An Illustrating Example

This RMarkdown file provides an illustrative example to demonstrate the use of our method. Consider the underlying function-on-scalar regression model:

$$Y_i(t_{ij}) = \Xi_0(t_{ij}; x_i) + \epsilon_{ij}, \quad \epsilon_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2),$$

where $Y_i \in L_2([0,1])$ is a function observed at a set of m_i points $\mathbf{t}_i = \{t_{i1}, \ldots, t_{im_i}\} \subseteq \mathbb{R}$, $\mathbf{x}_i \in \mathbb{R}^p$ is the corresponding observed covariate vector, and $\Xi_0(t; \mathbf{x}) := \mathbb{E}(Y(t) \mid \mathbf{x})$.

In this experiment, we consider $n = 100$ curves. Each curve $\{Y_i(t_{ij})\}_{j \in [m_i]}$ contains $m_i = m = 10$ observations at a regular grid of points $\{t_{ij} = \frac{j}{11} : j \in [m_i]\}$.

Two estimation methods are demonstrated:

1. **FBART**: The proposed functional BART.
2. **S-FBART**: The shape-constrained version of FBART with a monotonic increasing constraint.

Preparation and Data Generation

```r
rm(list=ls())
## required packages
packages <- c("devtools","splines2", "ggplot2", "dplyr",
              "LaplacesDemon", "doParallel", "foreach")
install_if_missing <- function(pkg) {
  if (!require(pkg, character.only = TRUE)) {
    install.packages(pkg, dependencies = TRUE)
    library(pkg, character.only = TRUE)
  } else{ return('INSTALLED')}
}
lapply(packages, install_if_missing)
```

```
## [[1]]
## [1] "INSTALLED"
##
## [[2]]
## [1] "INSTALLED"
##
## [[3]]
## [1] "INSTALLED"
##
## [[4]]
## [1] "INSTALLED"
##
## [[5]]
## [1] "INSTALLED"
##
## [[6]]
## [1] "INSTALLED"
```

```
##
## [[7]]
## [1] "INSTALLED"
```

```
if(!require("FBART", character.only = TRUE))
{
    cat("Install the FBART package...\n")
    install.packages('FBART_1.0.tar.gz', repos = NULL, type="source")
}
source('functions.R')
## Setting
n = 100  # sample size
p = 2   # covariate dim
m = 10  # num of observed observations
J = 5    # num of basis
T = 5    # num of trees
degree = 3    # cubic B-spline
sigma = 2; sigma2 = sigma^2   # error var
MC_iter = 3000; num_burn = 2000
## true regression map in case 2
x_to_y = function(t_vec,x){
    return(
        (4*x[1] + 1)/(x[2] + 2) * t_vec +
        2*(x[2]*(x[1]<=0.5)+1)*tan( 8/9*(t_vec + t_vec^(x[2]*4) - 1)*pi/2)
    )
}
```

