Solving Continuous Linear Least-Squares Problems by Iterated Projection

BY RALF JUENGLING

Department of Computer Science, Portland State University
PO Box 751 Portland, OR 97207 USA

Email: juenglin@cs.pdx.edu

Abstract

I present a new divide-and-conquer algorithm for solving continuous linear least-squares problems. The method is applicable when the column space of the linear system relating data to model parameters is "translation invariant". The central operation is a matrix-vector product, which makes the method very easy to implement. Secondly, the structure of the computation suggests a straightforward parallel implementation.

A complexity analysis for sequential implementation shows that the method has the same asymptotic complexity as well-known algorithms for discrete linear least-squares. For illustration we work out the details for the problem of fitting quadratic bivariate polynomials to a piecewise constant function.

1 Introduction

The linear least-squares problem is to determine the parameter values β^* for a function f that minimizes the sum of squared differences between data values $\{d_i\}$ and function values $\{f(x_i;\beta)\}$ for corresponding points $\{x_i\}$,

$$\beta^* = \operatorname{argmin}_\beta \sum_{i=1}^{m} (d_i - f(x_i;\beta))^2. \tag{1}$$

The problem is a *linear least-squares* problem if the model f is linear in the parameters, that is,

$$f(x;\beta) = \sum_{k=1}^{n} \beta_k f_k(x).$$

A classical way of determining β goes as follows:

1. Express (1) in matrix form,

   $$\beta^* = \min_\beta \|d - A\,\beta\|, \tag{2}$$

   where

   $$A = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \dots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_n(x_2) \\ \vdots & \vdots & & \vdots \\ f_1(x_m) & f_2(x_m) & \dots & f_n(x_m) \end{pmatrix}, \tag{3}$$

   and d is the vector of data values.

2. Construct and solve the so called *normal equations*

   $$A^T A\,\beta = A^T d. \tag{4}$$

Assuming $m > n$ (more data values than parameters) and that A has full rank, the solution to (4) may be written in terms of the *pseudoinverse* A^+ of A,

$$\beta = (A^T A)^{-1} A^T d = A^+ d. \tag{5}$$

Geometrically speaking, $\tilde{d} = A\beta$ is the point in the range of A closest to d in the Euclidean norm. The pseudoinverse projects d to the solution β, which may be thought of as the "coordinates" of $\tilde{d}$ in the column space of A.

In the following we consider the continuous, linear version of problem (1),

$$\beta^* = \operatorname{argmin}_\beta \int_\Omega \left(d(x) - \sum_k \beta_k f_k(x) \right)^2 \mathrm{dx}. \tag{6}$$

Note that in the continuous setting we assume there is a "data function" $d(x)$, defined over some domain Ω. An example is single-band image data, which is often interpreted as a piecewise constant function over the image domain [3].

Section 4 introduces the key ideas of the new method and explains its prerequisites. Section 4 gives the algorithm and discusses its computational merits. As an example, in Section 4 I work out the elements of the algorithm for the problem of fitting a bivariate quadratic polynomial. The method for solving (6) that I call *iterated projection* works for any dimension of Ω, but for the sake of clear exposition I assume Ω has dimension 2 and $\Omega = \Omega_{2h} = [-h, h] \times [-h, h]$.

