Investigation of Sparse Hierarchical Regularization for Basis Expansion Methods

Exploration and Expansion of Nonparametric Regression via `HierBasis`

David Fleischer Annik Gougeon

Last Update: 04 May, 2018

Contents

1 Introduction

The method of nonparametric regression regularization described in Haris, Shojaie, and Simon (2016b) provides a flexible framework and implementation of a sparse hierarchical penalty via the `R` package `HierBasis`. The proposal offered by Haris, Shojaie, and Simon (2016b) outlines a convex penaltization and estimation technique that is suggested to be well-suited to high-dimensional problems. In particular, we wish to verify and expand upon the `HierBasis` framework in the context of sparse additive modelling, focusing on the problem of prediction of a continuous response and variable selection.

1.1 Problem Description

We restrict the attention of this project to focus on the problem of regression of a continuous response $y = [y_n, ..., y_n] \in \mathbb{R}^n$ on a high-dimensional design matrix $\mathbb{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]^T = [X_1, ..., X_p] \in \mathbb{R}^{n \times p}$, such that

$$\boldsymbol{x}_i = [x_{i1}, ..., x_{ip}] \quad \text{(observation } i\text{)}$$
$$X_j = [x_{1j}, ..., x_{nj}]^T \quad \text{(predictor } j\text{)}.$$

We consider the problem of estimating additive components $\{f_j\}_{j=1}^p$ of the additive model

$$y_i = \sum_{j=1}^{p} f_j(x_{ij}) + \varepsilon_i,$$

for a sparse set of active features $f_j(x_{ij}) \neq 0$. The proposal offered by Haris, Shojaie, and Simon (2016b) considers the class of basis expansion estimators (Cencov (1962)) defined by a finite set of basis functions $\{\psi_k(z)\}_{k=1}^K$, with some notion of increasing complexity (in k) and for a truncation level K to be adaptively selected. Let $\Psi_K^{(j)} \in \mathbb{R}^{n\times K}$ be the basis expansion corresponding to the j^{th} predictor X_j, with $(i,k)^{th}$ entry associated with observation x_{ij} and basis function ψ_k,

$$\Psi_{K,(i,k)}^{(j)} = \psi_k(x_{ij}), \quad 1 \leq k \leq K,\ 1 \leq i \leq n.$$

Then, through the basis expansion functions, the design matrix $\mathbb{X} \in \mathbb{R}^{n\times p}$ maps to a set of p $(n \times K)$ matrices

$$\mathbb{X} \overset{\psi}{\mapsto} \left\{\Psi_K^{(j)} \in \mathbb{R}^{n\times K}\right\}_{j=1}^p.$$

Of present interest is the set of polynomial basis functions $\{\psi_k(z)\}_{k=1}^K = \left\{z^k\right\}_{k=1}^K$ so that

$$X_j = \begin{bmatrix} x_{1j} \\ \vdots \\ x_{nj} \end{bmatrix} \mapsto \Psi_K^{(j)} = \begin{bmatrix} \psi_1(x_{1j}) & \psi_2(x_{1j}) & \cdots & \psi_K(x_{1j}) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(x_{nj}) & \psi_2(x_{nj}) & \cdots & \psi_K(x_{nj}) \end{bmatrix} = \begin{bmatrix} x_{1j} & x_{1j}^2 & \cdots & x_{1j}^K \\ \vdots & \vdots & \ddots & \vdots \\ x_{nj} & x_{nj}^2 & \cdots & x_{nj}^K \end{bmatrix}.$$

We estimate the additive components f_j by the sparse additive `HierBasis` estimator $\widehat{f}_j$ given by

$$\widehat{f}_j(x_{ij}) = \sum_{k=1}^{K} \widehat{\beta}_{j,k}^{\texttt{SA-hier}} \psi_k(x_{ij}), \quad j = 1, ..., p,$$

such that the $j = 1, ..., p$ coefficient vectors $\widehat{\boldsymbol{\beta}}_j^{\texttt{SA-hier}} = \left[\widehat{\beta}_{j,1}^{\texttt{SA-hier}}, ..., \widehat{\beta}_{j,K}^{\texttt{SA-hier}}\right] \in \mathbb{R}^K$ are simultaneously estimated by the solution of the minimization problem

$$\left[\widehat{\boldsymbol{\beta}}_1^{\texttt{SA-hier}}, ..., \widehat{\boldsymbol{\beta}}_p^{\texttt{SA-hier}}\right] = \underset{\boldsymbol{\beta}_1,...,\boldsymbol{\beta}_p}{\arg\min} \left\{ \frac{1}{2} \left\| y - \sum_{j=1}^{p} \Psi_K^{(j)} \boldsymbol{\beta}_j \right\|_2^2 + \lambda \sum_{j=1}^{p} \Omega_j(\beta_j) + \frac{\lambda^2}{\sqrt{n}} \sum_{j=1}^{p} \left\| \Psi_K^{(j)} \boldsymbol{\beta}_j \right\|_2 \right\}, \tag{1}$$

where

$$\Omega_j(\boldsymbol{\beta}_j) = \frac{1}{\sqrt{n}} \sum_{k=1}^{K} w_k \left\| \Psi_{k:K}^{(j)} \boldsymbol{\beta}_{j,k:K} \right\|_2,$$

for weights $w_k = k^m = (k-1)^m$ penalization weights for the k^{th}-order basis estimator, $\Psi_{k:K}^{(j)}$ denotes the submatrix of columns $k, k+1, ..., K$ of $\Psi_K^{(j)}$, and $\boldsymbol{\beta}_{j,k:K}$ denotes the corresponding subvector of $\boldsymbol{\beta}_j$.

The penalty described in (1) is defined by two terms. The first term containing Ω_j's is designed to provide a data-driven method of selecting the basis complexity/truncating the degree of the basis functions to some

adaptively selected level $K_0 \leq K$. This term is derived from the hierarchical group lasso penalty (Zhao, Rocha, and Yu (2009)) and leads to hierarchical sparsity of the fitted parameters. That is, $\widehat{\beta}_{j,k} = 0 \implies \widehat{\beta}_{j,k'} = 0$ for all $k' \geq k$.

The second term in the sparse additive `HierBasis` penalty $\frac{\lambda^2}{\sqrt{n}} \sum_{j=1}^{p} \left\| \Psi_K^{(j)} \boldsymbol{\beta}_j \right\|_2$ imposes sparsity across the predictors $X_1, ..., X_p$ and induces additional sparsity across the solution space $\left\{ \widehat{\boldsymbol{\beta}}_j^{\texttt{SA-hier}} \right\}_{j=1}^{p}$.

1.2 Problem Convexity

It is important to mention the convexity properties contained within this problem. First, we note that the `HierBasis` penalty, $\Omega(\beta)$ is of the hierarchical group lasso form (Zhao, Rocha, and Yu (2009)). That is, it belongs to the CAP (Composite Absolute Penalties) family of penalties, which enables the solution to achieve hierarchical sparsity. According to Zhao, Rocha, and Yu (2009), CAP estimators lead to stable estimates along with a more effective use of degrees of freedom. However, they do not generally result in sparser estimates than the lasso (Tibshirani (1996)).

We define the CAP penalty for hierarichal solution. Introduce a node that corresponds to some group of variables, say G_k, and let there be a total of n nodes. For every group G_k, define $G_{k:n}$ as the groups that should only be added to the model after G_k. For example, in a model with both main and interaction effects, the group of interaction effects can only be added after its main effects are included in the model. In the `HierBasis` case, this consists of the higher order terms of the set of basis functions, ψ_n.

Then, a hierarchical sparsity inducing CAP penalty can be defined as

$$T(\beta) = \sum_{i=1}^{n} \alpha_i \cdot \left\| (\beta_{G_i}, \beta_{G_{i:n}}) \right\|_{\gamma_i},$$

where $\alpha_m > 0, \forall m$ and $1 \leq \gamma_i < \infty$. Note that α_m is a correction factor in the case that a coefficient appears in numerous groups. An important theorem from Zhao, Rocha, and Yu (2009) allows us to obtain convexity:

Theorem (Zhao, Rocha, and Yu (2009)) If $\gamma_i \geq 1, \forall i = 1, ...n$, then $T(\beta)$ is convex. Furthermore, if the loss function L is convex in β, then the objective function of the CAP optimization problem is convex.

It follows that our estimators $\widehat{\beta}_1^{\texttt{SA-hier}}, ..., \widehat{\beta}_p^{\texttt{SA-hier}}$ are indeed convex.

1.3 Solving the `HierBasis` Estimators

To solve the sparse additive problem Haris, Shojaie, and Simon (2016b) first solves the equivalent univariate problem by applying the results of Zhao, Rocha, and Yu (2009), Jenatton et al. (2010), Jenatton et al. (2011). By writing the problem in the form

$$\min_{v \in \mathbb{R}^p} \left\{ \|u - v\|_2^2 + \lambda \Omega(v) \right\}$$

where Ω is a hierarchical penalty of the form described above. We may apply the following proximal gradient descent algorithm (with complexity $O(p)$) (Jenatton et al. (2011)). Let $\Psi = UV$ such that $U \in \mathbb{R}^{n \times K}$, $U^T U / n = \mathbb{I}_K$, can be obtained via a QR decomposition on the basis expansion matrix of Ψ. Then, the univariate `HierBasis` problem

$$\widehat{\boldsymbol{\beta}}^{\texttt{hier}(K)} = \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^K} \left\{ \frac{1}{2} \|y - \Psi_K \boldsymbol{\beta}\|_2^2 + \frac{\lambda}{\sqrt{n}} \sum_{k=1}^{K} \left(k^m - (k-1)^m\right) \|\Psi_{k:K} \boldsymbol{\beta}_{k:K}\|_2 \right\}$$

is solved by reformulating it in a proximal-gradient-descent-friendly format

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^K} \left\{ \frac{1}{2} \left\| U^T y / n - \boldsymbol{\beta} \right\|_2^2 + \lambda \sum_{k=1}^{K} w_k \|\boldsymbol{\beta}_{k:K}\|_2 \right\}$$

which itself can be solved via a coordinate descent algorithm (Haris, Shojaie, and Simon (2016b))

Algorithm 1 Solving the Univariate `HierBasis` Problem

procedure HierBasis($y, U, \lambda, \{w_k\}_{k=1}^K$)
  Initialize $\boldsymbol{\beta}^{(1)} = \cdots = \boldsymbol{\beta}^K \leftarrow U^T y/n$
  for k = K, ..., 1 **do**
    Update $\boldsymbol{\beta}_{k:K}^{k-1} \leftarrow \left(1 - \frac{w_k \lambda}{\|\boldsymbol{\beta}_{k:K}^k\|_2}\right)_+ \boldsymbol{\beta}_{k:K}^k$
  return $\boldsymbol{\beta}^1$