Data Generation

```
## DGP
seed = 1; set.seed(seed)
# covariates
X = matrix(nrow = n, ncol = p); for(j in 1:p){X[,j] = runif(n)}
# discretization
n_discrete = 500; x_discrete = seq(0,1,length.out = n_discrete+2)[2:(n_discrete+1)]
# sampling points
ts = matrix(rep(seq(0,1,length.out = m+2)[2:(m+1)], each = n), n , m)
t_list = rep(list(seq(0,1,length.out = m+2)[2:(m+1)]), n)
# Basis matrix
Phi_list = vector("list", n)
for(i in 1:n)
{
    Phi_list[[i]] = bSpline(x = ts[i,], df = J, degree = degree,
                            knots = seq(0,1,length.out=J-degree+1)[-c(1,J-degree+1)],
                            Boundary.knots = c(0,1),intercept = TRUE)# of size m*J
}

# observations
ys = matrix(nrow = n, ncol = m)
ys_noiseless = matrix(nrow = n, ncol = m)
Y_list = vector(length = n, mode = "list")
Y_list_noiseless = vector(length = n, mode = "list")
for(i in 1:n)
{
```

```r
    ys_noiseless[i,] = x_to_y(ts[i,],  X[i,])
    ys[i,] =  ys_noiseless[i,] + rnorm(m,0,sqrt(sigma2))
    Y_list_noiseless[[i]] = ys_noiseless[i,]
    Y_list[[i]] = ys[i,]
}

## preliminary estimates
betas = matrix(nrow = n, ncol = J)
n_discrete = 500; x_discrete = seq(0,1,length.out = n_discrete+2)[2:(n_discrete+1)]
Basis_mat = bSpline(x = x_discrete, df = J, degree = degree,
                            knots = seq(0,1,length.out=J-degree+1)[-c(1,J-degree+1)],
                            Boundary.knots = c(0,1),intercept = TRUE)# of size n_discrete*J
for(i in 1:n)
{
    t_i = c(ts[i,],0,1)
    y_i = ys[i,]; y_i = c(y_i, ys[i,which.min(ts[i,])],ys[i,which.max(ts[i,])])
    spline_predict_values = approx(x = t_i, y = y_i, xout = x_discrete, method = 'linear')$y
    betas[i, ] = as.vector(lm(spline_predict_values ~ Basis_mat - 1)$coefficient)
}
# ---------------------# ---------------------# ---------------------# ---------------------
# Test data
n_test = 2
m_test = 50
#locations
X_test = matrix(c(0.2,0.8,0.2,0.8), nrow = 2, ncol = p)
#realizations
ts_test = matrix(rep(seq(0,1,length.out = m_test+2)[2:(m_test+1)], each = n_test), n_test , m_test)
t_list_test = rep(list(seq(0,1,length.out = m_test+2)[2:(m_test+1)]), n_test)
# Basis matrix,E matrice and  transformed observations
Phi_list_test = vector("list", n_test)
for(i in 1:n_test)
{
    Phi_list_test[[i]] = bSpline(x = ts_test[i,], df = J, degree = degree,
                            knots = seq(0,1,length.out=J-degree+1)[-c(1,J-degree+1)],
                            Boundary.knots = c(0,1),intercept = TRUE)# of size m*J
}
# observations
ys_test = matrix(nrow = n_test, ncol = m_test)
ys_noiseless_test = matrix(nrow = n_test, ncol = m_test)
Y_list_test = vector(length = n_test, mode = "list")
Y_list_noiseless_test = vector(length = n_test, mode = "list")
for(i in 1:n_test)
{
    ys_noiseless_test[i,] = x_to_y(ts_test[i,],  X_test[i,])
    ys_test[i,] =  ys_noiseless_test[i,] + rnorm(m_test,0,sqrt(sigma2))
    Y_list_noiseless_test[[i]] = ys_noiseless_test[i,]
    Y_list_test[[i]] = ys_test[i,]
}
```

