# Splines In Stan

***Milad Kharratzadeh***
***milad.kharratzadeh@gmail.com (mailto:milad.kharratzadeh@gmail.com)***

***24 October, 2017***

# Overview

In this document, we discuss the implementation of splines in Stan. We start by providing a brief introduction to splines and then explain how they can be implemented in Stan. We also discuss a novel prior that alleviates some of the practical challenges of spline models.

# Introduction to Splines

Splines are continuous, piece-wise polynomial functions. B-splines or basis splines are the building blocks of spline functions: any spline function of a given degree can be expressed as a linear combination of B-splines of that degree. There are two parameters that uniquely define a family of B-spline functions: (i) the polynomial degree, $p$; and (ii) a non-decreasing sequence of knots, $t_1, \dots, t_q$. The order of a spline family is defined as $p+1$. B-splines of order 1 ( $p=0$) are a set of piece-wise constant functions:

$$B_{i,1}(x) := \begin{cases} 1, & \text{if} \quad t_i \leq x < t_{i+1} \\ 0, & \text{otherwise,} \end{cases}$$

where $B_{i,k}$ denotes the $i$'th member of a family of B-splines of order $k$ (or equivalently of degree $k-1$). B-splines of higher orders are defined recursively:

$$B_{i,k}(x) := \omega_{i,k} B_{i,k-1}(x) + (1-\omega_{i+1,k}) B_{i+1,k-1}(x).$$

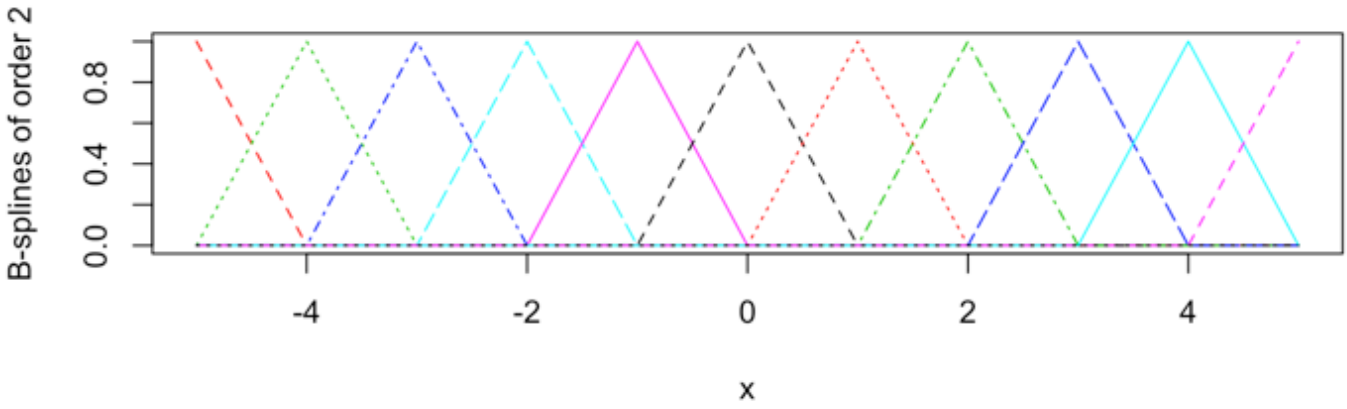
where

$$\omega_{i,k} := \begin{cases} \frac{x-t_i}{t_{i+k-1}-t_i}, & \text{if} \quad t_i \neq t_{i+k-1} \\ 0, & \text{otherwise.} \end{cases}$$