With the univariate `HierBasis` estimators solved, we may now introduce the solution to the sparse additive `HierBasis` estimators using a block coordinate descent algorithm (Haris, Shojaie, and Simon (2016b))

Algorithm 2 Solving the Sparse Additive `HierBasis` Problem

procedure AdditiveHierBasis($y, \left\{\Psi_K^{(j)}\right\}_{j=1}^p, \lambda, \{w_k\}_{k=1}^K$, `maxiter`)
  Initialize $\beta_j \leftarrow 0$ for $j = 1, ..., p$
  while $l \leq$ `maxiter` **and** not converged **do**
    for $j = 1, ..., p$ **do**
      Set $r_{-j} \leftarrow y - \sum_{j' \neq j} \Psi_K^{(j')} \boldsymbol{\beta}_{j'}$
      Set $\tilde{w}_1 = w_1 + \lambda$, $\tilde{w}_k = w_k$, for $k = 2, ..., K$
      Update $\boldsymbol{\beta}_j \leftarrow \arg\min \left\{ \frac{1}{2n} \left\| r_{-j} - \Psi_K^{(j)} \boldsymbol{\beta} \right\|_2^2 + \frac{\lambda}{\sqrt{n}} \sum_{k=1}^K \tilde{w}_k \left\| \Psi_{k:K}^{(j)} \boldsymbol{\beta}_{j,k:K} \right\|_2 \right\}$
  return $\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_p$

1.4 Proposal

Of consideration for this project, we wish to tackle the following questions:

(1) Can the `hierbasis` estimator procedure offer a material gain over the lasso estimator (Tibshirani (1996))? Preliminary tests, as well as the `hierbasis` documentation (Haris, Shojaie, and Simon (2016a)), suggest a marginal sparsity improvement with no worse predictive power, but at the cost of computational complexity.

(2) The `hierbasis` documentation (Haris, Shojaie, and Simon (2016a)) references a mixing parameter α controlling the relative importance of the hierarchical and the sparsity-inducing penalities. How does the manipulation of this parameter affect its performance? Is it feasible to select α through cross-validation?

(3) What is the effect of changing the form of the weights $w_k = k^m - (k-1)^m$ in the hierarchical penalty Ω? The documentation suggests implementing $m = 2$ or $m = 3$. Why are these two values optimal, and how does the procedure perform when another m is selected?

(4) How does the `hierbasis` estimator procedure and R package perform on new datasets and simulations? Is it feasible to use this method for large datasets, considering the compuation time. How does it compare to the lasso estimator (Tibshirani (1996)) in this regard?

2 Methods

2.1 The `hierbasis2` Package

We have create a companion package to `HierBasis`, named `hierbasis2`, in order to implement the above tests and features of the above proposal. This new package retains all of the user-facing functionality of the

original `HierBasis` package, but now permits the user to manipulate some additional parameters, as well as introducing some new functions. That is, the new library has been designed with this project in mind, allowing us to explore the properties of the original `HierBasis` package in a modular, readable, and concise format.

2.1.1 Installation

As is the case for `HierBasis`, installation of `hierbasis2` can be done via `devtools::install_github`

```
#install.packages(devtools)
library(devtools)
install_github("dfleis/hierbasis2")
library(hierbasis2)
```

3 Simulations

3.1 Comparison with `glmnet`

3.1.1 Linear Models

We begin with a simple comparison of the `HierBasis` estimators to their lasso counterparts (via `glmnet` Friedman, Hastie, and Tibshirani (2010)) in the prediction and variable selection of a simple linear model

$$y_i = \boldsymbol{x}_i \boldsymbol{\beta} + \varepsilon_i,$$

for $\boldsymbol{x}_i \sim \mathcal{N}_p(0, \mathbb{I}_p)$ and $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ such that the noise variance σ^2 satisfies

$$\text{SNR} = \frac{1}{\sigma^2 (n-1)} \sum_{i=1}^{n} \left(\boldsymbol{x}_i \boldsymbol{\beta} \right)^2,$$

for a fixed signal-to-noise ratio $\text{SNR} = 3$.

```
#===== parameters =====#
set.seed(400)
n <- 1000 # number of observations
p <- 9 # number of predictors (excluding intercept)
SPARSE_PCT <- 0.5 # proportion of sparse predictors
# which predictors are sparse?
SPARSE_IDX <- sample(2:(p + 1), size = floor(SPARSE_PCT * p))
SNR <- 3 # signal-to-noise ratio (controls noise dispersions)
beta <- rnorm(p + 1, 0, 10)
beta[SPARSE_IDX] <- 0

#===== generate data =====#
X <- matrix(rnorm(n * p), ncol = p) # iid normal deviates
# compute noiseless response
ytrue <- cbind(1, X) %*% beta
# compute noise dispersion satisfying the SNR ratio
sigma2 <- sum(ytrue^2)/(n - 1) * 1/SNR
eps <- rnorm(n, 0, sqrt(sigma2))
# compute perturbed response
y <- ytrue + eps
```

```
### split data into training and validation sets
X_train <- X[1:(n/2),]; X_valid <- X[(n/2 + 1):n,]
y_train <- y[1:(n/2)]; y_valid <- y[(n/2 + 1):n]
```

3.1.1.1 Prediction Performance

For both the lasso and `HierBasis` estimators we will perform 10-fold cross-validation to select the tuning parameter λ. For computational considerations we limit the maximum number of basis elements to $K =$ `nbasis` $= 10$

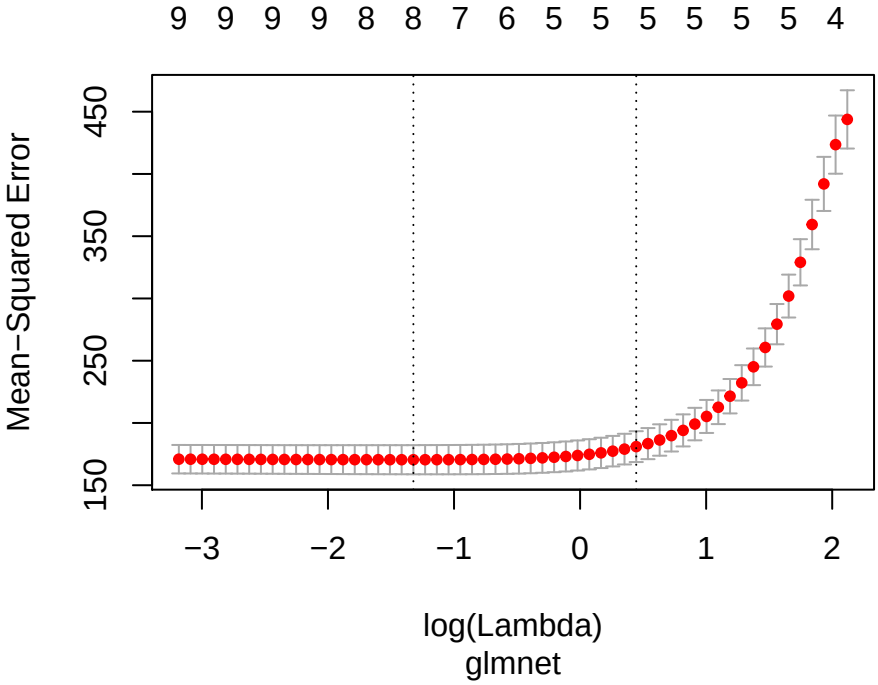
```
#===== fit models =====#
# lasso
pt <- proc.time()
mod.glmnet.cv <- cv.glmnet(x = X_train, y = y_train, alpha = 1)
proc.time() - pt
```

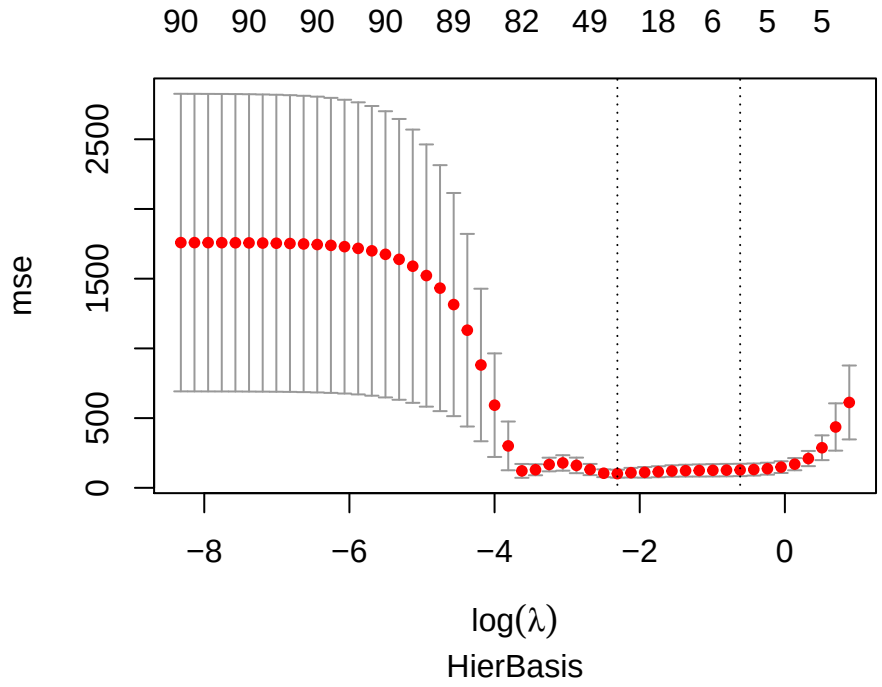
```
##    user  system elapsed
##   0.497   0.016   0.822
```

```
# additive hierbasis
pt <- proc.time()
mod.ahb.cv <- cv.additivehierbasis(X = X_train, y = y_train)
proc.time() - pt
```

```
##    user  system elapsed
##   0.712   0.017   0.765
```

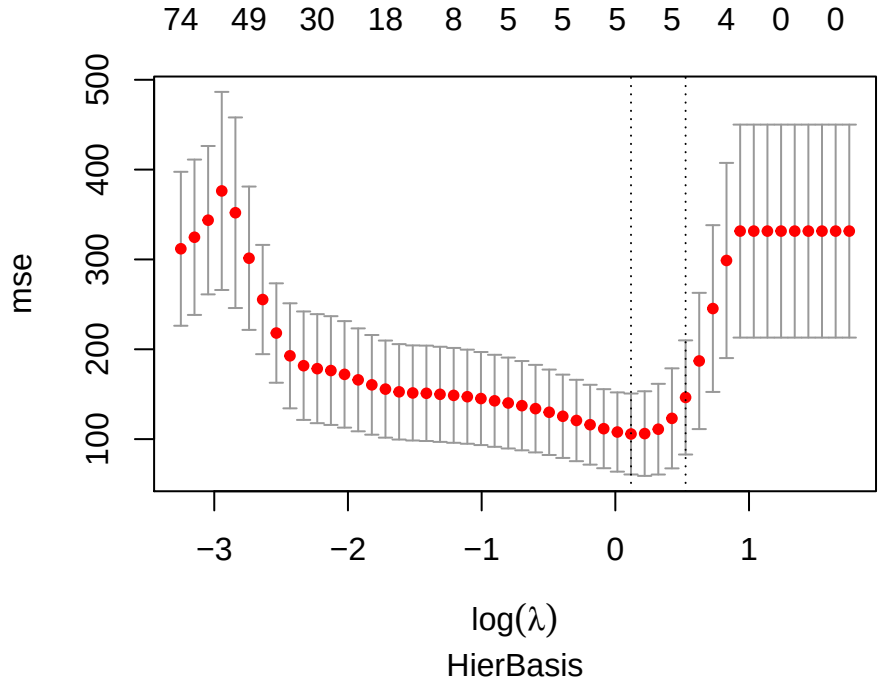
Below we present the test-error from the cross-validation procedure plotted against the natural logarithm of the tuning parameter, with the number of active features (`glmnet`) and number of active basis elements (`HierBasis`) printed above the figure.





