Smoothing Splines

Robert Dahl Jacobsen

February 5, 2022

Here I document the math used in the Julia package *SmoothSpline* that performs regression with B-splines using a Tikhonov regularisation.

The *SmoothSpline* package is a port of the *smooth.spline* function from R. During the implementation I discovered two special things in *smooth.spline*'s implementation that I have not seen in its documentation:

- Two matrix traces are computed. In the code the first two and last two entries on the diagonal are *not* included in the trace.
- In one computation $\frac{1}{3}$ is hard-coded to be 0.333. This approximation is a bit crude when the word size of the computer is 64 bit.

I will comment more on these points later in this document.

1 B-splines

My source for B-splines is the excellent book [3] with very detailed algorithms. The only downside for a Julia implementation is that all algorithms use 0-based arrays. I do not include further details about about evaluation of B-splines here, but note that we have algorithms for computing the values of B-splines and all of their derivatives.

In *SmoothSpline* we use cubic B-splines, so the general degree p from [3] is always $p = 3$. We therefore use a more compact notation here: The i'th spline of degree p is in [3] denoted $N_{i,p}$, but is here simply denoted N_i. However, to avoid "magic" occurences of 3, I still use p in the following to denote the degree.

A collection of B-splines are determined solely by their knots. Boundaries are handled by reusing the boundary knots: If we have m distinct breakpoints $u_1, \dots, u_m$ we construct the B-splines from the knots, where we have augmented by p endpoints in each end (u_1 and u_m repeated p times each).

The function *bs* from the {splines} package in R can be used to compute B-splines.

2 Regression with B-splines

We consider observations (x_i, y_i) for $i = 1, \dots, n$. It is allowed to have multiple observations with similar x values (and y values). Observation i has an associated weight w_i, that by default is 1. Observations with similar x values are grouped by summing their weights and taking the average of thier y values. In the following we assume that all x values are distinct and that the observations are sorted by their x values.

The choice of B-splines (that is, the choice of their knots) of course influences everything. In *smooth.spline* the number of internal knots m is determined from the number of observations n. Let

$$a_1 = \log_2(50), \quad a_2 = \log_2(100), \quad a_3 = \log_2(140), \quad a_4 = \log_2(200)$$

and

$$m' = \begin{cases} n, & n < 50, \\ 2^{a_1 + (a_2 - a_1)(n-50)/150}, & 50 \leq n < 200, \\ 2^{a_2 + (a_3 - a_2)(n-200)/600}, & 200 \leq n < 800, \\ 2^{a_3 + (a_4 - a_3)(n-800)/2400}, & 800 \leq n < 3200, \\ 200 + (n - 3200)^{0.2}, & n \geq 3200. \end{cases}$$

Then $m = [m']$. The breakpoints $u_1, \dots, u_m$ are computed from the observations in the following way. Define $j : \{1, \dots, m\} \to \{1, \dots, n\}$ by

$$j(i) = \left\lfloor \frac{n-1}{m-1}(i-1) + 1 \right\rfloor.$$

That is, the indices are equally distanced before being rounded. Then

$$u_i = x_{j(i)}, \quad 1 \leq i \leq m.$$

We want to perform regression with m B-splines on the interval $[x_1, x_n]$. That is, compute an approximation with the function

$$f(x) = \sum_{j=1}^{M} \beta_j N_j(x).$$

The design matrix of the regression problem is the matrix $\boldsymbol{X}$ of size $n \times m$ with entries

$$X_{i,j} = N_j(u_i).$$

The weights are collected in a diagonal matrix $\boldsymbol{W} = \operatorname{diag}(w_i, i = 1, \dots, n)$. To enforce a smoother interpolant we choose the β's with a least squares criterion and limit the curvature (measured by the second derivative):

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\sqrt{\boldsymbol{W}}\boldsymbol{\beta}\|^2 + \lambda \int_{x_1}^{x_n} \left\{ f^{(2)}(t) \right\}^2 dt.$$

We can also write this as a Tikhonov regularisation

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\sqrt{\boldsymbol{W}}\boldsymbol{\beta}\| + \lambda\boldsymbol{\beta}^T\boldsymbol{\Sigma}\boldsymbol{\beta},$$

where the regularisation term $\boldsymbol{\Sigma}$ is the Gram matrix of size $m \times m$ with entries

$$\Sigma_{i,j} = \int_{x_1}^{x_n} N_i^{(2)}(t)N_j^{(2)}(t)\,dt.$$

For each value of $\lambda > 0$, we have a solution $\boldsymbol{\beta}$ to the corresponding regression problem, namely the solution to the equation

$$\boldsymbol{A}\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X} + \lambda\boldsymbol{\Sigma}\right)\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{y} = \widetilde{\boldsymbol{y}}. \tag{1}$$

