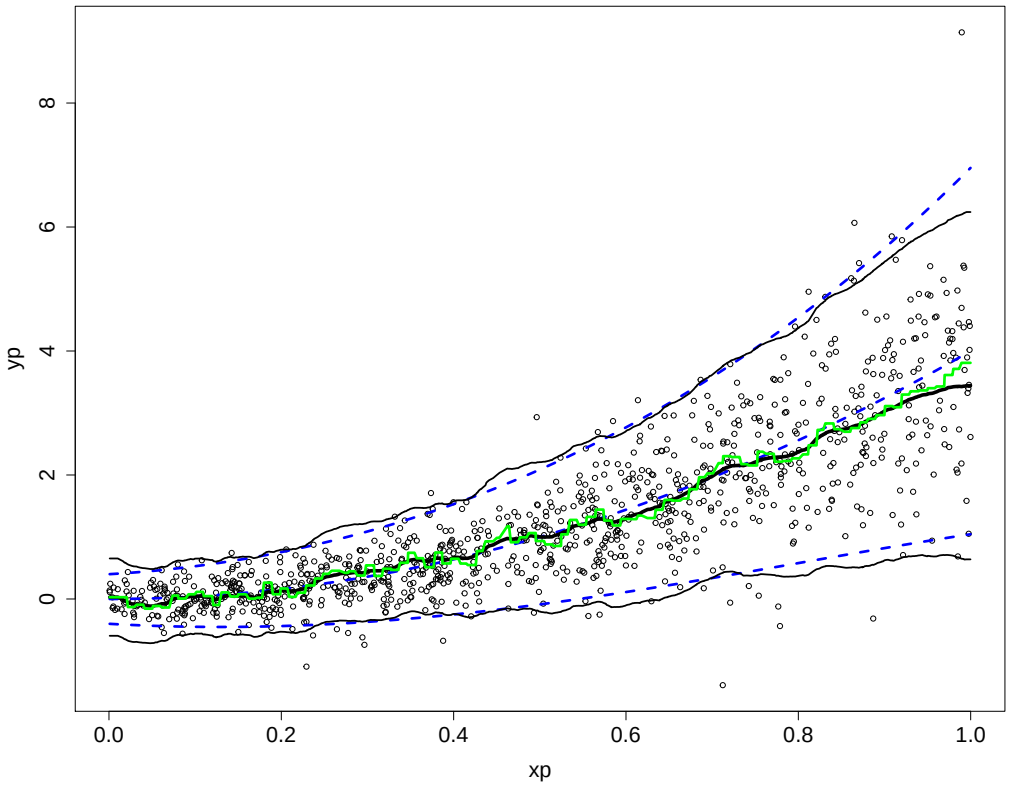
5.1 Plot posterior means of $f(x)$ and $s(x)$

First let's look at the posterior means of $f(x)$ and $s(x)$ at the test x values.

We'll plot the data, the true $f(x) \pm 2\,s(x)$ and the estmated $\hat{f}(x) \pm 2\,\hat{s}(x)$ where the "hats" indicate the posterior means.

The posterior means are returned in the list components **mmean** and **smean**.

```
#test data
plot(xp,yp,pch=1,cex=.8,cex.axis=1.5,cex.lab=1.5) #plot data

##true
lines(xp,fxp,col="blue",lwd=3,lty=2) #true f
lines(xp,fxp+2*sxp,col="blue",lwd=3,lty=2) #true f + 2s
lines(xp,fxp-2*sxp,col="blue",lwd=3,lty=2) #true f - 2s

##estimated
lines(xp,resp$mmean,col="black",lty=1,lwd=4) #estimate of f
lines(xp,resp$mmean+2*resp$smean,col="black",lty=1,lwd=2) #fhat + 2 shat
lines(xp,resp$mmean-2*resp$smean,col="black",lty=1,lwd=2) #fhat - 2 shat
lines(xp,resdefp$mmean,col="green",lty=1,lwd=3) #estimate of f from the default setting
```



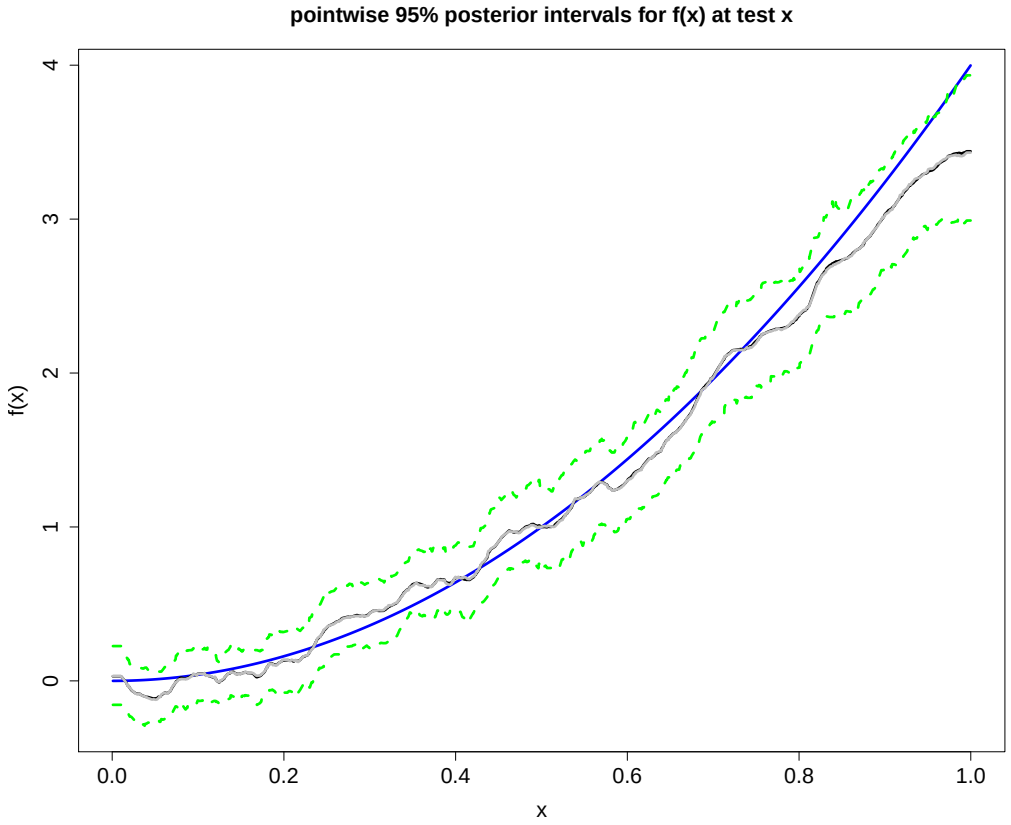
The posterior means of both $f(x)$ and $s(x)$ do a nice job of tracking the true values.