One empirical observation with the cross-validation procedure is that the method used for automatically selecting the tuning parameters λ appears to set the lower bound of the sequence to be too low. If we instead manually set λ to be within the range of $[e^{-3}, e^{1.5}]$ we find cross-validation plots that appear to be more reasonable.

```
##    user  system elapsed
##   0.684   0.017   0.724
```



Using the validation sets we may compare the performance of the regression procedures numerically, using the tuning parameters corresponding to the least testing error, $\lambda_{\min}$, and the one-standard-error rule, $\lambda_{1\text{se}}$.

```
### predict validation sets ###
yhat.glmnet.min <- predict(mod.glmnet.cv, X_valid, s = mod.glmnet.cv$lambda.1se)
yhat.glmnet.1se <- predict(mod.glmnet.cv, X_valid, s = mod.glmnet.cv$lambda.min)
yhat.ahb.min <- predict(mod.ahb.cv, X_valid, lam.idx = mod.ahb.cv$lambda.min.idx)
yhat.ahb.1se <- predict(mod.ahb.cv, X_valid, lam.idx = mod.ahb.cv$lambda.1se.idx)

err.glmnet.min <- yhat.glmnet.min - y_valid
err.glmnet.1se <- yhat.glmnet.1se - y_valid
err.ahb.min <- yhat.ahb.min - y_valid
err.ahb.1se <- yhat.ahb.1se - y_valid

mse.glmnet.min <- mean(err.glmnet.min^2); mse.glmnet.min.sd <- sd(err.glmnet.min)
mse.glmnet.1se <- mean(err.glmnet.1se^2); mse.glmnet.1se.sd <- sd(err.glmnet.1se)
mse.ahb.min <- mean(err.ahb.min^2); mse.ahb.min.sd <- sd(err.ahb.min)
mse.ahb.1se <- mean(err.ahb.1se^2); mse.ahb.1se.sd <- sd(err.ahb.1se)

mse <- cbind(
  c(mse.glmnet.min, mse.glmnet.1se, mse.ahb.min, mse.ahb.1se),
  c(mse.glmnet.min.sd, mse.glmnet.1se.sd, mse.ahb.min.sd, mse.ahb.1se.sd))
colnames(mse) <- c("MSE", "MSE.SD")
rownames(mse) <- c("glmnet.min", "glmnet.1se", "HierBasis.min", "HierBasis.1se")
round(mse, 2)
```

```
##                   MSE MSE.SD
## glmnet.min    182.59  13.51
## glmnet.1se    171.75  13.11
## HierBasis.min 196.52  14.01
## HierBasis.1se 257.78  16.03
```

We see that for this regression task the `HierBasis` estimator performs comparably to the lasso, at the cost of added computational (and model) complexity. It should be noted that the linear model is not the scenario the additive `HierBasis` estimator was designed for, and so matching performance with the lasso informs us that the `HierBasis` estimator remains capable of handling simpler response-predictor relationships.

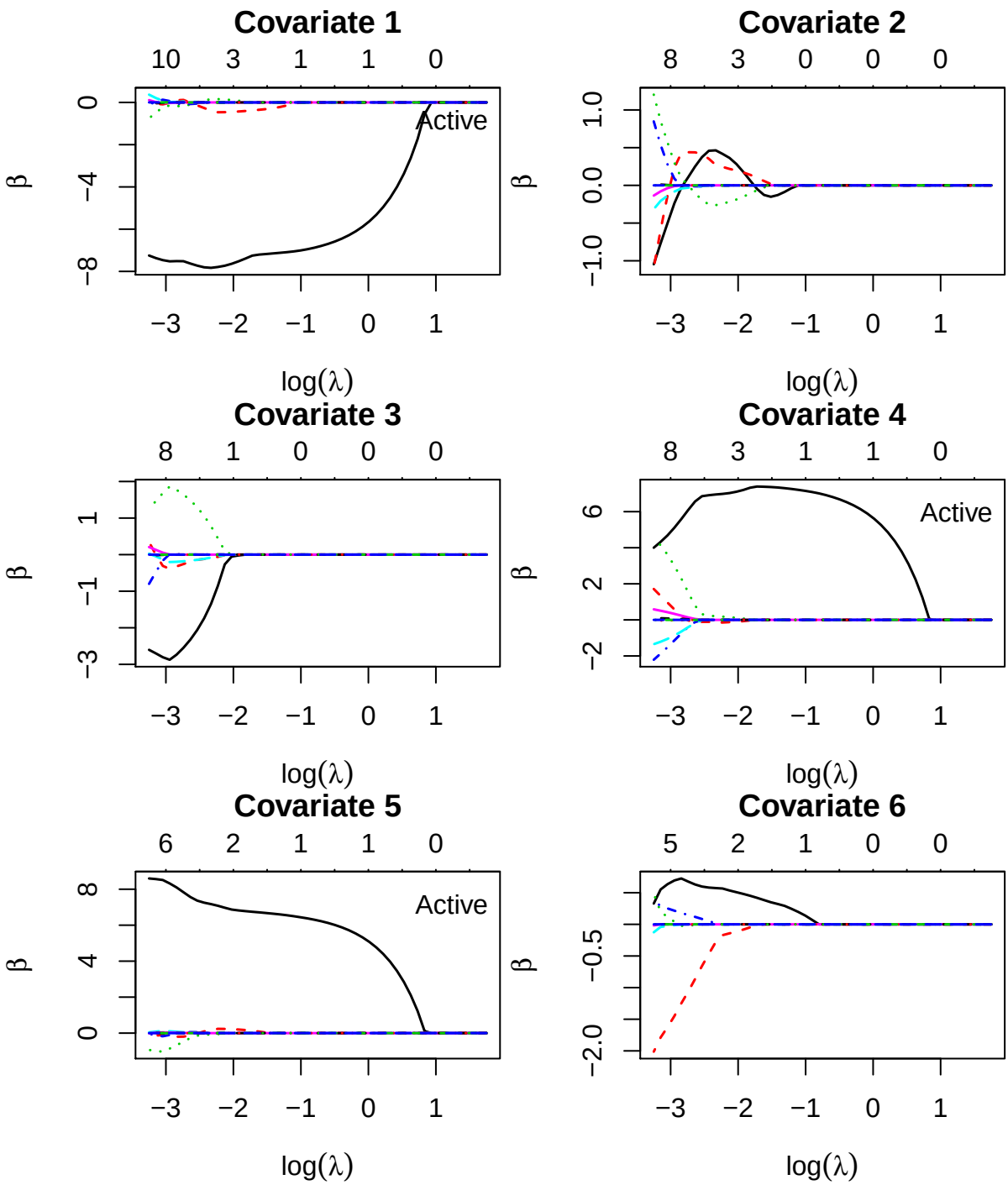
3.1.1.2 Variable Selection

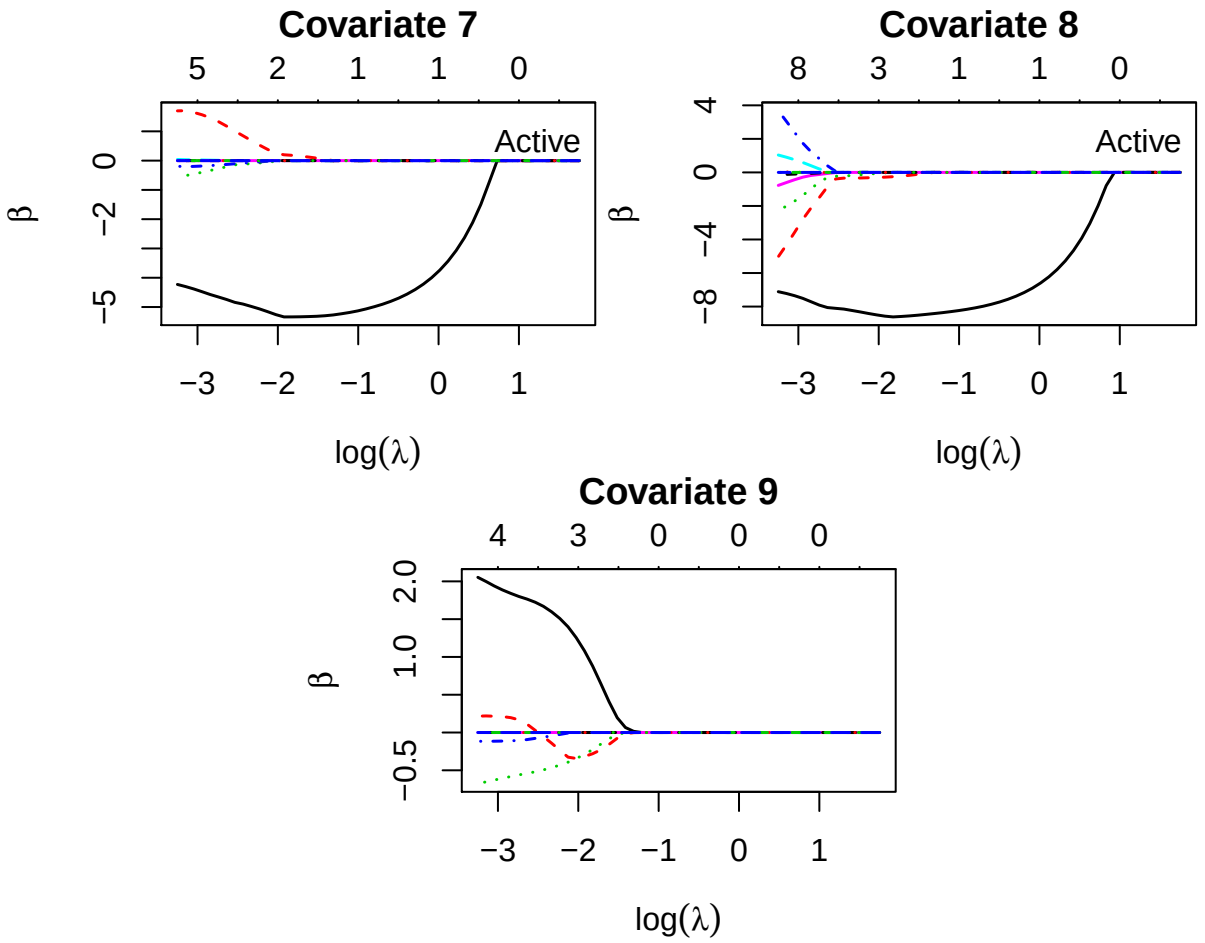
Prior to the comparison between the lasso and `HierBasis` estimators' ability to correctly select active features, we explore the properties of the `HierBasis` estimates. First, we investigate the paths the coefficients take as a function of the tuning parameter λ. For the simulation above we find the following coefficients (with nonzero values corresponding to the active features of $\mathbb{X}$)

```
round(beta[2:length(beta)], 2) # exclude intercept
```

