Math 496: Report on Nonparametric Estimation with Splines

Genevieve Mendoza, Shengxuan Chen, and Hao Yin

2023-07-27

Theory of Splines

Splines are piecewise polynomial functions with some differentiability and continuity conditions, typically used for interpolation or smoothing of points. Their name derives from the historical usage of strips of wood which could be stretched up or down to interpolate points. For the same reason, the set of points which define the intervals of the piecewise function are known as knots.

The simplest way to calculate a spline estimation of a 2-dimensional cloud of points is to use the x-values as knots for the *truncated power basis.* We will also need to choose the order for the spline, which we will denote M. An M-spline r has degree $M-1$ and, since it is a piecewise polynomial, has continuous $r', \dots, r^{(m-2)}$ at each knot ξ_i. The most common method is to use 4-splines, which are cubic.

Given some knots $\tau_1, \dots, \tau_k$, and an order M, we can construct a truncated power basis. It takes the form:

$$x^0, x^1, \dots, x^{M-1}, (x-\tau_1)_+^{M-1}, \dots, (x-\tau_K)_+^{M-1}$$

This is a basis for the space of cubic splines with these knots, a four dimensional vector space over $\mathbb{R}$. Any such spline can be written as a linear combination of the elements of this basis (denoted ψ_j):

$$r(x) = \sum_{j=1}^{k+M} \beta_j \psi_j(x)$$

Because the spline takes this form, polynomial regression splines can be found with OLS, minimizing the sum of squared errors.

$$\begin{aligned} J &= k + M \\ \hat{y} &= \beta_1 \psi_1(x) + \cdots + \beta_J \psi_J(x) \\ \hat{\boldsymbol{\beta}} &= \arg\min \sum \left(y_i - \sum_{j=1}^{J} \beta_j \psi_j(x_i) \right) \end{aligned}$$

Citation: *All of Nonparametric Statistics*, Wasserman, and "Nonparametric regression using kernel and spline methods", Opsomer and Breidt 2016.

B-splines and linear smoothers

B-splines (short for basis splines) are a basis with compact support, which makes computations less intensive. Most software packages thus perform spline regression by projecting onto the basis of B-splines. To calculate B-splines, we start by defining (arbitrary) additional $2M$ knots outside of the support of the function to estimate.

$$\tau_1 \leq \dots \leq \tau_m \leq \tau_1 \leq \dots \leq \tau_k \leq \tau_{k+m+1} \leq \dots \leq \tau_{k+2m}$$

Then, the B-spline functions are defined recursively in terms of lower degree basis functions, starting with the zeroth degree basis functions, which is a simple (scaled) indicator function per knot. The i-th basis function

of degree k is denoted by (B_{i,k}(x) and is defined recursively as follows:

$$B_{i,0}(x) = c\mathbf{1}\{\tau_i, \tau_{i+n}\}$$
$$\sum_i B_{i,0}(x) = 1$$

Then, higher order B-splines are calculated through recursion:

$$B_{i,k+1}(x) = \frac{x - \tau_i}{\tau_{i+k+1} - \tau_i} B_{i,k}(x) + \frac{\tau_{i+k+2} - x}{\tau_{i+k+2} - \tau_{i+1}} B_{i+1,k}(x)$$

The degree of the b-spline basis functions determines the smoothness of the resulting spline function. Higher degree basis functions result in smoother curves, but can also be more computationally expensive to calculate.

Application to density estimation

We can also use our method of regression by OLS on the spline basis functions for density estimation after transforming the problem. We used the `logspline` R package to do this.

Regression with Splines

We used the `logspline` R package, which is based on the theory of Kooperberg and Stone. The method is to estimate $\log(f)$ by a function $s \in S$ with maximum likelihood estimation We are choosing from the space S_0 of cubic smoothing splines with fixed knots at order statistics. Then, normalize with a constant c, so $\widehat{\log} f = \widehat{\ell} = \hat{s} + c(\hat{s})$, where c makes $\int \exp\{\widehat{\ell}\} = 1$. This can be done by iterating over combinations of B-splines and calculating MLEs. This reduces the problem of density estimation to OLS, and the main choice we have is knot selection. This comes from "A study of logspline density estimation", Charles Kooperberg and Charles J. Stone (1991) and "Large Sample Inference for Log-Spline", Stone (1990)

Regression: Buja, Hastings, and Tibshirani

We also used the pspline package in our calculations. The method used by this package is described here. Let $h_i = x_{i+1} - x_i$, Δ and $(n-2) \times n$ matrix, C a symmetric matrix. Δ and C are tridiagonal, $\Delta_{i,i} = \frac{1}{h_i}$, $\Delta_{i,i+1} = -(\frac{1}{h_i} + \frac{1}{h_{i+1}})$, and $\Delta_{i,i+2} = \frac{1}{h_{i+1}}$, and $C_{i-1,i} = C_{i,i-1} = \frac{h_i}{6}$ and $C_{i,i} = \frac{h_i + h_{i+1}}{3}$.

$$K = \Delta^t C^{-1} \Delta, S = (I + \lambda K)^{-1}, \hat{y} = Sy$$

The below are examples for $n = 5$.

$$\Delta = \begin{bmatrix} \frac{1}{x_2 - x_1} & -(\frac{1}{x_2 - x_1} + \frac{1}{x_3 - x_2}) & \frac{1}{x_3 - x_2} & 0 & 0 \\ 0 & \frac{1}{x_3 - x_2} & -(\frac{1}{x_3 - x_2} + \frac{1}{x_4 - x_3}) & \frac{1}{x_4 - x_3} & 0 \\ 0 & 0 & \frac{1}{x_4 - x_3} & -(\frac{1}{x_4 - x_3} + \frac{1}{x_5 - x_4}) & \frac{1}{x_5 - x_4} \end{bmatrix}$$

$$C = \begin{bmatrix} \frac{(x_2 - x_1) + (x_3 - x_2)}{3} & \frac{x_3 - x_2}{6} & 0 \\ \frac{x_3 - x_2}{6} & \frac{(x_3 - x_2) + (x_4 - x_3)}{3} & \frac{x_4 - x_3}{6} \\ 0 & \frac{x_4 - x_3}{6} & \frac{(x_4 - x_3) + (x_5 - x_4)}{3} \end{bmatrix}$$

Solving for the smoothing spline is equivalent to minimizing

$$\|y - \hat{y}\|^2 + \lambda \hat{y}^t K \hat{y}$$

where $K = \Delta^t C^{-1} \Delta$, and $\hat{y} = (I + \lambda K)^{-1} y$.

Citation: "Linear Smoothers and Additive Models," Buja, Hastie, Tibshirani. Annals of Statistics, 17:2, 1989, 453–510.

Natural Splines and Linear Smoothers

Another reason to use splines is that minimizing the penalized sum of squares gives a natural spline

$$M(\lambda) = \sum_i (y_i - \hat{r_n}(x_i))^2 + \lambda J(r)$$

which has $\xi_i = x_i$, knots at the data points. These splines are also linear smoothers.

$$\ell_i(x) \approx \frac{1}{f(x_i)} K_h(x_i)$$

This result is proven in Silverman 1984. The method of penalized regression is elaborated below.

Penalized Regression (Hao)

Notably, the risk of overfitting rises as we increase the number of knots.

In this case, the penalty is used to prevent overfitting, therefore we don't need to worry about the selection of knot size.

To ensure sufficient flexibility and allow the penalty to prevent overfitting, we can simply tie a sizable number of knots—let's say 20, for example.

As explained before, the second derivative of a quadratic plus function is

$$\frac{d^2}{dx^2}(x-t)_+^2 = 2(x-t)_+^0$$

That means if we are using quadratic splines, then the spline's second derivative jumps by an amount $2b_k$ at the k^{th} knot, and this is shown in the red circles.

Overfitting occurs when there are many jumps (many knots) and the jumps are unconstrained.

The sum of the squared jumps is

$$4\sum_{k=1}^{K} b_k^2$$

The sum of squared residuals and the sum of squared leaps are two quantities that we want to be as minimal as possible, so we add them with a weight λ.

$$\hat{\beta} = \arg \min_{\beta} \sum_{i=1}^{n} \Big\{ Y_i - \big(\beta_0 + \beta_1 x + \beta_2 x^2 + b_1 (x - t_1)_+^2 + \cdots + b_K (x - t_K)_+^2 \big) \Big\}^2 + \lambda \sum_{k=1}^{K} b_k^2.$$

Alternatively, one can use this equation from Wasserman, All of Nonparametric Statistics.

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{J} \beta_j \psi_j(x_i) \right)^2 + \lambda \sum_{j=1}^{J} \beta_j^2$$

The penalty parameter can be any value between 0 and ∞, inclusive. If $\lambda = \infty$, then $b_1, \ldots, b_k$ are constrained to be zero since any positive value of these parameters makes the penalty infinite. This makes the quadratic spline equal to a single quadratic polynomial.

If $\lambda = 0$ then there is no penalty and we are using OLS or ordinary least squares.

In fact, a decent middle ground between these two options is to select an appropriate λ.

For example, we compare the fits with λ equal to 0, 5, and 10^{10} in the right graph. We can see that the fit with $\lambda = 10^{10}$ has little flexibility. It is a quadratic polynomial fit. The fit with $\lambda = 0$ is the OLS to a 25-knot spline to be an overfit. Using $\lambda = 5$ is a good compromise between these two extremes.

One question is how to find the optimal choice of λ that gives us the most accurate predictions of new data. Cross-validation is a technique that can precisely select λ.

We can remove one data point, estimate the regression function using the remaining data, and then use the estimated regression function to forecast the deleted observation to assess how well utilizing a certain trial value of works for prediction. Each data point in the sample can be used as the deleted point in this process, which can be repeated n times. As a result, we have n different estimates of the predicted squared prediction error when employing this.

To be specific, it follows the basic steps:

For $j = 1, \ldots, n$, let $\hat{s}(\cdot; \lambda, -j)$ be the regression function estimated using this and with the j^{th} data point deleted.

$$\sum_{i \neq j} \Big\{ Y_i - \Big(\beta_0 + \beta_1 x + \beta_2 x^2 + b_1 \left(x - t_1\right)_+^2 + \cdots + b_K \left(x - t_K\right)_+^2 \Big) \Big\}^2 + \lambda \sum_{k=1}^{K} b_k^2.$$

Let $s(\cdot; \lambda, -j)$ be the prediction of the value of Y_j using the other observations. Define

$$CV(\lambda) = n^{-1} \sum_{j=1}^{n} \left\{ Y_j - s\left(X_j; \lambda, -j\right) \right\}^2$$

to be the average squared error from these n predictions.