Notice that the posterior mean estimates from the default run (resdefp, solid green line) and the alternative run (resp, solid black line) are very similar. The default setting gives us a somewhat less smooth estimate.

5.2 Inference for f

We'll plot the true f and pointwise 95% intervals for $f(x)$ at each test x.
We can do this using the **mean**, **m.lower**, and **m.upper** components of the **resp** list.

```
rgy = range(resp$mmean,resp$m.upper,resp$m.lower)
plot(range(xp),rgy,xlab="x",ylab="f(x)",cex=.8,cex.axis=1.5,cex.lab=1.5,type="n")
lines(xp,fxp,col="blue",lwd=3,lty=1) #true f
lines(xp,resp$mmean,col="black",lwd=3,lty=1)
lines(xp,resp$m.5,col="grey",lwd=3,lty=1)
lines(xp,resp$m.upper,col="green",lwd=3,lty=2)
lines(xp,resp$m.lower,col="green",lwd=3,lty=2)
title(main="pointwise 95% posterior intervals for f(x) at test x",cex.main=1.5)
```



The posterior means and medians are so similar that the median (grey) overwrites the mean (black).
If you wanted different probabilities for the intervals you can change the `q.upper` and `q.lower` values in the call to `predict.rbart`.

Or you can just compute the intervals directly from `mdraws` or `sdraws`:

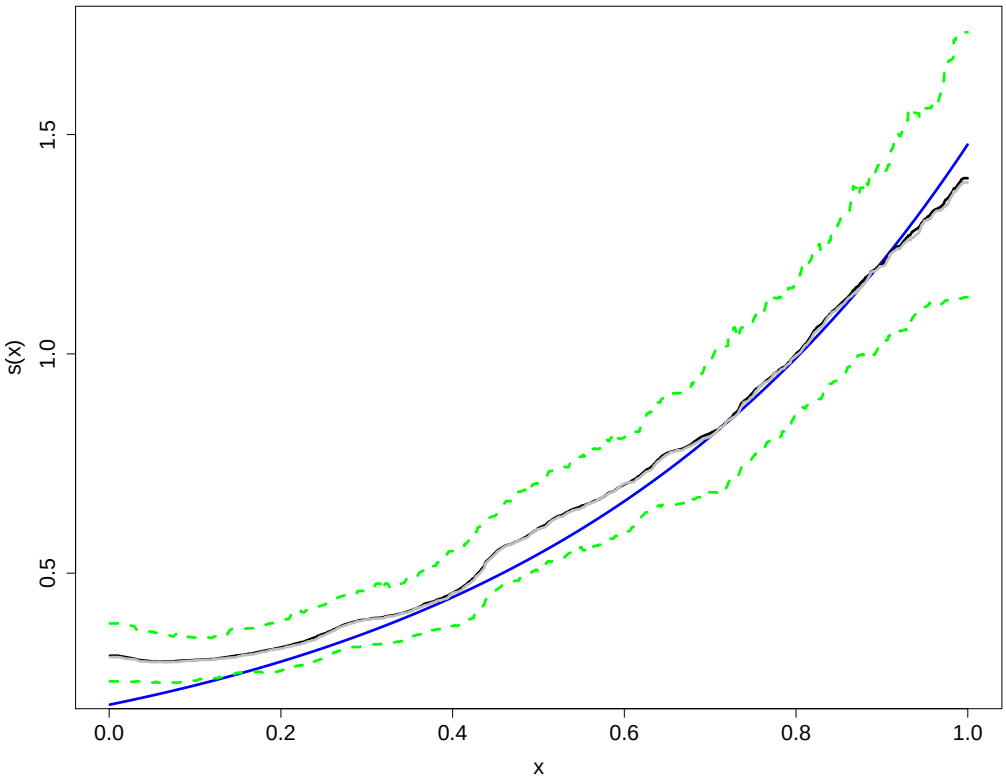
```
oldmqupper = apply(resp$mdraws,2,quantile,probs=c(.975)) #compute quantile of each column.
summary(abs(oldmqupper-resp$m.upper)) #compare to results returned from predict.rbart.
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0       0       0       0       0
```

5.3 Inference for s

We can do the same for the function s.

```
rgy = range(resp$smean,resp$s.upper,resp$s.lower)
plot(range(xp),rgy,xlab="x",ylab="s(x)",cex=.8,cex.axis=1.5,cex.lab=1.5,type="n")
lines(xp,sxp,col="blue",lwd=3,lty=1) #true s
lines(xp,resp$smean,col="black",lwd=3,lty=1)
lines(xp,resp$s.5,col="grey",lwd=3,lty=1)
lines(xp,resp$s.upper,col="green",lwd=3,lty=2)
lines(xp,resp$s.lower,col="green",lwd=3,lty=2)
```



Notice that both the mean intervals and the standard deviation intervals can "miss" a bit at the end points where there are not observations all around. In this case there is less information and the shrinkage implicit in the priors takes over.

