Smoothing Spline Regression in R

Nathaniel E. Helwig (http://stat.umn.edu/~helwig)
Department of Psychology & School of Statistics
University of Minnesota

January 04, 2021

- 1 Introduction
  - 1.1 Motivation and Goals
  - 1.2 smooth.spline Function
  - 1.3 ss Function (npreg Package)
  - 1.4 Comparison of Approaches
- 2 Definition
  - 2.1 Model Form
  - 2.2 Penalized Least Squares
  - 2.3 Smoothing Parameter Influence
  - 2.4 Penalty Order Influence
- 3 Computational Details
  - 3.1 Function Representation
  - 3.2 Coefficient Estimates
  - 3.3 Fitted Values and Smoothing Matrix
  - 3.4 Effective Degrees of Freedom
- 4 Smoothing Parameter Selection
  - 4.1 Ordinary Cross-Validation
  - 4.2 Generalized Cross-Validation
  - 4.3 AIC and BIC
  - 4.4 REML and ML
- 5 Example 1: Prestige from Income
  - 5.1 Overview of Data
  - 5.2 Analyses and Results
- 6 Example 2: Motorcycle Accident
  - 6.1 Overview of Data
  - 6.2 Analyses and Results

1 Introduction

1.1 Motivation and Goals

Smoothing splines are a powerful approach for estimating functional relationships between a predictor X and a response Y. Smoothing splines can be fit using either the `smooth.spline` function (in the **stats** package) or the `ss` function (in the **npreg** package). This document provides theoretical background on smoothing splines, as well as examples that illustrate how to use the `smooth.spline` and `ss` functions. As I demonstrate in this tutorial, the two functions have very similar syntax, but the `ss` function offers some additional options.

1.2 smooth.spline Function

Syntax:

```
smooth.spline(x, y = NULL, w = NULL, df, spar = NULL, lambda = NULL, cv = FALSE,
              all.knots = FALSE, nknots = .nknots.smspl,
              keep.data = TRUE, df.offset = 0, penalty = 1,
              control.spar = list(), tol = 1e-6 * IQR(x), keep.stuff = FALSE)
```

1.3 ss Function (npreg Package)

Syntax:

```
ss(x, y = NULL, w = NULL, df, spar = NULL, lambda = NULL,
   method = c("GCV", "OCV", "GACV", "ACV", "REML", "ML", "AIC", "BIC"),
   m = 2L, periodic = FALSE, all.knots = FALSE, nknots = .nknots.smspl,
   knots = NULL, keep.data = TRUE, df.offset = 0, penalty = 1,
   control.spar = list(), tol = 1e-6 * IQR(x), bernoulli = TRUE)
```

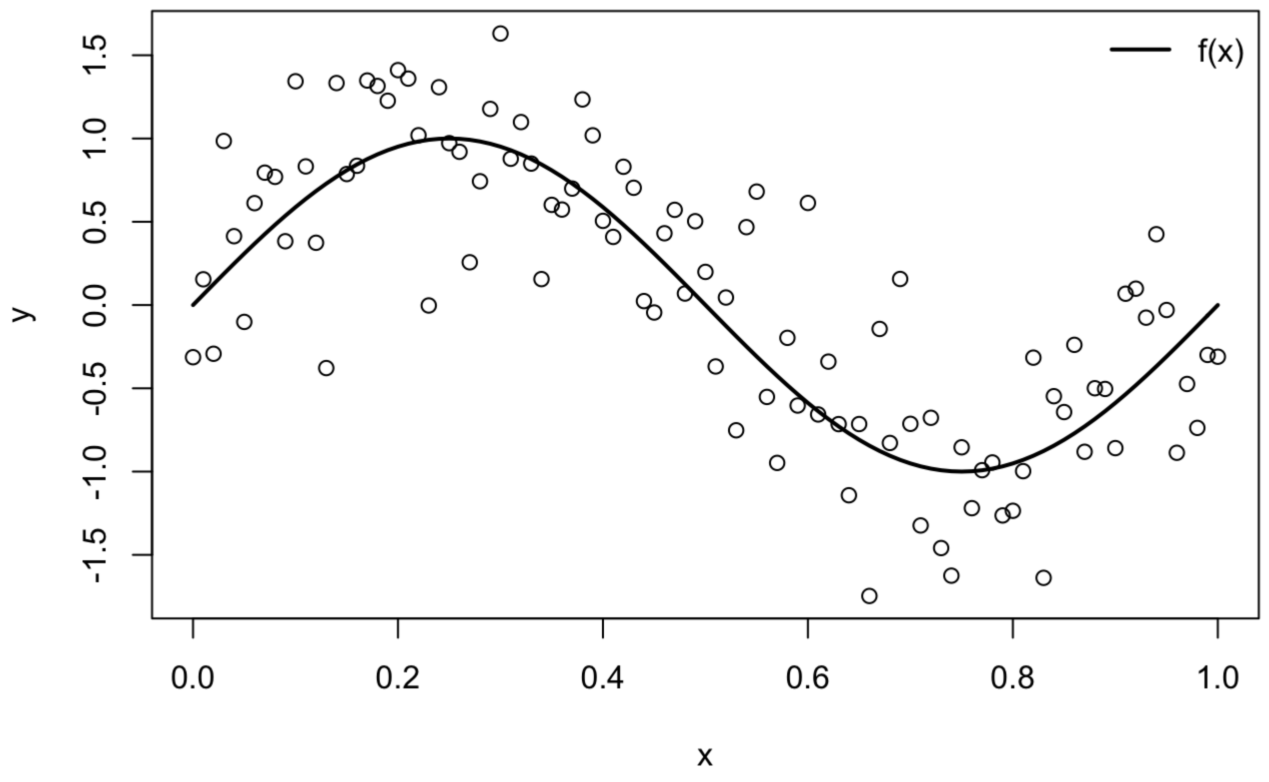
1.4 Comparison of Approaches

Compared to the `smooth.spline` function, the `ss` function has

1. more smoothing parameter selection methods
2. more spline types (linear, cubic, quintic)
3. an option for periodicity constraints
4. an option for user-specified knot values
5. corresponding summary and plot methods

To see how these functions perform in practice, let's look at a simulated example. Specifically, let's simulate some data with a (periodic) functional relationship that has some noise.

```
# define function
n <- 101
x <- seq(0, 1, length.out = n)
fx <- sin(2 * pi * x)

# generate noisy data
set.seed(1)
y <- fx + rnorm(n, sd = 0.5)

# plot data and f(x)
plot(x, y)             # data
lines(x, fx, lwd = 2)  # f(x)
legend("topright", legend = "f(x)", lty = 1, lwd = 2, bty = "n")
```



Estimate function using `smooth.spline` and `ss` functions with 10 knots:

```
# load 'npreg' package
library(npreg)

# fit using ss
mod.ss <- ss(x, y, nknots = 10)
mod.ss
```

```
##
## Call:
## ss(x = x, y = y, nknots = 10)
##
## Smoothing Parameter  spar = 0.3294827   lambda = 1.431201e-05
## Equivalent Degrees of Freedom (Df) 6.432432
## Penalized Criterion (RSS) 19.31262
## Generalized Cross-Validation (GCV) 0.2181113
```

```
# fit using smooth.spline
mod.smsp <- smooth.spline(x, y, nknots = 10)
mod.smsp
```

```
## Call:
## smooth.spline(x = x, y = y, nknots = 10)
##
## Smoothing Parameter  spar= 0.3108197  lambda= 0.001590704 (12 iterations)
## Equivalent Degrees of Freedom (Df): 6.348786
## Penalized Criterion (RSS): 19.36494
## GCV: 0.2183157
```