The value of that minimizes CV (λ) is regarded as the best since CV (λ) calculates the expected squared prediction error. The regression function's final estimate makes use of all the information and the value of that minimizes CV.

The $1 + p + K$ parameters, the intercept, the p coefficients of the powers x to x^p, and the K coefficients of the plus functions are present if we employ a pth degree spline with K knots.

If $\lambda = 0$, then all of these parameters are free to vary. If $\lambda = +\infty$, the estimated coefficients of the plus function are all be constrained to equal 0, so there are only $1 + p$ free parameters.

If $0\lambda + \infty$, then the "effective number of parameters" should be somewhere between $1 + p$ and $1 + p + K$. We use the complex matrix to measure the effective number of parameters.

$$\mathbf{X} = \begin{pmatrix} 1 & X_1 & \cdots & X_1^p & (X_1 - t_1)^p & \cdots & (X_1 - t_K)^p \\ 1 & X_2 & \cdots & X_2^p & (X_2 - t_1)^p & \cdots & (X_2 - t_K)^p \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_n & \cdots & X_n^p & (X_n - t_1)^p & \cdots & (X_n - t_K)^p \end{pmatrix}$$

If D is a square matrix $(1 + p + K)(1 + p + K)$ with all off-diagonal elements equal to zero and its diagonal elements equal to $1 + p$ zeros then K ones, then this is the case. When this is the case, the effective number of parameters, also known as the effective degrees of freedom (DF), is

$$DF(\lambda) = trace\left\{ \left(\mathbf{X}^\top \mathbf{X}\right) \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{D}\right)^{-1} \right\}$$

If $\lambda = 0$

$$DF(0) = trace\left\{\left(\mathbf{X}^\top\mathbf{X}\right)\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\right\} = trace\left(\mathbf{I}_{1+p+K}\right) = 1 + p + K$$

Let $\hat{s}(\cdot;\lambda,-j)$ be the estimated regression function using all the data. Then

$$\hat{\sigma}^2(\lambda) = \frac{\sum\left\{Y_i - \hat{s}\left(X_i;\lambda\right)\right\}^2}{n - DF(\lambda)}$$

A simpler method to measure CV (λ) is to calculate the generalized cross-validation statistic (GCV)

$$GCV(\lambda) = \frac{n^{-1}\sum\left\{Y_i - \hat{s}\left(X_i;\lambda\right)\right\}^2}{\left\{1 - \frac{DF(\lambda)}{n}\right\}^2}$$

Compared with CV (λ), $GCV(\lambda)$ only uses the estimate $\hat{s}$ computed from all the data. Of course, this would lead to some derivation from the correct value.

we can define AIC for P-splines as

$$AIC(\lambda) = n\log\left\{\hat{\sigma}^2(\lambda)\right\} + 2DF(\lambda)$$

We can then select λ by minimizing AIC.

There are several methods for the same question. Let's make a comparation. Time series analysts created AIC, parametric statistics gave rise to CV, and nonparametric regression, notably spline estimation, introduced GCV. Researchers from three independent teams approached the same broad issue and came up with three comparable, yet distinct, solutions. Given that Cp is comparable to CV, GCV, and AIC, there were actually four similar responses.

Let's make a summary. Splines that are fitted by penalized least squares are known as penalized splines. The sum of squared residuals and a penalty for the spline's roughness are reduced by penalized least squares. A nonnegative penalty parameter, here indicated by, multiplied by the sum of the squared coefficients of the plus functions is a typical roughness penalty. It's important to choose correctly, and generalized cross-validation (GCV) is a useful tool for doing so.

Numerical Studies (Shengxuan)

To further explore the properties and behaviors of splines, we decided to conduct four numerical experiments: Density estimation with simulated data, density estimation with real data, spline regression with simulated data and spline regression with real data.

Density estimation with simulated data

To start, we would consider a density estimation problem with known underlying distribution to test the performance of the spline density estimator. This will require an R package called "logspline". We would like to compare this with the kernel density estimator (KDE).

Experiment setup

In order to generate random samples, we have to first define the distribution where the samples are drawn from. To make the density estimation more challenging, we will consider a 3-mode Gaussian mixture that has the following parameters: Mean = (-5, -2, 4), s.t.d. = (0.6, 0.6, 1.5), p = (0.25, 0.25, 0.5). The "p" is the probability of drawing sampling from a particular mode. We will generate 1000 data points for this study.

To select bandwidth or parameters of the "logspline" function, each package already implemented their own selection method. For kernel density estimator, "bw.bcv" was used. For "logspline", the built-in selection method is used.

To evaluate to performance of each method, we approximate the integrated mean squared error (IMSE) for each method by performing Monte Carlo simulation (1000 times) and Riemann sum for approximation of the integral over x.

Results and discussion

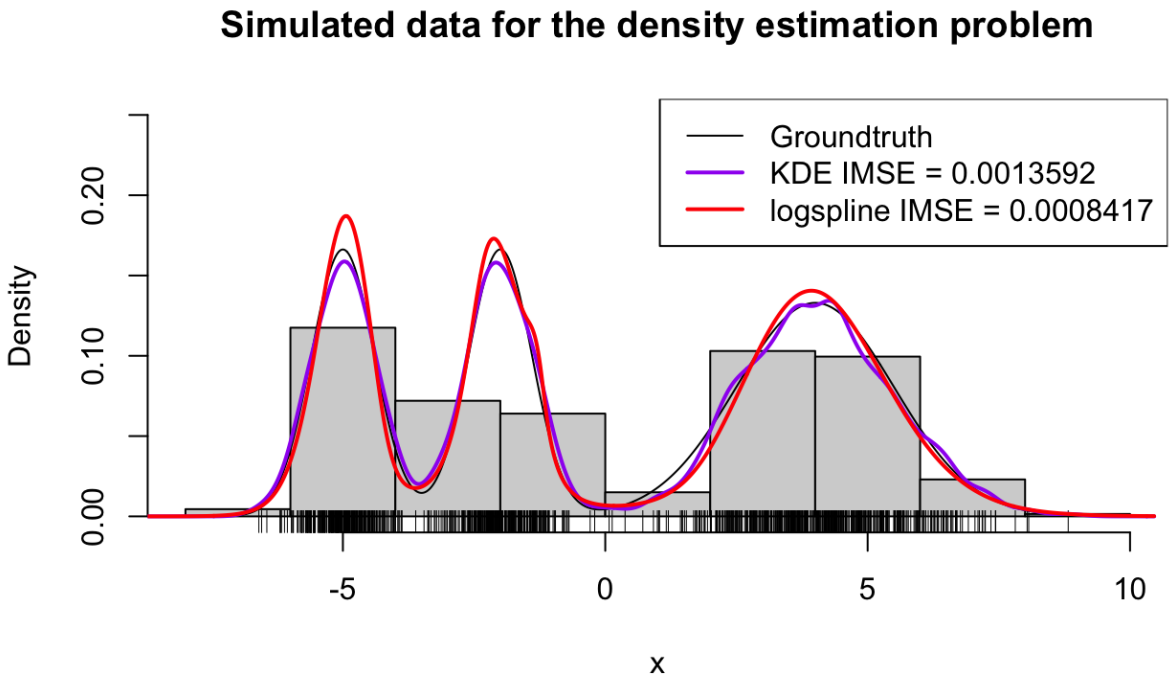

Figure 1: Density estimation with simulated data

In this figure, the rug of the data points, the true distribution, the histogram of the data and the density estimates from the two methods for one realization were plotted. Looking at the estimated densities from both methods, both of them performed very well visually. A little bit over estimate at the peak for spline and a more jaggy estimate for KDE is observed. In terms of IMSE, logspine" out performed KDE.

Density estimation with real data

It is always good to examine how methods work for real life data to make sure results we see in simulation can be translated to practice. In this experiment, we choose the Fiji Earthquake depth data from Wasserman (2005). It contains 1000 observations.

Experiment setup

To solve the problem of evaluating algorithm with no known distribution (`groundtruth`), we splitted the data randomly in training (25%) and test set (75%). The large portion of the data is used to generate a reliable reference or approximated ground truth. By having less data, we put more pressure on the estimator and is easier for us to spot the difference between the two methods.

The bandwidth selection procedure is the same as the previous experiment.

Since we don't know the true distribution (`groundtruth`), the evaluation method needs to be different. Here, we proposed two ways to evaluate the algorithms: log-liklihood given the test data and comparing the empirical cdf of the test data with the estimated cdf's.

Computing the log-likelihood of the two model given the test data will give us something related to the probability of observing the test data given the estimated model being true. The higher the log-likelihood, the better the data fits the model, which is desired.

When the estimated density is exactly the same as the true density, the distance between their cdf will be 0. by comparing the distance between the cdf's also gives us a measure of the performance. Here, L^2 distance was used for simplicity.

