SPLINTER USER GUIDE

A tutorial for downloading, installing and using SPLINTER, with example problems.

Draft for SPLINTER version 4.0.

Bjarne Grimstad

January 26, 2018

Contents

1 Introduction to SPLINTER

The intention with this document is to inform and guide users of SPLINTER – a library for multivariate function approximation with splines.

SPLINTER is an open-source library implemented in C++. It is publicly available at `github.com/bgrimstad/splinter`, and is distributed under the MPL 2.0 license [2].

1.1 Reader prerequisites

The reader should be acquainted with the following topics:

- Linear algebra: solving linear systems, the concept of sparsity, inner product, and vector norms.
- Spline Theory: some knowledge about the B-spline and its properties.

1.2 Installation

SPLINTER can be installed in the following ways:

- C/C++: compile from source files or link to released binaries (see `https://github.com/bgrimstad/splinter/releases`.
- Python: Install *splinterpy* via PyPI or run released binaries.

1.3 Third-party dependencies

SPLINTER is packaged with the following third-party libraries:

- The C++ template library EIGEN for linear algebra [3]
- The JSON for Modern C++ library by Niels Lohmann for saving and loading B-splines to JSON files [4].

2 Background on B-splines

SPLINTER implements multivariate B-splines, also known as tensor product B-splines owing to their construction which utilizes the Kronecker product. Before diving into the multivariate case, let us consider the form of a B-spline in a single variable.

2.1 Univariate B-splines

A B-spline $f : \mathbb{R} \to \mathbb{R}^n$ of degree p in the variable x is expressed as

$$f(x) = \sum_{i=0}^{l-1} P_i B_{i,p}(x; \boldsymbol{t}), \tag{1}$$

where $\{B_{i,p}\}_{i=0}^{l-1}$ are B-spline *basis functions*. A B-spline is defined by three parameters:

- the *control points* $\{P_i\}_{i=0}^{l-1} \in \mathbb{R}^n$;
- the degree $p \in \mathbb{N}_0$;
- and the *knot sequence* $\boldsymbol{t} = \{t_i\}_{i=0}^{l+p} \in \mathbb{R}$.

A control point P_i may be interpreted as the weight to put on basis function B_i. The shape of the B-spline is often controlled by modifying the control points – as in the famous Bézier curves. The degree and knot sequence define the basis functions as follows[1]

$$\begin{aligned} B_{i,p}(x; \boldsymbol{t}) &= \frac{x - t_i}{t_{i+p} - t_i} B_{i,p-1}(x; \boldsymbol{t}) + \frac{t_{i+p+1} - x}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(x; \boldsymbol{t}), \\ B_{i,0}(x; \boldsymbol{t}) &= \begin{cases} 1, & t_i \leq x < t_{i+1}, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{2}$$

By definition, the basis functions are piecewise polynomial functions. The avid reader will notice that the basis functions are defined recursively, with the base case being the zeroth degree, piecewise constant basis functions $\{B_{i,0}\}$. Furthermore, a degree p basis function is defined as the convex combination of two basis functions of degree $p - 1$. The B-spline owes its numerical stability mainly to this construction.

For the definition of the B-spline basis functions to make sense, some restrictions must be put on the knot sequence. First of all, the definition requires that the knot sequence has $l + p + 1$ values, or *knots*. The knots defines the partition of the domain; that is, where the basis functions are non-zero (have support). Secondly, it is required that the knot sequence is a non-decreasing sequence of real numbers, i.e. $t_0 \leq t_1 \leq \cdots \leq t_{l+p}$. A

[1]Division by zero is handled by a '0/0 = 0' convention.

knot sequence is said to be *regular* if it also abides to the requirement that $t_i < t_{i+p+1}$. The maximum knot multiplicity of any knot in a regular knot sequence is thus $p+1$; that is, no knot may be repeated more than $p+1$ times.