Note that the two functions produce objects that contain (nearly) identical results:

```
# compare returned objects
names(mod.ss)
```

```
##  [1] "x"          "y"          "w"          "yin"        "tol"        "data"
##  [7] "lev"        "cv.crit"    "pen.crit"   "crit"       "df"         "spar"
## [13] "lambda"     "fit"        "call"       "sigma"      "logLik"     "aic"
## [19] "bic"        "penalty"    "method"
```

```
names(mod.smsp)
```

```
##  [1] "x"          "y"          "w"          "yin"        "tol"
##  [6] "data"       "no.weights" "lev"        "cv.crit"    "pen.crit"
## [11] "crit"       "df"         "spar"       "ratio"      "lambda"
## [16] "iparms"     "auxM"       "fit"        "call"
```

```
# rmse between solutions
sqrt(mean(( mod.ss$y - mod.smsp$y )^2))
```

```
## [1] 0.003092318
```

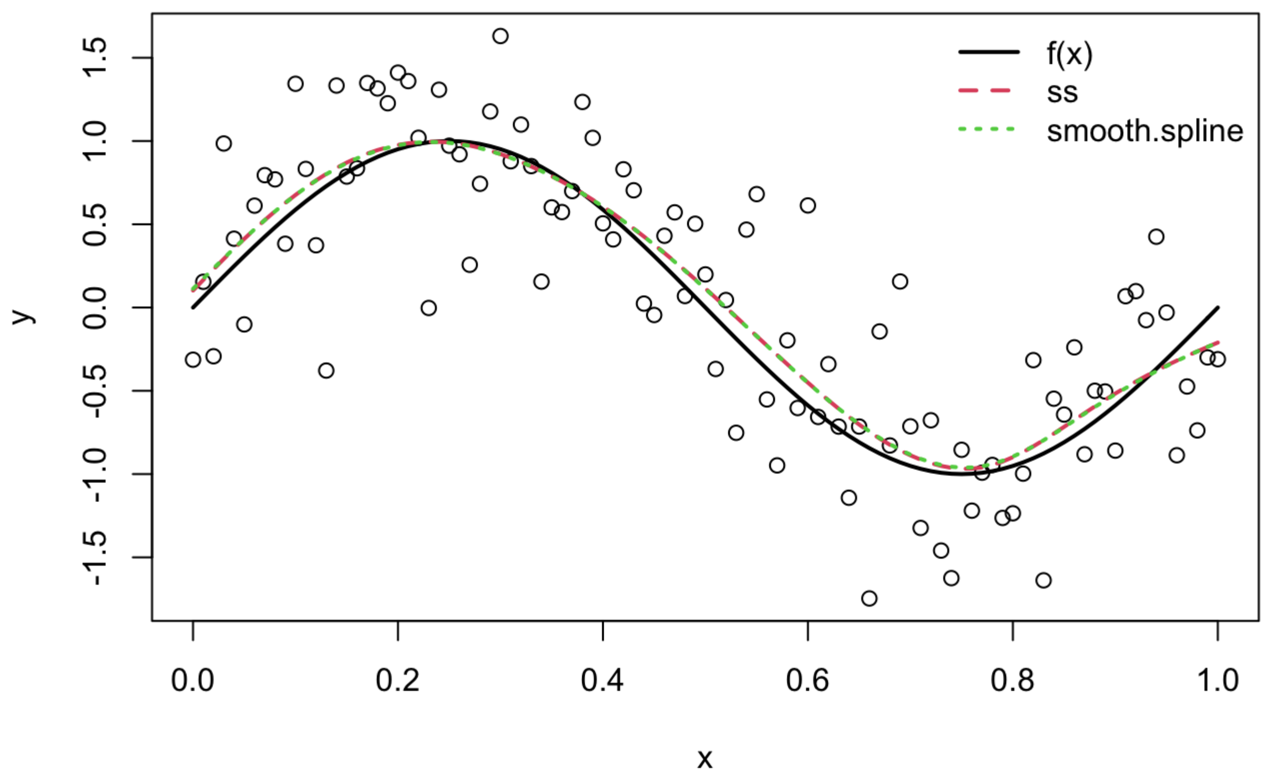
```
# rmse between solutions and f(x)
sqrt(mean(( fx - mod.ss$y )^2))
```

```
## [1] 0.08363143
```

```
sqrt(mean(( fx - mod.smsp$y )^2))
```

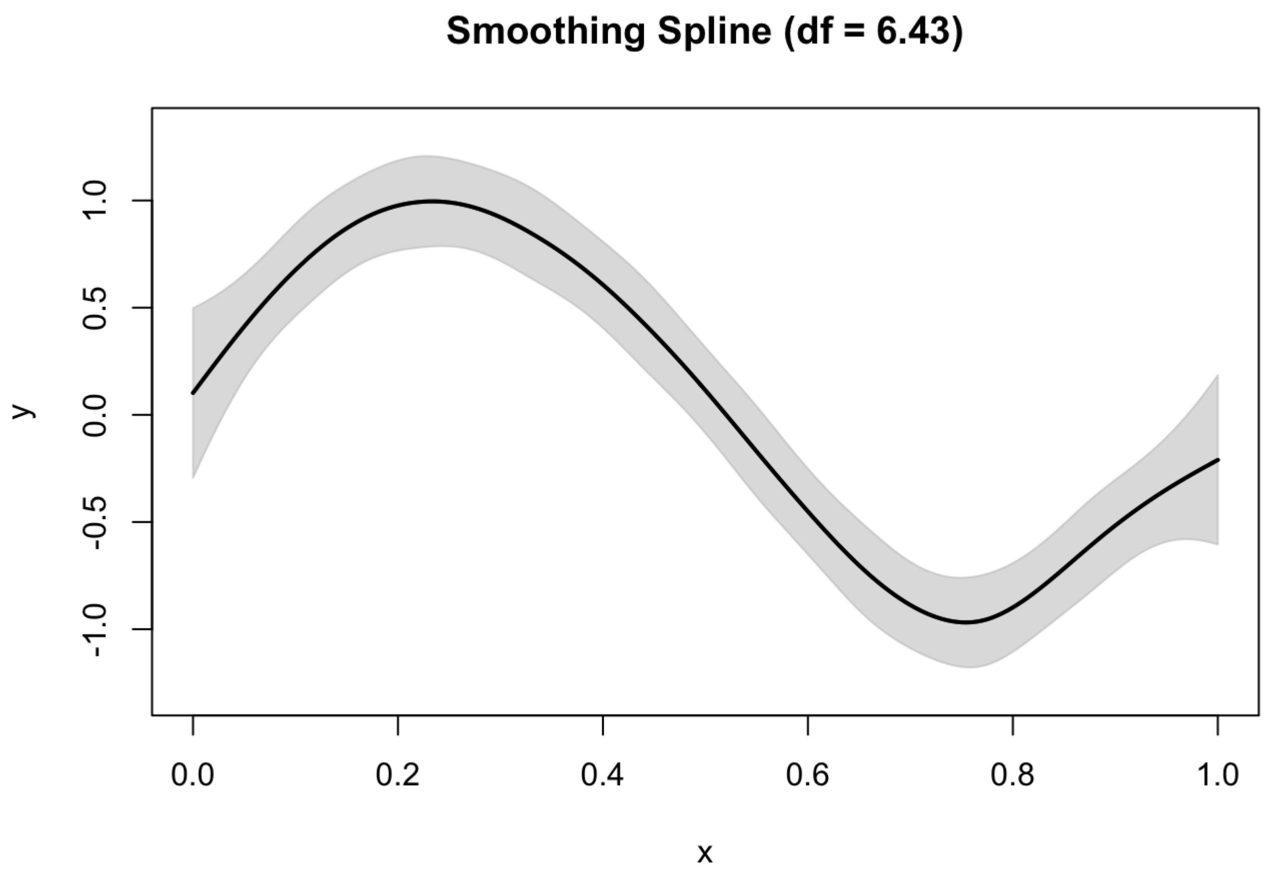
```
## [1] 0.08389557
```

```
# plot results
plot(x, y)
lines(x, fx, lwd = 2)
lines(x, mod.ss$y, lty = 2, col = 2, lwd = 2)
lines(x, mod.smsp$y, lty = 3, col = 3, lwd = 2)
legend("topright",
       legend = c("f(x)", "ss", "smooth.spline"),
       lty = 1:3, col = 1:3, lwd = 2, bty = "n")
```