Results and discussion

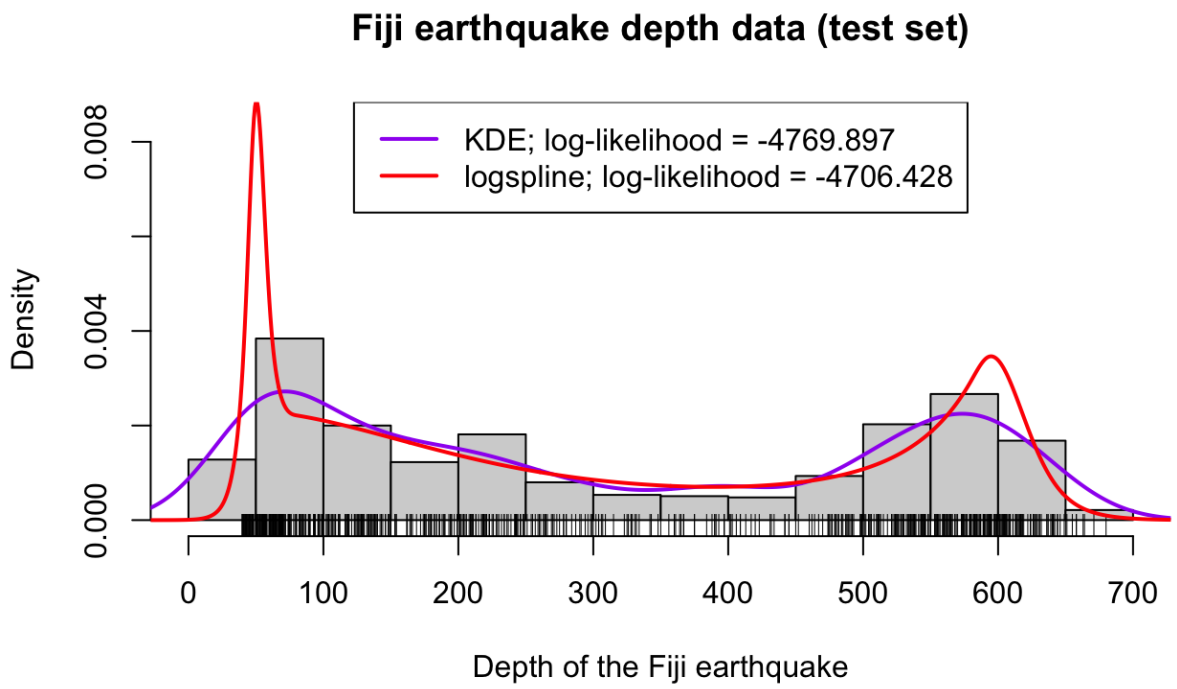

Figure 2: Density estimation with Fiji earthquake depth data

The estimated densities were plotted in figure 2, along with the rug and histogram of the test set data. The two methods behaved very differently here. The logspline estimate is very spiky near 0 and kernel density estimate is more jagged and with obvious edge bias. This is possibly due to the logspline model being more flexible, and perhaps the number of data points is not enough for "bw.bcv" to choose an optimal bandwidth for KDE. In terms of log-likelihood, logspline outperformed KDE.

By looking at the cdf plot, visually they are surprisingly similar. The edge bias of the KDE estimate can be seen near 0. The L^2 distance also indicates the logspline estimate is closer to the test set distribution.

Spline regression with simulated data

In this numerical study, we are also interested in the application of splines to regression problems. In this experiment, we will start with simulation data. At the same time, this is a good place to look into the effect

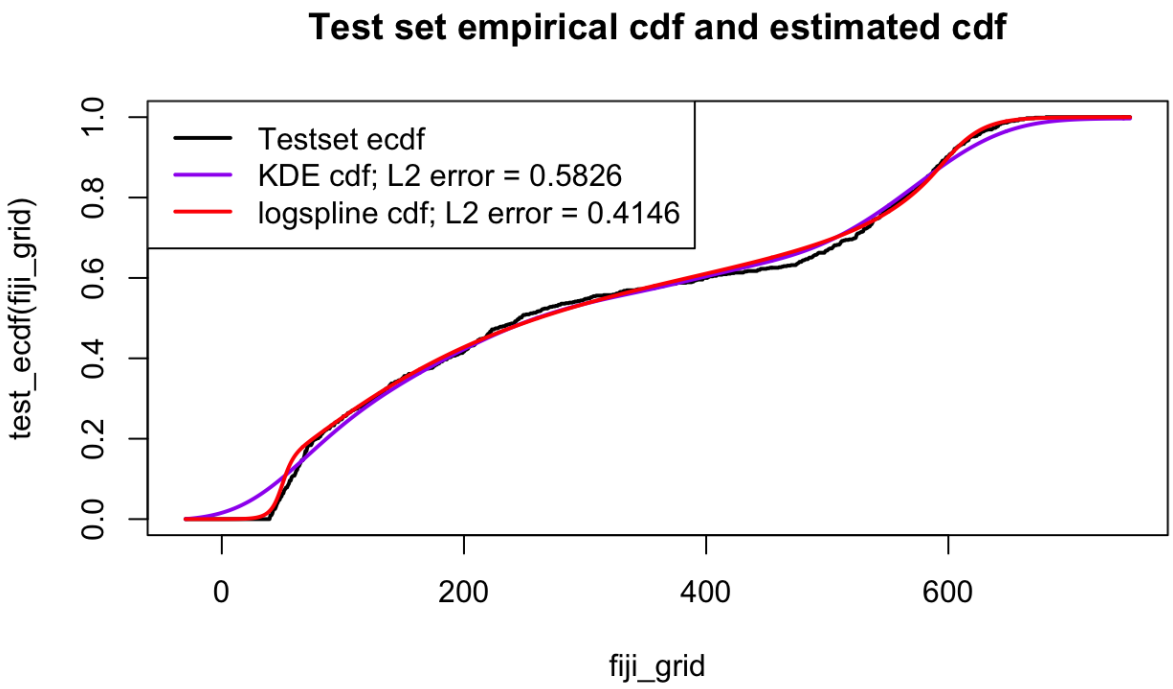

Figure 3: Empirical cdf plot for the test set data and estimated CDFs

of penalization term on the estimate, since we have the true distribution in simulation study. We will need the R package "spline"in this and the following study. As a comparison, we decided to use local polynomial regression implement by function "loess" and penalized spline regression function in package `pspline`.

Experiment setup

To make the regression problem more challenging, here we consider a pairwise constant function and added white Gaussian noise with s.t.d. 0.5. 1024 observations, one at each x value, were generated. Once the data set is generated, the data set was split to training (75%) and test set (25%).

We choose 2nd degree polynomial to allow more flexible model since the package only allow degree 0-2. For spline, we decided to use cubic spline since it is very common. The evaluation of the method is the same as the density estimation problem with simulated data.

Since there is no written function for bandwidth selection in either cases, bandwidth selection is implemented using custom written cross validation algorithm. In this algorithm, the training set was split to 3 fold and the model was used to fit onto two sets. The sum of square validation error was computed using the thrid fold of the training data and this process repeats for two more times by holding out other folds. The cross validation error was averaged and the bandwidth the minimized the averaged cross validation error was used as the final bandwidth. This method was used for bandwidth in local polynomial, the degrees of freedom in spline regression and the tuning parameter λ in penalized spline regression.

Results and discussion

Bandwidth selection for local polynomial went well. With some trial-and-error, we were able to identify the approximated bandwidth that minimizes the cross validation error.

For spline, the degree of freedom selection potentially had some issues. R threw warning messages about the