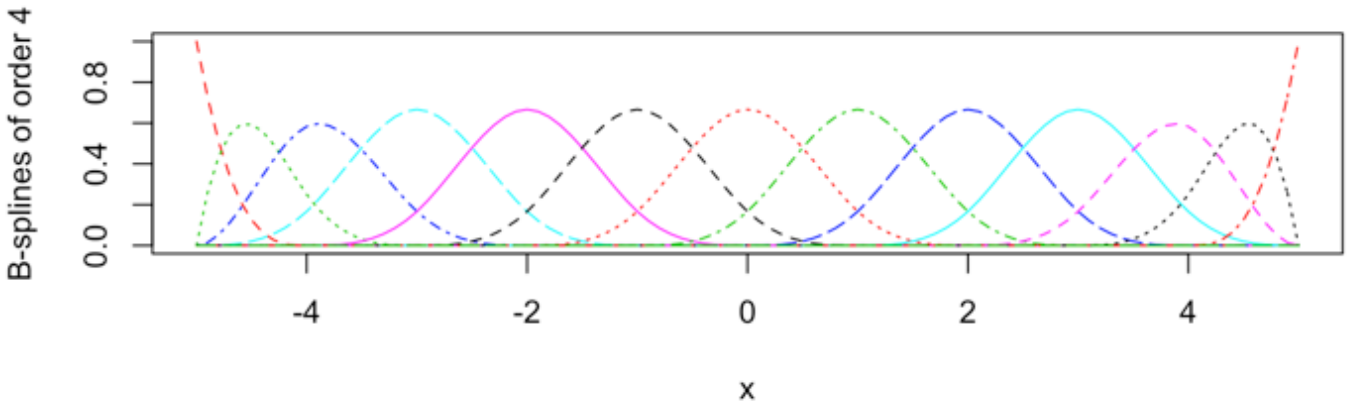
Thus, at a given point, $x$, a B-spline function of order $k$ is a linar combination of two B-splines of order $k-1$. To have well-defined B-splines of order $k$ covering the whole span of the knots, i.e., the interval $[t_1, t_q)$, the sequence of knots have to be extended as follows:

$$\text{extended knots} : \underbrace{t_1, \dots, t_1}_{k-1 \text{ times}}, \underbrace{t_1, t_2, t_3, \dots, t_q}_{\text{original knot sequence}}, \underbrace{t_q, \dots, t_q}_{k-1 \text{ times}}.$$

The recursive formulation described above is defined over the extended knots. If we do not extend the sequence of knots, we will not have B-spline functions covering the areas around the edges of the interval $[t_1, t_q)$. The second-order B-spline, $B_{i,2}$, has, in general, two linear pieces which join continuously at $t_{i+1}$ to form a piecewise linear function and vanishes outside the interval $[t_i, t_{i+2})$. A family of second-order B-splines for knots $= \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$ is shown below:



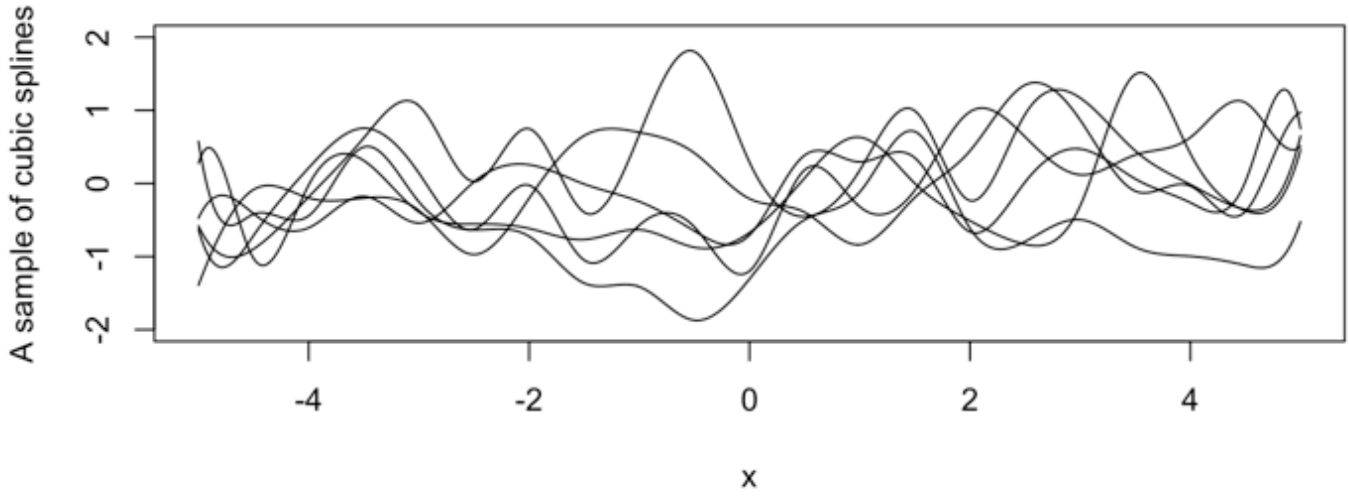

As we can see, all the first-order B-splines, except the first and the last ones, consist of two linear pieces. Those two splines are defined around the edge of the interval, and because of the knot extension, have two equal knots. Similarly, B-splines of order $k$ consist of $k$ pieces of polynomials with degree $k - 1$ (except those B-splines near the borders). These $k$ pieces are joined continuously at $k - 1$ interior knots and moreover, are $k - 2$ times differentiable. For instance, B-splines of order 4 have differenitals of orders 1 and 2; that is why the fourth-order B-splines (which consist of cubic polynomials) are widely used in practice. In the Figure below, we show a family of fourth-order B-splines:



We are now ready to define splines. A spline of order $k$ (degree $k-1$) with knot sequence $\mathbf{t} = t_1, \ldots, t_q$ is defined as linear combination of the B-splines, $B_{i,k}$, corresponding with that knot sequence. The set of all such splines can be denoted as follows:

$$\mathbf{S}_{k,t}(x) = \{\sum_i a_i B_{i,k}(x), \quad a_i \in \mathbb{R}\}.$$

A sample of fourth-order splines with $a_i$ independently drawn from $\mathcal{N}(0,1)$ is shown below:




# Fitting Splines in Stan

## Building B-splines in R

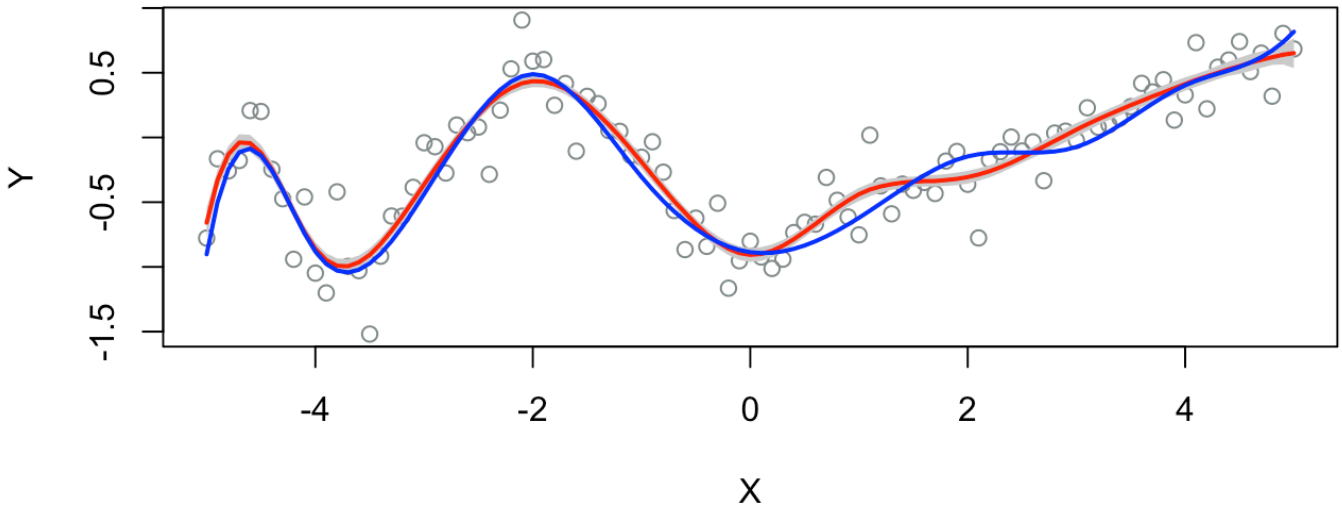
We are interested in using splines for regression; given an order and a set of knots, we would like to estimate the spline coefficients, $a_i$, that give the best fit to the data. For now, we focus on the univariate case and assume that the data consists of pairs of observation $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ and our goal is to predict $y$ from $x$. There are some readily available functions in R to generate the B-splines. These functions can be used to build a design matrix. Then, this design matrix can be passed on to Stan for a usual linear regression. The code below demonstrates how this can be done for a toy example:

```
library("splines")
library("rstan")
X <- seq(from=-5, to=5, by=.1) # generating inputs
B <- t(bs(X, knots=seq(-5,5,1), degree=3, intercept = TRUE)) # creating
the B-splines
num_data <- length(X); num_basis <- nrow(B)
a0 <- 0.2 # intercept
a <- rnorm(num_basis, 0, 1) # coefficients of B-splines
Y_true <- as.vector(a0*X + a%*%B) # generating the output
Y <- Y_true + rnorm(length(X),0,.2) # adding noise
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
sm<-stan_model("fit_basis.stan")
fit<-sampling(sm,iter=500,control = list(adapt_delta=0.95))
```