Some extra features of the `ss` function include the “plot” and “summary” methods:

```
# plot method
plot(mod.ss)
```



```
# summary method
mod.sum <- summary(mod.ss)
mod.sum
```

```
## 
## Call:
## ss(x = x, y = y, nknots = 10)
## 
## Residuals:
##       Min        1Q    Median        3Q       Max 
## -1.180772 -0.252248 -0.004237  0.295601  1.063916 
## 
## Approx. Signif. of Parametric Effects:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05191    0.04499   1.154   0.2514
## x           -0.31243    0.28435  -1.099   0.2747
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Approx. Signif. of Nonparametric Effects:
##               Df Sum Sq Mean Sq F value   Pr(>F)    
## s(x)       4.432  17.60  3.9712   19.45 2.09e-12 ***
## Residuals 94.568  19.31  0.2042                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.4519 on 94.57 degrees of freedom
## Multiple R-squared:  0.7203,    Adjusted R-squared:  0.7038
## F-statistic: 43.7 on 5.432 and 94.57 DF,  p-value: <2e-16
```

Note that the `summary` function produces a printout that resembles what one would see from the `lm` or `glm` function—with the noteworthy exception that an additional table for the nonparametric effects is included.

2 Definition

2.1 Model Form

Suppose we an independent sample of n observations $(x_i, y_i) \sim F_{X,Y}$ from some continuous bivariate distribution $F_{X,Y}$, and consider the nonparametric regression model

$$y_i = f(x_i) + \epsilon_i$$

where $f(\cdot)$ is some unknown “smooth” function, and $\epsilon_i \overset{\text{iid}}{\sim} (0, \sigma^2)$ are iid error terms with mean zero and variance σ^2. This implies that $f(x_i)$ is the conditional mean of y_i given x_i. The goal is to estimate the unknown function $f(\cdot)$ from the sample of data.

2.2 Penalized Least Squares

To estimate $f(\cdot)$, a smoothing spline minimizes the *penalized least squares functional*

$$f_\lambda = \min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda J_m(f)$$

where $J_m(f) = \int |f^{(m)}(z)|^2 dz$ is a penalty term that quantifies the lack of parsimony of the function estimate, and $\lambda > 0$ is the *smoothing parameter* that controls the influence of the penalty. Note that $f^{(m)}(\cdot)$ denotes the m -th derivative of $f(\cdot)$, and $\mathcal{H} = \{f : J_m(f) < \infty\}$ is the space of functions with square integrable m-th derivative.

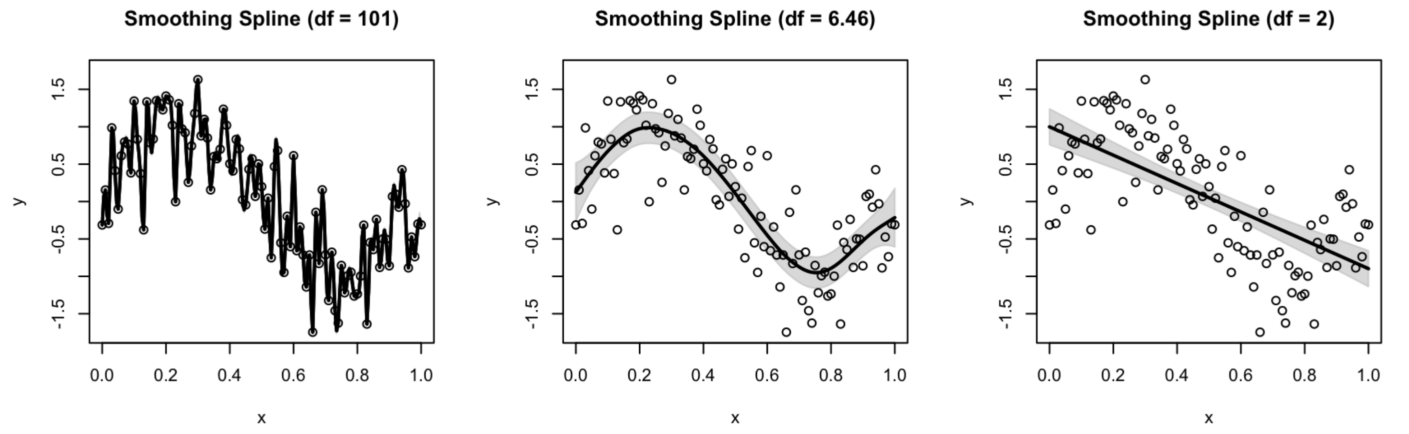
2.3 Smoothing Parameter Influence

As $\lambda \to 0$ the penalty has less influence on the penalized least squares functional So, for very small values of λ, the function estimate f_λ essentially minimizes the residual sum of squares.

As $\lambda \to \infty$ the penalty has more influence the penalized least squares functional So, for very large values of λ, the function estimate f_λ is essentially constrained to have a zero penalty, i.e., $J_m(f_\lambda) \approx 0$.

As λ increases from 0 to ∞, the function estimate f_λ is forced to be smoother with respect to the penalty functional $J_m(\cdot)$. The goal is to find the λ that produces the "correct" degree of smoothness for the function estimate.

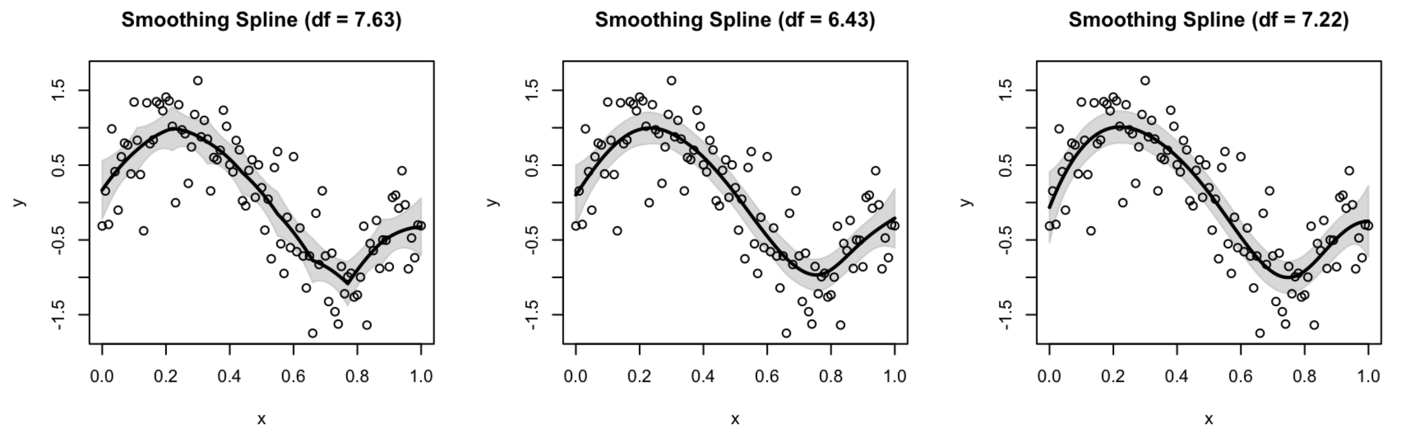
```r
# subplots (1 x 3)
par(mfrow = c(1,3))

# lambda = 1e-15 (df = n)
mod.ss0 <- ss(x, y, all.knots = TRUE, lambda = 1e-15)
plot(mod.ss0, ylim = c(-1.75, 1.75))
points(x, y)

# GCV selection
mod.ss <- ss(x, y, all.knots = TRUE)
plot(mod.ss, ylim = c(-1.75, 1.75))
points(x, y)

# lambda = 100 (df = m)
mod.ss10 <- ss(x, y, all.knots = TRUE, lambda = 100)
plot(mod.ss10, ylim = c(-1.75, 1.75))
points(x, y)
```