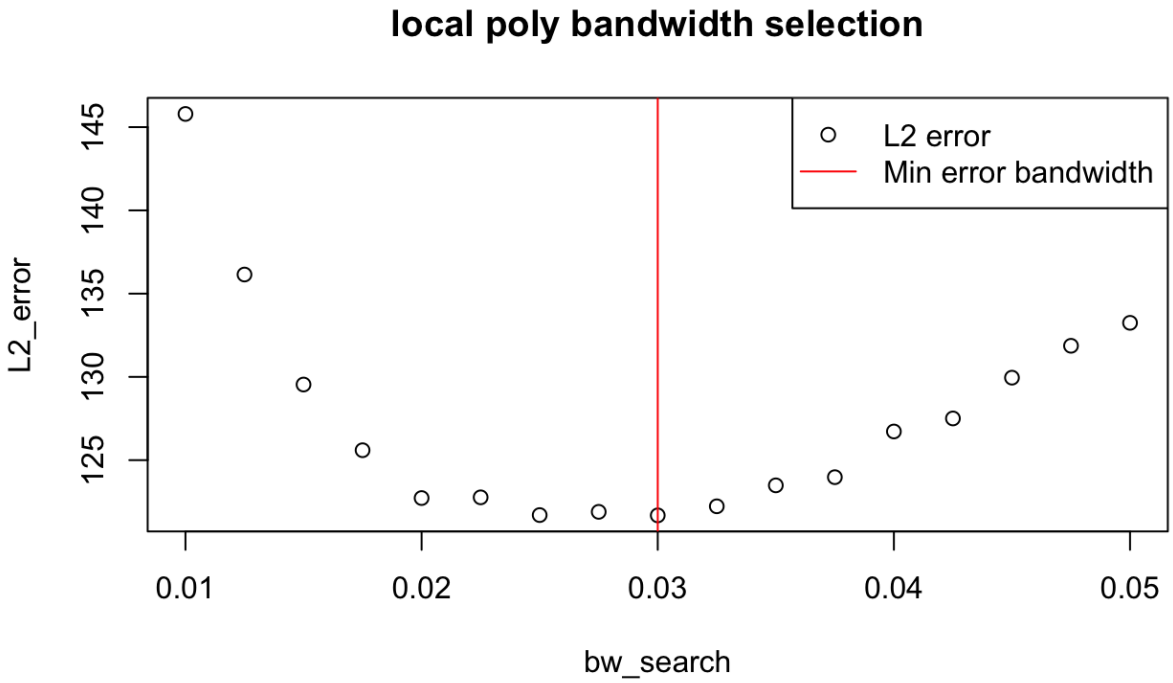

Figure 4: Bandwidth selection plot for local polynomial regression

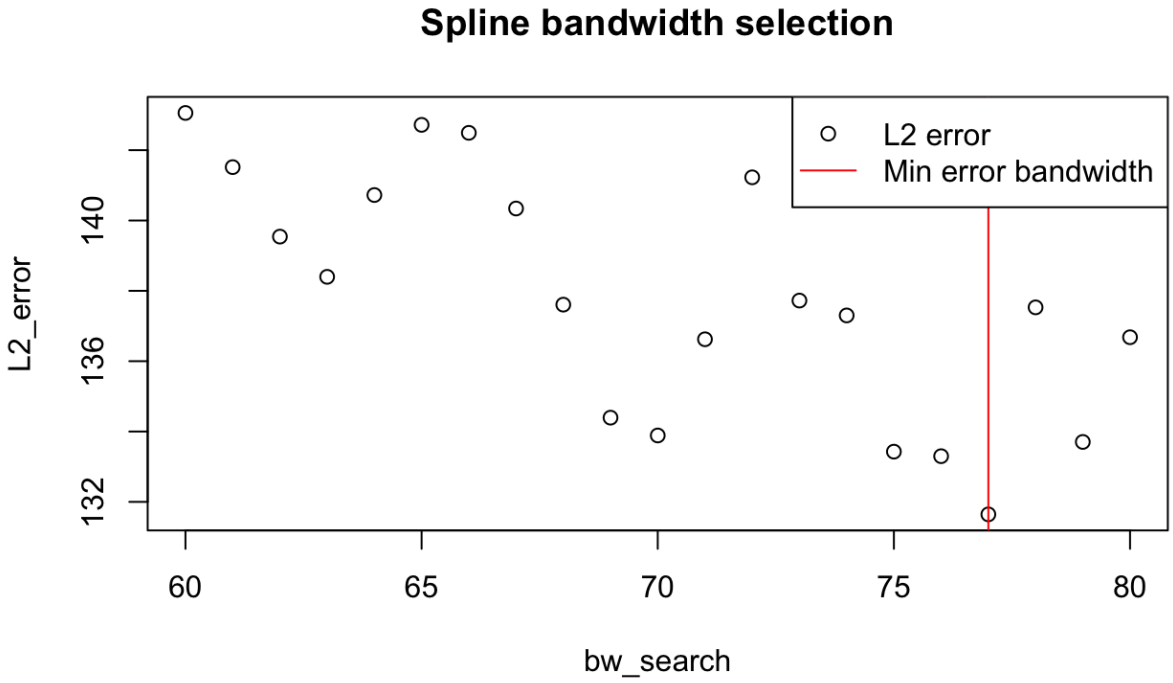

Figure 5: Bandwidth selection plot for spline regression

matrix condition and knot locations near the boundary. This might be due to the degree of freedom being to small or too large. When looking at the objective function values, it goes up and down when varying the degree of freedom. Regradless, we still picked the one that minimizes the cross validation error from the set of degree of freedom values we choice.

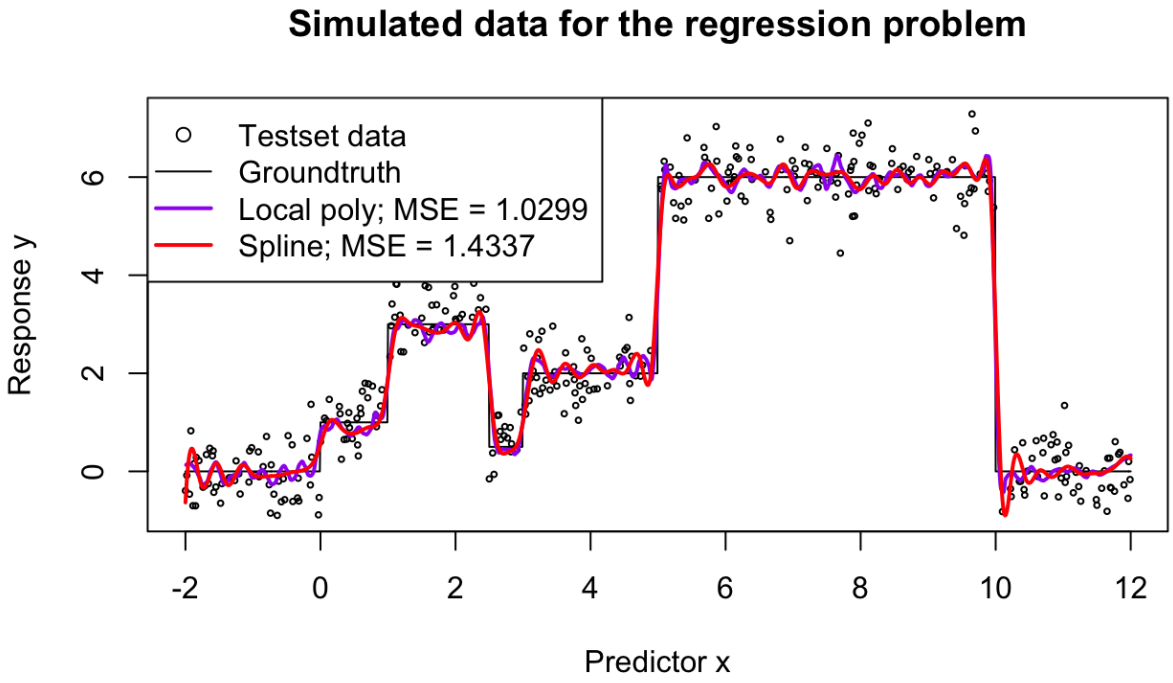

Figure 6: Regression problem with simulated data

Despite the challenges we had in the hyperparameters step, the fit for both of the methods work pretty nicely. In terms of IMSE, the local polynomial actually outperformed spline regression. This is perhaps expected since the ringing artifacts of the spline is much stronger than local polynomial.

To understand better how the penalization term works and if it helps improve the prediction, one can try the same procedure with penalized spline. The "pspline" package does not allow us to specify the degree of spline so only the tuning parameter λ was chosen using cross validation. The cross validation error for λ is somewhat surprising at the first glance. The λ that minimizes the cross validation is the one that is barely above 0. The cross validation goes way up when λ was set to 0 and it also went up monotonically as we increased λ, for the set of λ we tried.

Even though the λ selection had weird phenomenon, the selected λ actually resulted in a smaller IMSE compared to regular spline regression implemented by "spline". The local polynomial IMSE is still the lowest out of the three.

As we varied the λ value, the smoothing effect clearly showed up in the fit, which is very satisfying to see. This numerical results are align with the theory of spline regression.

Interestingly, when setting λ to 0, the resulting model is not the same as the model from "spline" and the model is severely overfit. This can be explained by the way "pspline" implements spline regression: pspline does not specify the order of spline and only uses λ to control for smoothness of the model; while "spline" package uses the spline order or the degree of freedom of the marix to control the behaviors of the model. When we set λ to 0, the "pspline" function will find a spline model that has arbitary order and try to fit that to the data, which causes the overfit.

