PSQR: A Stable and Efficient Penalized Spline Algorithm

Christopher P. Calderon *
Department of Computational & Applied Mathematics, Rice University

Josue G. Martinez
Department of Statistics, Texas A&M University

Raymond J. Carroll
Department of Statistics, Texas A&M University and

Danny C. Sorensen
Department of Computational & Applied Mathematics, Rice University

May 1, 2009

Abstract

We introduce an algorithm for reliably computing quantities associated with several types of semiparametric mixed models in situations where the condition number on the random effects matrix is large. The algorithm is numerically stable and efficient. It was designed to process penalized spline (P-spline) models without making unnecessary numerical approximations. The algorithm, PSQR (P-splines via QR), is formulated in terms of QR decompositions. PSQR can treat both exactly rank deficient and ill-conditioned matrices. The latter situation often arises in large scale mixed models and/or when a P-spline is estimated using a basis with poor numerical properties, e.g. a truncated power function (TPF) basis. We provide concrete examples where unnecessary numerical approximations introduce both subtle and dramatic errors that would likely go undetected, thus demonstrating the importance of using this reliable numerical algorithm. Simulation results studying a univariate function and a longitudinal data set are used to demonstrate the algorithm. Extensions and the utility of the method in more general semiparametric regression applications are briefly discussed. MATLAB scripts demonstrating implementation are provided in the Supplemental Materials.

Keywords: Demmler-Reinsch Basis, Mixed Models, P-Spline, Semiparametric Regression

*Calderon’s work was funded by NIH grant T90 DK070121-04. Martinez was supported by a postdoctoral training grant (CA90301). Sorensen was supported partially by AFOSR grant FA9550-09-1-0225 and by NSF grants CCF-0634902. Carroll’s research was supported by a grant from the National Cancer Institute (CA57030) and by Award Number KUS-CI-016-04, made by King Abdullah University of Science and Technology (KAUST).
1 Introduction

We introduce a numerically stable and efficient algorithm for constructing penalized spline (P-spline) models minimizing a quadratic penalty without making unnecessary numerical approximations. Several P-spline applications require the numerical solution of many least squares problems; let these different solutions be indexed by $\alpha$, i.e. $(\hat{\beta}_\alpha, \hat{u}_\alpha) \equiv \arg\min_{(\beta, u)} \| y - (X\beta + Zu) \|_2^2 + \alpha \| DP u \|_2^2$, where $y$ is a vector containing noisy function samples, $\alpha$ is a given smoothing parameter, $D_P$ is a penalty matrix, $X$ the fixed effects matrix, and $Z$ the random effects matrix. Many different $\alpha$ values are proposed in an attempt to find the “best” smoothing parameter, $\hat{\alpha}$, using some criterion, e.g. GCV (1). P-splines are useful in part because they can parsimoniously represent curves using ideas coming from low-rank regression splines and because they can be formulated in terms of linear mixed models (as done above). The latter feature facilitates the formulation of more complex semiparametric models (2), (3). However, existing algorithms can encounter problems either with efficiency and/or numerical stability when the the condition number of $Z$ is large (4), (5).

Our algorithm reliably computes various quantities associated with a given $\alpha > 0$ if the proposed $Z$ is ill-conditioned or even exactly rank-deficient. The numerical stability is attractive in situations where the user has little or no control over the design matrices $X$ and $Z$ used in the P-spline problem. The algorithm was motivated by the need to accurately and efficiently compute with ill-conditioned $Z$ associated with a particular class of P-spline problems using the truncated power function (TPF) basis (1). When the algorithm is tailored to the TPF basis, we demonstrate that it is extremely efficient because solutions to the collection of least squares problems can be computed in a vectorized fashion for a large grid of $\alpha$’s. In addition, a variety of other statistical quantities such as the residual sum of squares and the GCV can also be computed in a vectorized fashion. The algorithm also shows promise in complex random effects models where nonlinear subject specific curves need to be estimated. Possible high throughput applications include analyzing mass spectrometry signals containing low frequency baseline contamination (6) and longitudinal modeling (3).

Concrete examples illustrating the utility of our algorithm in the situations described above are provided. We also provide examples illustrating how seemingly innocuous numerical approximations can introduce both subtle and dramatic errors in the P-spline fit and these errors would likely go unnoticed by many practitioners. As the size of the data and complexity of P-spline
models considered increases, the types of issues numerical addressed here will likely become more relevant.

The remainder of the article is organized as follows: Section 2 introduces the basic notation, reviews established P-splines results, and highlights the connection to mixed models (1). Section 3 presents a new numerical algorithm tailored to dealing with generic P-spline problems. We also provide some explicit formulas for computing quantities commonly needed in P-spline applications using quantities computed as a byproduct of this algorithm. Section 4 presents benchmark simulation results. The last application presented is used to highlight the dangers of introducing unnecessary numerical approximations. Section 5 presents the conclusion and outlook. MATLAB codes for fitting general P-splines with our algorithm are provided in the Supporting Material.

## 2 Background and Notation

Regression splines are utilized heavily in scatterplot smoothing and semiparametric regression (1). The ability to parsimoniously represent nonlinear curves makes regression splines attractive. Below we provide some basic features and notation used in regression splines.

### 2.1 Basic Regression Splines

The basic regression problem considered here is to approximate a continuous (at least once differentiable) nonlinear function, \( f(\cdot) \) using noisy discrete noisy measurements, \( y_i = f(x_i) + \epsilon_i \). Most P-spline approximations take the form

\[
y_i \equiv \eta_0 + \eta_1 x_i \ldots \eta_p x_i^p + \sum_{j=1}^{K} \zeta_j B_j(x_i),
\]

where the \( B_j(\cdot) \) represent spline basis functions. This approximation serves as the building block for several more complicated semiparametric models (1). We introduce the following notation:

\[
Z \equiv [B_1, \ldots, B_K]^T, u \equiv [\zeta_1, \ldots, \zeta_K]^T, \beta \equiv [\eta_0, \ldots, \eta_p]^T, y \equiv [y_1, \ldots, y_m]^T, X_i = [1, \ldots, x_i^p], X \equiv [X_1^T, \ldots, X_m^T]^T.
\]

With this notation we can rewrite Equation (1) as

\[
y = X\beta^a + Zu = C\beta^a,
\]

where \( C = [X, Z] \) and \( \beta^a = [\beta^T, u^T]^T \). The goal is to determine the \( \beta^a \) that minimizes

\[
(y - C\beta^a)^\top (y - C\beta^a) + \alpha(\beta^a)^\top D^\top D\beta^a.
\]
Let the solution be denoted $\hat{\beta}_a^\alpha$ for the given penalty matrix $D$ and $\alpha$.