where `fit_basis.stan` performs a simple regression:

```
data {
  int num_data;
  int num_basis;
  vector[num_data] Y;
  vector[num_data] X;
  matrix[num_basis, num_data] B;
}

parameters {
  row_vector[num_basis] a_raw;
  real a0;
  real<lower=0> sigma;
  real<lower=0> tau;
}

transformed parameters {
  row_vector[num_basis] a;
  vector[num_data] Y_hat;
  a = a_raw*tau;
  Y_hat = a0*X + to_vector(a*B);
}

model {
  a_raw ~ normal(0, 1);
  tau ~ cauchy(0, 1);
  sigma ~ cauchy(0, 1);
  Y ~ normal(Y_hat, sigma);
}
```

The true curve (in blue), the fitted curve (in red), and the 50% uncertainty intervals (in grey) are shown below:




## Building B-splines in Stan

It is also possible to form B-splines directly in Stan. Having all the code in the same place (from forming the B-splines to fitting them to the data) gives us more flexibility and allows us to adjust the knots locations or order of B-splines in the Stan program. The Stan recursive function for building the B-splines is shown below. Its output is $B_{ind,order}(t)$ for the set of given extended knots.

```
functions {
  vector build_b_spline(real[] t, real[] ext_knots, int ind, int order)
;
  vector build_b_spline(real[] t, real[] ext_knots, int ind, int order)
{
    // INPUTS:
    //    t:          the points at which the b_spline is calculated
    //    ext_knots:  the set of extended knots
    //    ind:        the index of the b_spline
    //    order:      the order of the b-spline
    vector[size(t)] b_spline;
    vector[size(t)] w1 = rep_vector(0, size(t));
    vector[size(t)] w2 = rep_vector(0, size(t));
    if (order==1)
      for (i in 1:size(t)) // B-splines of order 1 are piece-wise const
ant
        b_spline[i] = (ext_knots[ind] <= t[i]) && (t[i] < ext_knots[ind
+1]);
    else {
      if (ext_knots[ind] != ext_knots[ind+order-1])
        w1 = (to_vector(t) - rep_vector(ext_knots[ind], size(t))) /
             (ext_knots[ind+order-1] - ext_knots[ind]);
      if (ext_knots[ind+1] != ext_knots[ind+order])
        w2 = 1 - (to_vector(t) - rep_vector(ext_knots[ind+1], size(t)))
/
                 (ext_knots[ind+order] - ext_knots[ind+1]);
      // Calculating the B-spline recursively as linear interpolation o
f two lower-order splines
      b_spline = w1 .* build_b_spline(t, ext_knots, ind, order-1) +
                 w2 .* build_b_spline(t, ext_knots, ind+1, order-1);
    }
    return b_spline;
  }
}
```

This function can then be called, e.g., in the `transformed data` block, to form the B-splines. An example code, `b_spline.stan`, is shown below.

```
data {
  int num_data;             // number of data points
  int num_knots;            // num of knots
  vector[num_knots] knots;  // the sequence of knots
```