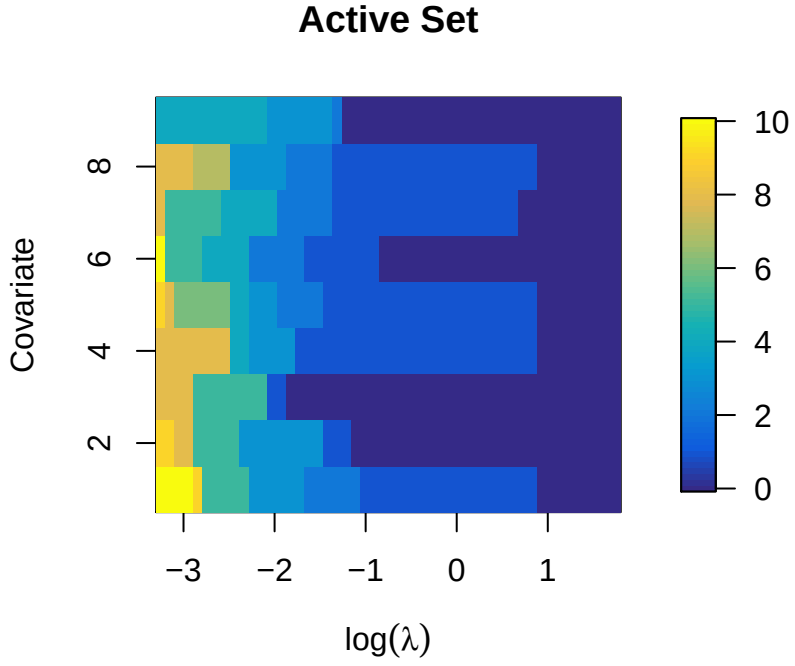
```
## [1] -6.83  0.00  0.00  8.61  7.20  0.00 -5.75 -9.02  0.00
```

Plotted below are a set of graphs illustrating the coefficient paths (excluding the intercept), with each predictor presented in a separate plot. Plotted above each figure is the number of active hierarchical basis features for the corresponding value of λ, with the diffrent lines corresponding to the different hierarchical basis features


Covariate 1
10 3 1 1 0
Active
β
log(λ)
Covariate 2
8 3 0 0 0
β
log(λ)
Covariate 3
8 1 0 0 0
β
log(λ)
Covariate 4
8 3 1 1 0
Active
β
log(λ)
Covariate 5
6 2 1 1 0
Active
β
log(λ)
Covariate 6
5 2 1 0 0
β
log(λ)



We may also visualize the total number of active features the `HierBasis` estimators detect, where a feature is said to be active if all its basis estimates are estimated to zero. Plotted are the number of active features against the tuning parameter, with the covariate index marked on the y axis.



To compare the lasso and `HierBasis` estimators we display the estimates of $\boldsymbol{\beta}$ in both the cases of the one-standard-error rule tuning parameter λ_{1se}.

```
betahat.glmnet <- coef(mod.glmnet.cv, s = mod.glmnet.cv$lambda.1se)
betahat.ahb <- coef(mod.ahb.cv, lam.idx = mod.ahb.cv$lambda.1se.idx)

round(beta, 2)
```

```
##  [1] 14.73 -6.83  0.00  0.00  8.61  7.20  0.00 -5.75 -9.02  0.00
```

```
round(as.numeric(betahat.glmnet), 2)
```

```
##  [1] 15.85 -6.06  0.00  0.00  6.10  5.50  0.00 -4.18 -7.11  0.00
```

```
round(betahat.ahb$intercept, 2)
```

```
## lam.13
##  16.54
```

```
round(betahat.ahb$X[1:3,], 2)
```

```
##           X.1 X.2 X.3  X.4  X.5 X.6  X.7   X.8 X.9
## basis.1 -3.38   0   0 3.08 2.84   0 -1.5 -3.92   0
## basis.2  0.00   0   0 0.00 0.00   0  0.0  0.00   0
## basis.3  0.00   0   0 0.00 0.00   0  0.0  0.00   0
```

Once again, we find `HierBasis` to be comparable to the lasso. In this case, both estimates correctly select the active features after the application of the one-standard-error rule.

3.1.2 Additive Models

The sparse additive `HierBasis` framework was original envisaged to be particularly well-suited to additive modelling. For this reason we now consider the (potentially nonlinear) relationship

$$y_i = \sum_{j=1}^{p} f_j(x_{ij}) + \varepsilon_i,$$

as defined in the introduction, and with ε_i defined analogously to the linear case (with SNR $= 3$). For our simulation we consider the additive relationship

$$y_i = 2 + 5f_1(x_{i1}) + 3f_2(x_{i2}) + 4f_3(x_{i3}) + 6f_4(x_{i4}) + \varepsilon_i,$$

such that

$$\begin{aligned}
f_1(x) &= x \\
f_2(x) &= (2x-1)^2 \\
f_3(x) &= \frac{2\sin(2\pi x)}{2-\sin(2\pi x)} \\
f_4(x) &= 0.1\sin(2\pi x) + 0.2\cos(2\pi x) + 0.3\sin^2(2\pi x) + 0.4\cos^3(2\pi x) + 0.5\sin^3(2\pi x).
\end{aligned}$$

We generate our features $\boldsymbol{x}_i \overset{\text{iid}}{\sim} \mathcal{N}_p(0,1)$ from i.i.d. standard normal deviates with n and p set immediately below.

```r
#===== data parameters =====#
set.seed(680)
n <- 1000 # number of observations
p <- 9 # number of predictors (excluding intercept)
SNR <- 3 # signal to noise ratio
nbasis <- 10

#===== functions =====#
f1 <- function(x) x
f2 <- function(x) (2 * x - 1)^2
f3 <- function(x) 2 * sin(2 * pi * x)/(2 - sin(2 * pi * x))
f4 <- function(x) 0.1 * sin(2 * pi * x) + 0.2 * cos(2 * pi * x) +
  0.3 * sin(2 * pi * x)^2 +
  0.4 * cos(2 * pi * x)^3 +
  0.5 * sin(2 * pi * x)^3

#===== generate data =====#
X <- matrix(rnorm(n * p), ncol = p) # iid normal deviates
y1 <- 5 * f1(X[, 1])
y2 <- 3 * f2(X[, 2])
y3 <- 4 * f3(X[, 3])
y4 <- 6 * f4(X[, 4])
ytrue <- 2 + y1 + y2 + y3 + y4

# compute noise dispersion satisfying the SNR ratio
sigma2 <- sum(ytrue^2)/(n - 1) * 1/SNR
eps <- rnorm(n, 0, sqrt(sigma2))

# compute perturbed response
y <- ytrue + eps

### split data into training and validation sets ###
X_train <- X[1:(n/2),]
X_valid <- X[(n/2 + 1):n,]
y_train <- y[1:(n/2)]
y_valid <- y[(n/2 + 1):n]
```

3.1.2.1 Prediction

Once again, we set $K = \texttt{nbasis} = 10$ to be the number of basis features to consider, and perform 10-fold cross-validation over the set of tuning parameters λ.