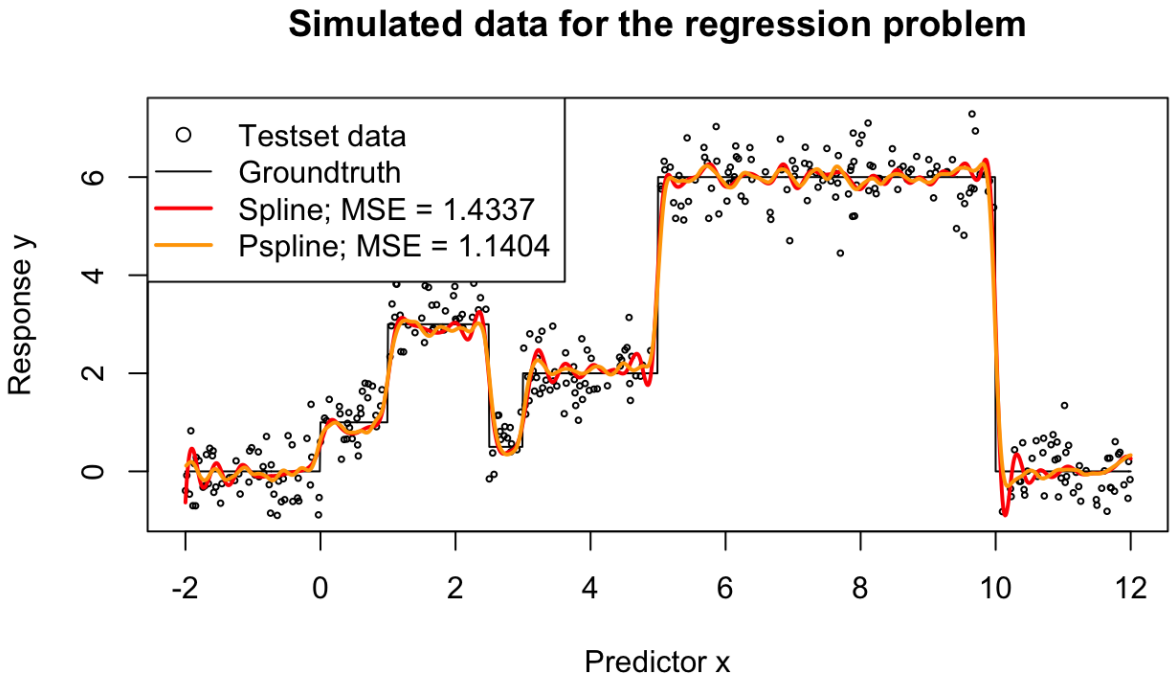

Figure 7: Comparison of the regular spline and penalized spline regression

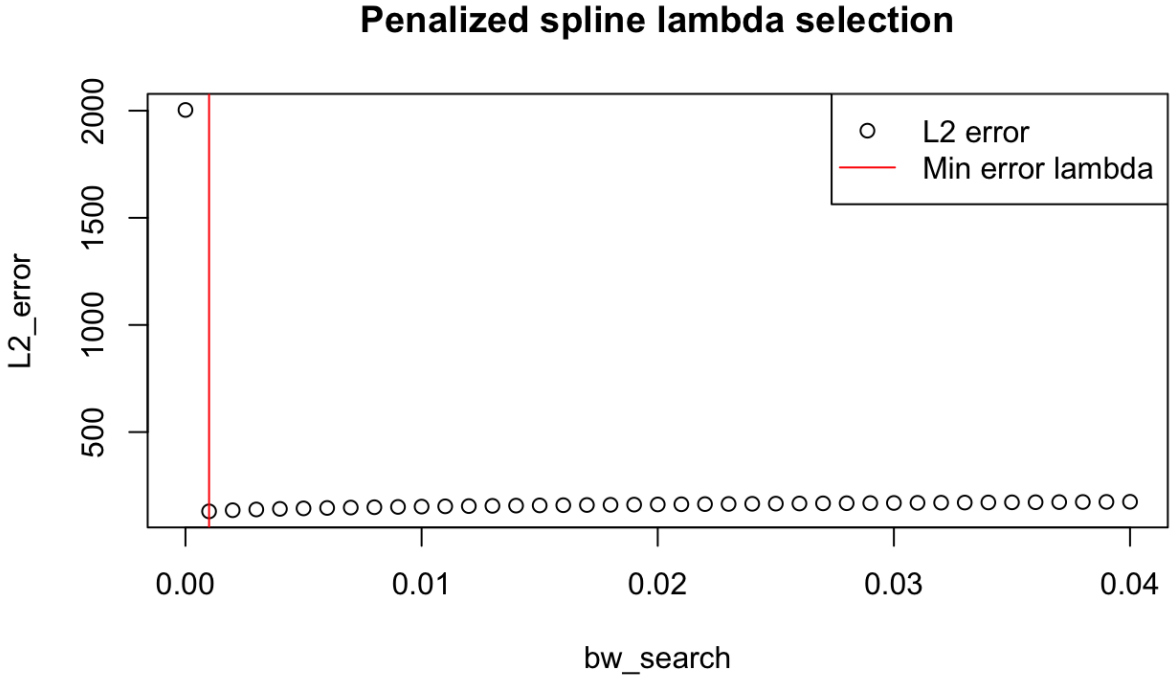

Figure 8: Tuning parameter selection for penalized spline regression

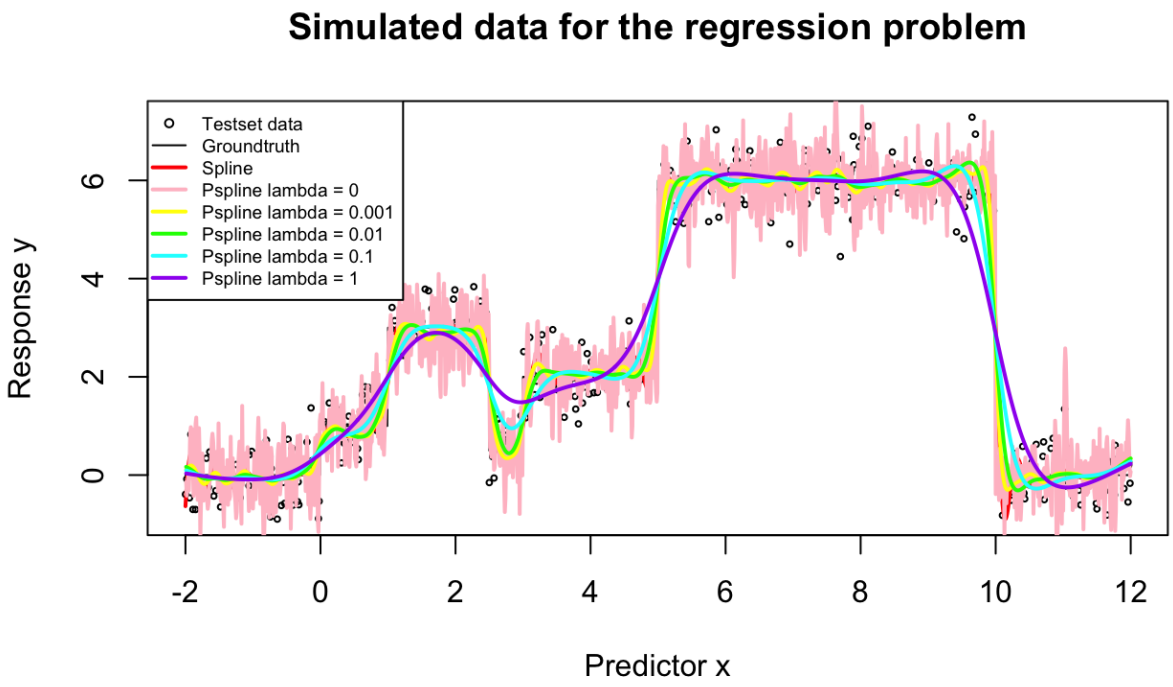

Figure 9: The effect of tuning parameter on the estimate

The small λ is perferred mainly due to the fact we have a very not smooth function. When applying penalty on the second derivatives of the estimate, we are enforcing smoothness of the function. However, the starcase funciton is discontinuous in theory and has many "jumps". This is the exacy opposite of what the penalty term is trying to do. As the results, in order to minimize the cross validation error, the penalty term needs to be kept very small while not to small to cause overfit. Because of this mismatch of the two packages, this comparison is perhaps not the very fair and we should look more into R packages that allows one to specify the order of the spline while also have the penalty feature available.

Spline regression with real data

Last but not least, we would liek to try out the spline regression on real data. The dataset we choose is the Lidar data from Wasserman (2005). Here, the "logratio" is the response and the "range" is the predictor.

Experiment setup

The data was split to training (70%) and test set (30%). The bandwidth selection is the same as the last experiment for both local polynomial and the spline regression. Only regular spline using the "spline" package was considered in the experiment. The evaluation criterion is the sum of squared serror of the test set data.

Results and discussion

The bandwidth selection for local polynomial was also very reasonable. The R again threw error for the selection of degree of freedom in spline regression, although the objective function values did not look at bad as the last experiment.

Quite interestingly, both methods performed very similarly for this problem. The IMSEs were almost identical and it was hard to conclude which method is more appropriate. In addition, the test set data was not big

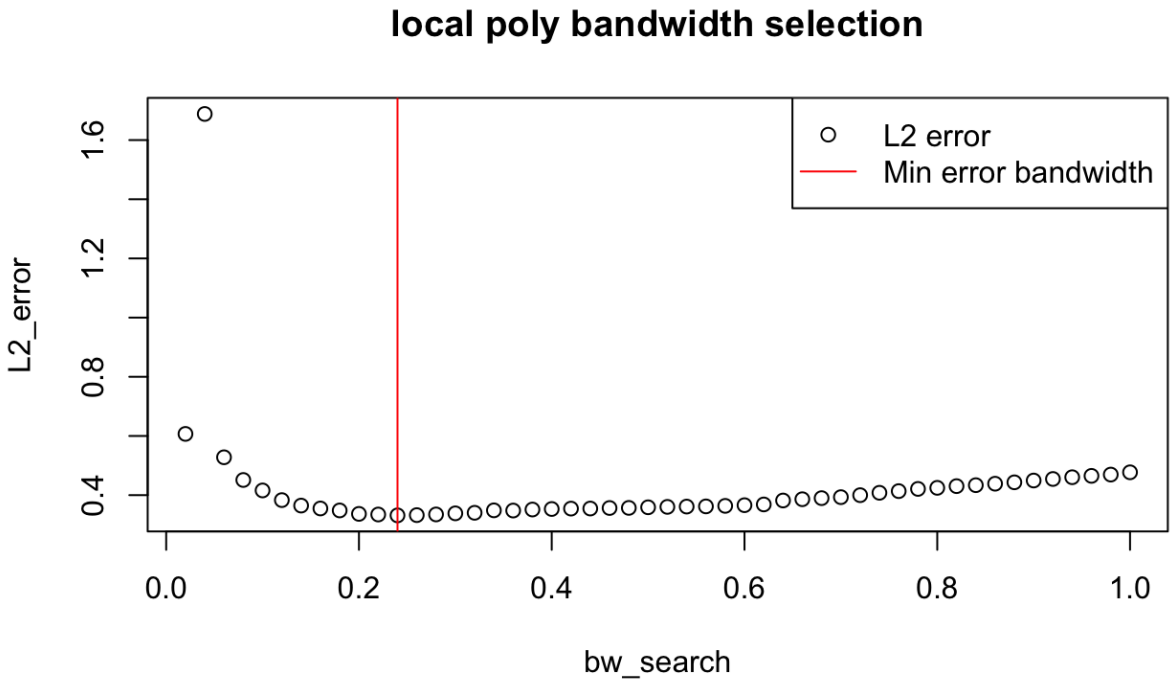

Figure 10: The bandwidth selection for local polynomial regression with Lidar data