5.4 Checking for Fit and Heteroskedasticity: `plotFunctionDraws`

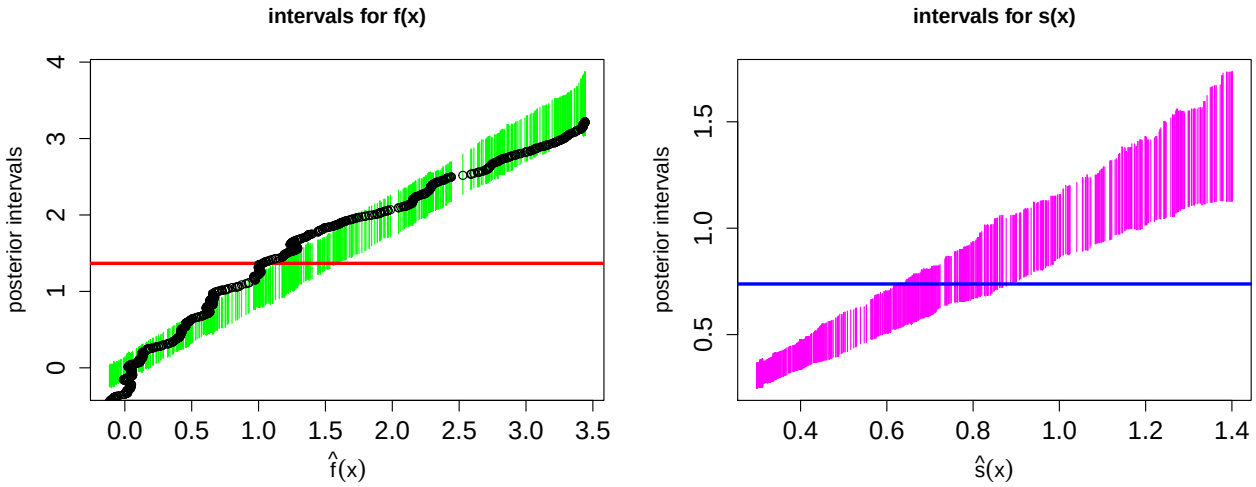
The function `plotFunctionDraws` provides a plot of the information in `mdraws` or `sdraws`.

First we will get a overall summary of the errors from the residual standard deviation and store it in `shat`. In addition, let's get the least squares predictions at `xp` and compare them to the inference for $f(x)$ from HBART.

```
shat = sqrt(mean((yp-resp$mmean)^2))
lmfit = lm(y~x,data.frame(x,y)); yhatlm = predict(lmfit,data.frame(x=xp))
```

Now we use `plotFunctionDraws` to look at `mdraws` (left panel) and `sdraws` (right panel).

```
par(mfrow=c(1,2))
plotFunctionDraws(resp$mdraws,complevel=mean(y), probs=c(.05,.95),
   xlab=expression(hat(f)(x)), pts=yhatlm, ptscol="black",
   cex.lab=1.2, cex.axis=1.4, main="intervals for f(x)")
plotFunctionDraws(resp$sdraws, complevel=shat, xlab=expression(hat(s)(x)),
   intervalcol="magenta", linecol="blue",
   cex.lab=1.2, cex.axis=1.4, main="intervals for s(x)")
```



In the left plot, $\hat{f}(x_j)$ for x_j in `xp` are plotted on the x-axis and the green intervals are 90% intervals for $f(x_j)$. The horizontal red line is plotted at the mean of `y` and the black points are the fitted values from the linear regression. The clear separation of the green intervals from horizontal red line indicates strong evidence for a function f which captures some `x` dependence. The fact that most of the fitted values from the linear regression are in the green intervals suggest that the evidence against linearity is not overwhelming, a conclusion that seems reasonable given a glance back at the data. However, the systematic nature of the difference between the between the linear fits and the f inference makes a linear f doubtful. This illustrates the value of considering all intervals simultaneously.

In the right plot $\hat{s}(x_j)$ for x_j in `xp` are plotted on the x-axis and the magenta intervals are 95% intervals for $s(x_j)$. The horizontal blue line is plotted at `shat`. The clear departure of the intervals from the horizontal indicates strong evidence for heteroskedasticity.

Arguments for `plotFunctionDraws` used above:

- **complevel**: value for horizontal comparison line.
- **probs**: quantiles to use for the posterior interval, e.g. (.05,.95) gives a 90% interval.
- **pts**: A point is plotted at (x_j, pts_j).
- **ptscol**: color used to plot points pts.
- **intervalcol**: color for the posterior intervals.
- **linecol**: color for the horizontal line.
- Additional arguments are passed on to `graphics::plot`.