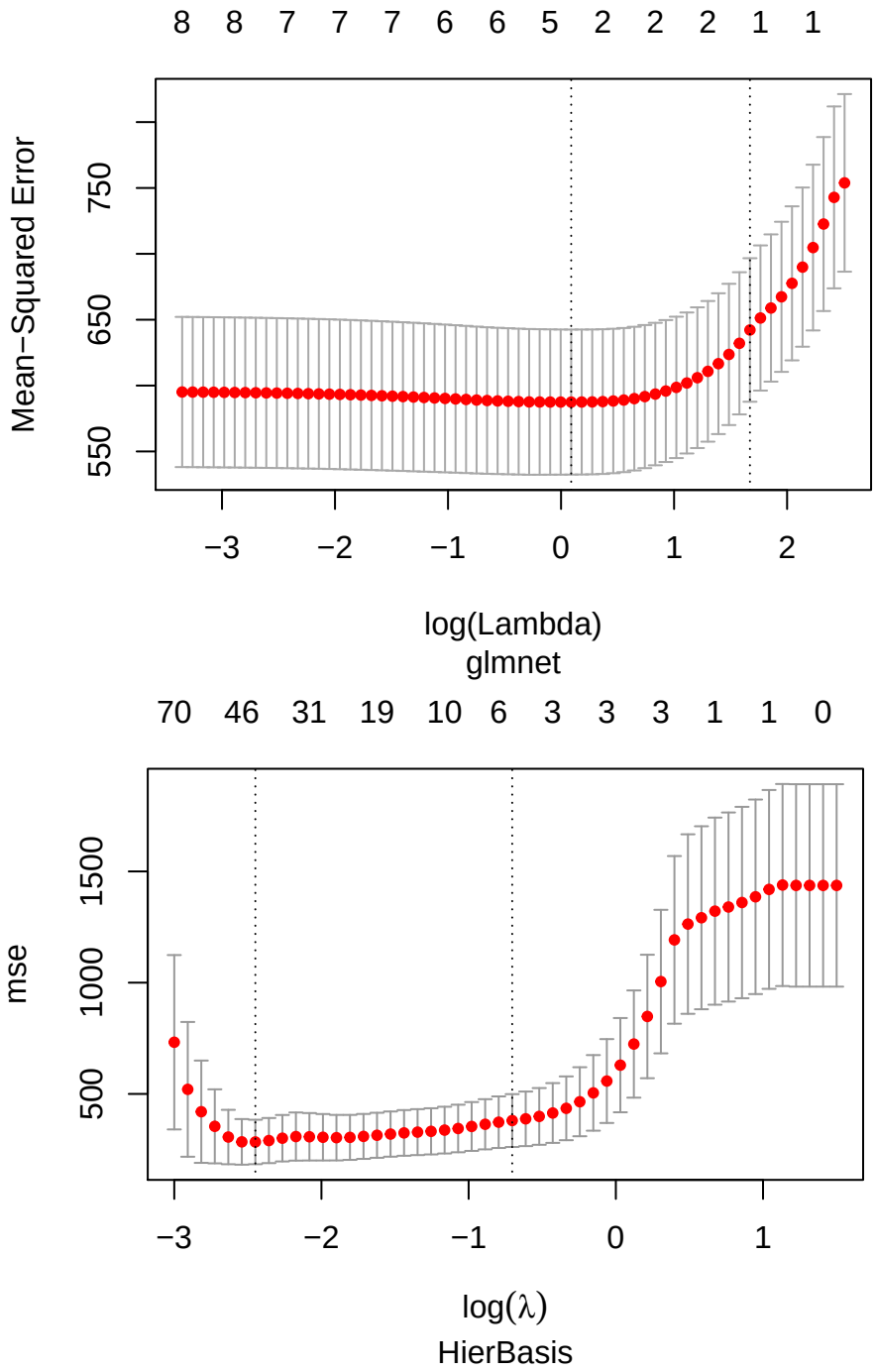
```r
#===== fit models =====#
# lasso
pt <- proc.time()
mod.glmnet.cv <- cv.glmnet(x = X_train, y = y_train, alpha = 1)
proc.time() - pt
```

```
##    user  system elapsed
##   0.191   0.003   0.198
```

```r
# additive hierbasis
pt <- proc.time()
mod.ahb.cv <- cv.additivehierbasis(X = X_train, y = y_train, nbasis = nbasis,
```

```
                                  lambdas = exp(seq(1.5, -3, length.out = 50)))
proc.time() - pt
```

```
##    user  system elapsed
##   0.526   0.021   0.674
```



We again use the validation sets to compare estimator performance under both the least testing error, $\lambda_{\min}$, and the one-standard-error rule, λ_{1se}, tuning parameters

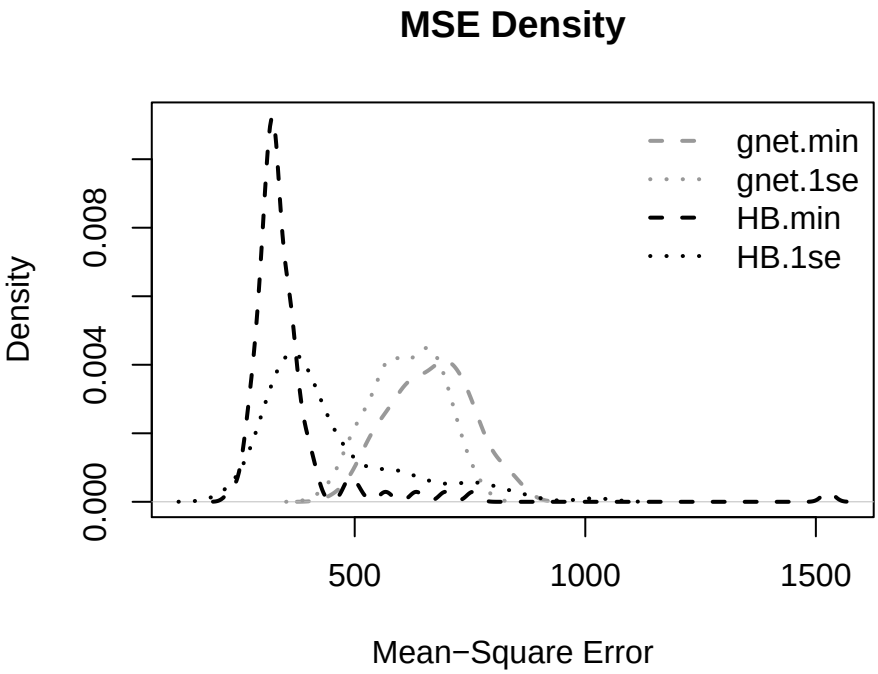
```
### predict validation sets ###
yhat.glmnet.min <- predict(mod.glmnet.cv, X_valid, s = mod.glmnet.cv$lambda.1se)
```

```
yhat.glmnet.1se <- predict(mod.glmnet.cv, X_valid, s = mod.glmnet.cv$lambda.min)
yhat.ahb.min <- predict(mod.ahb.cv, X_valid, lam.idx = mod.ahb.cv$lambda.min.idx)
yhat.ahb.1se <- predict(mod.ahb.cv, X_valid, lam.idx = mod.ahb.cv$lambda.1se.idx)

err.glmnet.min <- yhat.glmnet.min - y_valid
err.glmnet.1se <- yhat.glmnet.1se - y_valid
err.ahb.min <- yhat.ahb.min - y_valid
err.ahb.1se <- yhat.ahb.1se - y_valid

mse.glmnet.min <- mean(err.glmnet.min^2)
mse.glmnet.min.sd <- sd(err.glmnet.min)
mse.glmnet.1se <- mean(err.glmnet.1se^2)
mse.glmnet.1se.sd <- sd(err.glmnet.1se)
mse.ahb.min <- mean(err.ahb.min^2)
mse.ahb.min.sd <- sd(err.ahb.min)
mse.ahb.1se <- mean(err.ahb.1se^2)
mse.ahb.1se.sd <- sd(err.ahb.1se)

mse <- cbind(
  c(mse.glmnet.min, mse.glmnet.1se, mse.ahb.min, mse.ahb.1se),
  c(mse.glmnet.min.sd, mse.glmnet.1se.sd, mse.ahb.min.sd, mse.ahb.1se.sd))
colnames(mse) <- c("MSE", "MSE.SD")
rownames(mse) <- c("glmnet.min", "glmnet.1se", "HierBasis.min", "HierBasis.1se")
round(mse, 2)
```

```
##                  MSE MSE.SD
## glmnet.min    497.08  22.23
## glmnet.1se    449.94  21.17
## HierBasis.min 245.58  15.67
## HierBasis.1se 243.51  15.58
```

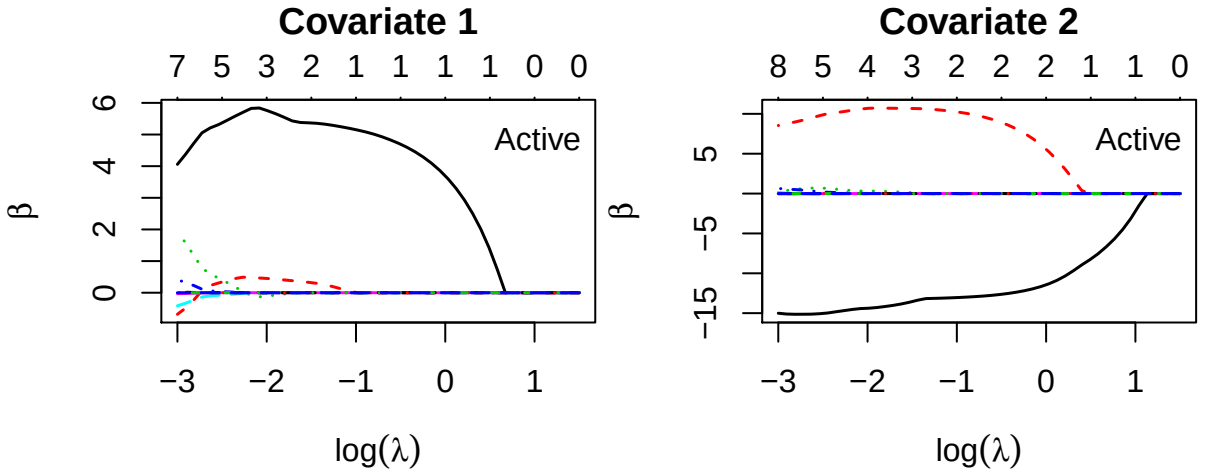
Although it may not be immediately obvious from the cross-validaiton figures, the numeric results demonstrate a clear advantage of the `HierBasis` estimator's ability to capture additive relationships in the data (at the cost of computational performance). We may see this comparison more clearly (and rigorously) by performing the above computations a number of times in order to generate a distribution of squared residuals. To this end, we repeat this process `nsims` $= 100$ times and plot the results below