2.4 Penalty Order Influence

Setting $m = 2$ produces a cubic smoothing spline, which penalizes the squared second derivative of the function. Cubic smoothing splines are the default in many software. Cubic smoothing splines estimate $f(\cdot)$ using piecewise cubic functions, which are connected at points known as “knots” (later defined). The function estimates have two continuous derivatives at the knots, ensuring a smooth estimate of the function and its derivatives. Setting $m = 1$ results in a linear smoothing spline (a piecewise linear function), and setting $m = 3$ produces a quintic smoothing spline (a piecewise quintic function).

```
mod.lin <- ss(x, y, nknots = 10, m = 1)
mod.cub <- ss(x, y, nknots = 10)
mod.qui <- ss(x, y, nknots = 10, m = 3)
par(mfrow = c(1,3))
plot(mod.lin, ylim = c(-1.75, 1.75))
points(x, y)
plot(mod.cub, ylim = c(-1.75, 1.75))
points(x, y)
plot(mod.qui, ylim = c(-1.75, 1.75))
points(x, y)
```



3 Computational Details

3.1 Function Representation

The Kimeldorf-Wahba representer theorem reveals that the function $f \in \mathcal{H}$ that minimizes the penalized least squares functional has the form

$$f_\lambda(x) = \sum_{v=0}^{m-1} \beta_v N_v(x) + \sum_{u=1}^{r} \gamma_u K_1(x, x_u^*)$$

where $\{N_v\}_{v=0}^{m-1}$ are known functions spanning the null space $\mathcal{H}_0 = \{f : J_m(f) = 0\}$, $K_1(\cdot,\cdot)$ is the known reproducing kernel function for the contrast space $\mathcal{H}_1 = \mathcal{H} \ominus \mathcal{H}_0$, and $\{x_u^*\}_{u=1}^{r}$ are the selected spline knots. Note that $\boldsymbol\beta = (\beta_0, \ldots, \beta_{m-1})^\top$ and $\boldsymbol\gamma = (\gamma_0, \ldots, \gamma_r)^\top$ are the unknown basis function coefficient vectors. The optimal solution uses all n data points as knots, but it is often possible to obtain good solutions using $r < n$ knots. In such cases, it is typical to place the knots at the quantiles of x_i.

3.2 Coefficient Estimates

Applying the Kimeldorf-Wahba representer theorem, the penalized least squares functional can be rewritten as

$$\frac{1}{n}\|\mathbf{y} - \mathbf{X}\boldsymbol\beta - \mathbf{Z}\boldsymbol\gamma\|^2 + \lambda\boldsymbol\gamma^\top\mathbf{Q}\boldsymbol\gamma$$

where $\mathbf{y} = (y_1, \ldots, y_n)^\top$ is the response vector, $\mathbf{X} = [N_v(x_i)]$ is the null space basis function matrix, $\mathbf{Z} = [K_1(x_i, x_u^*)]$ is the contrast space basis function matrix, and $\mathbf{Q} = [K_1(x_u^*, x_v^*)]$ is the penalty matrix. Note that this is the correct form for the penalty given that

$$J_m(f_\lambda) = \sum_{u=1}^{r}\sum_{v=1}^{r} \gamma_u\gamma_v K_1(x_u^*, x_v^*)$$

due to the reproducing property of the kernel function.

Given λ, the optimal basis function coefficients can be written as

$$\begin{bmatrix} \hat{\boldsymbol\beta}_\lambda \\ \hat{\boldsymbol\gamma}_\lambda \end{bmatrix} = \begin{bmatrix} \mathbf{X}^\top\mathbf{X} & \mathbf{X}^\top\mathbf{Z} \\ \mathbf{Z}^\top\mathbf{X} & \mathbf{Z}^\top\mathbf{Z} + n\lambda\mathbf{Q} \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{X}^\top \\ \mathbf{Z}^\top \end{bmatrix} \mathbf{y}$$

where $(\cdot)^\dagger$ denotes the Moore-Penrose pseudoinverse. Note that the coefficient estimates are subscripted with λ because they depend on the chosen smoothing parameter. In other words, different choices of λ result in different coefficient estimates.

3.3 Fitted Values and Smoothing Matrix

The fitted values have the form

$$\begin{aligned} \hat{\mathbf{y}}_\lambda &= \mathbf{X}\hat{\boldsymbol\beta}_\lambda + \mathbf{Z}\hat{\boldsymbol\gamma}_\lambda \\ &= \mathbf{S}_\lambda\mathbf{y} \end{aligned}$$

where

$$\mathbf{S}_\lambda = [\,\mathbf{X} \quad \mathbf{Z}\,] \begin{bmatrix} \mathbf{X}^\top\mathbf{X} & \mathbf{X}^\top\mathbf{Z} \\ \mathbf{Z}^\top\mathbf{X} & \mathbf{Z}^\top\mathbf{Z} + n\lambda\mathbf{Q} \end{bmatrix}^\dagger \begin{bmatrix} \mathbf{X}^\top \\ \mathbf{Z}^\top \end{bmatrix}$$

is the *smoothing matrix*, which is the smoothing spline analogue of the “hat matrix” $\mathbf{H} = \mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$ in linear regression.

3.4 Effective Degrees of Freedom

In a parametric regression model, the degrees of freedom of a fit model is equivalent to the number of parameters, i.e., regression coefficients. However, this idea does not make sense for smoothing splines, where the number of coefficients could be equal to (or greater than!) the number of observations n.

In a nonparametric regression model, the effective (or equivalent) degrees of freedom (EDF) is defined as

$$\nu_\lambda = \mathrm{tr}(\mathbf{S}_\lambda)$$

where $\mathrm{tr}(\cdot)$ denotes the matrix trace function.

The EDF is subscripted with λ because the EDF changes as a function of λ

- As $\lambda \to 0$, the EDF approaches $m + r$ (null space dimension plus number of knots)
- As $\lambda \to \infty$, the EDF approaches m (null space dimension)

This definition of the model’s DF is a direct analogue of how the DF is defined in a multiple linear regression model: the number of coefficients is equal to the trace of the “hat matrix” in multiple linear regression.

4 Smoothing Parameter Selection

4.1 Ordinary Cross-Validation

The ordinary cross-validation (OCV) criterion, also known as the leave-one-out cross-validation (LOO-CV) criterion, seeks to find the λ that minimizes

$$\mathrm{OCV}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f_\lambda^{[i]}(x_i)\right)^2$$

where $f_\lambda^{[i]} \in \mathcal{H}$ is the function that minimizes the *leave-one-out* version of the penalized least squares functional, i.e.,

$$f_\lambda^{[i]} = \min_{f\in\mathcal{H}} \frac{1}{n}\sum_{j=1, j\neq i}^{n}(y_j - f(x_j))^2 + \lambda J_m(f)$$

which is the penalized least squares functional holding out the i-th observation’s data (x_i, y_i).