This can be solved fast with a few tricks. Both $\boldsymbol{X}$ and $\boldsymbol{\Sigma}$ are *banded* with lower and upper band p. That is, $\boldsymbol{X}_{i,j} = \boldsymbol{\Sigma}_{i,j} = 0$ when $|i-j| > p$. This implies that the Cholesky factorization of $\boldsymbol{A}$ is also banded with the same band. More specifically, we let $\boldsymbol{C}$ denote the upper triangular matrix of the Cholesky factorization such that

$$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X} + \lambda\boldsymbol{\Sigma}.$$

With this convention $\boldsymbol{C}^T$ is a lower triangular matrix and we can the use the classic forward and backward substitution to solve eq. (1): First solve $\boldsymbol{C}^T\boldsymbol{z} = \widetilde{\boldsymbol{y}}$ and then $\boldsymbol{C}\boldsymbol{\beta} = \boldsymbol{z}$.

Note that the matrix $\boldsymbol{A}$ may be ill-conditioned even with the Tikhonov term. In fact, I have seen substantial differences between $\boldsymbol{\beta}$'s computed with generic solvers and the special solvers used in *SmoothSpline*.

2.1 Computing matrices

Before solving eq. (1) we must compute the matrices involved.

In the *documentation* for *smooth.spline* (and therefore not under the same license as the *code*) it is noted that the Lagrange multiplier λ is data dependent. That is, an appropriate value for λ depends on the values of the observations. This makes it more delicate to choose λ.

But *smooth.spline* aids the user by offering a data independent parameter "spar", where $0 < \text{spar} \leq 1$, that is mapped to an appropriate λ. The map is

$$\lambda = r \cdot 256^{3\cdot\text{spar}-1}, \quad r = \frac{\operatorname{tr}(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})}{\operatorname{tr}(\boldsymbol{\Sigma})}.$$

In the code, however, it is not the standard traces that are computed – both in *SmoothSpline* and *smooth.spline*. The first two entries and the last two entries of the diagonals are discarded in the sum. I have not found documentation to explain this, but I *suspect* that it is to reduce the impact of the endpoint knots that occur in multiple splines – cf. section 1.

Furthermore, *smooth.spline* rescales the x values to the closed unit interval. This has an impact on $\boldsymbol{\Sigma}$, but the impact on $\boldsymbol{A}$ is de facto alleviated by the above mapping.

2.1.1 Design matrix

The design matrix to compute is straightforward once we know how to compute the splines.

2.1.2 Gram matrix

We have closed-form expressions for the Tikhonov matrix $\mathbf{\Sigma}$. All spline functions are supported on (a subset of) $[u_0, u_m]$ and we first note that

$$\Sigma_{i,j} = \int_{u_0}^{u_m} N_i^{(2)}(t) N_j^{(2)}(t)\, dt = \sum_{k=0}^{m-1} \int_{u_k}^{u_{k+1}} N_i^{(2)}(t) N_j^{(2)}(t)\, dt. \tag{2}$$

With partial integration each of the integrals can be computed as

$$\int_{u_k}^{u_{k+1}} N_i^{(2)}(t) N_j^{(2)}(t)\, dt = \Big[N_i'(t) N_j^{(2)}(t) \Big]_{u_k}^{u_{k+1}} - \int_{u_k}^{u_{k+1}} N_i'(t) N_j^{(3)}(t)\, dt.$$

On each interval $[u_k, u_{k+1}]$, every spline is a polynomial of degree at most three. Hence the third derivative is a constant and

$$\int_{u_k}^{u_{k+1}} N_i'(t) N_j^{(3)}(t)\, dt = N_j^{(3)}(u_k) \big(N_i(u_{k+1}) - N_i(u_k) \big).$$

Substituting all this into eq. (2) we note a telescoping sum and arrive at an expression:

$$\Sigma_{i,j} = N_i'(u_m) N_j^{(2)}(u_m) - N_i'(u_0) N_j^{(2)}(u_0) - \sum_{k=0}^{m-1} N_j^{(3)}(u_k) \big(N_i(u_{k+1}) - N_i(u_k) \big).$$