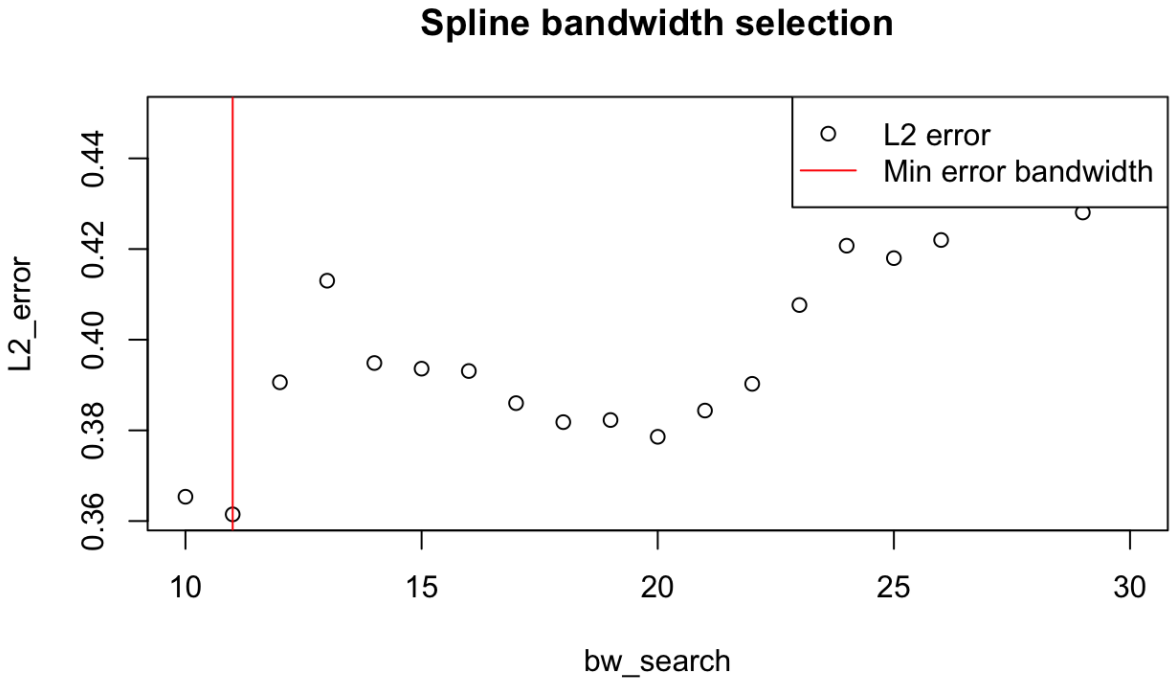

Figure 11: The bandwidth selection for spline regression with Lidar data

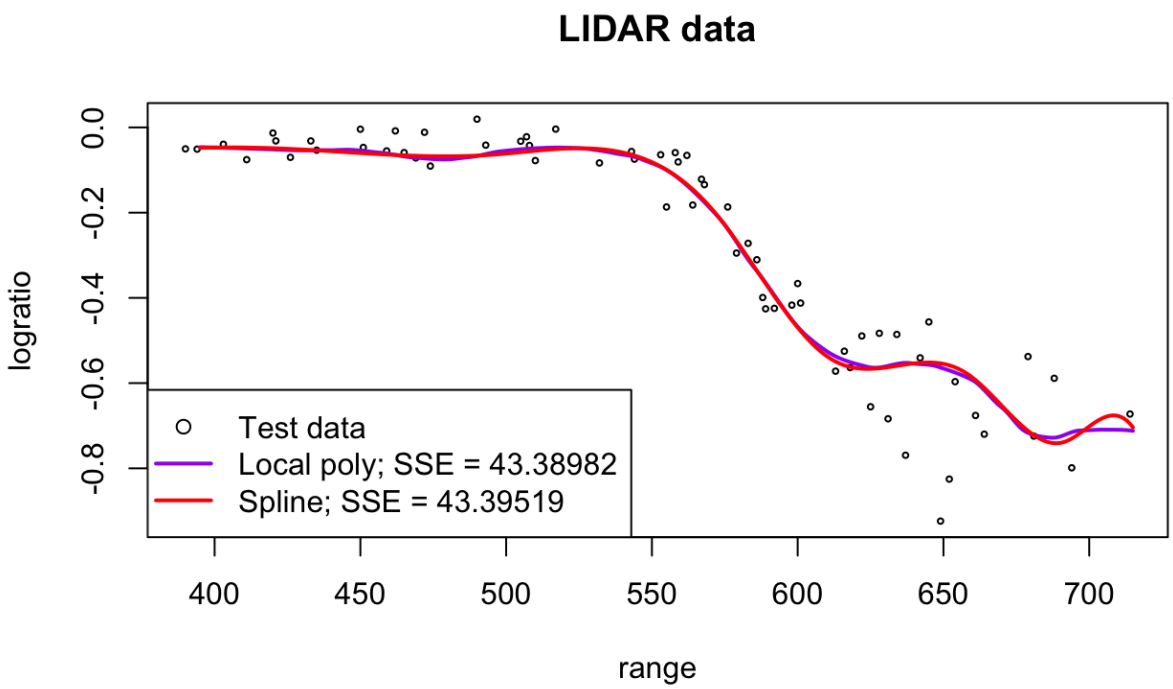

Figure 12: The regression fit for Lidar data

and majority of the SSE came from the tail of the dataset and unfortunately, there was little data at the tail after data splitting. This makes the resutls harder to interpret. Perhaps a slightly different data splitting ratio will help.

Conclusions

We dived into the theory of spline in nonparametric statistics and its usages in practice. Both regular and penalized spline were discussed from a theoretical standpoint. The numerical studies showcased the power of spline and demonstrated that both KDE, local polynomial regression and their spline version are useful in practice. By leveraging the different characteristics of different methods, one should choose the model that works the best for their application.