2 Linear Least-Squares by Iterated Projection

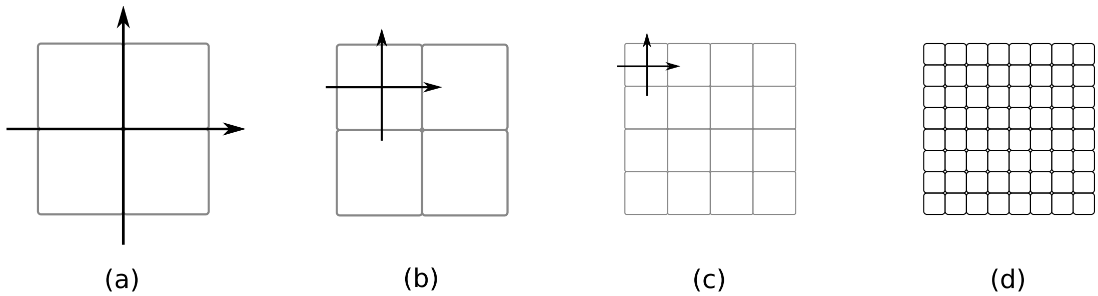

Figure 1. Domain Ω with coordinate origin at center (a). Ω partitioned into four subdomains Ω_1, Ω_2, Ω_3, Ω_4 in clockwise order (b), next finer partition (c), finest partition at which the data function d may be accurately represented over the subdomains (d). A local coordinate system is centered on each subdomain (only shown for upper left subdomains in (b) and (c)).

Recall that the pseudoinverse in (5) projects the data directly onto a solution vector. With iterated projection we also obtain the solution by projection. However, instead of directly projecting the data we project solutions of intermediate least-squares problems. Imagine we partition Ω into four equally sized subdomains Ω_1, ..., Ω_4 as shown in Figure 1(b), and that we are able to solve (6) for each subdomain, with solutions β_1^*, ..., β_4^*. Can we compute the solution β^* for Ω from the four solutions β_1^*, ..., β_4^* for Ω_1, ..., Ω_4? As we will see below, the answer is yes, under certain circumstances.

With "problem (6) for subdomain Ω_i" we mean that the integral in (6) is over Ω_i instead of Ω, and that the coordinates are with respect to a coordinate system centered on Ω_i (cf. Figure 1(b)). But centering the coordinate systems with respect to each Ω_i means the four integrals are over the same set of points $\Omega_h = [-h/2, h/2] \times [-h/2, h/2]$. These are important aspects of iterated projection: the problems corresponding to the four subdomains are four instances of the same problem (only the data term in (6) is different), as well as similar to the original problem ($\Omega = \Omega_h$ instead of $\Omega = \Omega_{2h}$).

A few more bits of notation are necessary to define the intermediate problems more precisely. We denote the coordinates (with respect to Ω_{2h}) of the four subdomain's coordinate origins by $x_1^0, ..., x_4^0$ and the solutions corresponding to the four subdomain problems by $\beta_1^*, ..., \beta_4^*$, respectively. In the continuous setting β^* are the coordinates of a function closest to d in the space F_{2h} of functions over Ω_{2h} that is spanned by $\{f_k|_{\Omega_{2h}}\}_{k=1...n}$. Likewise, the solution β_i^* for subdomain Ω_i are the coordinates of a function closest to $x \mapsto d(x + x_i^0)$, in the space F_h of functions over Ω_h spanned by $\{f_k|_{\Omega_h}\}_{k=1...n}$. We can use the space F_h to construct another space of functions over Ω_{2h},

$$\tilde{F}_{2h} = \{ f : \Omega_{2h} \to \mathbb{R} \mid x \mapsto f(x + x_i^0) \in F_h \text{ for all } x \in \Omega_h \text{ and } i = 1...4 \}.$$

Note that F_{2h} has dimension n, whereas $\tilde{F}_{2h}$ has dimension $4 \times n$. The answer to our question is that we can construct a solution β^* for (6) from the solutions β_i^* for the subdomains if F_{2h} is a subspace of $\tilde{F}_{2h}$,

$$F_{2h} \subset \tilde{F}_{2h}. \tag{7}$$

2.1 The Subspace Property

When does the "subspace property" (7) hold for a set of functions $\{f_k\}$? What we must verify is that for any set of parameters β there are parameters $\beta_1, ..., \beta_4$ such that

$$\sum_k \beta_k f_k(x - x_i^0) = \sum_k \beta_{i,k} f_k(x), \ \forall x \in \Omega_h, i = 1, ..., 4. \tag{8}$$

In other words, relation (7) holds when the space of functions (over $\mathbb{R}^2$) spanned by $\{f_k\}$ is *translation-invariant*. Examples are the spaces of polynomials of any finite order, or the space of functions spanned by sine-cosine pairs with same wave number.