### 2.2 Regularized Least Squares

We need to solve many LS problems in the following optimization problem:

$$\hat{\alpha} \equiv \arg\min_{\alpha} C(\hat{\beta}_a^\alpha),$$

where $C(\cdot)$ is some criterion function used to impose smoothness in the P-spline, e.g. GCV, AIC, AICc, etc.

The Demmler-Reinsch (DR) orthogonalization idea can, in principle, help in computations for efficiently selecting $\hat{\alpha}$ (1). The method is attractive because it only requires one to numerically process the full design matrix $C \equiv [X, Z] \in \mathbb{R}^{m \times n}$ once. After this often computationally costly step, one can then cheaply evaluate scalar functions like the residual mean square function, $\text{RSS}(\alpha) = \|y - \hat{y}\|^2_2$ or the degrees of freedom, $d_{\text{fit}}(\alpha) = \text{trace}\{C(C^T C + \alpha D)^{-1}C^T\}$, of the fit function. However numerical issues arise because the design matrix $C$ is often ill-conditioned. This is especially true for the TPF basis.

Often, ad hoc remedies have been introduced to overcome this ill-conditioning problem (4),(5) while retaining the computational advantages of DR type decompositions. The introduction of additional tunable parameters used only for computational purposes also occurs in other mixed model situations (4). However, the ramifications of these ad hoc modifications can be difficult to determine. As the size and complexity of semiparametric models grow, the subtle errors introduced can propagate in complicated and unknown ways. In Section 4.1 and 4.2 we will provide two concrete illustrative examples.

Given that P-splines using TPF basis have proven to be effective in various applications (3), (7), it would be useful to introduce accurate and efficient algorithms that avoid unnecessary approximations introduced only for computational purposes. It is well-known that QR methods can help one avoid such approximations (1),(4), (8), but to our knowledge a fast, reliable and efficient version that utilizes the advantages of both DR techniques and the numerically attractive features of the QR decomposition are currently lacking. Recall that $m,n$ are used denote the number of rows and columns, respectively, of our design matrix and $K$ denotes the number of spline basis functions. The algorithm we introduce is attractive in the case where $K << m$. 

4
It also helps reduce the computational load if $K << n$, e.g., only a small number of random effects are penalized. A longitudinal, subject specific curve simulation study gives a concrete example where this situation can occur. The algorithm outlined in the next section can process an arbitrary design matrix $C$. Rank degenerate cases (numerical and exact) are readily handled without making unnecessary approximations or introducing a cut-off criterion.

The proposed algorithm exploits another advantage of the mixed model interpretation of semi-parametric regression. That is, one can readily construct penalty matrices that concentrate either on random or fixed-effects. Although the penalty matrix $D$ acts on the entire parameter vector $\beta^a$ and is allowed to be exactly rank deficient, we can readily construct a symmetric positive definite matrix using sub-blocks of $D$. This matrix can be used to regularize some ill-conditioned, or exactly rank deficient, columns of $Z$. This issue is discussed further in next section. Although the core of the algorithm proposed can handle a fairly wide class of permissible penalty $D$’s, we tailor the algorithm to the fairly versatile P-spline structure discussed in (5), where $D$ is a diagonal matrix containing $0$’s in the columns corresponding to the fixed and unpenalized random effects and $1$’s in the columns corresponding to penalized random effects.

### 2.3 Motivation for Using the TPF Basis

The TPF basis is sometimes considered unattractive in applications because it often results in an ill-conditioned design matrix (9). The high condition number complicates using fast computational methods in P-spline problems (5). However, the TPF basis does have some redeeming qualities. It is attractive in part because each estimated regression coefficient corresponds to a single knot (which corresponds to a single basis function). This is advantageous because the basis can be made adaptive. For example one can easily increase, decrease or change the spacing of the knots (10). These types of changes correspond to QR insertions, deletions, or modifications (respectively) and can be computationally exploited in the algorithm we propose. Of course, the same comments apply to adding new observations to the P-spline design matrix. These computationally attractive features are somewhat cumbersome to obtain with other spline basis, e.g., B-splines. In addition, derivative estimation is trivial with the TPF basis regardless of the knot spacing used (1).
3 PSQR Overview

As discussed in the Introduction, we exploit the mixed model structure and partition the random effects into $P$ ("penalized") and $F$ ("free") columns. A key feature of the PSQR algorithm is its ability to exploit this partitioning to gain computational efficiency when solutions associated with several $\alpha$ and/or $D^P$ are required. A superscript $F$ and $P$ is used on both vectors and matrices, e.g. $u = [u^F, u^P]$ or $Z = [Z^F, Z^P]$. The total number of columns corresponding to the $P$ entries is denoted by $n^P$. In standard P-spline applications $n^P = K$, but examples where $n^P > K$ are studied in this paper. The fixed effects are not penalized in our formulation. However this case can readily be accounted for by relabelling the fixed effect as a random effect. Note that the partitioning allows us to see that the penalty term $(\beta^a)^\top D^T D \beta^a$ appearing in Equation (3) is equivalent to $(u^p)^\top (D^P)^\top D^P u^p$.