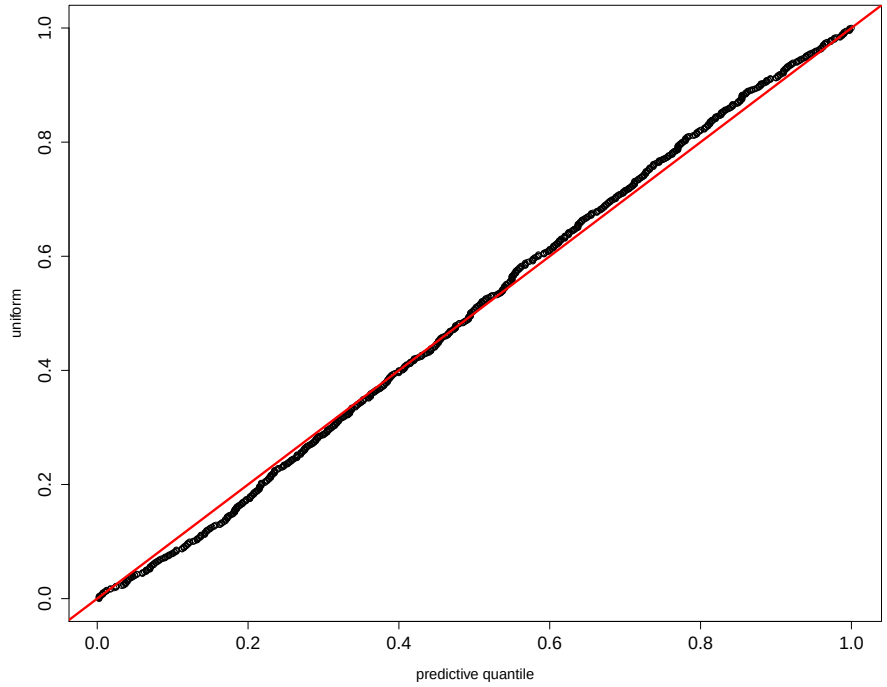
5.5 Predictive quantile-quantile plot: `hbartqqplot`

To assess the overall fit of the HBART model, we:

- pick a set of (x_j, y_j) points.
- draw repeatedly from the predictive distribution conditional on $x = x_j$ for each j.
- using the predictive draws, compute the quantile of y_j in the draws.
- Draw a qqplot of the y_j quantiles vs uniform(0,1) draws.

If the predictive were the "true" conditional, the y_j quantiles should look like uniform(0,1) draws. In small samples, the predictive may be dispersed relative to the true conditional even if the model is correct since the predictive reflects the estimation uncertainty. However, dramatic model failures can be evident from a marked departure from the $y = x$ straight line.

```
hbartqqplot(yp,resp,xlab="predictive quantile",ylab="uniform",
   cex.axis=1.4,cex.lab=1.2)
```



The points are pretty close to the reference 45-degree line $y = x$ indicating that we have enough information in the data to see that our HBART model has captured the essential relationship between y and x.

---

6 The used cars data

6.1 Read in the Cars data and run rbart: `rbartModelMatrix`

Now we will illustrate the use of `rbart` with real data having a higher dimensional x.

Of course, some of the plots we used for our one dimensional simlulated example are no longer available!!!

We will use the same prices of used cars data set used in Pratola et. al.

```
data(ucarprice)
ddf=ucarprice[,1:5] # price, trim, isOneOwner, mileage, year
dim(ddf)
```

```
## [1] 1000    5
```

```
head(ddf)
```

```
##   price trim isOneOwner mileage year
## 1 43995  550          f   36858 2008
## 2 44995  550          f   46883 2012
## 3 25999  550          f  108759 2007
## 4 33880  550          f   35187 2007
## 5 34895  550          f   48153 2007
## 6  5995  500          f  121748 2002
```

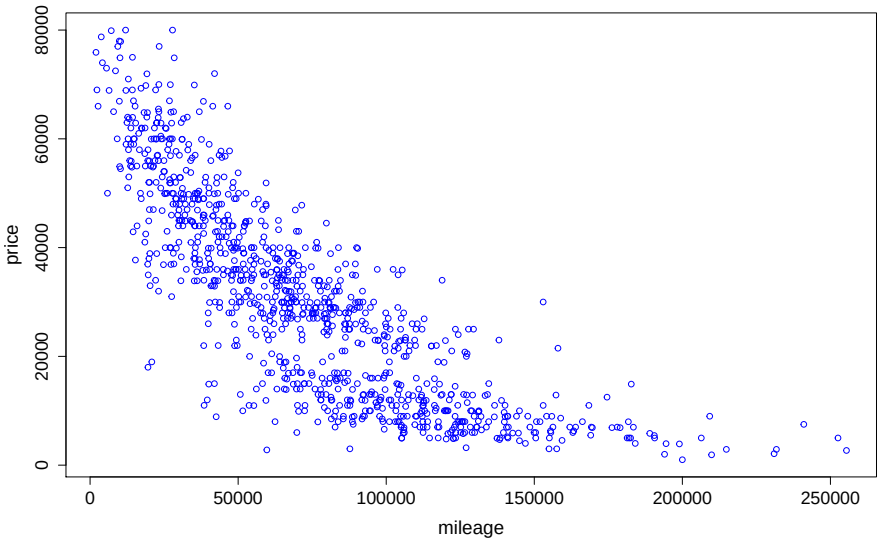
```
summary(ddf)
```

```
##      price          trim     isOneOwner    mileage            year
##  Min.   :  995   430  :143   f:841      Min.   :  1997   Min.   :1994
##  1st Qu.:12995   500  :127   t:159      1st Qu.: 40133   1st Qu.:2004
##  Median :29800   550  :591              Median : 67920   Median :2007
##  Mean   :30583   other:139              Mean   : 73652   Mean   :2007
##  3rd Qu.:43992                          3rd Qu.:100138   3rd Qu.:2010
##  Max.   :79995                          Max.   :255419   Max.   :2013
```

The response is `y` = the price of a used car. These cars are very nice Mercedes, so even the used car prices are quite high! The `x` include the mileage and year of the car as well as a categorical variable `trim` which relates to features of the car such as the type of material used in the interior. We have dropped some of the other variables to keep things simple (but we know we have kept the most important ones).