2.2 Solving the Linear Least-Squares Problem

We want to obtain the linear least-squares solution β^* on $\Omega = \Omega_{2h}$ from the solutions β_1^*, ..., β_4^* for Ω's subdomains Ω_1, ..., Ω_4, respectively. As we said above, we obtain β^* by projecting the solutions $\beta_1^*, ..., \beta_4^*$. Let a function $\tilde{f} \in \tilde{F}_{2h}$ be parameterized by $\tilde{\beta} \in \mathbb{R}^{4 \times n}$, $\tilde{\beta}^T = \left(\beta_1^T \, \beta_2^T \, \beta_3^T \, \beta_4^T\right)$, where β_i corresponds to the part of $\tilde{f}$ covering subdomain Ω_i, and let $f \in F_{2h}$ be parameterized by $\gamma \in \mathbb{R}^n$. The projector from $\tilde{F}_{2h}$ onto F_{2h} we seek maps $\tilde{\beta}$ to the vector

$$\beta^* = \mathrm{argmin}_\gamma \sum_i \int_{\Omega_h} \left(\sum_l \beta_{i,l} f_l(x) - \sum_k \gamma_k f_k(x - x_i^0) \right)^2 \mathrm{dx}. \tag{9}$$

Assume, for the moment, we know the projector, a $n \times (4\,n)$-matrix, and call it R_h. Using R_h, the solution to problem (6) is

$$\beta^* = R_h \tilde{\beta}^* = \left(\begin{array}{cccc} R_{h,1} & R_{h,2} & R_{h,3} & R_{h,4} \end{array} \right) \left(\begin{array}{c} \beta_1^* \\ \beta_2^* \\ \beta_3^* \\ \beta_4^* \end{array} \right) \tag{10}$$

($R_{h,i}$ denotes the obvious $n \times n$ submatrix of R_h). Equation (10) expresses the central idea and operation of the iterated projection algorithm: obtain solutions to smaller versions of the linear least-squares problem first, then derive the solution to the original problem by a simple matrix-vector product. The solutions to the smaller problems are obtained in the same way, by projecting solutions for sub-subdomains, using a similar projection operator $R_{h/2}$. The resulting algorithm is recursive. The recursion ends at subdomains over which the data function d is an element of the space spanned by $\{f_k\}$ over that domain, or when d may be approximated in that space with negligible error (Figure 1(d)).

2.3 The Projection Operator

We now turn to the problem of determining the projection matrix R_h used in (10). There are different ways of deriving R_h. In this section I describe one way, and in Section 4 I demonstrate another way by example.

To begin note that when subspace property (7) holds, for every element β in the range of R_h there is one vector $\tilde{\beta}$ for which (9) is exactly zero. If this is the case, then $\tilde{\beta}$ and $\beta = R_h \tilde{\beta}$ represent the same function over Ω_{2h}. We denote P_h the $(4\,n) \times n$-matrix that maps β (representing a function in F_{2h}) to the vector $\tilde{\beta}$ representing the same function in $\tilde{F}_{2h}$,

$$\tilde{\beta} = P_h \beta = \left(\begin{array}{c} P_{h,1} \\ P_{h,2} \\ P_{h,3} \\ P_{h,4} \end{array} \right) \beta. \tag{11}$$

Determining P_h for a particular set of functions $\{f_k\}$ is just another way of confirming that the subspace property (7) holds. P_h is relevant here because concrete expressions for it are easy to derive, and we can express R_h in terms of P_h (below).