From its definition, it may seem that evaluating the OCV for a given λ requires fitting the model n different times, i.e., once holding out each (x_i, y_i) for $i = 1, \ldots, n$. However, this is not the case… It can be shown that the OCV can be evaluated using the results from the (single) model fit to the full sample of n observations.

Specifically, the OCV can be rewritten as

$$\mathrm{OCV}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - f_\lambda(x_i)}{1 - s_{ii(\lambda)}}\right)^2$$

where $f_\lambda(x_i)$ are the fitted values from the full solution, and $s_{ii(\lambda)}$ is the i-th diagonal element of the smoothing matrix.

4.2 Generalized Cross-Validation

The generalized cross-validation criterion (GCV) is an improvement of the OCV. Note that the OCV criterion can be interpreted as a weighted least squares criterion where the weights have the form $w_i = (1 - s_{ii(\lambda)})^{-2}$. The leverages $s_{ii(\lambda)}$ will differ across observations, which implies that the OCV gives a different weight to each observation for the CV tuning. To equalize the influence of the observations on the tuning of λ, the GCV criterion seeks to find the λ that minimizes

$$\mathrm{GCV}(\lambda) = \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - f_\lambda(x_i))^2}{(1 - \nu_\lambda/n)^2}$$

which is the OCV criterion with the leverages $s_{ii(\lambda)}$ replaced by their average values: $\frac{1}{n}\sum_{i=1}^{n} s_{ii(\lambda)} = \nu_\lambda/n$

4.3 AIC and BIC

If we assume the errors are iid Gaussian, we could use information criteria to select the smoothing parameter λ. Note that assuming $\epsilon_i \overset{\text{iid}}{\sim} N(0, \sigma^2)$ implies that $y_i \overset{\text{ind}}{\sim} N(f(x_i), \sigma^2)$.

Given an independent sample of n observations, the log-likelihood function has the form

$$l(\lambda, \sigma^2) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f_\lambda(x_i))^2 - \frac{n}{2}\log(\sigma^2) - \frac{n}{2}\log(2\pi)$$

which depends on the smoothing parameter and the error variance. In most cases σ^2 is unknown, so the maximum likelihood estimate $\sigma_\lambda^2 = (1/n)\sum_{i=1}^{n}(y_i - f_\lambda(x_i))^2$ can be used in its place.

Substituting σ_λ^2 for σ^2 gives the log-likelihood as a function of λ

$$\tilde{l}(\lambda) = l(\lambda, \sigma_\lambda^2) = -\frac{n}{2} - \frac{n}{2}\log(\sigma_\lambda^2) - \frac{n}{2}\log(2\pi)$$

which only depends on λ through $\log(\sigma_\lambda^2)$, i.e., the other terms are constants.

Note: maximizing $\tilde{l}(\lambda)$ would not be a useful way to select λ (reader: can you think of why not?).

Akaike’s An Information Criterion (AIC) and Schwarz’s Bayesian Information Criterion (BIC) select the smoothing parameter λ by adding a penalty to the log-likelihood $\tilde{l}(\lambda)$. Specifically, the AIC and BIC seek to find the λ that minimizes

$$\begin{aligned}\mathrm{AIC}(\lambda) &= -2\tilde{l}(\lambda) + 2\nu_\lambda \\ \mathrm{BIC}(\lambda) &= -2\tilde{l}(\lambda) + \log(n)\nu_\lambda\end{aligned}$$

which penalize the log-likelihood by a weight (either 2 or $\log(n)$) multiplied by the EDF of the function estimate.

4.4 REML and ML

Assume that $\boldsymbol{\gamma} \sim N(\mathbf{0}, \frac{\sigma^2}{n\lambda}\mathbf{Q}^{-1})$ and $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma^2\mathbf{I})$, where $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_n)^\top$ is the error vector. This implies that $\mathbf{y} \sim N(\mathbf{X}\boldsymbol{\beta}, \sigma^2\boldsymbol{\Sigma}_\lambda)$, where

$$\boldsymbol{\Sigma}_\lambda = \frac{1}{n\lambda}\mathbf{Z}\mathbf{Q}^{-1}\mathbf{Z}^\top + \mathbf{I}$$

is the part of the covariance matrix that depends on λ.

Given an independent sample of n observations, the log-likelihood function has the form

$$L(\lambda, \sigma^2) = -\frac{1}{2}\left\{\sigma^{-2}\mathbf{r}^\top\boldsymbol{\Sigma}_\lambda^{-1}\mathbf{r} + \log(|\boldsymbol{\Sigma}_\lambda|) + n\log(\sigma^2) + n\log(2\pi)\right\}$$

where $\mathbf{r} = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}$. In most cases σ^2 is unknown, so the maximum likelihood estimate

$$\sigma^2_{\lambda(\mathrm{ML})} = \frac{1}{n}\mathbf{r}^\top\boldsymbol{\Sigma}_\lambda^{-1}\mathbf{r}$$

can be used in its place.

Substituting $\sigma^2_{\lambda(\mathrm{ML})}$ for σ^2 gives the maximum likelihood criterion:

$$\mathrm{ML}(\lambda) = -\frac{1}{2}\left\{n + \log(|\boldsymbol{\Sigma}_\lambda|) + n\log(\mathbf{r}^\top\boldsymbol{\Sigma}_\lambda^{-1}\mathbf{r}) + n\log(2\pi/n)\right\}$$

which depends on λ through the $\boldsymbol{\Sigma}_\lambda$ matrix.

The restricted maximum likelihood (REML) function has the form

$$R(\lambda, \sigma^2) = L(\lambda, \sigma^2) - \frac{1}{2}\left\{\log(|\mathbf{X}^\top\boldsymbol{\Sigma}_\lambda^{-1}\mathbf{X}|) - m\log(2\pi\sigma^2)\right\}$$

which implies that the REML estimate of σ^2 has the form

$$\sigma^2_{\lambda(\mathrm{REML})} = \frac{1}{n-m}\mathbf{r}^\top\boldsymbol{\Sigma}_\lambda^{-1}\mathbf{r}$$

Substituting $\sigma^2_{\lambda(\mathrm{REML})}$ for σ^2 gives the restricted maximum likelihood criterion:

$$\mathrm{REML}(\lambda) = -\frac{1}{2}\left\{\tilde{n} + \log(|\boldsymbol{\Sigma}_\lambda|) + \tilde{n}\log(\mathbf{r}^\top\boldsymbol{\Sigma}_\lambda^{-1}\mathbf{r}) + \tilde{n}\log(2\pi/\tilde{n}) + \log(|\mathbf{X}^\top\boldsymbol{\Sigma}_\lambda^{-1}\mathbf{X}|)\right\}$$

where $\tilde{n} = n - m$ is the degrees of freedom corresponding to $\sigma^2_{\lambda(\mathrm{REML})}$.

5 Example 1: Prestige from Income

5.1 Overview of Data

The `Prestige` dataset (from the **car** package) contains the prestige of $n = 102$ Canadian occupations from 1971, as well as the average income of the occupation. We will use a smoothing spline to explore the relationship between prestige and income.

First, let's load the data and visualize the relationship between income (X) and prestige (Y).