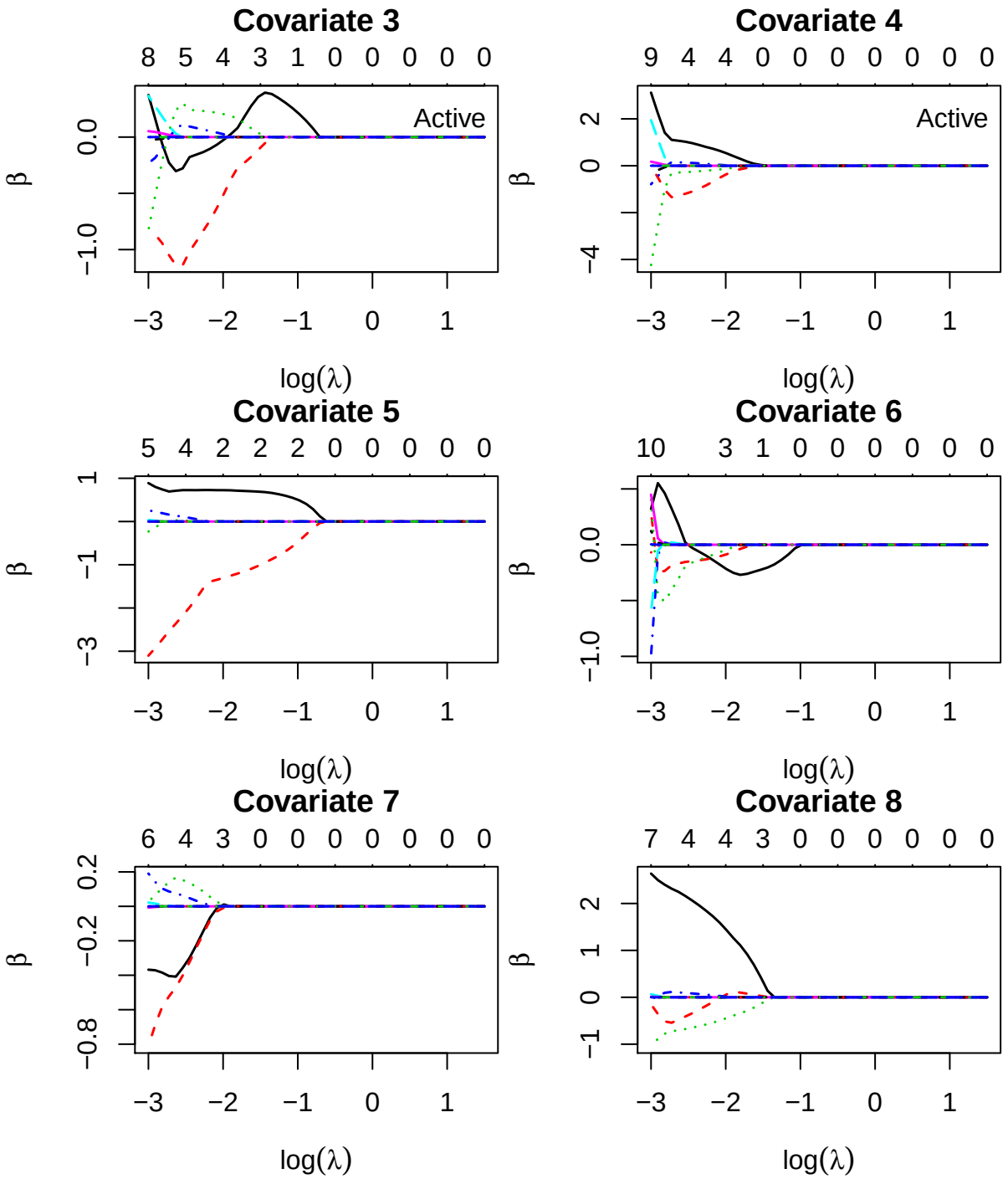
```
##    user  system elapsed
##  60.493   1.654  64.004
```

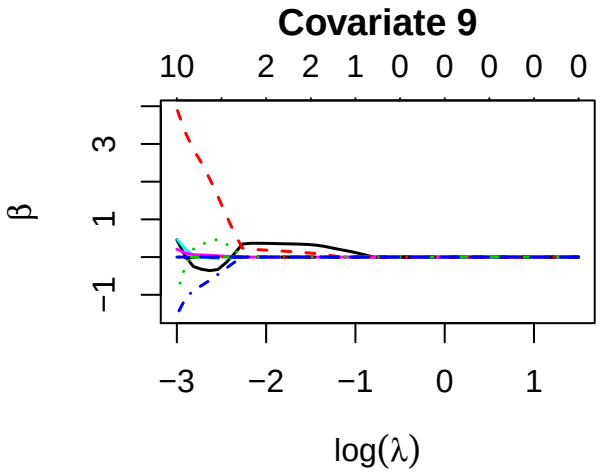


3.1.2.2 Variable Selection

We repeat the figures from the linear model variable selection section for the additive modelling environment. First, we investigate `HierBasis`'s ability to select the correct features from the design matrix via the coefficient plots as a function of λ




Covariate 3
8 5 4 3 1 0 0 0 0 0
Active
β
0.0
−1.0
−3 −2 −1 0 1
log(λ)
Covariate 4
9 4 4 0 0 0 0 0 0 0
Active
β
2
0
−4
−3 −2 −1 0 1
log(λ)
Covariate 5
5 4 2 2 2 0 0 0 0 0
β
1
−1
−3
−3 −2 −1 0 1
log(λ)
Covariate 6
10 3 1 0 0 0 0 0 0
β
0.0
−1.0
−3 −2 −1 0 1
log(λ)
Covariate 7
6 4 3 0 0 0 0 0 0 0
β
0.2
−0.2
−0.8
−3 −2 −1 0 1
log(λ)
Covariate 8
7 4 4 3 0 0 0 0 0 0
β
2
1
0
−1
−3 −2 −1 0 1
log(λ)



3.2 Manipulating the Mixing Parameter α

The `HierBasis` documentation references a mixing parameter α which controls the degree to which the two penalty terms are balanced according to

$$\alpha\lambda\sum_{j=1}^{p}\Omega_j\left(\boldsymbol{\beta}_j\right)+(1-\alpha)\frac{\lambda^2}{\sqrt{n}}\sum_{j=1}^{p}\left\|\Psi_K^{(j)}\boldsymbol{\beta}_j\right\|_2^2.$$

That is, α controls the balance between the hierarchical penalty and the sparsity penalty. A natural question is to ask what affect does varying α have on variable selection? To address this we fit an additive `HierBasis` model specifying $\alpha \in \left\{\frac{i}{n_\alpha}\right\}_{i=1}^{n_\alpha}$. For this simulation study we consider the same additive model as before

$$y_i = 2 + 5f_1(x_{i1}) + 3f_2(x_{i2}) + 4f_3(x_{i3}) + 6f_4(x_{i4}) + \varepsilon_i,$$

with f_1, f_2, f_3, f_4 the same as in the previous sections. We fit the additive `HierBasis` model specifying `nbasis` $= 10$ and selecting $\lambda_{1\text{se}}$ as the tuning parameter corresponding to the fitted values of $\widehat{\boldsymbol{\beta}}$. For each value of α we check whether the active predictors (X_1, X_2, X_3, X_4) are correctly identified as being active via the fitted $\widehat{\boldsymbol{\beta}}$, i.e., we check whether $\widehat{\boldsymbol{\beta}}_1, ..., \widehat{\boldsymbol{\beta}}_4 = \mathbf{0}$ and whether $\widehat{\boldsymbol{\beta}}_5, ..., \widehat{\boldsymbol{\beta}}_9 = \mathbf{0}$. We perform this process `nsims` $= 50$ times in order to get an estimate of the rates of variable selection/rejection, as well as computing the corresponding type I and type II errors.

```
#===== functions =====#
f1 <- function(x) x
f2 <- function(x) (2 * x - 1)^2
f3 <- function(x) 2 * sin(2 * pi * x)/(2 - sin(2 * pi * x))
f4 <- function(x) 0.1 * sin(2 * pi * x) + 0.2 * cos(2 * pi * x) +
  0.3 * sin(2 * pi * x)^2 +
  0.4 * cos(2 * pi * x)^3 +
  0.5 * sin(2 * pi * x)^3

#===== data parameters =====#
set.seed(680)
nsims <- 25
n_alpha <- 20

alpha <- seq(0, 1, length.out = n_alpha + 1); alpha <- alpha[alpha != 0]
n <- 1500 # number of observations
```

```r
p <- 9 # number of predictors (excluding intercept)
SNR <- 4 # signal to noise ratio

is_nonzero <- c(rep(T, 4), rep(F, p - 4))

#===== SIMULATE =====#
pt <- proc.time()
sim <- replicate(nsims, {
  #===== generate data =====#
  X <- matrix(rnorm(n * p), ncol = p) # iid normal deviates
  y1 <- 5 * f1(X[, 1])
  y2 <- 3 * f2(X[, 2])
  y3 <- 4 * f3(X[, 3])
  y4 <- 6 * f4(X[, 4])
  ytrue <- 2 + y1 + y2 + y3 + y4
  # compute noise dispersion satisfying the SNR ratio
  sigma2 <- sum(ytrue^2)/(n - 1) * 1/SNR
  eps <- rnorm(n, 0, sqrt(sigma2))
  # compute perturbed response
  y <- ytrue + eps

  ### split data into training and validation sets
  X_train <- X[1:(n/2),]; X_valid <- X[(n/2 + 1):n,]
  y_train <- y[1:(n/2)]; y_valid <- y[(n/2 + 1):n]

  #===== fit models =====#
  mods <- vector(mode = "list", length = length(alpha))
  select.tab <- matrix(nrow = p, ncol = length(alpha))
  err <- vector(mode = 'numeric', length = length(alpha))

  for (i in 1:length(alpha)) {
    mods[[i]] <- cv.additivehierbasis(X = X_train, y = y_train, alpha = alpha[i])

    beta.1se <- coef(mods[[i]], lam.idx = mods[[i]]$lambda.1se.idx)
    yhat.1se <- predict(mods[[i]]$model.fit, new.X = X_valid,
                        lam.idx = mods[[i]]$lambda.1se.idx)
    beta.select <- apply(beta.1se$X, 2, function(b) sum(abs(b)) != 0)

    err[i] <- mean((yhat.1se - y_valid)^2)
    select.tab[,i] <- beta.select
  }

  list("select" = select.tab, "err" = err)
})
proc.time() - pt
```

```
##    user  system elapsed
## 494.680  25.435 955.351
```

Now, we compute the estimated classification rates

$$
\begin{aligned}
p_{00,j} &= \widehat{\mathbb{P}}\left(\widehat{\boldsymbol{\beta}}_j = \mathbf{0} \mid X_j \text{ is inactive}\right) \quad \text{(True Negative)} \\
p_{10,j} &= \widehat{\mathbb{P}}\left(\widehat{\boldsymbol{\beta}}_j \neq \mathbf{0} \mid X_j \text{ is inactive}\right) \quad \text{(Type I)} \\
p_{01,j} &= \widehat{\mathbb{P}}\left(\widehat{\boldsymbol{\beta}}_j = \mathbf{0} \mid X_j \text{ is active}\right) \quad \text{(Type II)} \\
p_{11,j} &= \widehat{\mathbb{P}}\left(\widehat{\boldsymbol{\beta}}_j \neq \mathbf{0} \mid X_j \text{ is active}\right) \quad \text{(True Positive)},
\end{aligned}
$$

such that

$$
\widehat{\mathbb{P}}\left(\widehat{\boldsymbol{\beta}}_j = \mathbf{0} \mid X_j \text{ is (in)active}\right) = \frac{1}{\texttt{nsims}} \sum_{l=1}^{\texttt{nsims}} \mathbf{1}_{\left\{\widehat{\boldsymbol{\beta}}_j = \mathbf{0} \mid X_j \text{ is (in)active}\right\}}
$$

so that the estimates of the classification rates are

$$
p_{n_1 n_2} = \frac{1}{p} \sum_{j=1}^{p} p_{n_1 n_2, j}, \quad n_1, n_2 = 0, 1.
$$