For example, we can easily see that there is a strong, nonlinear relationship between the price of the used cars and the mileage.

```
plot(ddf$mileage, ddf$price, xlab="mileage", ylab="price",
   cex.axis=1.4, cex.lab=1.5, col="blue")
```



Let's do a linear fit to compare HBART to.

```
lmcars = lm(price~.,ddf)
summary(lmcars)
```

```
##
## Call:
## lm(formula = price ~ ., data = ddf)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -22201  -4537  -1585   3497  30398
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.270e+06  1.966e+05 -26.808  < 2e-16 ***
## trim500      2.421e+03  8.726e+02   2.775  0.00563 **
## trim550      4.754e+03  8.348e+02   5.695 1.63e-08 ***
```

```
## trimother    9.601e+03  8.660e+02  11.087  < 2e-16 ***
## isOneOwnert  1.522e+03  6.386e+02   2.383  0.01734 *
## mileage     -1.371e-01  8.014e-03 -17.103  < 2e-16 ***
## year         2.644e+03  9.788e+01  27.011  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7089 on 993 degrees of freedom
## Multiple R-squared:  0.8526, Adjusted R-squared:  0.8517
## F-statistic: 957.5 on 6 and 993 DF,  p-value: < 2.2e-16
```

To fit HBART, we first need to express the categorical variable `trim` as dummies.
Notice how a factor with k levels is codes as k dummies, not $k-1$ as is done in linear regression.
This gives BART/HBART greater flexibility to identify the most appropriate coding.

```
x = rbartModelMatrix(ddf[,-1])
head(x)
```

```
##      mileage year trim1 trim2 trim3 trim4 isOneOwner1 isOneOwner2
## [1,]   36858 2008     0     0     1     0           1           0
## [2,]   46883 2012     0     0     1     0           1           0
## [3,]  108759 2007     0     0     1     0           1           0
## [4,]   35187 2007     0     0     1     0           1           0
## [5,]   48153 2007     0     0     1     0           1           0
## [6,]  121748 2002     0     1     0     0           1           0
```

Now we can run HBART

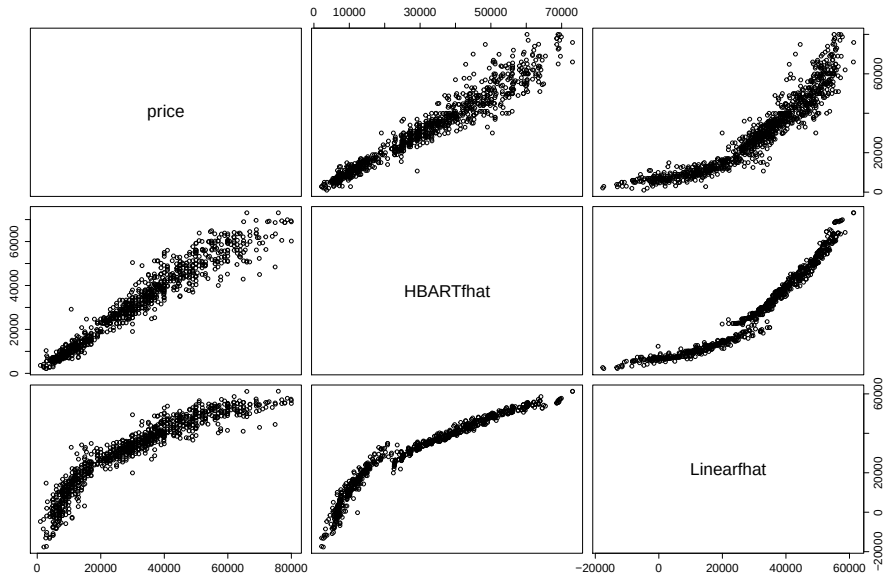
```
set.seed(99)
resc = rbart(x,ddf$price,tc=5) #run the HBART MCMC
rescp = predict(resc,x) #get the inference on the training x
```

Let's compare the fitted values from HBART and the linear regression with `y=price`.

```
fitmat = cbind(ddf$price,rescp$mmean,lmcars$fitted)
colnames(fitmat) = c("price","HBARTfhat","Linearfhat")
cor(fitmat)
```

```
##                price HBARTfhat Linearfhat
## price      1.0000000 0.9686283  0.9233775
## HBARTfhat  0.9686283 1.0000000  0.9564317
## Linearfhat 0.9233775 0.9564317  1.0000000
```

```
pairs(fitmat,cex.axis=1.4,cex.lab=1.5)
```