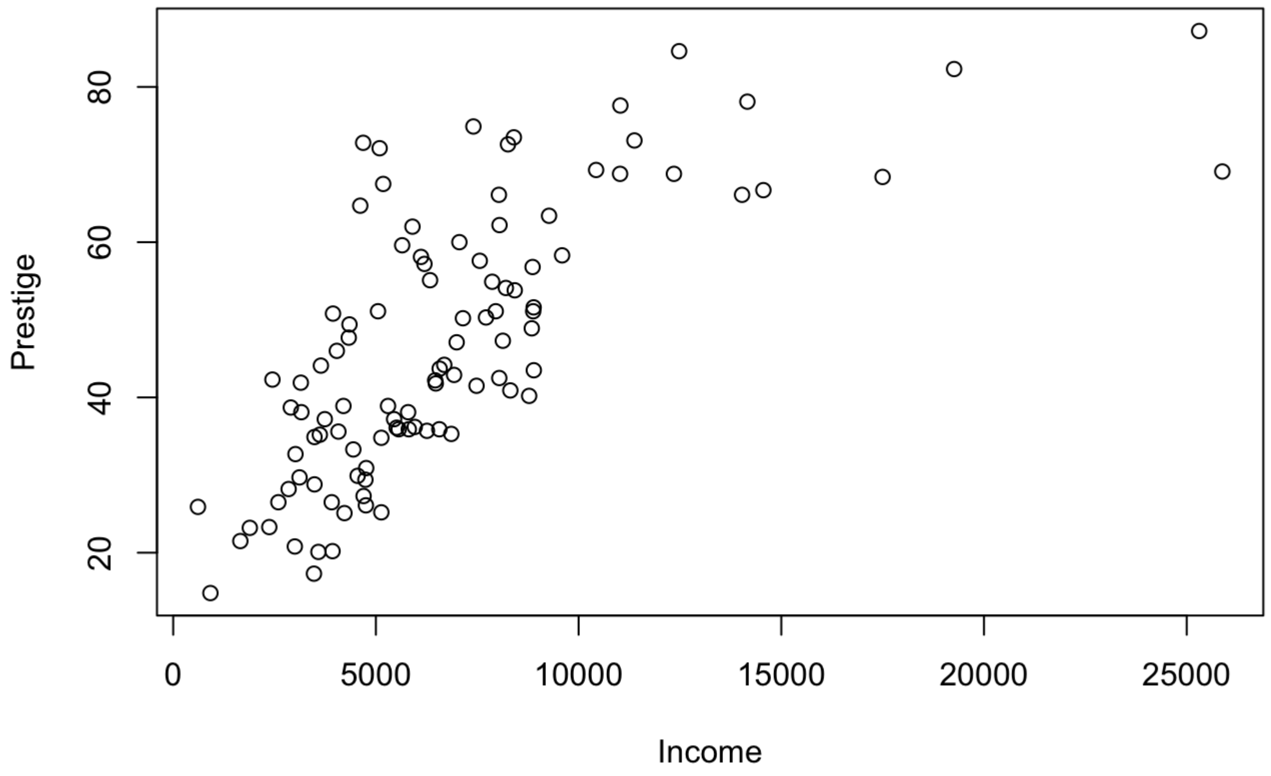
```
# load data
library(car)
```

```
## Loading required package: carData
```

```
data(Prestige)
head(Prestige)
```

```
##                      education income women prestige census type
## gov.administrators       13.11  12351 11.16     68.8   1113 prof
## general.managers         12.26  25879  4.02     69.1   1130 prof
## accountants              12.77   9271 15.70     63.4   1171 prof
## purchasing.officers      11.42   8865  9.11     56.8   1175 prof
## chemists                 14.62   8403 11.68     73.5   2111 prof
## physicists               15.64  11030  5.13     77.6   2113 prof
```

```
# plot data
plot(Prestige$income, Prestige$prestige,
     xlab = "Income", ylab = "Prestige")
```



Note: the relationship looks non-linear. For occupations that earn less than $10K, there is a strong (positive) linear relationship between income and prestige. However, for occupations that earn between $10K and $25K, the relationship has a substantially different (attenuated) slope.

5.2 Analyses and Results

R code to fit model (using default number of knots)

```
# fit model
mod.ss <- with(Prestige, ss(income, prestige))
mod.ss
```

```
##
## Call:
## ss(x = income, y = prestige)
##
## Smoothing Parameter  spar = 0.4923283   lambda = 0.0002148891
## Equivalent Degrees of Freedom (Df) 3.467512
## Penalized Criterion (RSS) 12118.05
## Generalized Cross-Validation (GCV) 127.3134
```

Note: the model has an EDF of $\nu_\lambda = 3.47$.

To get more details on the model fit, we can use the summary function:

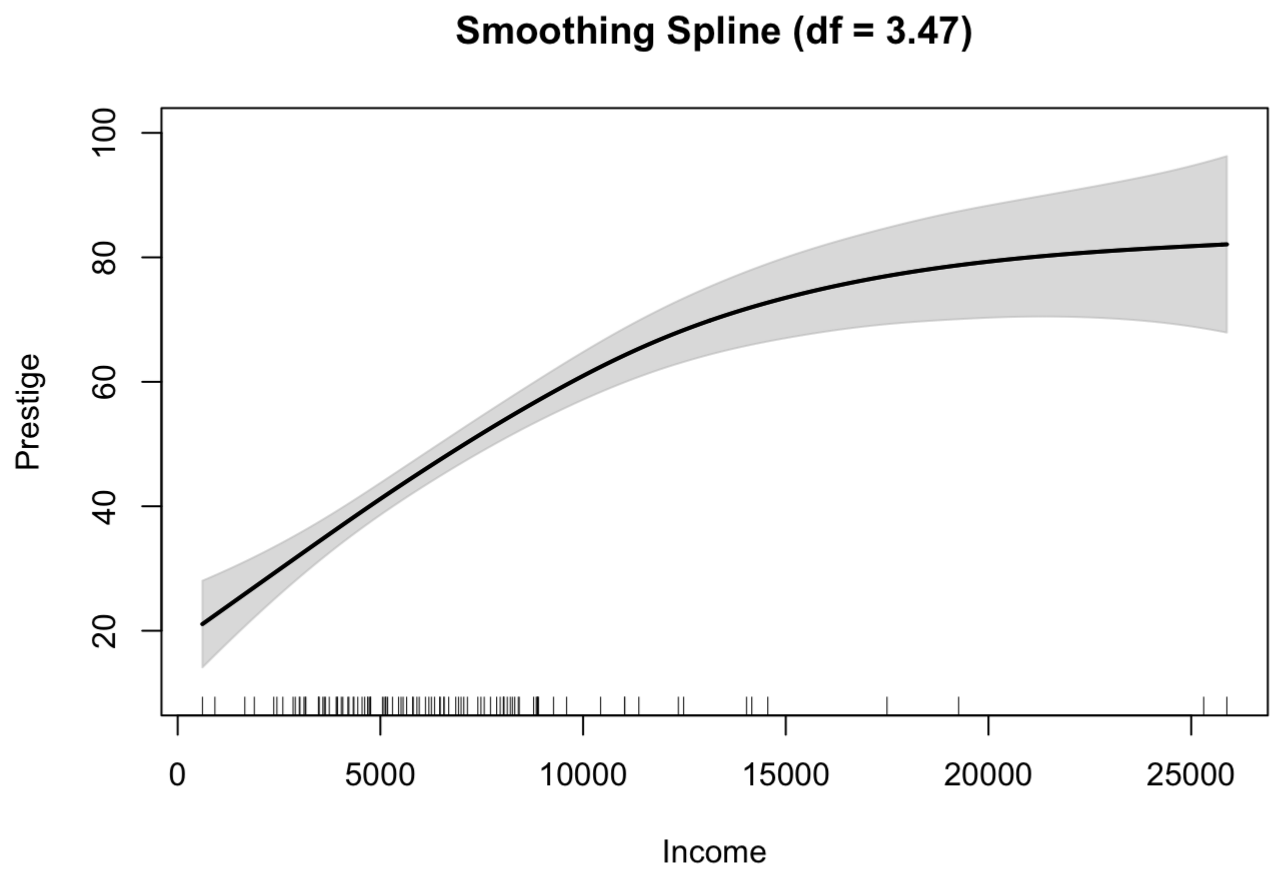
```
# summarize fit
summary(mod.ss)
```

```
##
## Call:
## ss(x = income, y = prestige)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.991  -7.727  -2.453   7.586  33.006
##
## Approx. Signif. of Parametric Effects:
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)    63.08      2.228  28.310 0.000e+00 ***
## x              61.01      8.035   7.593 1.807e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approx. Signif. of Nonparametric Effects:
##               Df Sum Sq Mean Sq F value    Pr(>F)
## s(x)       1.468   2125    1448   11.78 0.0001596 ***
## Residuals 98.532  12118     123
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.09 on 98.53 degrees of freedom
## Multiple R-squared:  0.5949,    Adjusted R-squared:  0.5845
## F-statistic: 57.35 on 2.468 and 98.53 DF,  p-value: <2e-16
```