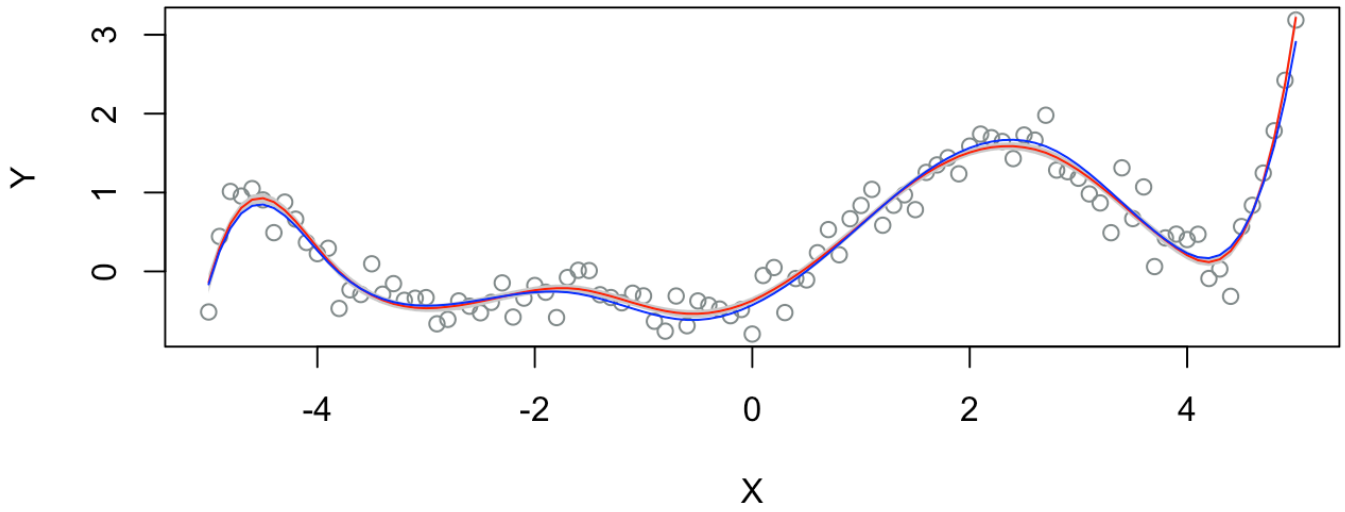
```
  int spline_degree;        // the degree of spline (is equal to order
- 1)
  real Y[num_data];
  real X[num_data];
}

transformed data {
  int num_basis = num_knots + spline_degree - 1; // total number of B-s
plines
  matrix[num_basis, num_data] B;  // matrix of B-splines
  vector[spline_degree + num_knots] ext_knots_temp;
  vector[2*spline_degree + num_knots] ext_knots; // set of extended kno
ts
  ext_knots_temp = append_row(rep_vector(knots[1], spline_degree), knot
s);
  ext_knots = append_row(ext_knots_temp, rep_vector(knots[num_knots], s
pline_degree));
  for (ind in 1:num_basis)
    B[ind,:] = to_row_vector(build_b_spline(X, to_array_1d(ext_knots),
ind, spline_degree + 1));
  B[num_knots + spline_degree - 1, num_data] = 1;
}

parameters {
  row_vector[num_basis] a_raw;
  real a0;  // intercept
  real<lower=0> sigma;
  real<lower=0> tau;
}

transformed parameters {
  row_vector[num_basis] a; // spline coefficients
  vector[num_data] Y_hat;
  a = a_raw*tau;
  Y_hat = a0*to_vector(X) + to_vector(a*B);
}

model {
  // Priors
  a_raw ~ normal(0, 1);
  a0 ~ normal(0, 1);
  tau ~ normal(0, 1);
  sigma ~ normal(0, 1);
```

```
  //Likelihood
  Y ~ normal(Y_hat, sigma);
}
```