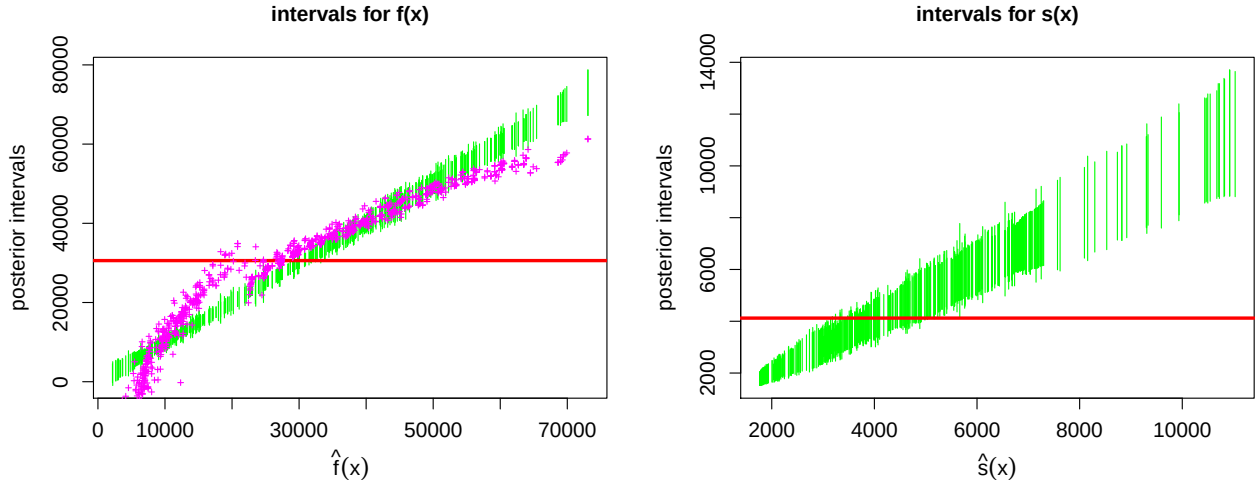


The HBART fitted values ($\hat{f}(x)$) clearly track y much better than the linear fits.
The plots suggest that there may be heteroskedasticity!!

6.2 Checking for Heteroskedasticity

We use `plotFunctionDraws` applied to `rescp$mdraws` and `rescp$sdraws` it assess the evidence for non-linearity in the mean ($f(x)$) and heteroskedasticity ($s(x)$).

```
par(mfrow=c(1,2))
plotFunctionDraws(rescp$mdraws,complevel=mean(ddf$price), xlab=expression(hat(f)(x)),
   pts=lmcars$fitted, ptscol="magenta",ptspch=3,ptscex=.5,
   cex.lab=1.2, cex.axis=1.2, main="intervals for f(x)")
plotFunctionDraws(rescp$sdraws, xlab=expression(hat(s)(x)),
   cex.lab=1.2, cex.axis=1.2, main="intervals for s(x)")
```

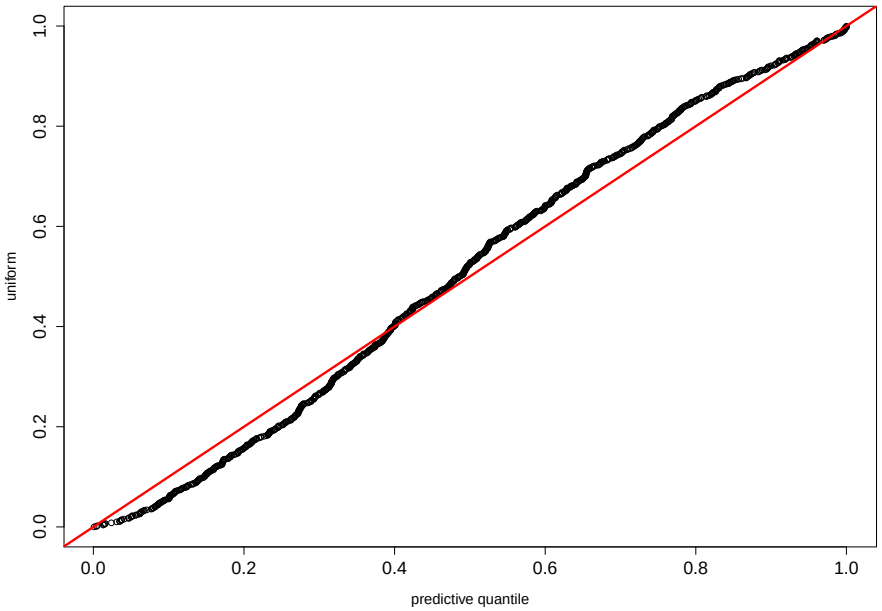


In the left panel we see that the fitted values from the linear regression are well outside the posterior intervals for $f(x)$. There is strong evidence against linearity.

In the right panel, we see that the intervals for $s(x)$ separate clearly from the horizontal reference line.
There is strong evidence for heteroskedasticity.

6.3 Predictive quantile-quantile plot

```
hbartqqplot(ddf$price, rescp, xlab="predictive quantile", ylab="uniform",
   cex.axis=1.4, cex.lab=1.2)
```



Overall, the qqplot follows the 45% line.
The departure is consistent with a predictive which is over-dispersed relative to the true conditional distributions.

---

7 BART from rbart, more `rbart` parameters

In this section we explore some more options in the fundamental `rbart` function.
In particular, we show how to get a standard BART fit.
We will continue to use the used cars data.

We can control the number of trees used in the ensembles:

- **ntree**: Number of trees used in the ensemble for the mean function f.
- **ntreeh**: Number of trees used in the ensemble for the standard deviation function s.

The MCMC uses four types of moves:

- The fundamental birth/death move in which pairs of bottom child nodes are added (*birth*) or deleted (*death*) from a tree in an ensemble.
- Peturb, in which the cutpoint for a node decision rule is allowed to change.
- change of variable, the decision rule, including the chosen variable (component of x), associated with an interior node change.

- Rotate, the Pratola (2016) move in which a tree is "rotated". This move enables more dramatic changes in the tree structure.

Several parameters allow the user to choose the probability with which these different types of moves are proposed by the Metropolis-Hastings move for a single tree in an ensemble.

The `pbd` parameter controls the probability of a birth/death move.
You can give a single number in which case this is used as the probability for both the f and s ensembles or you can give a vector of length 2, where the first number gives the probability for the f ensemble and the second number is for the s ensemble.

Let's use these parameters to run BART using `rbart`.
We can do this by:

- Using a single tree for the s ensemble.
- Setting the birth/death probability equal to zero for single tree representing s.

With these settings, the s ensemble is a single tree consisting of a single node.