Note: the model has a coefficient of determination of $R^2 = 0.5949$.

The estimated functional relationship can be plotted using the `plot` function:

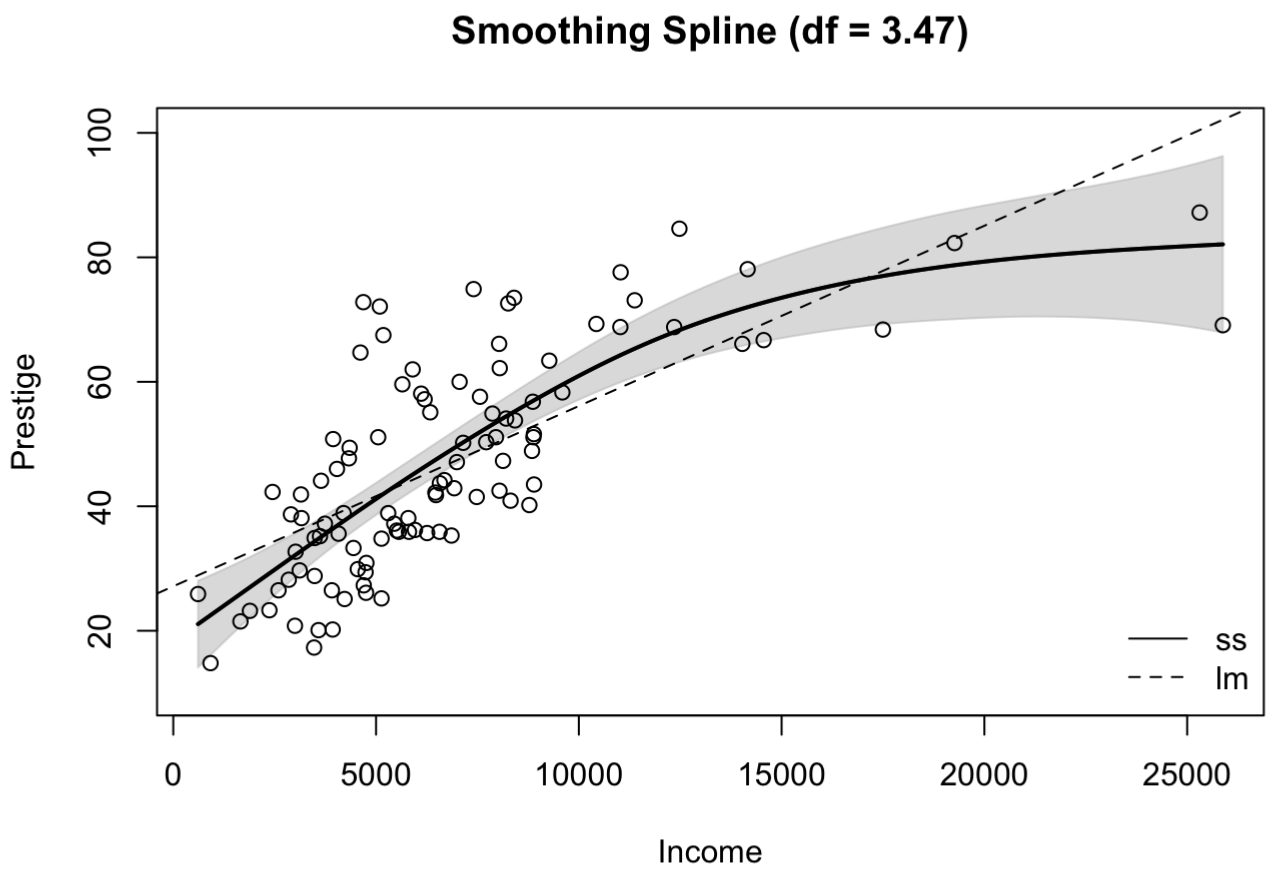
```
# plot fit
plot(mod.ss, xlab = "Income", ylab = "Prestige")
rug(Prestige$income)  # add rug to plot
```



The gray shaded area denotes a 95% Bayesian “confidence interval” for the unknown function.

Compare to `lm` fit:

```
# plot ss fit
plot(mod.ss, xlab = "Income", ylab = "Prestige")

# add lm fit
abline(coef(lm(prestige ~ income, data = Prestige)), lty = 2)

# add data and legend
with(Prestige, points(income, prestige))
legend("bottomright", legend = c("ss", "lm"), lty = 1:2, bty = "n")
```



Note: the `lm` fit does not fall completely within the Bayesian CIs, suggesting that the nonparametric model should be preferred.

6 Example 2: Motorcycle Accident

6.1 Overview of Data

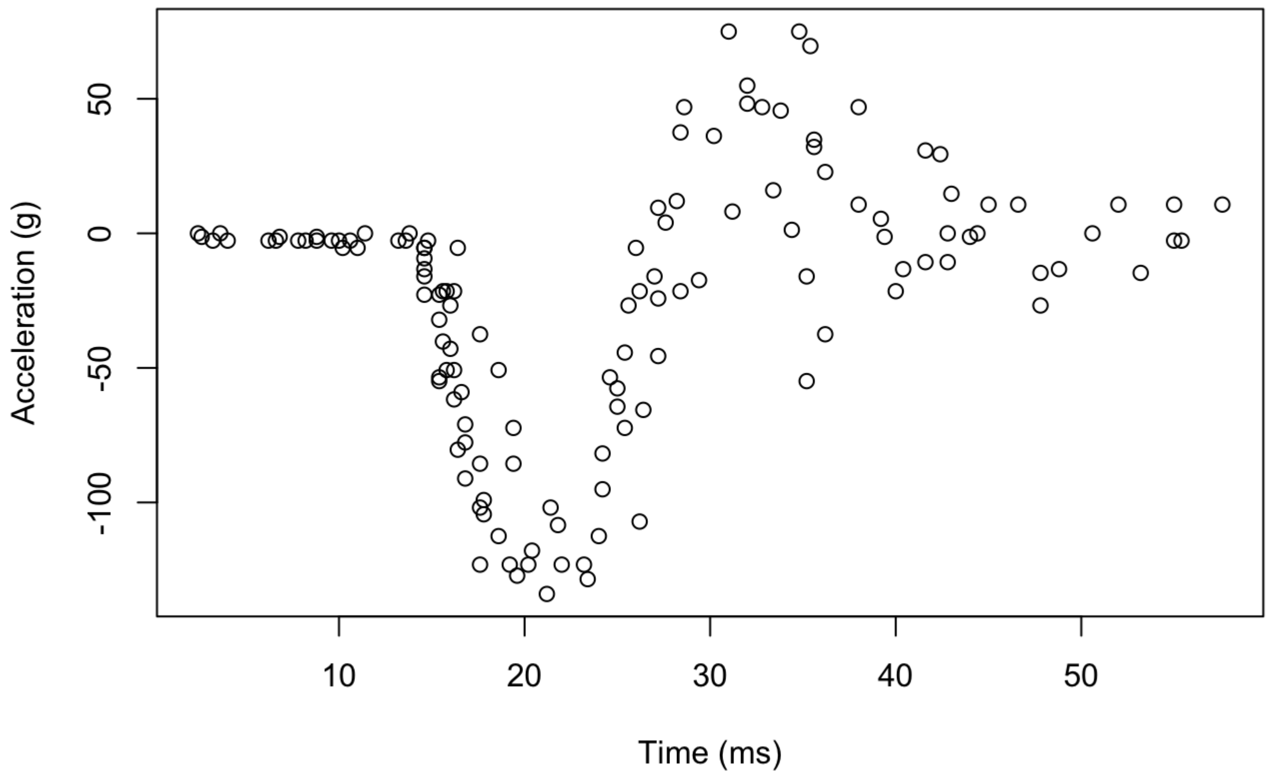
The `mcycle` dataset (from the **MASS** package) contains $n = 133$ pairs of time points (in ms) and observed head accelerations (in g) that were recorded in a simulated motorcycle accident. We will use a smoothing spline to explore the relationship between time and acceleration.