Before providing the algorithmic details, we state in fairly simple terms what the algorithm accomplishes. PSQR performs a series of matrix operations that result in output that can be used to solve the quadratic programming problem in augmented form using structured QR factorizations. The QR factors allow us to easily find the $\hat{\beta}^a$ minimizes

$$\left\| \begin{pmatrix} C & \sqrt{\alpha} D \end{pmatrix} \beta^a - \begin{pmatrix} y \\ 0 \end{pmatrix} \right\|_2^2.$$ 

It is well-known in the P-spline literature, see Appendix B of Ref. (1), that this is the proper numerical formulation. However, we are unaware of efficient and stable P-spline methods for computing this solution for multiple candidate $\alpha$ or $D^P$'s that do not introduce arbitrary cut-off criteria. Our numerical solution computes the expensive QR factorization of $C$ only once, without pivoting. After this, the algorithm only operates on matrices containing $O(n^P)$ rows and columns. This facilitates processing multiple candidate $\alpha$ and $D^P$'s. The specific details of the general algorithm are as follows:

The PSQR Algorithm:
Matrix Factorization Steps

1. Obtain the QR decomposition of \( C = QR \).

2. Partition result above as: \( QR = (Q^F, Q^P) \begin{pmatrix} R_{11}^F & R_{12}^F \\ 0 & R_{22}^P \end{pmatrix} \).

3. Obtain the SVD of \( R_{22}^P = USV^T \).

4. Form the following:
   \[
   \tilde{Q} = (Q^F, Q^P U), \quad \tilde{V} = \begin{pmatrix} I & 0 \\ 0 & V^T \end{pmatrix},
   \]
   \[
   \tilde{R} = \begin{pmatrix} R_{11}^F & R_{12} V \\ 0 & S \end{pmatrix} \equiv \begin{pmatrix} \tilde{R}_{11} & \tilde{R}_{12} \\ 0 & \tilde{R}_{22} \end{pmatrix},
   \]
   \[
   b = \begin{pmatrix} \tilde{Q}^T y \\ 0 \end{pmatrix} \equiv \begin{bmatrix} b^F \\ b^P \end{bmatrix}.
   \]

The steps above only need to be carried out once for a given design matrix \( C \). The computational cost of these operations is \( O(mn^2) \) flops (8) and provide the factorization \( C = \tilde{Q}\tilde{R}\tilde{V} \). Subsequently one can propose a sequence of candidate smoothing parameters, \( \alpha \) and/or \( D^P \), and obtain the corresponding least squares solution by repeating the steps below.
**Solution Steps: Repeat for each regularization/smoothing parameter**

5. For each given $\alpha$ (and/or $D_P$) form: $\tilde{D}_\alpha = \sqrt{\alpha} D_P V$ and $\tilde{W}_\alpha = \left( \begin{array}{c} S \\ \tilde{D}_\alpha \end{array} \right)$.

6. Obtain the QR decomposition $\tilde{W}_\alpha = Q'R'$.

7. Form $c = (R')^{-1}(Q')^\top \left( \begin{array}{c} b_P \\ 0 \end{array} \right)$.

8. Solve $\hat{\beta}_\alpha^a = \left( \tilde{R}^{-1}_{11}(b^F - \tilde{R}_{12}c) \right)$.

In the solution steps, the QR decomposition for $\tilde{W}_\alpha$ in Step 6, costing up to $O((n_P)^3)$ flops, will dominate the computational load in most P-spline applications. In the special case $D_P = I$, which we highlight later, we can reduce the cost of Steps 5-7 to $O(n_P)$. Recall that in typical P-spline applications $n_P$ is much smaller than the number of observations, i.e. rows of $C$, so the cost of Steps 5-8 is small relative to that of Steps 1-4 regardless of the structure of $D_P$.

In order to better understand the PSQR algorithm, note that

$$\begin{pmatrix} SV^\top \\ D_P \end{pmatrix} = \begin{pmatrix} S \\ D_P V \end{pmatrix} V^\top \equiv \begin{pmatrix} \tilde{R}_{22} \\ D_P V \end{pmatrix} V^\top,$$

and when solving for the $n_P$ penalized terms, we carry out computations in a coordinate system where $\tilde{R}_{22}$ is diagonal and convert back to the standard Cartesian basis in the final step. A meaningful solution of the PSQR algorithm requires $R_{11}^F$ to be full rank. Ill-conditioning can readily be checked in the unpenalized terms by computing the condition number of $R_{11}^F$. If all terms are penalized, the algorithm is guaranteed to have a meaningful mathematical solution for any $\alpha > 0$ and positive definite $(D_P)^\top D_P$. In numerical implementation, one would need to ensure that when $\tilde{D}_\alpha$ is appended to $S$ in Step 5 that the condition number is such that numerically meaningful results can be obtained with the the machine precision available. Note that the diagonal nature of $\tilde{R}_{22}$ (i.e. $S$) allows one to easily control the lowest singular value associated with the regularized least squares problem. This can be done by finding the columns corresponding to the smallest computed singular values of $S$ and then choosing the corresponding
columns from $D^P V$. With these columns, form a new matrix and compute the singular values. For a fixed $D^P$ the SVD just mentioned only needs to be done once; $\sqrt{\alpha}$ multiplied by the smallest of these singular values provides a lower bound of the smallest singular value associated with the penalized regression problem. We recommend letting the machine precision limit the range of $\alpha$ that can be used and to not combine penalized regression problems with singular value truncation as is sometimes done (4).

The PSQR algorithm can be used by semiparametric regression schemes seeking least squares solutions using any valid penalty matrix, but was designed to achieve efficient DR type computations in the case that $D^P = I$. This choice of $D^P$ is popular when one uses a TPF basis in regression splines (1). The $F/P$ partitioning allows us to solve the least squares problem using two different QR factorizations. In the $D^P = I$ case, we can efficiently carry out the second QR factorization and matrix inversion executed in Steps 6-7. To see this, note that these steps aim at finding the vector $x$ that minimizes

$$\left\| \begin{pmatrix} S \\ \sqrt{\alpha} V \end{pmatrix} x - \begin{pmatrix} b^P \\ 0 \end{pmatrix} \right\|_2^2 = \left\| \begin{pmatrix} I, 0 \\ 0, V \end{pmatrix} \begin{pmatrix} S \\ \sqrt{\alpha} I \end{pmatrix} x - \begin{pmatrix} b^P \\ 0 \end{pmatrix} \right\|_2^2.$$

Givens rotations (8) can be used to efficiently provide the QR decomposition needed to find the $x$ yielding the minimum, denoted by $c$. The solution is given by the expression $c = \Lambda^{-1} S b^P$ where $\Lambda = S^\top S + \alpha I$ is a diagonal matrix. $\sqrt{\Lambda}$ is the upper triangular matrix of a QR decomposition available after applying a sequence of Givens rotations to the sparse matrix $\begin{pmatrix} S, \sqrt{\alpha} I \end{pmatrix}^\top$. The result of applying the same sequence of Givens rotations to the vector containing $b^P$ in the norm expression above is $(\sqrt{\Lambda})^{-1} S b^P$. The Givens rotations are computationally cheap and utilize quantities already constructed in Steps 1-4. Specifically one is able to perform Steps 5-7 in $O(n^P)$ flops. MATLAB scripts illustrating the details are provided in the Supporting Materials. Step 8 is unchanged.