The second ingredient we need is the inner product on the parameter space of F_{2h} that corresponds to the "least-squares" (Euclidean) norm,

$$\begin{aligned}\langle f, g\rangle_{2h} &= \int_{\Omega_{2h}} f(x)\, g(x)\, \mathrm{dx} \\ &= \int_{\Omega_{2h}} \Big(\sum_k \beta_k f_k(x)\Big)\Big(\sum_l \gamma_l f_l(x)\Big)\mathrm{dx} \\ &= \beta^T \begin{pmatrix} \int_{\Omega_{2h}} f_1 f_1 & \int_{\Omega_{2h}} f_1 f_2 & \cdots & \int_{\Omega_{2h}} f_1 f_n \\ \int_{\Omega_{2h}} f_2 f_1 & \int_{\Omega_{2h}} f_2 f_2 & & \vdots \\ \vdots & \vdots & & \int_{\Omega_{2h}} f_{n-1} f_n \\ \int_{\Omega_{2h}} f_n f_1 & \int_{\Omega_{2h}} f_n f_2 & \cdots & \int_{\Omega_{2h}} f_n f_n \end{pmatrix} \gamma \\ &= \beta^T Q_{2h}\, \gamma. \end{aligned} \tag{12}$$

Q_{2h} is the Gram matrix of the functions $\{f_k\}$. Inner products corresponding to the subdomain-function spaces are defined in the same way,

$$\langle f|_{\Omega_h}, g|_{\Omega_h}\rangle_h = \beta_i^T \begin{pmatrix} \int_{\Omega_h} f_1 f_1 & \int_{\Omega_h} f_1 f_2 & \cdots & \int_{\Omega_h} f_1 f_n \\ \int_{\Omega_h} f_2 f_1 & \int_{\Omega_h} f_2 f_2 & & \vdots \\ \vdots & \vdots & & \int_{\Omega_h} f_{n-1} f_n \\ \int_{\Omega_h} f_n f_1 & \int_{\Omega_h} f_n f_2 & \cdots & \int_{\Omega_h} f_n f_n \end{pmatrix} \gamma_i = \beta_i^T Q_h\, \gamma_i. \tag{13}$$

With P_h and Q_h we can express the least-squares property (9), which is the defining property of R_h, without using integrals

$$\begin{aligned} R_h\, \tilde{\beta} &= \mathrm{argmin}_\gamma \sum_i \int_{\Omega_h} \Big(\sum_l \beta_{i,l} f_l(x) - \sum_k \gamma_k f_k(x - x_i^0)\Big)^2 \mathrm{dx} \\ &= \mathrm{argmin}_\gamma \sum_i \int_{\Omega_h} \Big(\sum_l \beta_{i,l} f_l(x) - \sum_k (P_h\gamma)_{i,k} f_k(x)\Big)^2 \mathrm{dx} \\ &= \mathrm{argmin}_\gamma \sum_i (\beta_i - (P_h\gamma)_i)^T Q_h\, (\beta_i - (P_h\gamma)_i) \end{aligned} \tag{14}$$

Since the sum in (14) is quadratic in γ we can set its gradient w.r.t γ to zero and solve for γ to find the minimizer. The result is the following expression for R_h,

$$R_h = \left(\sum_{i=1}^4 P_{h,i}^T Q_h P_{h,i}\right)^{-1} \left(P_{h,1}^T Q_h \quad P_{h,2}^T Q_h \quad P_{h,3}^T Q_h \quad P_{h,4}^T Q_h \right). \tag{15}$$

3 Computational Characteristics

The following algorithm ITERATED_PROJECTION_LSQ is a very simple instance of iterated projection; it requires that the data function is piecewise constant over square regions of a regular partition of Ω into $m \times m = 2^l \times 2^l$ regions. Not included in the algorithm description is the construction of the projection matrices R_h. These are assumed to have been precomputed for all relevant scales prior to calling ITERATED_PROJECTION_LSQ.

ITERATED_PROJECTION_LSQ(D, h)

Input. D: $m \times m$ data matrix, h: scale of domain

Output. β^*: least-squares solution over domain $\Omega = [-h, h] \times [-h, h]$

Steps.