Model Fitting

```r
seed = 0
set.seed(seed)
```

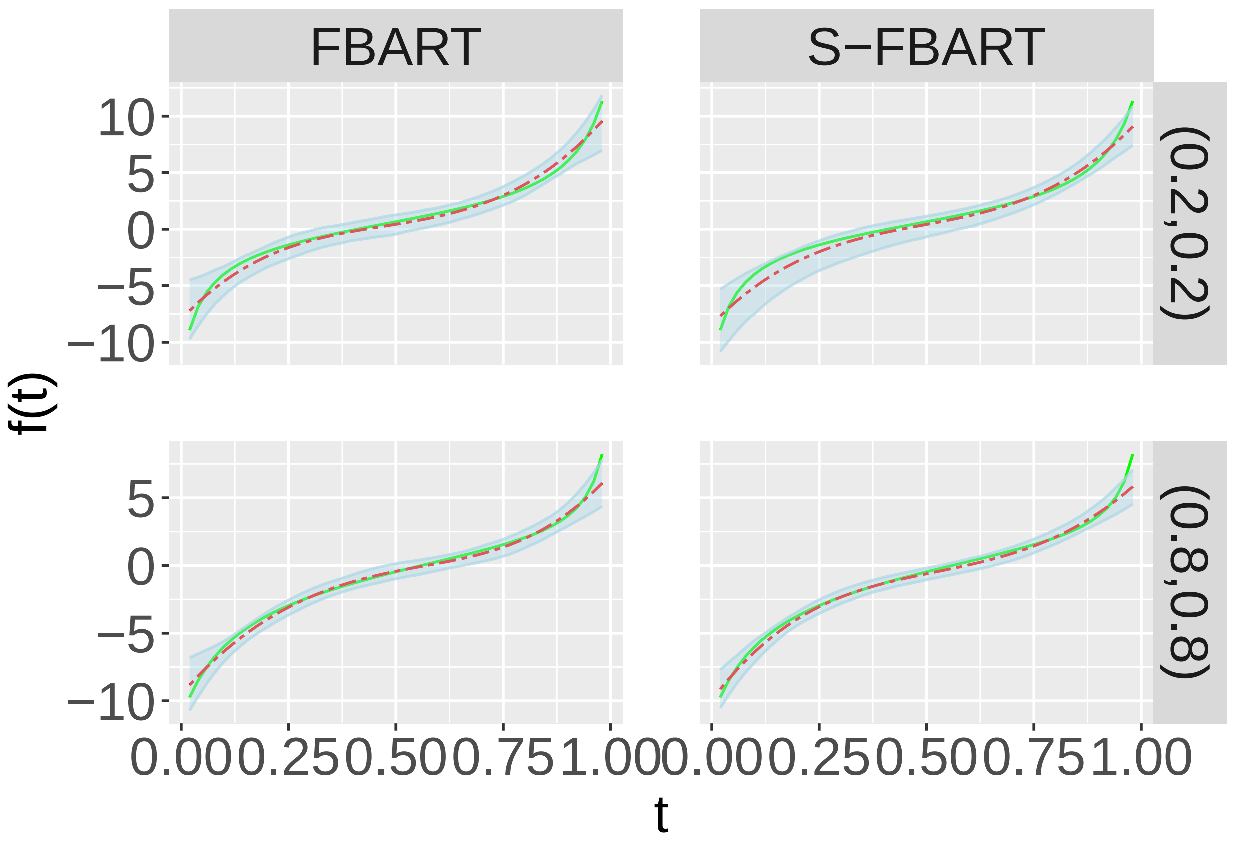
```r
# ---------------------# ---------------------# ---------------------# ---------------------
# FBART
# in a few seconds
savefile_mcmc = ('Data/FBART.RData')
fit = fit_FBART(X = X, Y = Y_list, X_test = X_test, Phi_list = Phi_list,
                Phi_list_test = Phi_list_test, num_tree = T,
                betas = betas, is_constrained = FALSE, D = diag(J),
                num_burn = num_burn, num_thin = 1, num_save = MC_iter - num_burn)

sigma_hat = fit$sigma; Y_hat_list_test = fit$y_hat_test
save(sigma_hat, Y_hat_list_test, file = savefile_mcmc)

# ---------------------# ---------------------# ---------------------# ---------------------
# SFBART
# in a few minutes
savefile_mcmc = ('Data/SFBART.RData')
D = diag(J)
for(j in 2:J) D[j,j-1] = -1
lower = c(rep(-Inf,1),rep(0, J-1))
upper = rep(Inf, J)

fit = fit_FBART(X = X, Y = Y_list, X_test = X_test, Phi_list = Phi_list,
                Phi_list_test = Phi_list_test, num_tree = T,
                betas = betas, is_constrained = TRUE, D = D, lb = lower, ub = upper,
                num_burn = num_burn, num_thin = 1, num_save = MC_iter - num_burn)

sigma_hat = fit$sigma; Y_hat_list_test = fit$y_hat_test
save(sigma_hat, Y_hat_list_test, file = savefile_mcmc)
```