Recall that an appealing feature of P-splines is that $n^P \leq n << m$ can be used to accurately represent some complex data sets. Another appealing feature of our QR based approach is that columns/rows of $C$ can readily be modified, added, and/or deleted. In some spline applications, letting the data modify the design matrix is desirable (11),(12) and our formulation can account for many such changes in the design matrix using low-rank updates to the various decompositions. Perhaps the feature most important to statistical applications is that for a fixed design matrix one
can obtain most quantities of interest to semiparametric regression, e.g. residual sum of squares, $\hat{\beta}_a$, various traces and covariance matrices, etc., for several trial $\alpha$ and/or $D^P$ by repeating the solution steps and using the matrix decomposition in hand to cheaply compute various quantities. The subsection below provides some examples in the special case $D^P = I$. Note that the MATLAB code in the Supporting Material illustrates how to compute these quantities in a vectorized fashion.

### 3.1 Statistical Quantities Computable by PSQR

The following commonly needed quantities in regression can be readily computable from the PSQR algorithm where $D^P = I$ is assumed. The quantities below can be computed for multiple $\alpha$ values without repeating the matrix factorization steps, i.e. Steps 1-4.

- $d_{\text{fit}}(\alpha) = n - n^P + \sum_{i=1}^{n^P} S^2_{ii} / (S^2_{ii} + \alpha)$

- $d_{\text{res}}(\alpha) = m - 2d_{\text{fit}}(\alpha) + (n - n^P) + \sum_{i=1}^{n^P} (S^2_{ii} / (S^2_{ii} + \alpha))^2$

  The above corresponds to the residual degrees of freedom.

- $\text{RSS}(\alpha) = \|y - C\hat{\beta}^a\|_2^2 = \|y\|_2^2 - \|(b_F, Sc, \sqrt{2\alpha}, c)^T\|_2^2$

- $GCV(\alpha) = \text{RSS}(\alpha) / (1 - d_{\text{fit}}(\alpha)/m)^2$

- $\text{AIC}_c(\alpha) = \log[\text{RSS}(\alpha)] + 2(d_{\text{fit}}(\alpha) + 1) / (m - d_{\text{fit}}(\alpha) - 2)$

Next we briefly discuss some details and extensions of the computations above. The residual sum of squares comes from definition of $\hat{\beta}_a^a \equiv (\hat{\beta}_a^T, \hat{u}_a^T)^T$, i.e.

\[
\begin{pmatrix}
C \\
\sqrt{\alpha}D
\end{pmatrix} \perp
\begin{pmatrix}
y \\
0
\end{pmatrix} -
\begin{pmatrix}
C \\
\sqrt{\alpha}D
\end{pmatrix} \hat{\beta}^a_a.
\]

The trace computations are easy to accomplish because we used a DR type decomposition on $Z^P$. The free columns ($X^F$ and $Z^F$) contribute $n - n^P$ to the trace due to these matrices being unaffected...
by $D$. If one desired to solve for pointwise confidence bands on design points $x_i$ associated with a given $\alpha$ and $C$, this can be obtained by doing a vectorized solve for the smoother matrix using the identity matrix in place of the data vector $y$ (1).

4 Applications and Test Cases

4.1 Overview

The results are organized into two groups: In the first set of results, PSQR is applied to a P-spline problem associated with a univariate nonlinear function reported in the Appendix. The nonlinear function serves as the population mean in a longitudinal data application, e.g. $f_j(x) = h_j(x; \theta_j) + g(x)$. Different types of design matrices, possessing different condition numbers, are used to obtain estimates of $g(x)$ and $h_i(x; \theta_i)$ from noisy data. The various design matrices are also reported in the Appendix. The second set of results uses the same data generating mechanism, but the design matrix parameters are tuned to illustrate some possible consequences of unnecessary numerical approximations. In every case studied, we set $D^P = I$.

4.2 Application to Longitudinal Data Analysis

Here we show how one can tile design matrices presented, i.e., form a design matrix using repeated blocks of one form. We then use the resulting matrix in a longitudinal data type modeling situation. In some of the models considered, PSQR can be used to substantially reduce the work load because only $n_P$ terms need are penalized. In some design matrices considered, this is a small fraction of the overall design matrix and hence substantial savings occur in Steps 5-8.

We also study the original Demmler-Reinsch numerical approximation/regularization that motivated this study (5) and label this algorithm as the “DR algorithm”. We highlight some dangers of making unnecessary numerical approximations. The DR case was chosen for simplicity: technically any SVD truncation scheme can encounter similar problems (4).
4.2.1 Data Generating Process

Here the data is generated according to

\[ y_{ij}(x_i) = g(x_i) + h_j(x_i, \theta_j) + \varepsilon_{ij}, \]  

(5)

where \( \varepsilon_{ij} \) are Normal(0,1) and independent, \( x_i \) are equally spaced on \([0, 15]\) and \( j = 1, \ldots, G \). The \( h_j(x_i; \theta_j) = \theta_j^{(1)}(\omega) + \theta_j^{(2)}(\omega)\text{atan}(x_i - \mu)/\gamma + \theta_j^{(3)}(\omega)(x_i - \phi)^2 \) with \( \theta_j^{(i)}(\omega) \) denoting component \( i \) of \( \theta_j \) and \( \omega_j \) is used to indicate that \( \theta_j \) is random vector corresponding to sample \( j \); the parameters of \( h_j(x) \) and the nonlinear function \( g(x) \) is reported in Section 6.2. A mixture of normals is used for the random intercept term \( \theta^{(1)} \). Specifically a uniform random variable, \( p_j \) on \([0, 1]\) was drawn and if \( p_j \leq 0.75 \) then \( \theta_j^{(1)} \sim N(0.5, 0.25) \) otherwise \( \theta_j^{(1)} \sim N(-1.5, 0.25) \). The other \( \theta^{(i)} \)'s were distributed using independent Normal(0,1) variables. In this study, one “curve batch” consisted of \( G = 25 \) or \( 5 \) curves, each function sampled at \( m = 40 \) values of \( x_i \). A total of 5000 such curve batches were analyzed in our Monte Carlo study. The sampling grid was identical for each curve. In some cases considered, in addition to \( y_{ij} \) we also assume that an estimate of \( \partial y_{ij} = \partial f_j(x_i) + \varepsilon_{ij}' \) is available where \( \partial f(x_i) \equiv df(x)/d(x)|_{x=x_i} \) and \( \varepsilon_{ij}' \) denotes Normal(0,1) i.i.d. and also independent of the \( \varepsilon_{ij} \). The simultaneous use of derivative and function estimates facilitates in tuning the condition number of the design matrix. The PSQR algorithm was developed to treat ill-conditioned design matrices that can result in this type of situation.