So, we use `ntreeh=1` to get a single s tree and `pbd=c(0.7,0.0)` to say there is a 70% chance that the MH will try a birth/death move for the f ensemble and no chance of a birth/death move for the s ensemble (the single tree).

```
set.seed(99)
bartres = rbart(x,ddf$price,ntreeh=1,pbd=c(0.7,0.0), tc=5) #bart
bartresp = predict(bartres,x.text=x)
```

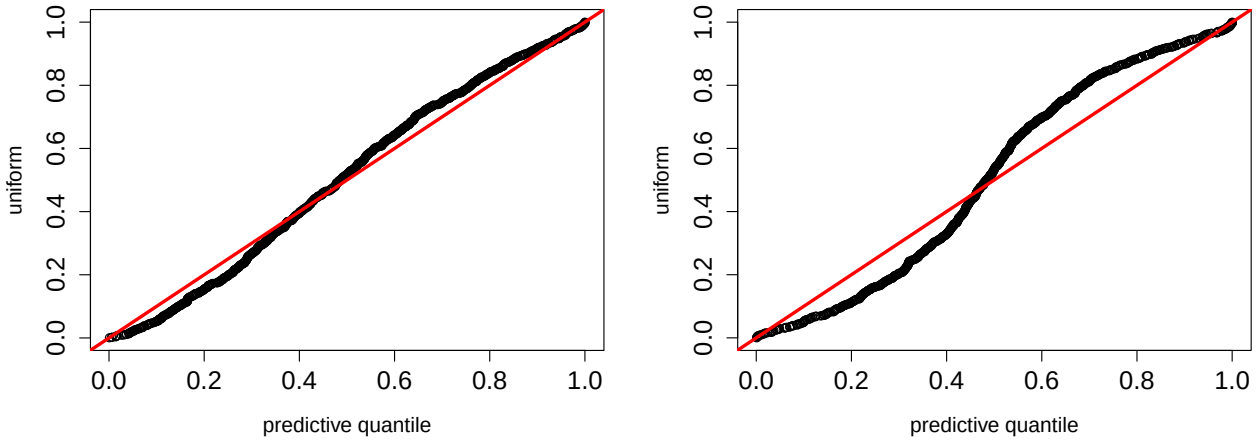
The structure of the list returned by `predict.rbart` is the same as an HBART fit.
The difference is that within a row of `sdraws` all the values are the same.

```
summary(apply(bartresp$sdraws,1,sd))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0       0       0       0       0       0
```

While this obviously has a lot of redundancy we can easily compare a HBART fit with a BART fit using `hbartqqplot`.

```
par(mfrow=c(1,2))
hbartqqplot(ddf$price, rescp, xlab="predictive quantile", ylab="uniform",
   cex.axis=1.4, cex.lab=1.2)
hbartqqplot(ddf$price, bartresp, xlab="predictive quantile", ylab="uniform",
   cex.axis=1.4, cex.lab=1.2)
```



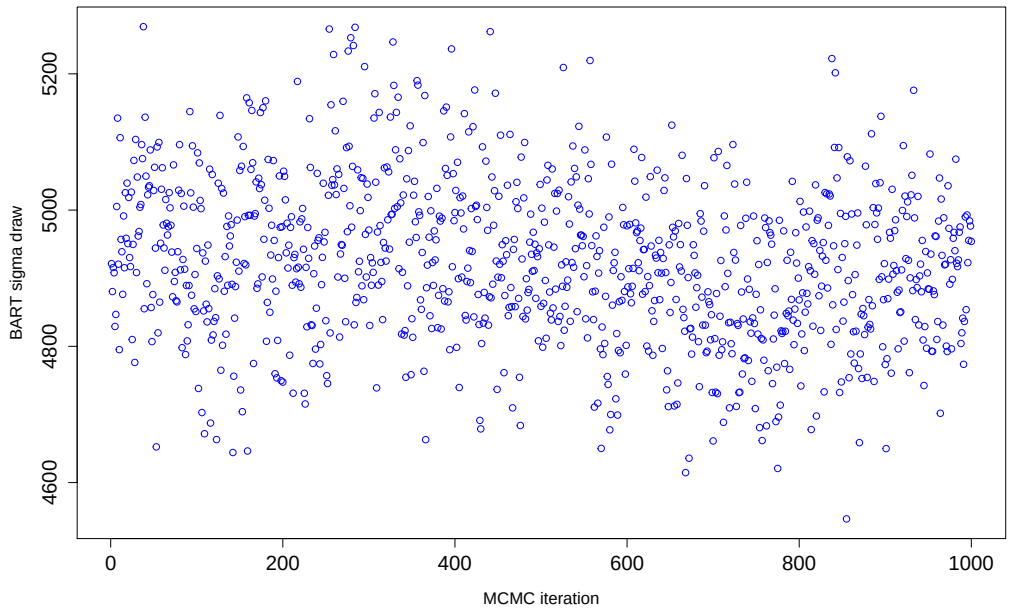
The dramatic improvement in the HBART inference over the BART inference is clear from the qqplots.

---

8 Checking MCMC burn in

In BART the errors are iid Normal, $Y_i = f(x_i) + \sigma Z_i, \;\; Z_i \sim N(0,1)$. A simple way to get a sense of whether or not the MCMC has converged is to look at the draws of the one parameter σ.

```
plot(bartresp$sdraws[,1], xlab="MCMC iteration", ylab="BART sigma draw",
   cex.axis=1.4, cex.lab=1.2, col="blue")
```



The draws are varying about a fixed level without substantial dependence so this plot indicates that at least as far as σ is concerned, the MCMC has burnt in and is not too dependent.