In *smooth.spline* the integral in eq. (2) is computed differently: On each interval $[u_k, u_{k+1}]$, the function $N_i^{(2)}$ is a polynomial of degree at most one. In the code, this polynomial is parameterized as

$$N_i^{(2)}(t) = a_{i,k} + b_{i,k}(t - u_k), \quad u_k \le t \le u_{k+1}.$$

Expanding parentheses in $N_i^{(2)} N_j^{(2)}$ we see that

$$N_i^{(2)}(t) N_j^{(2)}(t) = a_{i,k} a_{j,k} + \big(a_{i,k} b_{j,k} + a_{j,k} b_{i,k} \big)(t - u_k) + b_{i,k} b_{j,k} (t - u_k)^2.$$

With this expression and $\Delta_k = u_{k+1} - u_k$,

$$\begin{aligned}
&\int_{u_k}^{u_{k+1}} N_i^{(2)}(t) N_j^{(2)}(t)\, dt \\
&\quad = \int_0^{\Delta_k} a_{i,k} a_{j,k} + \big(a_{i,k} b_{j,k} + a_{j,k} b_{i,k} \big) s + b_{i,k} b_{j,k} s^2\, ds \\
&\quad = a_{i,k} a_{j,k} \Delta_k + \big(a_{i,k} b_{j,k} + a_{j,k} b_{i,k} \big) \frac{1}{2} \Delta_k^2 + b_{i,k} b_{j,k} \frac{1}{3} \Delta_k^3.
\end{aligned} \tag{3}$$

The slope $b_{i,k} = N_i^{(3)}(t)$, but *smooth.spline* only use the second order derivatives. We see readily that $a_{i,k} = N_i^{(2)}(u_k)$. Let $\Delta y_{i,k} = N_i^{(2)}(u_{k+1}) - N_i^{(2)}(u_k)$. Then $b_{i,k} = \Delta y_{i,k}/\Delta_k$. With these expressions, many of the Δ_k's can be removed from eq. (3):

$$\begin{aligned}
&\int_{u_k}^{u_{k+1}} N_i^{(2)}(t) N_j^{(2)}(t)\, dt \\
&\qquad = a_{i,k} a_{j,k} \Delta_k + \Big(a_{i,k} \frac{\Delta y_{j,k}}{\Delta_k} + a_{j,k} \frac{\Delta y_{i,k}}{\Delta_k} \Big) \frac{1}{2} \Delta_k^2 + \frac{\Delta y_{i,k}}{\Delta_k} \frac{\Delta y_{j,k}}{\Delta_k} \frac{1}{3} \Delta_k^3 \\
&\qquad = a_{i,k} a_{j,k} \Delta_k + \big(a_{i,k} \Delta y_{j,k} + a_{j,k} \Delta y_{i,k} \big) \frac{1}{2} \Delta_k + \Delta y_{i,k} \Delta y_{j,k} \frac{1}{3} \Delta_k.
\end{aligned}$$

In *smooth.spline* the value of $\frac{1}{3}$ is hard-coded to be 0.333. Even in small examples this can result in entry-wise deviations of about 1%.

Speed-ups From the definition we readily see that $\mathbf{\Sigma}$ is symmetric. We know from [3, P2.1] that the support of N_i is $[u_i, u_{i+p+1}]$ and therefore

$$\operatorname{supp} N_i \cap \operatorname{supp} N_j = \emptyset \quad \text{if } |i-j| > p.$$

This implies that $\Sigma_{i,j} = 0$ if $|i-j| > p$ and that the sum in eq. (2) goes from $\min\{i,j\}$ to $\min\{\max\{i,j\} + p + 1, m\}$.

2.2 Banded matrices

A banded matrix can be represented and stored efficiently [2, Section 1.2.5]. Furthermore, there are efficient algorithms for computing the Cholesky factorization of a banded matrix [2, Section 4.3].

In R, *smooth.spline* rely on the Fortran routines *dpbfa* and *dpbsl* from Linpack [1] to compute the Cholesky factorization and solving eq. (1), respectively.

References

[1] J. J. Dongarra et al. *LINPACK Users' Guide*. SIAM, 1979. DOI: 10.1137/1.9781611971811.

[2] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. 4th ed. Johns Hopkins University Press, 2013.

[3] Les Piegl and Wayne Tiller. *The NURBS Book*. 2nd ed. Springer-Verlag Berlin Heidelberg, 1997. DOI: 10.1007/978-3-642-59223-2.