Before we consider some of the great properties of the B-spline basis functions, let us introduce a compact form in which we may express the B-spline:

$$f(x) = \boldsymbol{P}\boldsymbol{B}_p(x; \boldsymbol{t}), \tag{3}$$

where we have stacked the control points in the matrix $\boldsymbol{P} = [P_0, \dots, P_{l-1}] \in \mathbb{R}^{n \times l}$, and the basis functions in the vector $\boldsymbol{B}_p = [B_{0,p}, \dots, B_{l-1,p}]^\mathsf{T} \in \mathbb{R}^l$. When it is clear from the context what the knot sequence is, we will simply write $B_{i,p}(x) \implies B_{i,p}(x; \boldsymbol{t})$, $\boldsymbol{B}_p(x) \implies \boldsymbol{B}_p(x; \boldsymbol{t})$ and $f(x) = \boldsymbol{P}\boldsymbol{B}_p(x)$.

2.2 Some properties of B-spline basis functions

The following holds true for a set of degree p B-spline basis functions $\{B_{i,p}\}$:

$$\begin{aligned} &B_{i,p}(x) \geq 0, \quad i = 1, \dots, l, \quad \forall x \in \mathbb{R} \\ &B_{i,p}(x) = 0, \quad \forall x \notin [t_i, t_{i+p+1}) \\ &\sum_{i=j-p}^{j} B_{i,p}(x) = 1, \quad \forall x \in [t_j, t_{j+1}) \end{aligned} \tag{4}$$

The first property in (4) says that the B-spline basis functions are always *non-negative*. The second property, often referred to as the *local support* property, shows us that a basis function $B_{i,p}$ is non-zero only in the interval $[t_i, t_{i+p+1}]$. This property, which shows us the sparse structure of B-splines basis functions, is heavily used by SPLINTER to avoid evaluation of zero-valued basis functions. The third property above, called the *partition-of-unity*, says that the basis functions always sum to one in any knot span. Notice that, due to the local support property, the sum is taken over $j - p, \dots, j$.

From the construction above, it is possible to derive several other useful properties of the B-spline. For example, the modeller of a B-spline may control its continuity properties by selecting appropriate knots and degree. For example, with $p = 3$ and a knot sequence without repeating knots (except for the start and end knots), the modeller can be assured that the B-spline is continuous to the second degree; i.e. has continuous second-order derivatives. A B-spline with $p = 3$ is referred to as a cubic B-spline, and it is perhaps the most widely used B-spline due to its smoothness. Higher degree B-splines are less frequently used since it usually is sufficient to just add more basis functions by increasing the number of knots, to gain more flexibility (freedom to fit other functions).

2.3 Multivariate B-splines

Now, let us consider the multivariate case in which we have m variables $\boldsymbol{x} = (x_1, \dots, x_m) \in \mathbb{R}^m$. A basis for multivariate B-splines is simply constructed by taking the Kronecker product of univariate basis functions. Let $\boldsymbol{B}_{p_k}(x_k; \boldsymbol{t}_k)$ be the l_k, degree p_k basis functions in variable x_k, defined by the knot sequence $\boldsymbol{t}_k$. Then, we may construct a multivariate basis as follows:

$$\boldsymbol{B}_{\boldsymbol{p}}(\boldsymbol{x}; \boldsymbol{t}_1, \dots, \boldsymbol{t}_n) := \bigotimes_{k=1}^{m} \boldsymbol{B}_{p_k}(x_k; \boldsymbol{t}_k), \tag{5}$$

where $\boldsymbol{p} = (p_1, \dots, p_n)$ are the degrees. At first, this may look a bit intimidating due the Kronecker products. It may help to think about the computation as picking a single univariate basis function for each variable, and then multiplying them together. The Kronecker products help us do this for all possible combinations of univariate basis functions, and stacks the results in the vector $\boldsymbol{B}_{\boldsymbol{p}}$.

A B-spline $f : \mathbb{R}^m \to \mathbb{R}^n$ in the variables $\boldsymbol{x} \in \mathbb{R}^m$ is then expressed similarly as in the univariate case, as

$$f(\boldsymbol{x}) = \sum_{i=0}^{L-1} P_i \boldsymbol{B}_{i,\boldsymbol{p}}(\boldsymbol{x}), \tag{6}$$

where $\{B_{i,\boldsymbol{p}}\}_{i=0}^{L-1}$ are the multivariate basis functions. The total number of combinations produced by the Kronecker products, is

$$L = \prod_{k=1}^{m} l_k. \tag{7}$$

These basis functions are defined on a rectilinear grid of congruent boxes (n-orthotopes). For the 1-D case the grid consists of intervals, for the 2-D case it consists of rectangles, for the 3-D case it consists of boxes, and so on.