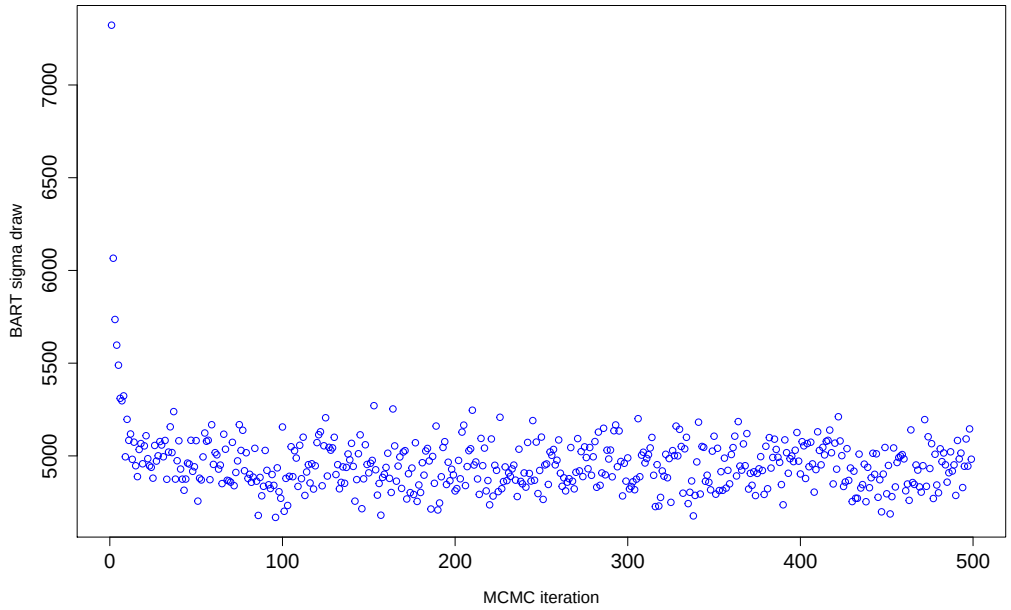
Let's run the BART MCMC again but this time skip the adapt and burn-in steps.

Run BART again:

```
ndbart = 500 #number of MCMC iterations
set.seed(99)
bartres1 = rbart(x,ddf$price,ntreeh=1,pbd=c(0.7,0.0),
   nadapt=0, nskip=0, ndpost=ndbart, tc=5) #BART, no adaptive learning, no burn-in
bartresp1 = predict(bartres1,x.text=x)
```

Plot the σ draws:

```
plot(bartresp1$sdraws[2:ndbart,1], xlab="MCMC iteration", ylab="BART sigma draw",
   cex.axis=1.4, cex.lab=1.2, col="blue") #(2:ndpbart) to drop the initial starting value
```



Now we can see the initial burn-in. As the MCMC iterates the trees in the ensemble evolve to find the fit in $f(x)$.

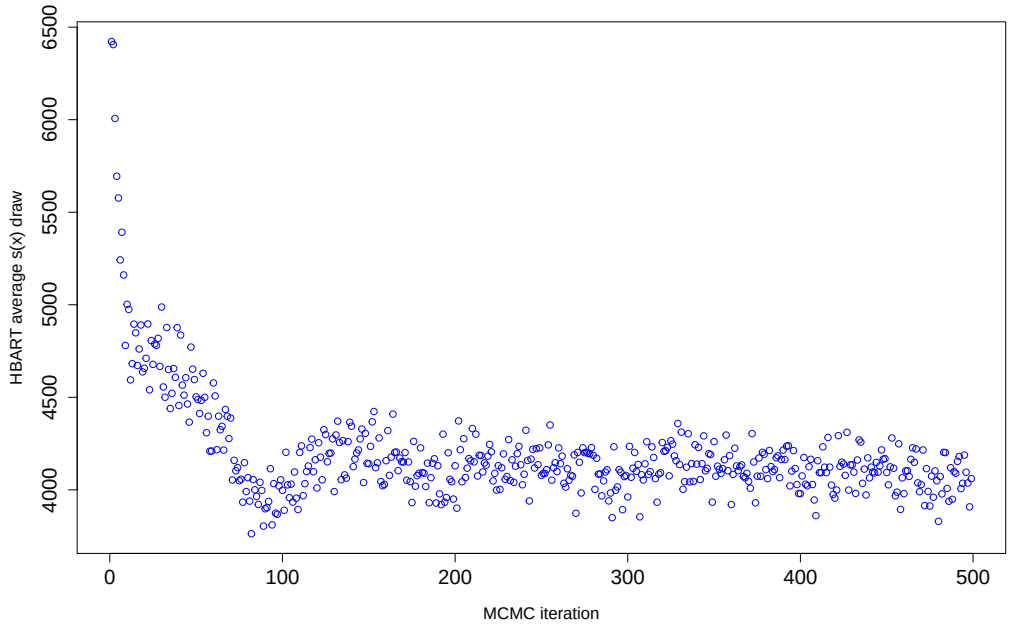
Now let's do the same thing for HBART. However, now we do not have the single parameter σ to monitor. A simple approach is to use the average of $s(x_i)$ over the training data.

Run HBART again without adaption or burn-in.

```
ndhbart=500
set.seed(99)
resc1 = rbart(x,ddf$price, nadapt=0, nskip=0, ndpost=ndhbart, tc=5) #run the HBART MCMC
rescp1 = predict(resc1,x) #get the inference on the training x
```

Now we plot the average $s(x)$:

```
plot(apply(rescp1$sdraws,1,mean)[2:ndhbart], xlab="MCMC iteration", ylab="HBART average s(x) draw",
   cex.axis=1.4, cex.lab=1.2, col="blue") #(2:ndpbart) to drop the initial starting value
```



Again, without the adaption and burn-in we see the initial phase of the MCMC as the trees learn f and s.