4.2.2 Semiparametric Model

Semiparametric regression models can be ignorant of the parametric form of \( h(\cdot ; \theta) \) (1), however we also study situations where this structure is assumed known. The interest is in both estimating the \( h_j \)'s and \( g \). For example, departures from the mean function, quantified through the \( h_j \)'s, were observed to have physical significance in the previous example and in (13),(14),and (15). In applications like proteomics or spectroscopic curve analysis, the interest is usually in the population function \( g(x) \) and the \( h_j \)'s are nuisance functions resulting from hard to avoid experimental drift and/or systematic baseline correction errors (6). To model these curves we use several different
design matrices. However, all models we consider have the following structure:

\[
y_B = \begin{pmatrix} X \\ X \\ \vdots \end{pmatrix} \beta + \begin{pmatrix} Z_{F1} & 0 \\ 0 & Z_{FG} \end{pmatrix} u_B^F + \begin{pmatrix} Z_{P1} & 0 \\ 0 & Z_{PG} \end{pmatrix} u_B^P + \begin{pmatrix} Z_P \end{pmatrix} u_P^P,
\]

where the subscript \( B \) on vectors is used to remind us that we are stacking \( G \) copies of discrete function samples each containing either \( m \) or \( 2m \) observations, the latter case occurs if we use derivative information. Another way of viewing this setup is say that the design matrices corresponding to \( h_j(x) \) and \( h_k(x) \) possess identical design matrices, but for \( i \neq k \) correspond to different P-spline coefficients.

In the design matrices we refer to as the “Generalized Additive Models” (GAM) case, we assume that the functional form of the basis functions used to generate the data are known, hence we only need to estimate the \( \theta^{(i)} \)'s for each curve sample \( j \). In the GAM case, the fixed effects are modified to \( X = (x_1 \ x_1^2; \ \ldots; \ x_m \ x_m^2) \). The \( Z_{Fi} \) each have 4 columns: one for the random intercept and the other three for the shape functions (in this case \( Z_{Pi} = 0 \)).

In the design matrices we refer to as “2 P-splines”, in contrast to the GAM case, we assume that the functional form of the basis functions describing the \( h_j \) is unknown to the analyst. Our design matrix attempts to find regression spline for both \( g \) and \( h_j \). The \( Z_{Fi} \) in this case include the linear polynomial terms for the \( G \) curves and the fixed effects matrix contains the quadratic polynomial term. The \( Z_{Pi} \) correspond to another knot sequence and TPF basis using \( K' \) knots uniformly spaced. The various design matrices used are written out in detail in the Appendix.

### 4.2.3 Results and Discussion

Figure 1 presents a representative sample of \( G = 25 \) curves along with three different estimates of these curves. The naive least squares estimate simply fits a P-spline to each of the curves. The influence of pooling information in the \( G \) curves as well as the derivative information is reported in Table 1. Here AMSE is computed for both the prediction of the individual curves as well as the estimate of the common population function \( g \). Not surprisingly, the GAM method using derivative information outperforms all other methods. When such functions are known, a substantial amount of dimension reduction can take place because only one large scale QR factorization is needed and then one computes factorizations associated with the relatively small \( R_{22}^P \) matrix to evaluate
different proposed smoothing parameters. Note that for large scale applications, this expensive QR factorization readily lends itself to parallelization. In the largest case studied in Table 1 the design matrix used by the 2 P-spline model used derivative and function information, each curve consisted of 40 function and 40 derivative scatterplot samples; the parameters $G = 25, K = 60, K' = 15$ along with the penalized and free polynomial terms resulted in a $C \in \mathbb{R}^{2000 \times 486}$. We evaluated the GCV over a grid of 1000 points logarithmically spaced between $1 \times 10^{-14}$ and $1 \times 10^{14}$, found the corresponding $\hat{\alpha}$ and $\hat{\beta}^a$, and $\hat{\gamma}$ in less than 2.2 seconds. All computations reported were carried out on a PC running MATLAB in Windows XP using an Intel Pentium D 3 GHz CPU using 3.5 GB RAM. The timing result provided is likely unrepresented of P-spline fits aiming at estimating a single curve, the number of columns is likely larger than is needed in typical tasks of this sort. To give one an idea of speed in the case just mentioned, we carried out the same operations as before, but this time fixing $p = 2$ using the “typical” TPF basis with $5000 \times 2$ observations of the function and derivative took $8.74 \times 10^{-2}$ seconds to compute when $K = 50$ ($3.95 \times 10^{-1}$ for $K = 100$).

![Figure 1](image)

Figure 1: (a) Sample of curve population observed. The curves labeled $f_i$ are generic samples, and those labeled $f_1$ and $f_2$ are estimated from noisy samples in panel (b) where the symbols denote the observed $y_i$ and the colored curves denote the various estimates of the color curves highlighted in (a). The grey curve shows the known “truth” and “LS” represents a standard least squares regression fit.