Appendix

The following R code was used to prepare the empirical section of this paper.

```
library(KernSmooth)
library(logspline)
library(splines)
library(pspline)

# SSE
SSE <- function(y_actual, y_fitted) {
  SSE <- sum((y_actual - y_fitted)^2, na.rm = TRUE)
}

# approximate MISE using Riemann sum
IMSE <- function(y_actual, y_fitted_mat, x_spacing) {
  # different trails are arranged as different column in y_fitted_mat
  IMSE <- x_spacing * sum(rowMeans((y_fitted_mat - y_actual)^2))
  return(IMSE)
```

```
}

# split data
k_fold_data_split <- function(data, k) {
  ran_ind <- sample(seq(1, nrow(data)), replace = FALSE)
  grouping_ind <- cut(seq(1, nrow(data)), breaks = k, labels = FALSE)
  output_list <- list(ran_ind, grouping_ind)
  names(output_list) <- c("ran_ind", "grouping_ind")
  return(output_list)
}

# KDE evaluate by the definition
density_predict <- function(olddatax, newdatax, h) {
  return(mean(dnorm(newdatax, mean = olddatax, sd = h)))
}

# # KDE alternative; slow
# density_predict = function(olddatax, newdatax, h) {
#   kernelValues <- rep(0, length(olddatax))
#     for(i in 1:length(olddatax)){
#         transformed = (newdatax - olddatax[i]) / h
#         kernelValues[i] <- dnorm(transformed, mean = 0, sd = 1) / h
#     }
#     return(sum(kernelValues) / length(olddatax))
# }

### Create simulated data for density estimation

# set RNG seed
set.seed(345)

# analytical expression for the mixture
density_expression <- function(x, p1, p2, mu1, mu2, mu3, sigma1, sigma2, sigma3) {
  p_of_x <- p1 * dnorm(x, mu1, sigma1) + p2 * dnorm(x, mu2, sigma2) +
    (1 - p1 - p2) * dnorm(v, mu3, sigma3)

  return(p_of_x)
}


# define the Gaussian mixture
density_simulator <- function(n, p1, p2, mu1, mu2, mu3, sigma1, sigma2, sigma3) {
  x <- matrix(0, n, 1)

  for (i in 1:n) {
    u <- runif(1)
    if (u < p1) {
      x[i] <- rnorm(1, mu1, sigma1)
    } else if (u < (p1 + p2)) {
      x[i] <- rnorm(1, mu2, sigma2)
    } else {
      x[i] <- rnorm(1, mu3, sigma3)
    }
  }

  return(x)
}

# generate simulated training and test data set
n_test <- 1000
p1 <- 1 / 4
p2 <- 1 / 4
mu1 <- -5
mu2 <- -2
mu3 <- 4
sigma1 <- 0.6
sigma2 <- 0.6
sigma3 <- 1.5

x_test <- density_simulator(n_test, p1, p2, mu1, mu2, mu3, sigma1, sigma2, sigma3)

# evaluation grid
lbound <- -10
ubound <- 10.46
spacing <- 0.02
num_of_points <- (ubound - lbound) / spacing + 1

### Kernel density estimation
bcv_bw <- bw.bcv(x_test)
kde_sim_display <- density(x_test,
  bw = bcv_bw,
```

```
  from = lbound, to = ubound, n = num_of_points
)
v <- kde_sim_display$x

### Spline density estimation
spline_sim_obj <- logspline(x_test, lbound, ubound)
spline_sim_display <- dlogspline(v, spline_sim_obj)

# compute approximated IMSE for kde
p_actual <- density_expression(v, p1, p2, mu1, mu2, mu3, sigma1, sigma2, sigma3)
Monte_Carlo_repetition <- 1000
p_fitted_mat <- matrix(0, nrow = length(v), ncol = Monte_Carlo_repetition)
for (i in 1:Monte_Carlo_repetition) {
  x_IMSE <- density_simulator(n_test, p1, p2, mu1, mu2, mu3, sigma1, sigma2, sigma3)

  bcv_bw <- bw.bcv(x_IMSE)
  kde_sim <- density(x_IMSE, bw = bcv_bw, from = lbound, to = ubound, n = num_of_points)
  p_fitted_mat[, i] <- kde_sim$y
}

kde_IMSE <- IMSE(p_actual, p_fitted_mat, spacing)

# compute approximated IMSE for spline
Monte_Carlo_repetition <- 1000
p_fitted_mat <- matrix(0, nrow = length(v), ncol = Monte_Carlo_repetition)
for (i in 1:Monte_Carlo_repetition) {
  x_IMSE <- density_simulator(n_test, p1, p2, mu1, mu2, mu3, sigma1, sigma2, sigma3)

  spline_sim_obj <- logspline(x_test, lbound, ubound)
  p_fitted_mat[, i] <- dlogspline(v, spline_sim_obj)
}

spline_IMSE <- IMSE(p_actual, p_fitted_mat, spacing)

#### visualization
hist(x_test,
  freq = FALSE, ylim = c(0, 0.25),
  main = "Simulated data for the density estimation problem", xlab = "x"
)
rug(x_test, ticksize = 0.05)
lines(v, density_expression(v, p1, p2, mu1, mu2, mu3, sigma1, sigma2, sigma3),
  lty = 1, lwd = 1, col = "black"
)
lines(v, kde_sim_display$y, lty = 1, lwd = 2, col = "purple")
lines(v, spline_sim_display, lty = 1, lwd = 2, col = "red")

legend("topright", c(
  "Groundtruth", paste("KDE IMSE =", round(kde_IMSE, 7)),
  paste("logspline IMSE =", round(spline_IMSE, 7))
),
col = c("black", "purple", "red"), lty = c(1, 1, 1), lwd = c(1, 2, 2)
)

rm(p_fitted_mat)


# create simulated data for regression

# set RNG seed
set.seed(345)

# define the regression function
r1 <- function(x) {
  y <- as.numeric(length(x)) # y=matrix(0,1,length(x))
  for (i in 1:length(x)) {
    if (x[i] < 0) {
      y[i] <- 0
    } else if (x[i] < 1) {
      y[i] <- 1
    } else if (x[i] < 2.5) {
      y[i] <- 3
    } else if (x[i] < 3) {
      y[i] <- 0.5
    } else if (x[i] < 5) {
      y[i] <- 2
    } else if (x[i] < 10) {
      y[i] <- 6
    } else {
      y[i] <- 0
    }
  }
```

```
  return(y)
}

# Generate simulated data
spacing <- .01
noise_mean <- 0
noise_sigma <- 0.5
x2 <- seq(from = -2, to = 12, by = spacing)
y2 <- r1(x2) + rnorm(length(x2), noise_mean, noise_sigma)
sim_staircase_data <- data.frame(x2, y2)

# x grid for evaluation
grid_x_2 <- sim_staircase_data$x2

# split data into training and test data sets
train_indices <- sample(1:nrow(sim_staircase_data), round(nrow(sim_staircase_data) * 0.75),
  replace = FALSE
)
sim_staircase_data_train_data <- sim_staircase_data[train_indices, ]
sim_staircase_data_test_data <- sim_staircase_data[-train_indices, ]

# split training data
k <- 3
training_folds <- k_fold_data_split(sim_staircase_data_train_data, k)

##### 3rd degree local polynomial model (bandwidth selection)
# y2_locpoly = locpoly(x2, y2, degree = 3, bandwidth = 0.25)
bw_search <- seq(0.01, 0.05, 0.0025)

L2_error_mat <- matrix(0, length(bw_search), k)
for (i in 1:length(bw_search)) {
  for (j in 1:k) {
    sim_staircase_data_train_data_not_j <-
      sim_staircase_data_train_data[training_folds$ran_ind[training_folds$grouping_ind != j], ]
    y2_loess <-
      loess(y2 ~ x2, span = bw_search[i], degree = 2, data = sim_staircase_data_train_data_not_j)
    # this loess predict will produce NA sometimes
    y2_loess_predict <- predict(
      y2_loess,
      sim_staircase_data_train_data$x2[training_folds$ran_ind[training_folds$grouping_ind == j]]
    )
    L2_error_mat[i, j] <-
      SSE(sim_staircase_data_train_data$y2[training_folds$ran_ind[training_folds$grouping_ind == j]], y2_loess_
          ↪ predict)
  }
}

L2_error <- rowMeans(L2_error_mat)
cv_bw_loess <- bw_search[which.min(L2_error)]
plot(bw_search, L2_error, main = "local poly bandwidth selection")
abline(v = cv_bw_loess, col = "red")
legend("topright", c("L2 error", "Min error bandwidth"),
  pch = c(1, NA), lty = c(NA, 1), col = c("black", "red")
)

# 3rd degree local polynomial model
y2_loess <- loess(y2 ~ x2,
  span = cv_bw_loess, degree = 2,
  data = sim_staircase_data_train_data
)
y2_loess_predict <- predict(y2_loess, grid_x_2)

##### Splines model (df selection)
bw_search <- seq(60, 80, 1)

L2_error_mat <- matrix(0, length(bw_search), k)
for (i in 1:length(bw_search)) {
  for (j in 1:k) {
    sim_staircase_data_train_data_not_j <-
      sim_staircase_data_train_data[training_folds$ran_ind[training_folds$grouping_ind != j], ]

    y2_spline <- lm(y2 ~ bs(x2, df = bw_search[i], degree = 3),
      data = sim_staircase_data_train_data_not_j
    )
    y2_spline_predict <- predict(
      y2_spline,
      data.frame(x2 = sim_staircase_data_train_data$x2[training_folds$ran_ind[training_folds$grouping_ind == j
          ↪ ]])
    )

    L2_error_mat[i, j] <-
```

```
      SSE(
        sim_staircase_data_train_data$y2[training_folds$ran_ind[training_folds$grouping_ind == j]],
        y2_spline_predict
      )
    }
}

L2_error <- rowMeans(L2_error_mat)
cv_bw_spline <- bw_search[which.min(L2_error)]
plot(bw_search, L2_error, main = "Spline bandwidth selection")
abline(v = cv_bw_spline, col = "red")
legend("topright", c("L2 error", "Min error bandwidth"),
  pch = c(1, NA), lty = c(NA, 1), col = c("black", "red")
)

# Splines model
y2_spline <- lm(y2 ~ bs(x2, df = cv_bw_spline, degree = 3), data = sim_staircase_data_train_data)
y2_spline_predict <- predict(y2_spline, data.frame(x2 = grid_x_2))

# compute approximated IMSE for locpoly
Monte_Carlo_repetition <- 1000
y_fitted_mat <- matrix(0, nrow = length(grid_x_2), ncol = Monte_Carlo_repetition)
for (i in 1:Monte_Carlo_repetition) {
  spacing <- .01
  noise_mean <- 0
  noise_sigma <- 0.5
  x2 <- seq(from = -2, to = 12, by = spacing)
  y2 <- r1(x2) + rnorm(length(x2), noise_mean, noise_sigma)
  sim_staircase_data_IMSE <- data.frame(x2, y2)

  y2_loess <- loess(y2 ~ x2, span = cv_bw_loess, degree = 2, data = sim_staircase_data_IMSE)
  y_fitted_mat[, i] <- predict(y2_loess, grid_x_2)
}

loess_IMSE <- IMSE(r1(x2), y_fitted_mat, spacing)

# compute approximated IMSE for spline
Monte_Carlo_repetition <- 1000
y_fitted_mat <- matrix(0, nrow = length(grid_x_2), ncol = Monte_Carlo_repetition)
for (i in 1:Monte_Carlo_repetition) {
  spacing <- .01
  noise_mean <- 0
  noise_sigma <- 0.5
  x2 <- seq(from = -2, to = 12, by = spacing)
  y2 <- r1(x2) + rnorm(length(x2), noise_mean, noise_sigma)
  sim_staircase_data_IMSE <- data.frame(x2, y2)

  y2_spline <- lm(y2 ~ bs(x2, df = cv_bw_spline, degree = 3), data = sim_staircase_data_IMSE)
  y_fitted_mat[, i] <- predict(y2_spline, data.frame(x2 = grid_x_2))
}

spline_IMSE <- IMSE(r1(x2), y_fitted_mat, spacing)

# Visualization
plot(sim_staircase_data_test_data$x2, sim_staircase_data_test_data$y2,
  xlab = "Predictor x", ylab = "Response y",
  main = "Simulated data for the regression problem", cex = 0.4
)
lines(grid_x_2, r1(grid_x_2), type = "l", lty = 1, lwd = 1, col = "black")
# lines(y2_locpoly$x, y2_locpoly$y, type="l", lty=1, lwd = 2,  col="green")
lines(grid_x_2, y2_loess_predict, type = "l", lty = 1, lwd = 2, col = "purple")
lines(grid_x_2, y2_spline_predict, lty = 1, lwd = 2, col = "red")
# lines(grid_x_2, y2_penalized_spline_predict, lty=1, lwd = 2, col="yellow")
legend("topleft", c(
  "Testset data", "Groundtruth",
  paste("Local poly; MSE =", round(loess_IMSE, 4)),
  paste("Spline; MSE =", round(spline_IMSE, 4))
),
lty = c(NA, 1, 1, 1), pch = c(1, NA, NA, NA),
col = c("black", "black", "purple", "red"), lwd = c(NA, 1, 2, 2)
)

##### penalized spline

# penalized splines model (bandwidth selection)
bw_search <- seq(0, 0.04, 0.001)

L2_error_mat <- matrix(0, length(bw_search), k)
for (i in 1:length(bw_search)) {
  for (j in 1:k) {
```

```
    sim_staircase_data_train_data_not_j <- sim_staircase_data_train_data[training_folds$ran_ind[training_folds$
        ↪ grouping_ind != j], ]