Estimation Results and Visualizations

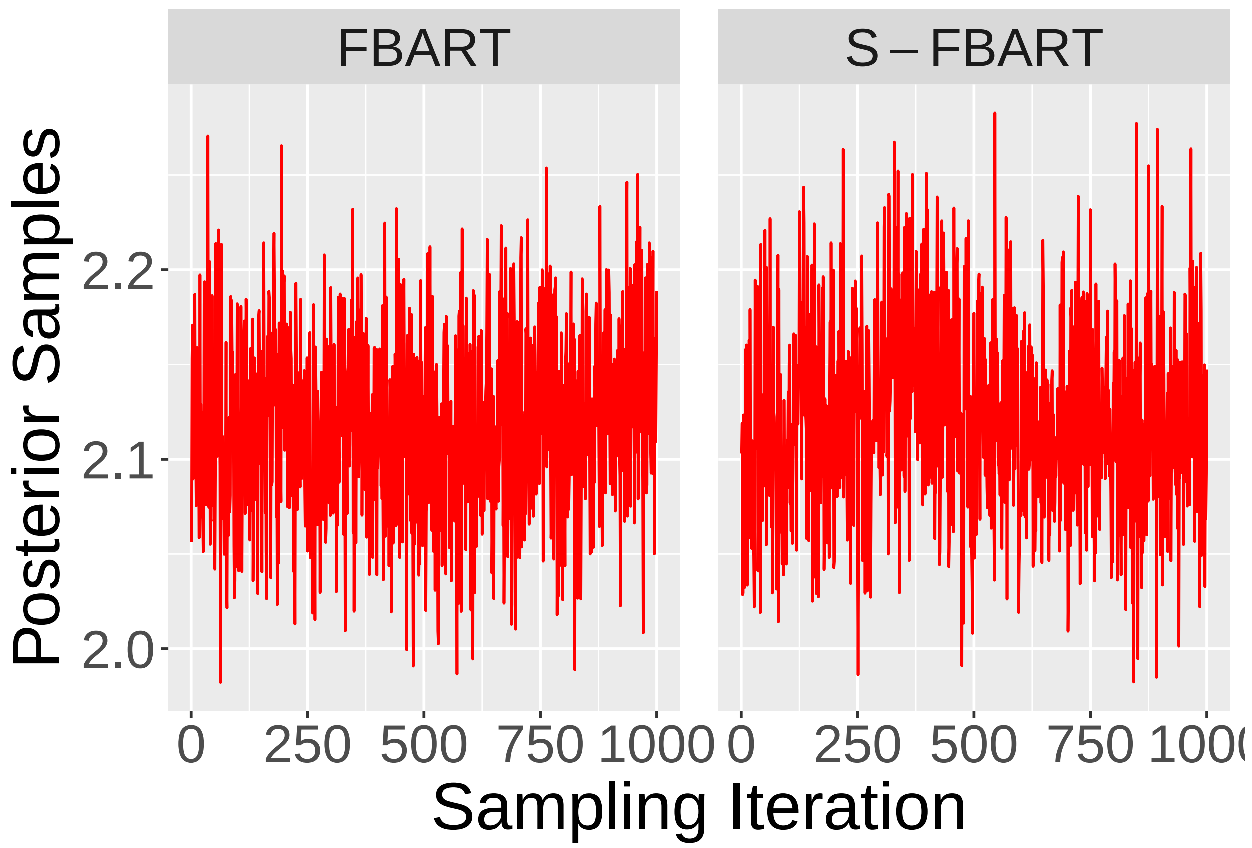
Finally, we depict the prediction results of the functional curves for FBART and S-FBART. The solid curves denote the true functions, the two-dashed curves denote the posterior mean functions, and the shaded areas represent the pointwise 95% credible intervals. We also present the trace plots of σ draws from FBART and S-FBART.

```r
savefile_mcmc = 'Data/FBART.RData'
load(savefile_mcmc)
sigma_hat_FBART = sigma_hat
i_test = 1
curve_summary = get_curve_summary(Y_hat_list_test)
df_curve = data.frame(t = t_list_test[[i_test]],
                      mean = curve_summary$y_hat_mean[[i_test]],
                      upper = curve_summary$y_hat_upper[[i_test]],
                      lower = curve_summary$y_hat_lower[[i_test]],
                      truth = Y_list_noiseless_test[[i_test]])
i_test = 2
curve_summary = get_curve_summary(Y_hat_list_test)
df_curve_temp = data.frame(t = t_list_test[[i_test]],
                      mean = curve_summary$y_hat_mean[[i_test]],
                      upper = curve_summary$y_hat_upper[[i_test]],
                      lower = curve_summary$y_hat_lower[[i_test]],
                      truth = Y_list_noiseless_test[[i_test]])
df_curve = rbind(df_curve, df_curve_temp)
```

```
df_curve$point = factor(rep(c('(0.2,0.2)','(0.8,0.8)'), each = m_test),
                        levels = c('(0.2,0.2)','(0.8,0.8)' ) )

savefile_mcmc = 'Data/SFBART.RData'
load(savefile_mcmc)
sigma_hat_SFBART = sigma_hat
i_test = 1
curve_summary = get_curve_summary(Y_hat_list_test)
df_curve_temp = data.frame(t = t_list_test[[i_test]],
                        mean = curve_summary$y_hat_mean[[i_test]],
                        upper = curve_summary$y_hat_upper[[i_test]],
                        lower = curve_summary$y_hat_lower[[i_test]],
                        truth = Y_list_noiseless_test[[i_test]])
i_test = 2
curve_summary = get_curve_summary(Y_hat_list_test)
df_curve_temp_temp = data.frame(t = t_list_test[[i_test]],
                        mean = curve_summary$y_hat_mean[[i_test]],
                        upper = curve_summary$y_hat_upper[[i_test]],
                        lower = curve_summary$y_hat_lower[[i_test]],