However ignorance of the functional form of the shape function is common and it is reassuring that the 2 P-Spline approach can work even in situations where the number of columns of the design matrix can grow rapidly. Given that condition number of the random effects matrix $Z$ typically grows rapidly in this situation (regardless of the basis) this posses serious computational
problems. The P-spline framework has a natural regularization parameter built into the regression. The high ill-conditioning of $Z$ often forces many researchers to introduce arbitrary parameters to provide numerical answers and hence another regularization. For example the DR algorithm advocated in Ref. (5) explicitly points this problem out; the algorithm they use needs a Cholesky factor of $C^T C$. Since the condition number of $C$ is large, this can be problematic even in small scale problems. To remedy this situation the authors obtain the Cholesky factor of $C^T C + \eta D$ instead. This regularized solution is then used to select an $\alpha$ which further smooths and is used to estimate $f$. The ad hoc parameter $\eta$ is suggested to be something small like $1 \times 10^{-10}$. Other works suggest that one should avoid such a situation by using a pivoted QR in conjunction with a truncated SVD instead (4), but again the cut-off criterion is arbitrary and using a poor criterion may obscure results. These benefits are perhaps seemingly minor, the next section demonstrates the practicality of these types of concerns using concrete examples.

4.2.4 Examples Illustrating the Dangers of ad hoc Regularization

The first example in this section utilizes data reported reported in Table 1. In the $G = 25$ GAM $f$, Case 1 in the Appendix, the condition number of $C^T C$ was $8.5 \times 10^{23}$ and that of $C^T C + \eta D$ was $5.4 \times 10^{18}$ and $1.1 \times 10^{17}$ for $\eta = 1 \times 10^{-10}$, $1 \times 10^{-6}$ (respectively). The cases with “(DR)” in Table 1 used the algorithm reported in (5) setting $\eta = 1 \times 10^{-10}$. The ad hoc regularization, or approximation, of replacing $C^T C$ by $C^T C + \eta D$ does measurably influence the numerically exact results reported in Table 1. Note that we simulated 5000 sets of scatterplot points, and processed each set of points 5 different times (once for each design matrix considered). The differences between the AMSE reported in the different columns are not random, the differences are due to systematic computation error. The standard deviation reflects uncertainty used by the random number generator introducing $\epsilon$. Each single noise realization was processed multiple times using the design matrices reported in the Appendix. As the size of the problem grows, the user will be forced to use a larger and larger $\eta$ only for numerical consideration and this will make the departure from the exact solution even larger.

Next we present a concrete example of a larger P-spline problem where numerical approximations introduce more serious errors. We increase the number of observations of $\{f(x_i), \partial f(x_i)\}$ to 80, $K$ to 160, and $K'$ to 60 in the 2 P-Spline $f, \partial f$ Case 4 design matrix. The condition number
Table 1: Measured average AMSE and standard deviation of the subject specific curves $h^{(i)}$ and the population curve $G$ in an MC simulation. Two different batch sizes were used; the first simulation contained $n_G = 25$ and the second used $n_G = 5$. Each individual curve was sampled over a uniform grid using a total of $m=40$ observations for each curve (for $\partial f$ and/or $f$). Each estimate of $g$ utilized the batch of $n_G f^{(i)}$ curves. A total of 1000 batches were simulated and the first two moments are reported (standard deviation in parenthesis). The design matrices corresponding to the various cases reported are given in the Appendix. The PSQR algorithm was used in all cases except the two rows where “(DR)” appears. These cases are discussed in Section 4.2.4.

<table>
<thead>
<tr>
<th>Method</th>
<th>$\langle |\hat{f}_i - f|_2^2 \rangle/m$</th>
<th>$\langle |\hat{g} - g|_2^2 \rangle/m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAM $f$, $\partial f$</td>
<td>0.1440(0.0224)</td>
<td>0.0712(0.0663)</td>
</tr>
<tr>
<td>GAM $f$</td>
<td>0.1830(0.0270)</td>
<td>0.0950(0.0674)</td>
</tr>
<tr>
<td>GAM $f$ (DR)</td>
<td>0.1909(0.0273)</td>
<td>0.0948(0.0673)</td>
</tr>
<tr>
<td>2 P-Splines $f$, $\partial f$</td>
<td>0.4320(0.0357)</td>
<td>0.1343(0.0682)</td>
</tr>
<tr>
<td>2 P-Splines $f$</td>
<td>0.4408(0.0402)</td>
<td>0.1170(0.0697)</td>
</tr>
<tr>
<td>Sample Size = 5 × $m$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAM $f$, $\partial f$</td>
<td>0.1804(0.0545)</td>
<td>0.3894(0.3914)</td>
</tr>
<tr>
<td>GAM $f$</td>
<td>0.2967(0.0814)</td>
<td>0.4923(0.3957)</td>
</tr>
<tr>
<td>GAM $f$ (DR)</td>
<td>0.3098(0.0778)</td>
<td>0.4901(0.3944)</td>
</tr>
<tr>
<td>2 P-Splines $f$, $\partial f$</td>
<td>0.6470(0.1041)</td>
<td>0.6670(0.3972)</td>
</tr>
<tr>
<td>2 P-Splines $f$</td>
<td>0.9980(0.1465)</td>
<td>0.5363(0.3959)</td>
</tr>
</tbody>
</table>

of this design matrix was $\approx 8.5 \times 10^{10}$, hence the condition of $C^TC$ used by the DR algorithm (5) was $\approx 7.2 \times 10^{21}$. We represent the common population function $g$ as a bold line in Fig. 2. We also plot the estimate of this function using the PSQR as well as the DR algorithms. Two different $\eta$ values, $1 \times 10^{-3}$ & $1 \times 10^{-6}$, were entertained when using the DR algorithm and they resulted in $C^TC + \eta D$ having a reduced condition number of $\approx 5.7 \times 10^{11}$ & $\approx 5.7 \times 10^{14}$, respectively.