Again, we may write the B-spline in a compact form as

$$f(\boldsymbol{x}) = \boldsymbol{P}\boldsymbol{B}_{\boldsymbol{p}}(\boldsymbol{x}), \tag{8}$$

where $\boldsymbol{B}_{\boldsymbol{p}}$ is a vector of multivariate basis functions, and the control points $\boldsymbol{P} \in \mathbb{R}^{n \times L}$.

3 Function approximation with splines

SPLINTER can be used for multivariate function approximation with splines. This section gives a quick exposition of the mathematics behind function approximation with splines.

3.1 Sampling

Consider the set of samples

$$D = \{(\boldsymbol{x}^i, \boldsymbol{y}^i)\}_{i=1}^N \tag{9}$$

where $\boldsymbol{y}^i = g(\boldsymbol{x}^i) + \boldsymbol{\epsilon}$ for some (typically unknown) function $g : \mathbb{R}^m \to \mathbb{R}^n$. The variables ϵ are stochastic and represent uncertainties such as noise. Usually, the stochastic variables are assumed to be normally distributed, i.e. $\epsilon_i \sim \mathcal{N}(0, \sigma^2), i = 1, \ldots, n$. And for many practical applications, they are simply ignored.

TODO: Write about sampling on a regular grid versus scattered samples. Show figure.

3.2 Approximation

The unknown function g can be approximated by a B-spline

$$f(\boldsymbol{x}) = \boldsymbol{P}\boldsymbol{B}_{\boldsymbol{p}}(\boldsymbol{x}), \tag{10}$$

by computing control points $\boldsymbol{P}$ so that $f \approx g$. This computation can be formulated as an optimization problem

$$\begin{aligned} \boldsymbol{P}^\star &= \arg\min_{\boldsymbol{P}} \sum_{i=1}^N ||g(\boldsymbol{x}^i) - f(\boldsymbol{x}^i)||_2^2 \\ &= \arg\min_{\boldsymbol{P}} \sum_{i=1}^N ||\boldsymbol{y}^i - \boldsymbol{P}\boldsymbol{B}_{\boldsymbol{p}}(\boldsymbol{x}^i)||_2^2, \end{aligned} \tag{11}$$

where we have denoted the optimal control points $\boldsymbol{P}^\star$.

An important property of f is that it is linear in the control points $\boldsymbol{P}$. This helps us a lot since it simplifies the problem of finding $\boldsymbol{P}^\star$ to solving a linear system of equations. To arrive at this solution requires some manipulation of the above problem.

Let us start by creating some big matrices to get rid of the summation. We stack the $\boldsymbol{y}^i$ values in a matrix $Y \in \mathbb{R}^{N \times n}$, so that a row i of Y corresponds to $\boldsymbol{y}^i$ transposed. We also stack the evaluated basis functions in a matrix $X \in \mathbb{R}^{N \times L}$ so that row i corresponds to $\boldsymbol{B}_{\boldsymbol{p}}(\boldsymbol{x}^i)$ transposed.

Now, we may formulate the optimization problem as follows

$$\boldsymbol{P}^\star = \arg\min_{\boldsymbol{P}} ||Y - X\boldsymbol{P}^\mathsf{T}||_F^2, \tag{12}$$

where we now utilize the Frobenius norm since we are dealing with matrices. Skipping some details, it can be shown that if we differentiate this objective function with respect to $\boldsymbol{P}$, and set it equal to zero, we obtain the equations

$$X^{\mathsf{T}} X \boldsymbol{P}^{\mathsf{T}} = X^{\mathsf{T}} Y. \tag{13}$$

This is a linear system of equations which we may solve for $\boldsymbol{P}^{\star}$ using a linear solver. It may not be evident from the equations, but the matrix $X^{\mathsf{T}} X$ is very sparse since most of the B-spline basis functions are zero at a sample point $\boldsymbol{x}^i$. We may utilize this structure to speed up solve times by applying linear solvers specialized for sparse systems, such as the SparseLU solver in Eigen.

Before considering more advanced cases, let us think a moment about the above optimization problem and what it actually solves. We know that the problem is linear, and thus, there are three possible outcomes when solving for the control points.