                        truth = Y_list_noiseless_test[[i_test]])
df_curve_temp = rbind(df_curve_temp, df_curve_temp_temp)
df_curve_temp$point = factor(rep(c('(0.2,0.2)','(0.8,0.8)'), each = m_test),
                          levels = c('(0.2,0.2)','(0.8,0.8)' ) )

df_curve = rbind(df_curve, df_curve_temp)
df_curve$method = factor(rep(c('FBART','S-FBART'), each = m_test*2),
                          levels = c('FBART','S-FBART' ) )


ggplot(df_curve) + geom_path(aes(x = t, y = truth),color = 'green', linetype = 'solid') +
    geom_path(aes(x = t, y = mean),color = 'red', linetype = 'twodash') +
    geom_path(aes(x = t, y = lower),color = 'lightblue', alpha=0.7) +
    geom_path(aes(x = t, y = upper),color = 'lightblue', alpha=0.7) +
    geom_ribbon(aes(x = t, ymin=lower, ymax=upper),fill = 'lightblue', alpha=0.4) +
    facet_grid(rows=vars(point), cols = vars(method),scales='free_y')+
    labs(x = 't', y = 'f(t)') +
    theme(legend.key.size = unit(1, 'cm'),legend.title = element_text(size=20),legend.text = element_te
    theme(axis.text=element_text(size=20),axis.title=element_text(size=20)) +
    theme(strip.text=element_text(size=20)) + scale_alpha_manual(values=c(1,0.5)) +
    theme(panel.spacing = unit(2, 'lines'))
```



```
ggsave(filename = 'Figures/Curve.pdf', width = 10, height = 7)

num_samples = MC_iter - num_burn
df = data.frame(iter = rep(seq(num_samples), times = 2),
                value = c(sigma_hat_FBART,sigma_hat_SFBART),
                method = rep(c('FBART','S-FBART'), each = num_samples) )
df$method = factor(df$method, levels = c('FBART','S-FBART'))

ggplot(df) + geom_path(aes(x = iter, y = value), color = 'red') +
    facet_grid(cols=vars(method),scales='free_y',labeller = label_parsed) +
    labs(x = 'Sampling Iteration', y =  "Posterior Samples")+
    theme(legend.key.size = unit(1, 'cm'),legend.title = element_text(size=20),legend.text = element_te
    theme(axis.text=element_text(size=20),axis.title=element_text(size=25)) +
    theme(strip.text=element_text(size=20)) + scale_alpha_manual(values=c(1,0.5)) +
    theme(panel.spacing = unit(1, 'lines'))
```



```
ggsave(filename = 'Figures/Traceplot.pdf', width = 10, height = 10)
```