The left panel of Figure 2 plots the various estimates. PSQR approximates the true function $g$ with higher fidelity ($\|\hat{g} - g\|_2^2 \approx 0.13$) but, more importantly, captures the main features of the true curve whereas both DR methods perform poorly in regards to both numerical ($\|\hat{g} - g\|_2^2 \approx 1.7, 1.25$) and qualitative criterion. Perhaps more disturbingly, such an error introduced by unnecessary numerical approximations would likely go unnoticed if features of the underlying
Figure 2: Example where innocuous numerical approximations fail severely. The bold curve displays the function of interest. Three independent realizations, i.e. batch of scatterplot points, were smoothed to demonstrate that the phenomena is not due to some pathological perturbation. The same design matrix was used to process all the data, but different unnecessary numerical approximations were introduced in the cases labeled DR. See text for details. The PSQR algorithm makes no such approximations. The main features of the true curve are captured in the PSQR case, but not in the others.

function were completely unknown. The right panel of this figure illustrates why this occurs. The underlying data consists of batches of two curves, \( f^{(j)}(x_i) = g(x_i) + h^{(j)}(x_i) + \epsilon \) and \( \partial f^{(j)}(x_i) = \partial g(x_i) + \partial h^{(j)}(x_i) + \epsilon' \). The \( h^{(j)} \) and \( g \) functions each have very different degrees of smoothness. A fairly large number of knots (\( K \) and \( K' \)) needs to be used to resolve both functions. If the aforementioned design matrix parameters are large enough and we have adequate signal to noise, then exact GCV possesses two minimum, see curve labelled PSQR in Figure 3. The \( \eta \) parameter introduced complicates accurately computing the portion of the GCV curve related to small \( \alpha \), i.e., related to small singular values. Since the GCV computed with the DR algorithm only sees the minimum near large \( \alpha \), it selects this spurious local minimum and hence dramatically over-
Figure 3: The GCV corresponding to the cases studied in Figure 2 is plotted as a function of $\alpha$. The computational parameter introduced in DR does not allow one to find the global minimum of the true GCV located at small $\alpha$. This error can be introduced by any scheme introducing an arbitrary singular value cut-off. The PSQR algorithm avoids the need to specify such a parameter.

smooths. Note that for the local minima associated with larger $\alpha$ around $\approx 1-5$ the exact position of the minimum is still inaccurate. The shift of this minima also explains the “minor” differences observed between PSQR and DR in Table 1.

We make no claim that the design matrix, knot spacing, etc. are making optimal use of the data in the previous example. Our fit is likely far from “optimal”. The point of this example was to show that large systematic bias can be introduced by unnecessary numerical approximations or ad hoc regularization. Although the DR type approximation as shown in (5) is known to have poor numerical properties (1), this DR based algorithm was likely advocated because of its computational speed. What may not be generally appreciated is that severe approximation errors can be introduced which are not easily detected. Note that truncating singular-values prematurely (4) can suffer similar types of errors as shown in this example. Since our method can handle exactly
rank deficient matrices, we never need to introduce an SVD truncation criterion or other *ad hoc* parameter to stabilize the algorithm. With this contribution, we have eliminated some major computational concerns associated with using poorly conditioned spline basis to compute $Z$ in this type of P-spline problem. Both algorithms introduced can be applied to any basis, not just the TPF basis, but the fast specialized version of PSQR we reported exploited the TPF structure.

5 Conclusions and Outlook

We have introduced a new algorithm, PSQR, for computing P-splines that is fast, numerically stable, and requires no tunable computational parameters. PSQR was designed to treat general ridge regression P-spline problems with a design matrix $C \in \mathbb{R}^{m \times n}$ and a matrix $D$ whose square $(D^T D)$ was non-negative definite. One expensive QR factorization of $C$ was performed and then subsequent matrix operations were performed on matrices with $n_P$ columns where the subscript denotes the number of columns subject to penalization / regularization. In typical P-spline applications, accurate curve approximations can be obtained with $n_P \ll m$. This property is computationally attractive because large data sets can be represented parsimoniously with a single P-spline. After one QR factorization of $C$, different smoothing parameters can be attempted without having to redo the expensive QR factorization of $C$. PSQR can accomplish these tasks by doing repeated operations on matrices possessing $n_P$ columns. The QR basis of the algorithm readily allows low rank QR column or row insertions and deletions (8), (11), (10).

Perhaps most importantly, PSQR can reliably handle exactly rank deficient design matrices, $Z$, without requiring any unnecessary computational parameters or cut-off criterion to be specified. We have demonstrated concrete cases where *ad hoc* parameters can cause both subtle and dramatic approximation errors that would likely go unnoticed by an investigator blindly trusting the output of a method introducing these types of unnecessary computational parameters. Note that our examples were not excessively large in regards to modern data sets, and as the problem dimension grows the types of issues we highlighted will likely become a greater concern to existing P-spline computational approaches. In some applications, knot locations are selected by the data (12). It has already been suggested that data-driven knot location might be useful in the P-spline setting (16). Our algorithm removes a computational concern about such data-driven knot procedures, e.g. design matrices where $\kappa_i = \kappa_j$ for $i \neq j$ are not computationally problematic for our
algorithm precisely because the algorithm handles exact rank deficiency in \( Z \) without numerical approximations. When PSQR exploited the matrix structure induced by the TPF basis on certain P-spline problems (1), we demonstrated that an extremely efficient and reliable P-spline fit could be obtained and our algorithm removed many of the numerical concerns associated with using a TPF basis in regression spline applications. We have made MATLAB scripts demonstrating the general and special case \( D^P = I \) of the PSQR algorithm via the Supplemental Materials.

**SUPPLEMENTAL MATERIALS**

**Title:** Brief description. (file type)

**MATLAB Demo:** Collection of MATLAB scripts containing code to perform the PSQR algorithm for general penalty matrix \( D^P \) and fast special case \( D^P = I \). In latter we show how to do vectorized computations of some standard statistical quantities (PSQRmfiles.zip)

6 Appendix

6.1 Design Matrices Studied

The design matrix, \( C \) defined in Equation (6) requires the specification of \( X, Z^P, Z^{P_i}, \) and \( Z^{F_i} \). Recall the later two random effects design matrices are identical for each subject and that \( h^{(j)}(x_i)\theta^{(1)}_j(\omega) + \theta^{(2)}_j(\omega)\tan((x_i - \mu)/\gamma) + \theta^{(3)}_j(\omega)(x_i - \phi)_+^2 \) where we set \( \mu = 7.5, \gamma = 6, \) and \( \phi = 9 \). We introduce \( \psi^1(x) = 1, \psi^2(x) = \tan((x - \mu)/\gamma), \) and \( \psi^3(x) = (x_i - \phi)_+^2 \). The \( \partial \) symbol appearing below represents differentiation with respect to \( x \), e.g. \( \partial x^2 = 2x \).