Thus,

$$
\begin{aligned}
\widehat{\mathbb{P}}\left(\text{correct variable selection/rejective}\right) &= p_{00} + p_{11} \\
\widehat{\mathbb{P}}\left(\text{incorrect variable selection/rejection}\right) &= p_{01} + p_{10} \\
\widehat{\mathbb{P}}\left(\text{type I error}\right) &= p_{10} \\
\widehat{\mathbb{P}}\left(\text{type II error}\right) &= p_{01}.
\end{aligned}
$$

and

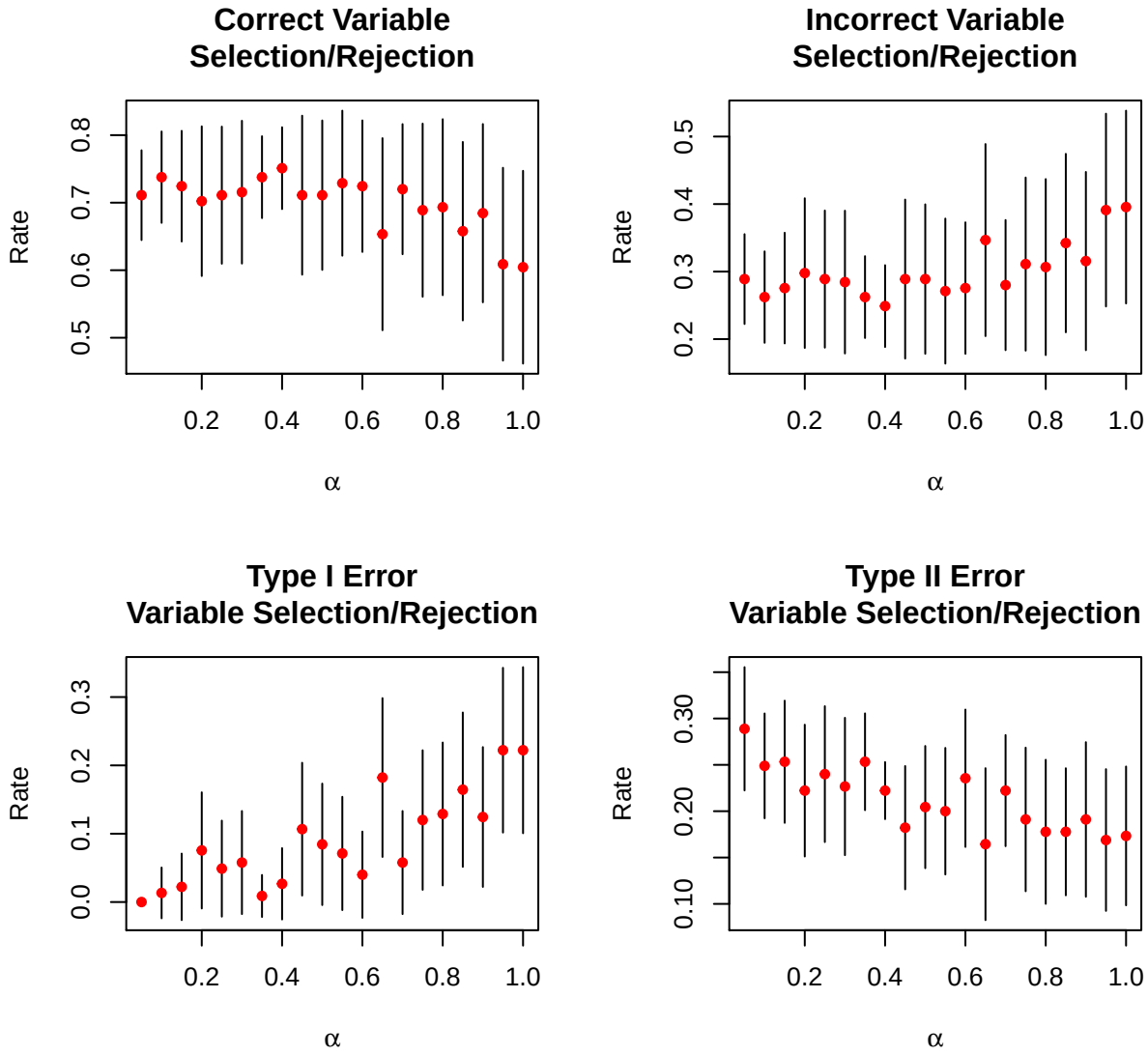
$$
\widehat{\text{Var}}\left(p_{n_1 n_2}\right) = \frac{1}{p^2 \cdot \texttt{nsims}} \sum_{l=1}^{\texttt{nsims}} \sum_{j=1}^{p} p_{n_1 n_2, j} \cdot (1 - p_{n_1 n_2, j}), \quad n_1, n_2 = 0, 1.
$$

To this end,

```
p_00j <- Reduce("+",
                lapply(sim["select",],
                       function(ss) apply(ss, 2,
                                          function(s) (s == F) & (is_nonzero == F))))/nsims
p_11j <- Reduce("+",
                lapply(sim["select",],
                       function(ss) apply(ss, 2,
                                          function(s) (s == T) & (is_nonzero == T))))/nsims
p_01j <- Reduce("+",
                lapply(sim["select",],
                       function(ss) apply(ss, 2,
                                          function(s) (s == F) & (is_nonzero == T))))/nsims
p_10j <- Reduce("+",
                lapply(sim["select",],
                       function(ss) apply(ss, 2,
                                          function(s) (s == T) & (is_nonzero == F))))/nsims

p_1j <- p_11j + p_00j # total correct sel/rej for each predictor
p_0j <- p_01j + p_10j # total incorrect sel/rej for each predictor
```

```
# mean over all p variables
p_00 <- colMeans(p_00j); p_11 <- colMeans(p_11j)
p_10 <- colMeans(p_10j); p_01 <- colMeans(p_01j)
p_1 <- colMeans(p_1j) # total correct sel/rej
p_0 <- colMeans(p_0j) # total incorrect sel/rej

# empirical variances
v_00 <- apply(p_00j * (1 - p_00j), 2, sum)/p^2
v_11 <- apply(p_11j * (1 - p_11j), 2, sum)/p^2
v_01 <- apply(p_01j * (1 - p_01j), 2, sum)/p^2
v_10 <- apply(p_10j * (1 - p_10j), 2, sum)/p^2
v_1 <- apply(p_1j * (1 - p_1j), 2, sum)/p^2
v_0 <- apply(p_0j * (1 - p_0j), 2, sum)/p^2
```

Plotted below are the empirical selection/rejection probabilities (as defined above) as a function of α, with the corresponding standard deviations above and below the mean probabilities.



From these figures we find little evidence that manipulating α has a material impact on overall variable selection/rejection rates. That is, the probability that a feature is correctly identified as being either a

part or not a part of our additive model appears to be independent of α, with perhaps a slight decrease in classification rates as $\alpha \to 1$. One way to interpret this slight decrease in the feature detection performance of the `HierBasis` model is by noting that as $\alpha \to 1$ we find our penalty retains only the hierarchical term

$$\lambda \sum_{j=1}^{p} \Omega_j(\boldsymbol{\beta}_j) = \frac{\lambda}{\sqrt{n}} \sum_{j=1}^{p} \sum_{k=1}^{\mathtt{K}} w_k \left\| \Psi_{k:K}^{(j)} \boldsymbol{\beta}_{j,k:K} \right\|_2$$

and drops the second additional sparsity-inducing penalty

$$\frac{\lambda^2}{\sqrt{n}} \sum_{j=1}^{p} \left\| \Psi_{K}^{(j)} \boldsymbol{\beta}_j \right\|_2$$

The effect of dropping this secondary sparsity-inducing term is made more apparent by the type I and type II figures. As $\alpha \to 1$ and this additional sparsity-inducing term vanishes we expect estimates $\widehat{\boldsymbol{\beta}}$ that are hierarchically sparse (i.e. only the first few orders of the basis expansion are expected to be estimated as nonzero), but are nonsparse across the p input features $\mathbb{X} = (X_1, ..., X_p)$. That is, as $\alpha \to 1$ we expect to see more features to be estimated as nonzero/influencial, leading a greater occurence of type I errors.

Likewise, as $\alpha \to 0$, we find that the hierarchical-sparsity term vanishes while the additional sparsity-inducing term is able to exert its full force on the penalty. As a result we expect the solutions $\widehat{\boldsymbol{\beta}}$ to be sparse with respect to the input features $\mathbb{X}$, but nonsparse with respect to a hierarchical structure (i.e. higher degrees of the basis expansion will be estimated as nonzero). So, as $\alpha \to 0$, we expect to see fewer features selected (but each selected feature is expected to have a higher-order basis expansion), leading to a greater occurence of type II errors.

The above simulation agrees with this interpretation of our mixing parameter α, as can be seen from the increasing rate of type I errors and decreasing rate of type II errors as a function of α.

3.3 Manipulating Penalty Weights w_k

3.3.1 Polynomial Growth: Manipulating m

We now look more closely at the hierarchical-sparsity inducing penalty

$$\Omega_j(\boldsymbol{\beta}_j) = \frac{1}{\sqrt{n}} \sum_{k=1}^{K} w_k \left\| \Psi_{k:K}^{(j)} \boldsymbol{\beta}_{j,k:K} \right\|_2,$$

where $w_k = k^m - (k-1)^m$ are penalization weights for the k^{th}-order basis estimator. According to Haris, Shojaie, and Simon (2016b), the tuning parameter m is similar to the number of bounded derivatives employed in the simple project estimator (Cencov (1962)) and recommends the use of $m = 2$ or 3 in order to achieve good results, and that $m = 3$ is analogous to applying a cubic spline for smoothing. We explore this suggestion, and other values of m, below.