In this example, we take in the set of knots as an input. Alternatively, we could define the knots within the Stan code. Usually a good way to set the knots is by setting them to the percentiles of the data. (In the next section, we discuss the choice of knots locations.) In the code below, we simulate the data and fit B-splines to it using the Stan program above.

```
library("splines")
library("rstan")
num_knots <- 10
spline_degree <- 3
num_basis <- num_knots + spline_degree - 1
X <- seq(from=-5, to=5, by=.1)
num_data <- length(X)
knots <- unname(quantile(X,probs=seq(from=0, to=1, length.out = num_kno
ts)))
a0 <- 0.2
a <- rnorm(num_basis, 0, 1)
B_true <- t(bs(X, df=num_basis, degree=spline_degree, intercept = TRUE)
)
Y_true <- as.vector(a0*X + a%*%B_true)
Y <- Y_true + rnorm(length(X), 0, 0.2)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
spline_model <- stan_model("b_spline.stan")
fit_spline <- sampling(spline_model, iter=500, control = list(adapt_del
ta=0.95))
```

The true curve (in blue), the fitted curve (in red), and the 50% undertainty intervals (in grey) are shown below:




# Location of Knots and the Choice of Priors

In practical problems, it is not always clear how to choose the number/location of the knots. Choosing too many/too few knots may lead to overfitting/underfitting. In this part, we introduce a prior that alleviates the problems associated with the choice of number/locations of the knots to a great extent.

Let us start by a simple observation. For any given set of knots, and any B-spline order, we have:

$$\sum_i B_{i,k}(x) = 1.$$

The proof is simple and can be done by induction. This means that if the B-spline coefficients, $a_i = a$, are all equal, then the resulting spline is the constant function equal to $a$ all the time. In general, the closer the consecutive $a_i$ are to each other, the smoother (less wiggly) the resulting spline curve is. In other words, B-splines are local bases that form the splines; if the the coefficients of nearby B-splines are close to each other, we will have less local variability.

This motivates the use of priors enforcing smoothness across the coefficients, $a_i$. With such priors, we can choose a large number of knots and do not worry about overfitting. Here, we propose a random-walk prior as follows:

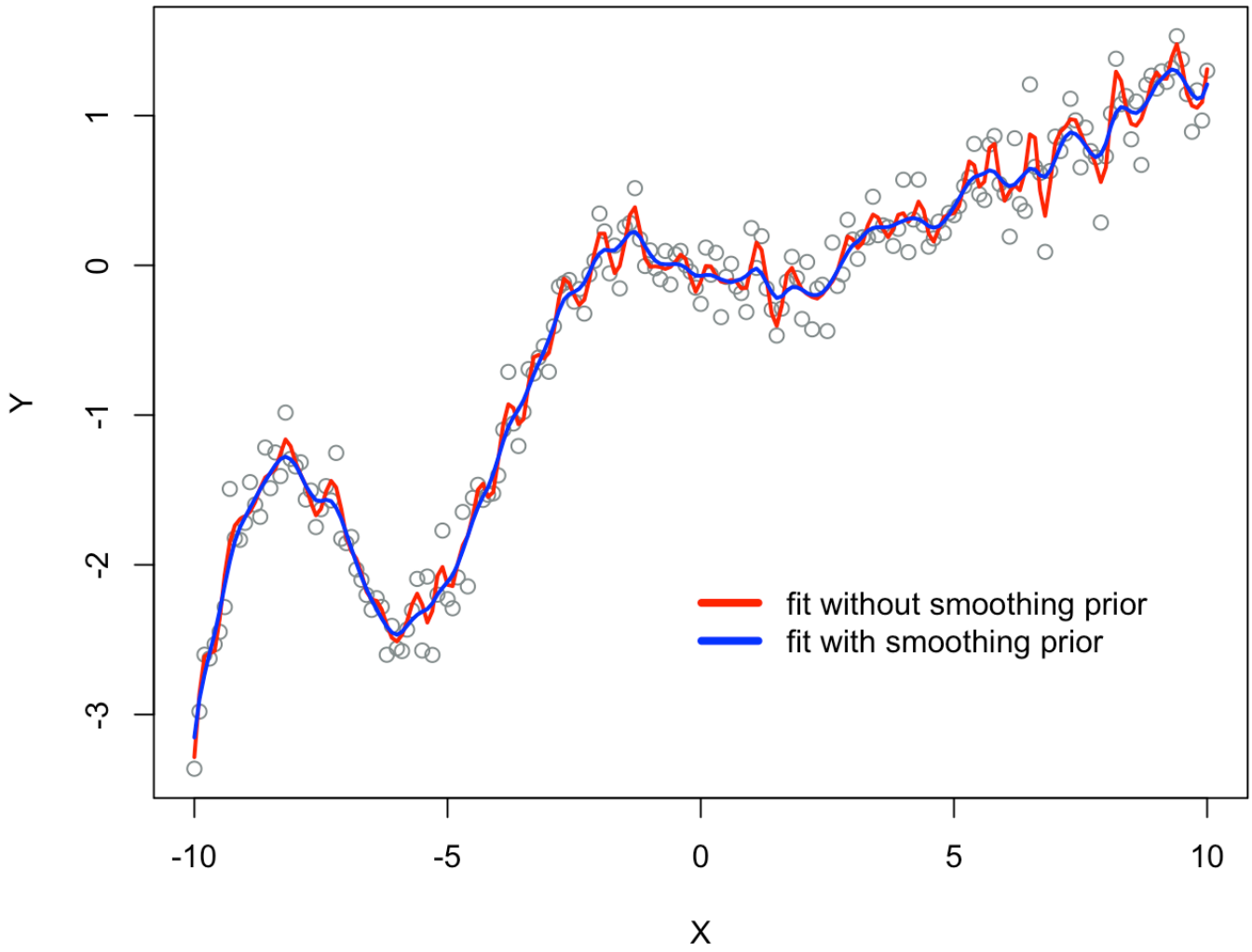
$$a_1 \sim \mathcal{N}(0, 1) \qquad a_i \sim \mathcal{N}(a_{i-1}, \tau) \qquad \tau \sim \mathcal{N}(0, 1)$$