    sim_staircase_data_train_data_not_j <- sim_staircase_data_train_data_not_j[order(sim_staircase_data_train_
        ↪ data_not_j$x2), ]

    y2_penalized_spline <- smooth.Pspline(sim_staircase_data_train_data_not_j$x2, sim_staircase_data_train_data
        ↪ _not_j$y2, norder = 2, spar = bw_search[i])
    y2_penalized_spline_predict <- predict(y2_penalized_spline, sim_staircase_data_train_data$x2[training_folds
        ↪ $ran_ind[training_folds$grouping_ind == j]])

    L2_error_mat[i, j] <- SSE(sim_staircase_data_train_data$y2[training_folds$ran_ind[training_folds$grouping_
        ↪ ind == j]], y2_penalized_spline_predict)
  }
}

L2_error <- rowMeans(L2_error_mat)
cv_bw_pspline <- bw_search[which.min(L2_error)]
plot(bw_search, L2_error, main = "Penalized spline lambda selection")
abline(v = cv_bw_pspline, col = "red")
legend("topright", c("L2 error", "Min error lambda"), pch = c(1, NA), lty = c(NA, 1), col = c("black", "red"))

# penalized splines model
sim_staircase_data_train_data <- sim_staircase_data_train_data[order(sim_staircase_data_train_data$x2), ]

y2_penalized_spline <- smooth.Pspline(sim_staircase_data_train_data$x2, sim_staircase_data_train_data$y2,
    ↪ norder = 2, spar = cv_bw_pspline)
y2_penalized_spline_predict <- predict(y2_penalized_spline, grid_x_2)

# compute IMSE for pspline
Monte_Carlo_repetition <- 1000
y_fitted_mat <- matrix(0, nrow = length(grid_x_2), ncol = Monte_Carlo_repetition)
for (i in 1:Monte_Carlo_repetition) {
  spacing <- .01
  noise_mean <- 0
  noise_sigma <- 0.5
  x2 <- seq(from = -2, to = 12, by = spacing)
  y2 <- r1(x2) + rnorm(length(x2), noise_mean, noise_sigma)
  sim_staircase_data_IMSE <- data.frame(x2, y2)

  sim_staircase_data_IMSE <- sim_staircase_data_IMSE[order(sim_staircase_data_IMSE$x2), ]

  y2_penalized_spline <- smooth.Pspline(sim_staircase_data_IMSE$x2, sim_staircase_data_IMSE$y2, norder = 2,
      ↪ spar = cv_bw_pspline)
  y_fitted_mat[, i] <- predict(y2_penalized_spline, grid_x_2)
}

Pspline_IMSE <- IMSE(r1(x2), y_fitted_mat, spacing)

# Visualization
plot(sim_staircase_data_test_data$x2, sim_staircase_data_test_data$y2,
  xlab = "Predictor x", ylab = "Response y",
  main = "Simulated data for the regression problem", cex = 0.4
)
lines(grid_x_2, r1(grid_x_2), type = "l", lty = 1, lwd = 1, col = "black")
lines(grid_x_2, y2_spline_predict, lty = 1, lwd = 2, col = "red")
lines(grid_x_2, y2_penalized_spline_predict, lty = 1, lwd = 2, col = "orange")
legend("topleft", c(
  "Testset data", "Groundtruth", paste("Spline; MSE =", round(spline_IMSE, 4)),
  paste("Pspline; MSE =", round(Pspline_IMSE, 4))
),
lty = c(NA, 1, 1, 1),
pch = c(1, NA, NA, NA),
col = c("black", "black", "red", "orange"), lwd = c(NA, 1, 2, 2)
)

# pspline lambda effect

# need variables from previous code chunk
lambda_vec <- c(0, 0.001, 0.01, 0.1, 1)
y2_penalized_spline_predict_mat <- matrix(0, nrow = length(grid_x_2), ncol = length(lambda_vec))

for (i in 1:length(lambda_vec)) {
  y2_penalized_spline <- smooth.Pspline(sim_staircase_data_train_data$x2,
    sim_staircase_data_train_data$y2,
    norder = 2, spar = lambda_vec[i]
  )
  y2_penalized_spline_predict_mat[, i] <- predict(y2_penalized_spline, grid_x_2)
}

# Visualization
```

```
plot(sim_staircase_data_test_data$x2, sim_staircase_data_test_data$y2,
  xlab = "Predictor x", ylab = "Response y",
  main = "Simulated data for the regression problem", cex = 0.4
)
lines(grid_x_2, r1(grid_x_2), type = "l", lty = 1, lwd = 1, col = "black")
lines(grid_x_2, y2_spline_predict, lty = 1, lwd = 2, col = "red")
lines(grid_x_2, y2_penalized_spline_predict_mat[, 1], lty = 1, lwd = 2, col = "pink")
lines(grid_x_2, y2_penalized_spline_predict_mat[, 2], lty = 1, lwd = 2, col = "yellow")
lines(grid_x_2, y2_penalized_spline_predict_mat[, 3], lty = 1, lwd = 2, col = "green")
lines(grid_x_2, y2_penalized_spline_predict_mat[, 4], lty = 1, lwd = 2, col = "cyan")
lines(grid_x_2, y2_penalized_spline_predict_mat[, 5], lty = 1, lwd = 2, col = "purple")

legend("topleft", c(
  "Testset data", "Groundtruth", "Spline", "Pspline lambda = 0",
  "Pspline lambda = 0.001", "Pspline lambda = 0.01", "Pspline lambda = 0.1", "Pspline lambda = 1"
),
lty = c(NA, 1, 1, 1, 1, 1, 1, 1), pch = c(1, NA, NA, NA, NA, NA, NA, NA),
col = c("black", "black", "red", "pink", "yellow", "green", "cyan", "purple"),
lwd = c(NA, 1, 2, 2, 2, 2, 2, 2), cex = 0.6
)


# real data for density estimation

# set RNG seed
set.seed(345)

# load data
Fiji_data <- read.table("/Users/Samson/Documents/Github/496-splines/fijiquakes.dat.txt", header = TRUE)

# split data into training and test data sets
train_indices <- sample(1:nrow(Fiji_data), round(nrow(Fiji_data) * 0.25),
  replace = FALSE
)

Fiji_train_data <- Fiji_data[train_indices, ]
Fiji_test_data <- Fiji_data[-train_indices, ]

# evaluation grid
lbound <- -30
ubound <- 750
spacing <- 0.75
num_of_points <- 1024

# kernel density estimation
# ucv_bw = bw.ucv(Fiji_train_data$depth) # failed
bcv_bw <- bw.bcv(Fiji_train_data$depth)
kde_fiji <- density(Fiji_train_data$depth, bw = bcv_bw, from = lbound, to = ubound, n = num_of_points)
fiji_grid <- kde_fiji$x

# spline model
spline_fiji_obj <- logspline(Fiji_train_data$depth, lbound, ubound)
spline_sim <- dlogspline(fiji_grid, spline_fiji_obj)

# find empirical cdf for kde, spline and the test data
test_ecdf <- ecdf(Fiji_test_data$depth)
kde_cdf <- cumsum(kde_fiji$y) * (fiji_grid[2] - fiji_grid[1])
spline_cdf <- cumsum(spline_sim) * (fiji_grid[2] - fiji_grid[1])

kde_l2 <- sqrt(sum((kde_cdf - test_ecdf(fiji_grid))^2))
spline_l2 <- sqrt(sum((spline_cdf - test_ecdf(fiji_grid))^2))

plot(fiji_grid, test_ecdf(fiji_grid),
  type = "l",
  main = "Test set empirical cdf and estimated cdf", lwd = 2
)
lines(fiji_grid, kde_cdf, col = "purple", lwd = 2)
lines(fiji_grid, spline_cdf, col = "red", lwd = 2)
legend("topleft", c("Testset ecdf", paste(
  "KDE cdf; L2 error =",
  round(kde_l2, 4)
), paste("logspline cdf; L2 error =", round(spline_l2, 4))),
lty = c(1, 1, 1), col = c("black", "purple", "red"), lwd = c(2, 2, 2)
)

# compute the log likelihood of each density given the test data
kde_pdf_val <- rep(0, length(Fiji_test_data$depth))
for (i in 1:length(Fiji_test_data$depth)) {
  kde_pdf_val[i] <- density_predict(Fiji_train_data$depth, Fiji_test_data$depth[i], bcv_bw)
}
LL_kde <- sum(log(kde_pdf_val))
```

```r
spline_pdf_val <- dlogspline(Fiji_test_data$depth, spline_fiji_obj)
LL_spline <- sum(log(spline_pdf_val))

LLR <- LL_spline / LL_kde

# visualization
hist(Fiji_test_data$depth,
  freq = FALSE,
  main = "Fiji earthquake depth data (test set)",
  xlab = "Depth of the Fiji earthquake", ylim = c(0, 0.0085)
)
rug(Fiji_test_data$depth, ticksize = 0.05)
lines(kde_fiji, lty = 1, lwd = 2, col = "purple")
lines(fiji_grid, spline_sim, lty = 1, lwd = 2, col = "red")
legend("top", c(
  paste("KDE; log-likelihood =", round(LL_kde, 3)),
  paste("logspline; log-likelihood =", round(LL_spline, 3))
),
lty = c(1, 1), col = c("purple", "red"), lwd = c(2, 2)
)

# real data for regression

# set RNG seed
set.seed(345)

# load data
lidar_data <- read.table("/Users/Samson/Documents/Github/496-splines/lidar.dat.txt", header = TRUE)

# evaluation grid
grid_lidar <- seq(from = 395, to = 715, by = 1)

# split data into training and testing data
train_indices <- sample(1:nrow(lidar_data), round(nrow(lidar_data) * 0.7),
  replace = FALSE
)

lidar_train_data <- lidar_data[train_indices, ]
lidar_test_data <- lidar_data[-train_indices, ]

# split training data
k <- 3
training_folds <- k_fold_data_split(lidar_train_data, k)

# local polynomial model (bandwidth selection)
bw_search <- seq(0.02, 1, 0.02)

L2_error_mat <- matrix(0, length(bw_search), k)
for (i in 1:length(bw_search)) {
  for (j in 1:k) {
    lidar_train_data_not_j <- lidar_train_data[training_folds$ran_ind[training_folds$grouping_ind != j], ]
    y2_loess <- loess(logratio ~ range, span = bw_search[i], degree = 2, data = lidar_train_data_not_j)
    y2_loess_predict <- predict(y2_loess, lidar_train_data$range[training_folds$ran_ind[training_folds$grouping
        ↪ _ind == j]])
    L2_error_mat[i, j] <- SSE(lidar_train_data$logratio[training_folds$ran_ind[training_folds$grouping_ind == j
        ↪ ]], y2_loess_predict)
  }
}

L2_error <- rowMeans(L2_error_mat)
cv_bw_loess <- bw_search[which.min(L2_error)]
plot(bw_search, L2_error, main = "local poly bandwidth selection")
abline(v = cv_bw_loess, col = "red")
legend("topright", c("L2 error", "Min error bandwidth"), pch = c(1, NA), lty = c(NA, 1), col = c("black", "red"
    ↪ ))


# local polynomial model
y2_loess <- loess(logratio ~ range, span = cv_bw_loess, degree = 2, data = lidar_train_data)
y2_lidar_loess_predict <- predict(y2_loess, grid_lidar)

# spline model (bandwidth selection)
bw_search <- seq(10, 30, 1)

L2_error_mat <- matrix(0, length(bw_search), k)
for (i in 1:length(bw_search)) {
  for (j in 1:k) {
    lidar_train_data_not_j <- lidar_train_data[training_folds$ran_ind[training_folds$grouping_ind != j], ]
    y2_spline <- lm(logratio ~ bs(range, df = bw_search[i], degree = 3), data = lidar_train_data_not_j)
```

```
    y2_spline_predict <- predict(y2_spline, data.frame(range = lidar_train_data$range[training_folds$ran_ind[
        ↪ training_folds$grouping_ind == j]]))
    L2_error_mat[i, j] <- SSE(lidar_train_data$logratio[training_folds$ran_ind[training_folds$grouping_ind == j
        ↪ ]], y2_spline_predict)
  }
}

L2_error <- rowMeans(L2_error_mat)
cv_bw_spline <- bw_search[which.min(L2_error)]
plot(bw_search, L2_error, main = "Spline bandwidth selection")
abline(v = cv_bw_spline, col = "red")
legend("topright", c("L2 error", "Min error bandwidth"), pch = c(1, NA), lty = c(NA, 1), col = c("black", "red"
    ↪ ))


# spline model
lidar_spline <- lm(logratio ~ bs(range, df = cv_bw_spline, degree = 3), data = lidar_train_data)
y2_lidar_spline_predict <- predict(lidar_spline, data.frame(range = grid_lidar))

# find SSE for kde
SSE_loess <- SSE(lidar_test_data$logratio, y2_lidar_loess_predict)

# find SSE for spline
SSE_spline <- SSE(lidar_test_data$logratio, y2_lidar_spline_predict)

# Visualization
plot(lidar_test_data$range, lidar_test_data$logratio,
  main = "LIDAR data", ylab = "logratio", xlab = "range", cex = 0.4
)
lines(grid_lidar, y2_lidar_loess_predict, lty = 1, lwd = 2, col = "purple")
lines(grid_lidar, y2_lidar_spline_predict, lty = 1, lwd = 2, col = "red")
legend("bottomleft", c(
  "Test data", paste("Local poly; SSE =", round(SSE_loess, 5)),
  paste("Spline; SSE =", round(SSE_spline, 5))
),
lty = c(NA, 1, 1),
pch = c(1, NA, NA), lwd = c(NA, 2, 2),
col = c("black", "purple", "red")
)
```