First, let's load the data and visualize the relationship between time (X) and acceleration (Y).

```
# load data
library(MASS)
data(mcycle)
head(mcycle)
```

```
##   times accel
## 1   2.4   0.0
## 2   2.6  -1.3
## 3   3.2  -2.7
## 4   3.6   0.0
## 5   4.0  -2.7
## 6   6.2  -2.7
```

```
# plot data
plot(mcycle$times, mcycle$accel,
     xlab = "Time (ms)", ylab = "Acceleration (g)")
```



Note: the relationship looks non-linear. The head acceleration is stable from 0-15 ms, drops from about 15-20 ms, rises from 20-30 ms, drops from 30-40 ms, and then begins to stabilize.

6.2 Analyses and Results

R code to fit model (using default number of knots)

```
# fit model
mod.ss <- with(mcycle, ss(times, accel))
mod.ss
```

```
##
## Call:
## ss(x = times, y = accel)
##
## Smoothing Parameter  spar = 0.1585867   lambda = 8.337281e-07
## Equivalent Degrees of Freedom (Df) 12.20781
## Penalized Criterion (RSS) 62034.66
## Generalized Cross-Validation (GCV) 565.4684
```

Note: the model has an EDF of $\nu_\lambda = 12.21$.

To get more details on the model fit, we can use the summary function:

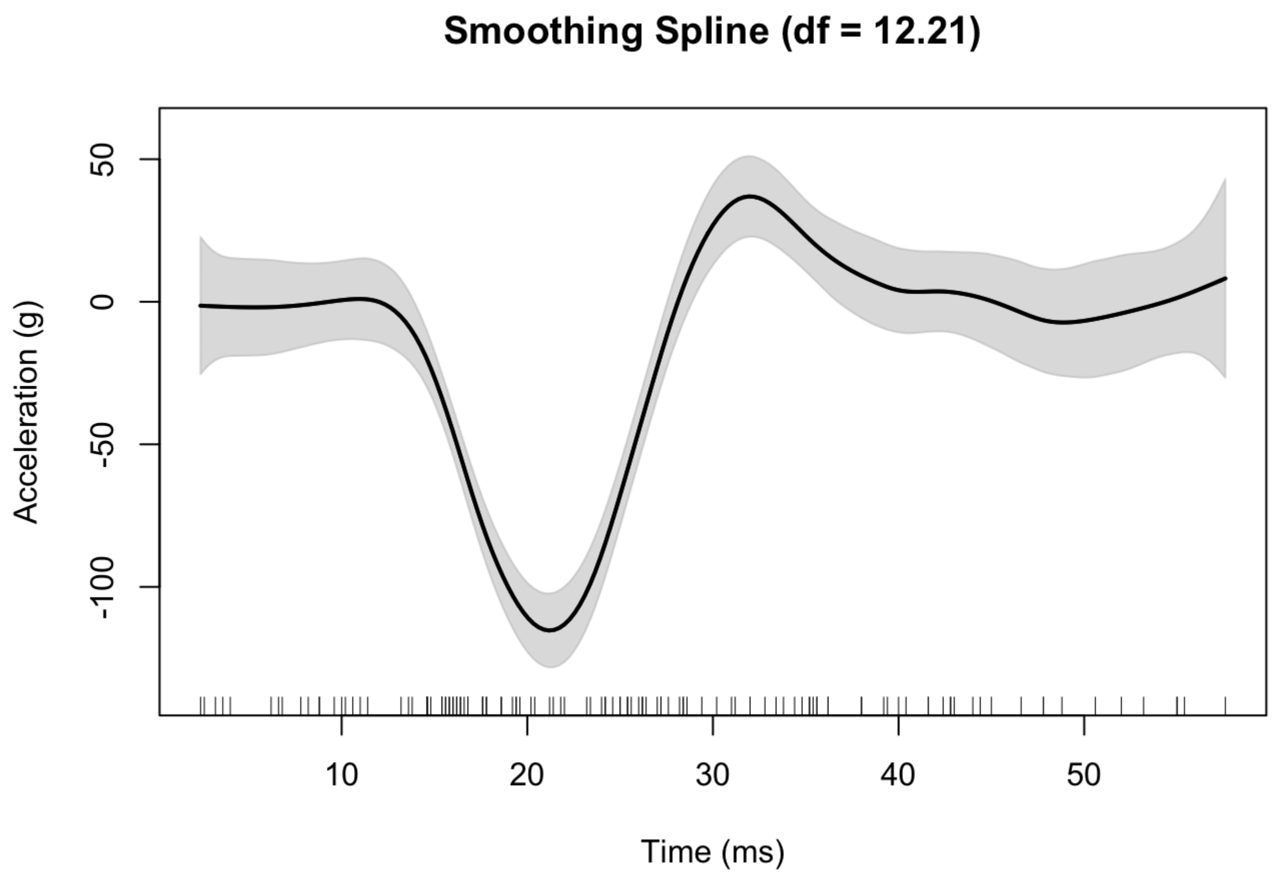
```
# summarize fit
summary(mod.ss)
```

```
##
## Call:
## ss(x = times, y = accel)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -76.7951 -12.5654  -0.8346  12.5823  50.5576
##
## Approx. Signif. of Parametric Effects:
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  -14.234      2.313  -6.154 1.018e-08 ***
## x              9.549     21.603   0.442 6.593e-01
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approx. Signif. of Nonparametric Effects:
##               Df Sum Sq Mean Sq F value Pr(>F)
## s(x)       10.21 210130 20585.3   40.08      0 ***
## Residuals 120.79  62035   513.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.66 on 120.8 degrees of freedom
## Multiple R-squared:  0.7999,    Adjusted R-squared:  0.7801
## F-statistic: 41.21 on 11.21 and 120.8 DF,  p-value: <2e-16
```

Note: the model has a coefficient of determination of $R^2 = 0.8$.

The estimated functional relationship can be plotted using the `plot` function:

```
# plot fit
plot(mod.ss, xlab = "Time (ms)", ylab = "Acceleration (g)")
rug(mcycle$times)  # add rug to plot
```



The gray shaded area denotes a 95% Bayesian “confidence interval” for the unknown function.