1. If $m = 1$:
2. $\beta^* \leftarrow$ project constant function $d(x) = D_{1,1}$ (Section 2.2)
3. else:
4. partition D, $D = \begin{pmatrix} D_1 & D_2 \\ D_4 & D_3 \end{pmatrix}$

5. $\beta_i^* \leftarrow$ ITERATED_PROJECTION_LSQ(D_i, $h/2$), for $i = 1, \ldots, 4$

6. $\beta^* \leftarrow R_h \begin{pmatrix} \beta_1^* \\ \beta_2^* \\ \beta_3^* \\ \beta_4^* \end{pmatrix}$ (Equation 10)

3.1 Computational Complexity of Iterated Projection

Algorithm ITERATED_PROJECTION_LSQ is formulated for two-dimensional domains. However, adapting the algorithm to domains of different dimensionality is straightforward and we discuss computational complexity for the general case (d dimensions). In the general case

- D is assumed to have m^d entries and is partitioned into 2^d parts in Step 4,
- the projection matrix R_h is of size $n \times 2^d n$,
- the depth of the recursion $l = \log_2 m$ is independent of d.

We assume that analytical expressions for R_h are available, thus, precomputing the projection matrices means evaluating those expressions for all l scales. Assuming that the amount of computation for evaluating each R_h-entry is independent of h, precomputing the projecting matrices requires $O(\log_2 m \times 2^d \times n^2)$ operations and $O(\log_2 m \times 2^d \times n^2)$ space.

Computing β^* in Step 2. requires $O(n)$ operations per entry of D, or $O(m^d \times n)$ operations total. Computing β^* in Step 6. requires a matrix-vector product of $O(2^d \times n^2)$. Step 6 is carried out $\sum_{i=0}^{l-1} \left(2^d\right)^i = \frac{2^{d \times l} - 1}{2^d - 1}$ times, resulting in a total operation count of $O\left(2^{d \times \log_2 m} \times n^2\right)$ or $O(m^d \times n^2)$ for Step 6.

Steps 5. and 6. together may be organized in a loop over all D_i. In that case the amount of space required by iterated projection (in addition to the space required for the projection matrices) is proportional to n and proportional to the depth of the recursion, $O(\log_2 m \times n)$.

3.2 Comparison to Classical Linear Least-Squares Algorithms

In discrete linear least-squares problems, the problem domain Ω has been abstracted away and its dimension usually plays no part in a complexity analysis. Instead, the complexity of those algorithms are expressed in the size of matrix A in (4), say $M \times n$ [4]. Setting $M = m^d$ for comparison, the cost of iterated projection is $O(M \times n^2)$ operations, and $O\left(\frac{2^d}{d} \times \log_2 M \times n^2\right)$ units of space (ignoring space required for entering the problem).

A classical algorithm for solving the normal equations (4) is Cholesky factorization,

CHOLESKY_LSQ(A, d)

Input. A: $M \times n$ model matrix, d: M data vector

Output. β^*: least-squares solution $\operatorname{argmin}_\beta \|A\beta - d\|_2$

Steps.

1. Compute $B = A^T A$
2. Compute $y = A^T d$
3. Factorize B, $B = R^T R$ (R upper triangular)
4. Solve $R^T x = y$
5. Solve $R\beta^* = x$

The costs of these steps are, respectively, $O(M \times n^2)$, $O(M \times n^2)$, $O(n^3)$, $O(n^2)$, and $O(n^2)$ operations. In most situations where iterated projection is applicable we could carry out Step 1. and Step 3. of CHOLESKY_LSQ in advance, and the asymptotic cost would be dominated by Step 2., $O(M \times n^2)$. Storing the Cholesky factors requires $O(n^2)$ units of space.

Other classical algorithm's complexity characteristics are essentielly the same ([4], Chapter 11), and we conclude that the asymptotic computational cost of iterated projection is the same as the cost of well-known algorithms for discrete linear least-squares.