1. $X^{\mathsf{T}} X$ has full rank and by solving for $\boldsymbol{P}^{\star}$ we will interpolate all sample points. If the sample points lie on a regular grid, we can always obtain this case by selecting appropriate knot sequences, so that we have enough basis functions/control points to interpolate all points.

2. The system is under-determined. There are several solutions.

3. The system is over-determined. There may be no solution.

3.3 Regularization

Regularization may improve the conditioning of the linear system and provide smoothing effect. Regularization can be incorparated by adding a term to the objective function

$$\boldsymbol{P}^{\star} = \arg\min_{\boldsymbol{P}} ||Y - X\boldsymbol{P}^{\mathsf{T}}||_F^2 + \alpha ||R\boldsymbol{P}^{\mathsf{T}}||_F^2, \tag{14}$$

where $R \in \mathbb{R}^{L \times L}$ is a regularization matrix. This optimization problem results in the equations

$$(X^{\mathsf{T}} X + \alpha R)\boldsymbol{P}^{\mathsf{T}} = X^{\mathsf{T}} Y. \tag{15}$$

SPLINTER currently supports $R = I$, and the second-order finite difference matrix discussed next.

3.3.1 The P-spline

A special type of regularization matrix is the second-order finite difference matrix denoted Δ^2. The matrix is constructed so that the product $\Delta^2 \boldsymbol{P}^{\mathsf{T}}$ consists of second-order finite differences of the control points: $P_i - 2P_{i-1} +$

P_{i-2}. When a B-spline is fitted with this regularization matrix, the second-order derivative (or an approximation of it) is minimized, and a smooth B-spline results. This B-spline is so nice that it was given its own name by its inventors Eilers and Marx [1]. The name is P-spline, since it is a penalized spline.

The control points of a P-spline results from the above equation by setting $R = \Delta^2$.

On a side note: The creation of Δ^2 for the multivariate case demands in return that a couple of your hairs turns white. Luckily, Patrick Robertson, a contributor to the SPLINTER library, already made that sacrifice for the greater good. You should send him a happy thought whenever you build a P-spline with SPLINTER.

3.4 Weighted least squares

Finally, we turn to the last extention of the optimization problem, which involves putting weights to the sample points. This is useful for problems in which some sample points are more “trustworthy” than others. For example, if each sample point represents multiple observation it is natural to express it as the mean value, accompanied by a variance or standard deviation. This variance may then be used weight the sample point, with more weight on samples with low variance, and lower weight on samples with high variance.

In SPLINTER, it is possible to assign to each sample a weight, to give $\{w_i\}_{i=1}^N$ weights. We put these weights on the diagonal of a matrix $W = \mathrm{diag}(w_1, \dots, w_N)$ and, as in weighted linear least squares, we multiply this with the sample points as follows

$$\boldsymbol{P}^\star = \arg\min_{\boldsymbol{P}} ||WY - WX\boldsymbol{P}^\mathsf{T}||_F^2 + \alpha||R\boldsymbol{P}^\mathsf{T}||_F^2. \tag{16}$$

Solving this yields the linear system

$$(X^\mathsf{T}WX + \alpha R)\boldsymbol{P}^\mathsf{T} = X^\mathsf{T}WY, \tag{17}$$

which we may solve for $\boldsymbol{P}^\star$.

As suggested above, the variance σ_i^2 of a sample point i can be used to weight the sample. A commonly used approach is to set $w_i = 1/\sigma_i^2$. Note that correlation between sample points is not supported in SPLINTER, since we only consider a diagonal matrix.

4 Interfacing SPLINTER

SPLINTER can be interfaced through its native C++ interface, or via interfaces to C and Python.

Some brief descriptions of the interfaces are given below. To see examples of their use please consult the code examples at `https://github.com/bgrimstad/splinter`.

4.1 C++ interface

4.2 C interface

4.3 Python interface

References

[1] Paul H. Eilers and Brian D. Marx. Flexible Smoothing with B-splines and Penalties. *Statistical Science*, 11(2):89–102, 1996.

[2] Bjarne Grimstad et al. SPLINTER: A library for multivariate function approximation with splines. `https://github.com/bgrimstad/splinter`, 2012. Accessed: 2015-05-16.

[3] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. `https://eigen.tuxfamily.org`, 2010.

[4] Niels Lohmann. Json for modern c++. `https://github.com/nlohmann/json`, 2016.