The Stan code, `b_spline_penalized.stan` , is the same as before except for the `transformed parameters` block:

```
transformed parameters {
  row_vector[num_basis] a;
  vector[num_data] Y_hat;
  a[1] = a_raw[1];
  for (i in 2:num_basis)
    a[i] = a[i-1] + a_raw[i]*tau;
  Y_hat = a0*to_vector(X) + to_vector(a*B);
}
```

To demonstrate the advantage of using this smoothing prior, we consider an extreme case where the number of knots for estimation is much larger than the true number of knots. We generate the synthetic data with 10 knots, but for the Stan program, we set the number of knots to 100 (ten times more than the true number). Then, we fit the model with and without the smoothing prior and compare the results. The two fits are shown in the figure below. We observe that without using the smoothing prior (red curve), the large number of knots results in a wiggly curve (overfitting). When the smoothing prior is used (blue curve), we achieve a much smoother curve.

```
library(rstan)
library(splines)
set.seed(1234)
num_knots <- 10 # true number of knots
spline_degree <- 3
num_basis <- num_knots + spline_degree - 1
X <- seq(from=-10, to=10, by=.1)
knots <- unname(quantile(X,probs=seq(from=0, to=1, length.out = num_kno
ts)))
num_data <- length(X)
a0 <- 0.2
a <- rnorm(num_basis, 0, 1)
B_true <- t(bs(X, df=num_basis, degree=spline_degree, intercept = TRUE)
)
Y_true <- as.vector(a0*X + a%*%B_true)
Y <- Y_true + rnorm(length(X), 0, 0.2)

num_knots <- 100; # number of knots for fitting
num_basis <- num_knots + spline_degree - 1
knots <- unname(quantile(X,probs=seq(from=0, to=1, length.out = num_kno
ts)))
rstan_options(auto_write = TRUE);
options(mc.cores = parallel::detectCores());
spline_model<-stan_model("b_spline.stan")
spline_penalized_model<-stan_model("b_spline_penalized.stan")
fit_spline<-sampling(spline_model,iter=500,control = list(adapt_delta=0
.95))
fit_spline_penalized<-sampling(spline_penalized_model,iter=500,control
= list(adapt_delta=0.95))
```




## License