4 Discussion

I have presented a new idea, iterated projection, for computing the solution to continuous linear least-squares problems. The idea is to project the data—which is thought of as a function—into a sequence of function spaces of decreasing dimensionality. The final projection in this sequence is into the space in which the solution is sought.

As discussed in Section 3.2, iterated projection is asymptotically as efficient as other linear least-squares algorithms. However, ITERATED_PROJECTION_LSQ is a strikingly simple algorithm, consisting essentially of a single operation (the matrix-vector product in Step 6) and is arguably easier to accelerate by special hardware. Secondly, the way the computation is organized in ITERATED_PROJECTION_LSQ makes it easy to distribute the work among multiple processors.

Classical linear least-squares algorithms and iterated projection do not solve exactly the same problem. The former solve the problem

$$\text{Find } \beta^* \text{ such that } \|A\,\beta - d\|_2 \text{ is minimal for } \beta = \beta^*, \tag{16}$$

iterated projection solves the problem (with f linear in β)

$$\text{Find } \beta^* \text{ such that } \int_\Omega (f(x;\beta) - d(x))^2\,\mathrm{dx} \text{ is minimal for } \beta = \beta^*. \tag{17}$$

In practice problem (17) is often approximated by a problem of the form (16), obtained through sampling d and f at selected points $\{x_i\}$ (cf. (1) to (3) in Section 1). In such situations, and when the subspace property (7) holds, iterated projection is a compelling alternative to discrete linear least-squares algorithms. It may give exact solutions (except for rounding errors), and it may require less computation when d is of appropriate form (e.g., functions in finite-element spaces; Step 2 in ITERATED_PROJECTION_LSQ needs to be adapted accordingly).

Another interesting aspect of iterated projection is that, since no linear system needs to be solved, the problems of underdetermined or singular systems do not occur. This recommends iterated projection for problems like least-squares based image reconstruction and segmentation [1, 2], where small regions often lead to underdetermined or singular least-squares problems.

5 Appendix–Iterated Projection with Quadratic Polynomials

As a concrete example, and to demonstrate an alternative to (15) for deriving the projection matrices, I consider problem (6) with

$$\begin{aligned} f_1(x) &= 1 \\ f_2(x) &= x_1 \\ f_3(x) &= x_2 \\ f_4(x) &= \frac{1}{2}x_1^2 \\ f_5(x) &= x_1 x_2 \\ f_6(x) &= \frac{1}{2}x_2^2. \end{aligned} \tag{18}$$

At first we verify that the space spanned by $f_{1,\ldots,}f_6$ is translation invariant. Let β represent any such function, $f(x;\,\beta) = \sum_{k=1}^{6} \beta_k\; f_k(x)$, x^0 be the origin of another coordinate system, and $f'(x';\,\beta')$ denote the representation with respect to the coordinate system centered at x^0. Function f' is the same as f if their values and that of all their derivatives are identical at any point. Choose x^0 as that point and write down the identities to obtain β' in terms of β and x^0.

$$\begin{aligned} \beta_1' &= \beta_1 + \beta_2\,x_1^0 + \beta_3\,x_2^0 + \frac{\beta_4}{2}\left(x_1^0\right)^2 + \beta_5\,x_1^0\,x_2^0 + \frac{\beta_6}{2}\left(x_2^0\right)^2 \\ \beta_2' &= \beta_2 + \beta_4\,x_1^0 + \beta_5\,x_2^0 \\ &\ldots \end{aligned}$$