3.3.1.1 Linear Model

Once again, we consider a linear model

$$y_i = \boldsymbol{x}_i \boldsymbol{\beta} + \varepsilon_i,$$

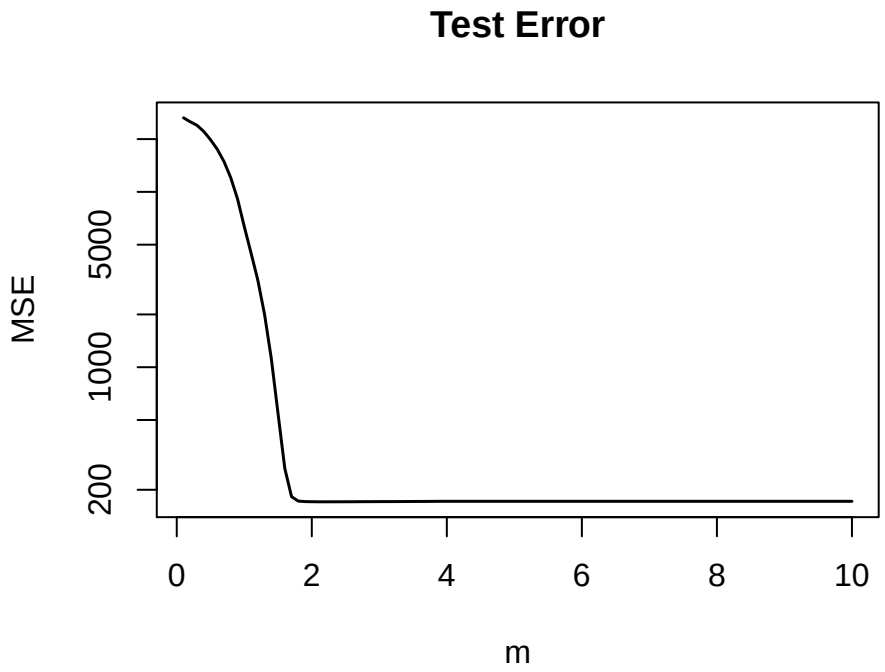
for $\boldsymbol{x}_i$, ε_i, and σ^2 as defined in the previous exploration of `HierBasis`'s performance in linear models. We will characterize the optimal value(s) of m by minimizing the MSE of a validation set (drawn from the same distribution as the training set). First, we generate our predictors and responses.

```r
#===== parameters =====#
set.seed(400)
n <- 1000 # number of observations
p <- 9 # number of predictors (excluding intercept)
SPARSE_PCT <- 0.5 # proportion of sparse predictors
# which predictors are sparse?
SPARSE_IDX <- sample(2:(p + 1), size = floor(SPARSE_PCT * p))
SNR <- 3 # signal-to-noise ratio (controls noise dispersions)
beta <- rnorm(p + 1, 0, 10)
beta[SPARSE_IDX] <- 0

#===== generate data =====#
X <- matrix(rnorm(n * p), ncol = p) # iid normal deviates
# compute noiseless response
ytrue <- cbind(1, X) %*% beta
# compute noise dispersion satisfying the SNR ratio
sigma2 <- sum(ytrue^2)/(n - 1) * 1/SNR
eps <- rnorm(n, 0, sqrt(sigma2))
# compute perturbed response
y <- ytrue + eps

### split data into training and validation sets
X_train <- X[1:(n/2),]; X_valid <- X[(n/2 + 1):n,]
y_train <- y[1:(n/2)]; y_valid <- y[(n/2 + 1):n]
```

Now, we fit multiple `HierBasis` models with varying values of m for fixed value of λ to be $\lambda = 0.25$.

```r
m.vec <- seq(0.1, 10, length.out = 100)
mse.train <- matrix(0, nrow = length(m.vec), ncol = 2)
mse.valid <- matrix(0, nrow = length(m.vec), ncol = 2)

pt <- proc.time()
for (i in 1:length(m.vec)) {
  mod.ahb <- additivehierbasis(X = X_train, y = y_train, m.const = m.vec[i],
                               nlam = 1, max.lambda = 0.25, lam.min.ratio = 1)
  yhat.train <- predict.additivehierbasis(mod.ahb, new.X = X_train)
  yhat.valid <- predict.additivehierbasis(mod.ahb, new.X = X_valid)

  err.train <- yhat.train - y_train
  err.valid <- yhat.valid - y_valid
  mse.train[i,1] <- mean(err.train^2); mse.train[i,2] <- sd(err.train)
  mse.valid[i,1] <- mean(err.valid^2); mse.valid[i,2] <- sd(err.valid)
}
proc.time() - pt
```

```
##    user  system elapsed
##   5.255   0.717  13.417
```



Interestingly, the analysis above selects an optimal value of m to be

```
# test error optimal m
m.vec[which(mse.valid[,1] == min(mse.valid[,1]))]
```

```
## [1] 2.2
```

terms of minimizing the validation MSE, which is within the range of the suggested values of $m = 2$ or 3.

3.3.1.2 Additive Model

We now consider the same additive model defined above. That is,

$$y_i = 2 + 5f_1(x_{i1}) + 3f_2(x_{i2}) + 4f_3(x_{i3}) + 6f_4(x_{i4}) + \varepsilon_i,$$

with f_1, f_2, f_3, f_4 the same as in the previous sections, and repeat the same process of validating m.

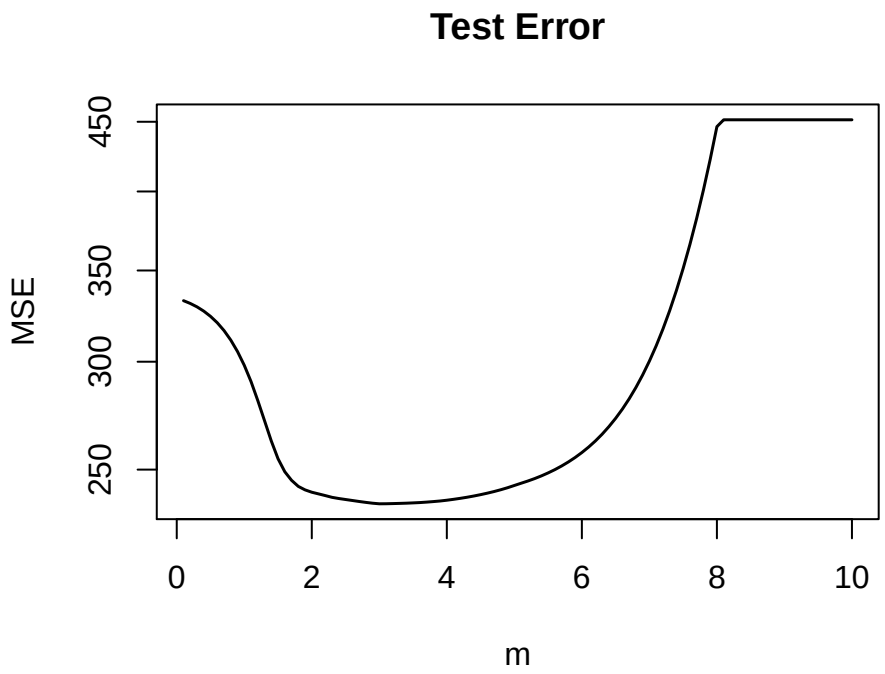
```
#===== data parameters =====#
set.seed(680)
n <- 1000 # number of observations
p <- 9 # number of predictors (excluding intercept)
SNR <- 3 # signal to noise ratio

#===== functions =====#
f1 <- function(x) x
f2 <- function(x) (2 * x - 1)^2
f3 <- function(x) 2 * sin(2 * pi * x)/(2 - sin(2 * pi * x))
f4 <- function(x) 0.1 * sin(2 * pi * x) + 0.2 * cos(2 * pi * x) +
  0.3 * sin(2 * pi * x)^2 +
  0.4 * cos(2 * pi * x)^3 +
  0.5 * sin(2 * pi * x)^3

#===== generate data =====#
X <- matrix(rnorm(n * p), ncol = p) # iid normal deviates
```

```
y1 <- 5 * f1(X[, 1])
y2 <- 3 * f2(X[, 2])
y3 <- 4 * f3(X[, 3])
y4 <- 6 * f4(X[, 4])
ytrue <- 2 + y1 + y2 + y3 + y4

# compute noise dispersion satisfying the SNR ratio
sigma2 <- sum(ytrue^2)/(n - 1) * 1/SNR
eps <- rnorm(n, 0, sqrt(sigma2))

# compute perturbed response
y <- ytrue + eps

### split data into training and validation sets ###
X_train <- X[1:(n/2),]; X_valid <- X[(n/2 + 1):n,]
y_train <- y[1:(n/2)]; y_valid <- y[(n/2 + 1):n]
```

We again will characterize the optimal value of m as the quantity minimizing the MSE for fixed $\lambda = 0.25$.

```
m.vec <- seq(0.1, 10, length.out = 100)
mse.train <- matrix(0, nrow = length(m.vec), ncol = 2)
mse.valid <- matrix(0, nrow = length(m.vec), ncol = 2)

pt <- proc.time()
for (i in 1:length(m.vec)) {
  mod.ahb <- additivehierbasis(X = X_train, y = y_train, m.const = m.vec[i],
                               nlam = 1, max.lambda = 0.25, lam.min.ratio = 1)
  yhat.train <- predict.additivehierbasis(mod.ahb, new.X = X_train)
  yhat.valid <- predict.additivehierbasis(mod.ahb, new.X = X_valid)

  err.train <- yhat.train - y_train
  err.valid <- yhat.valid - y_valid
  mse.train[i,1] <- mean(err.train^2); mse.train[i,2] <- sd(err.train)
  mse.valid[i,1] <- mean(err.valid^2); mse.valid[i,2] <- sd(err.valid)
}
proc.time() - pt
```

```
##    user  system elapsed
##   4.173   0.607   8.087
```




```
# test error optimal m
m.vec[which(mse.valid[,1] == min(mse.valid[,1]))]
```

```
## [1] 3
```

Once again, we note that the optimal m within the suggested range of $m = 2$ or 3, as noted in Haris, Shojaie, and Simon (2016b). The analysis above indicates that the optimal m value in terms of minimizing the MSE is found at $m = 3$ for this particular additive model.

References

Cencov, NN. 1962. "Estimation of an Unknown Density Function from Observations." In *Dokl. Akad. Nauk, Sssr*, 147:45–48.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1): 1–22. http://www.jstatsoft.org/v33/i01/.

Haris, Asad, Ali Shojaie, and Noah Simon. 2016a. *HierBasis: Nonparametric Regression and Sparse Additive Modeling*.

———. 2016b. "Nonparametric Regression with Adaptive Truncation via a Convex Hierarchical Penalty." *arXiv Preprint arXiv:1611.09972*.

Jenatton, Rodolphe, Julien Mairal, Guillaume Obozinski, and Francis Bach. 2011. "Proximal Methods for Hierarchical Sparse Coding." *Journal of Machine Learning Research* 12 (Jul): 2297–2334.

Jenatton, Rodolphe, Julien Mairal, Guillaume Obozinski, and Francis R Bach. 2010. "Proximal Methods for Sparse Hierarchical Dictionary Learning." In *ICML*, 487–94. 2010. Citeseer.

Tibshirani, Robert. 1996. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society. Series B (Methodological)*. JSTOR, 267–88.

Zhao, Peng, Guilherme Rocha, and Bin Yu. 2009. "The Composite Absolute Penalties Family for Grouped and Hierarchical Variable Selection." *The Annals of Statistics*. JSTOR, 3468–97.