**Case 1 “GAM, \( f’ \)”:**

\[
X = \begin{pmatrix} x_1, x_1^2 \\ \vdots \\ x_m, x_m^2 \end{pmatrix}, \quad Z^P = \begin{pmatrix} (\kappa_1 - x_1)_+^2 \ldots (\kappa_K - x_1)_+^2 \\ \vdots \\ (\kappa_1 - x_m)_+^p \ldots (\kappa_K - x_m)_+^2 \end{pmatrix}, \\
Z^{F_i} = \begin{pmatrix} \psi^1(x_1), \psi^2(x_1), \psi^3(x_1) \\ \vdots \\ \psi^1(x_m), \psi^2(x_m), \psi^3(x_m) \end{pmatrix}, \quad Z^{P_i} = 0.
\]
Case 2 “GAM, $f, \partial f$”:

\[
X = \begin{pmatrix}
  x_1, x_1^2 \\
  \vdots \\
  x_m, x_m^2 \\
  1, 2x_1 \\
  \vdots \\
  1, 2x_m
\end{pmatrix},
Z^P = \begin{pmatrix}
  \kappa_1 - x_1^2 & \cdots & \kappa_K - x_1^2 \\
  \vdots & \ddots & \vdots \\
  \kappa_1 - x_m^2 & \cdots & \kappa_K - x_m^2 \\
  2(\kappa_1 - x_1) & \cdots & 2(\kappa_K - x_1) \\
  \vdots & \ddots & \vdots \\
  2(\kappa_1 - x_m) & \cdots & 2(\kappa_K - x_m)
\end{pmatrix},
\]

\[
Z^{F_i} = \begin{pmatrix}
  \psi^1(x_1), \psi^2(x_1), \psi^3(x_1) \\
  \vdots \\
  \psi^1(x_m), \psi^2(x_m), \psi^3(x_m) \\
  \partial^1(x_1), \partial^2(x_1), \partial^3(x_1) \\
  \vdots \\
  \partial^1(x_m), \partial^2(x_m), \partial^3(x_m)
\end{pmatrix},
Z^{P_i} = 0.
\]

Case 3 “2 P-Spline, $f$”:

\[
X = \begin{pmatrix}
  x_1^2 \\
  \vdots \\
  x_m^2 \\
  1, x_1 \\
  \vdots \\
  1, x_m
\end{pmatrix},
Z^P = \begin{pmatrix}
  \kappa_1 - x_1^2 & \cdots & \kappa_K - x_1^2 \\
  \vdots & \ddots & \vdots \\
  \kappa_1 - x_m^2 & \cdots & \kappa_K - x_m^2 \\
  2(\kappa_1 - x_1) & \cdots & 2(\kappa_K - x_1) \\
  \vdots & \ddots & \vdots \\
  2(\kappa_1 - x_m) & \cdots & 2(\kappa_K - x_m)
\end{pmatrix},
\]

\[
Z^{F_i} = \begin{pmatrix}
  1, x_1 \\
  \vdots \\
  1, x_m
\end{pmatrix},
Z^{P_i} = \begin{pmatrix}
  \kappa_1^2 - x_1^2 & \cdots & \kappa_K^2 - x_1^2 \\
  \vdots & \ddots & \vdots \\
  \kappa_1^2 - x_m^2 & \cdots & \kappa_K^2 - x_m^2 \\
  2(\kappa_1^2 - x_1) & \cdots & 2(\kappa_K^2 - x_1) \\
  \vdots & \ddots & \vdots \\
  2(\kappa_1^2 - x_m) & \cdots & 2(\kappa_K^2 - x_m)
\end{pmatrix}.
\]

Case 4 “2 P-Spline, $f, \partial f$”:

21
\[ X = \begin{pmatrix} x_1^2 \\ \vdots \\ x_m^2 \\ 2x_1 \\ \vdots \\ 2x_m \end{pmatrix}, \quad Z^P = \begin{pmatrix} (\kappa_1 - x_1)_+^2 \ldots (\kappa_K - x_1)_+^2 \\ \vdots \\ (\kappa_1 - x_m)_+^2 \ldots (\kappa_K - x_m)_+^2 \\ 2(\kappa_1 - x_1)_+ \ldots 2(\kappa_K - x_1)_+^2 \\ \vdots \\ 2(\kappa_1 - x_m)_+ \ldots 2(\kappa_K - x_m)_+^2 \end{pmatrix}, \]

\[ Z_{F_i} = \begin{pmatrix} 1, x_1 \\ \vdots \\ 1, x_m, \\ 0, 1 \\ \vdots \\ 0, 1 \end{pmatrix}, \quad Z_{P_i} = \begin{pmatrix} (\kappa'_1 - x_1)_+ + \ldots (\kappa'_{K'} - x_1)_+ \\ \vdots \\ (\kappa'_1 - x_m)_+ + \ldots (\kappa'_{K'} - x_m)_+ \\ 1_{(\kappa'_1 - x) > 0}(x_1) \ldots 1_{(\kappa'_{K'} - x) > 0}(x_1) \\ \vdots \\ 1_{(\kappa'_1 - x) > 0}(x_m) \ldots 1_{(\kappa'_{K'} - x) > 0}(x_m) \end{pmatrix}, \]

where \( \{\kappa'_i\}_{i=1}^{K'} \) is a uniformly spaced knot sequence distinct from \( \{\kappa_i\}_{i=1}^{K} \) using \( \kappa'_1 = \kappa_1 + 10\pi/K \) and \( \kappa'_{K'} = \kappa_K - 10\pi/K \) with \( K' < K \) and \( 1_{(\kappa'_1 - x) > 0}(\cdot) \) is the indicator function taking a value of 1 if \( (\kappa'_1 - x) > 0 \) and 0 otherwise. Note that similar results to those shown here using design matrices with \( Z_{P_i} \) corresponding to quadratic terms were obtained.

### 6.2 Known Nonlinear Function

\[
y = -\left(2.5 \sin(2\pi \frac{x}{15}) + 9 \exp\left(-\frac{(x-3)^2}{2}\right) + 5 \exp\left(-\frac{(x-10)^2}{2}\right) - 2\right)/3; \tag{7}
\]

Note: The domain of the function above is \([-15, 0]\). The plots displayed in this study reflect this function about zero and use the domain \([0, 15]\) instead.

### References