To derive matrix P_h in (11) we set x^0 to $\left(\frac{-h}{2}, \frac{-h}{2}\right)$, $\left(\frac{h}{2}, \frac{-h}{2}\right)$, $\left(\frac{h}{2}, \frac{h}{2}\right)$, and $\left(\frac{-h}{2}, \frac{h}{2}\right)$, respectively, and obtain

$$P_{h,1}=\begin{pmatrix} 1 & \frac{-h}{2} & \frac{-h}{2} & \frac{h^2}{8} & \frac{h^2}{4} & \frac{h^2}{8} \\ & 1 & & \frac{-h}{2} & \frac{-h}{2} & \\ & & 1 & & \frac{-h}{2} & \frac{-h}{2} \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix} \qquad P_{h,2}=\begin{pmatrix} 1 & \frac{h}{2} & \frac{-h}{2} & \frac{h^2}{8} & \frac{-h^2}{4} & \frac{h^2}{8} \\ & 1 & & \frac{h}{2} & \frac{-h}{2} & \\ & & 1 & & \frac{h}{2} & \frac{-h}{2} \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix}$$

$$P_{h,4}=\begin{pmatrix} 1 & \frac{-h}{2} & \frac{h}{2} & \frac{h^2}{8} & \frac{-h^2}{4} & \frac{h^2}{8} \\ & 1 & & \frac{-h}{2} & \frac{h}{2} & \\ & & 1 & & \frac{-h}{2} & \frac{h}{2} \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix} \qquad P_{h,3}=\begin{pmatrix} 1 & \frac{h}{2} & \frac{h}{2} & \frac{h^2}{8} & \frac{h^2}{4} & \frac{h^2}{8} \\ & 1 & & \frac{h}{2} & \frac{h}{2} & \\ & & 1 & & \frac{h}{2} & \frac{h}{2} \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 1 \end{pmatrix}.$$

Matrix Q_h in (13) can be seen to be

$$Q_h = \begin{pmatrix} h^2 & & & \frac{1}{24}h^4 & & \frac{1}{24}h^4 \\ & \frac{1}{12}h^4 & & & & \\ & & \frac{1}{3}h^4 & & & \\ \frac{1}{24}h^4 & & & \frac{1}{320}h^6 & & \frac{1}{576}h^6 \\ & & & & \frac{1}{144}h^6 & \\ \frac{1}{24}h^4 & & & \frac{1}{576}h^6 & & \frac{1}{320}h^6 \end{pmatrix} \tag{19}$$

These are the ingredients needed to derive R_h via (15).

5.1 Deriving R_h by QR Factorization

One algorithm for solving least-squares problem is by QR factorization. It consists of computing the (reduced) QR factorization of A in (4), $A = Q\,R$, by which we obtain an orthonormal basis for the column space of A. Next, the data vector is expanded in that orthonormal basis, and, finally, an expansion in the basis of interest (the columns of A) is obtained by solving a triangular system.

I now show how to derive R_h by applying essentially this algorithm, but in the continuous setting. The polynomials (18) do not form an orthogonal basis with respect to the inner product (12), otherwise Q_h in (19) were diagonal. An orthornomal basis for the space spanned by (18) are the Legendre polynomials up to order two. Defined over $\Omega = [-h, h] \times [-h, h]$, these are

$$\begin{aligned} \phi_1(x) &= \frac{1}{h}\frac{1}{2} \\ \phi_2(x) &= \frac{1}{h^2}\frac{\sqrt{3}}{2}x_1 \\ \phi_3(x) &= \frac{1}{h^2}\frac{\sqrt{3}}{2}x_2 \\ \phi_4(x) &= \frac{1}{h^3}\frac{\sqrt{5}}{4}\left(3\,x_1^2 - h^2\right) \\ \phi_5(x) &= \frac{1}{h^3}\frac{3}{2}x_1 x_2 \\ \phi_6(x) &= \frac{1}{h^3}\frac{\sqrt{5}}{4}\left(3\,x_2^2 - h^2\right). \end{aligned}$$

The “QR factorization” of $A = (\, f_1 \;\; f_2 \;\; f_3 \;\; f_4 \;\; f_5 \;\; f_6 \,)$ over Ω is

$$\begin{aligned} (\, f_1 \;\; f_2 \;\; f_3 \;\; f_4 \;\; f_5 \;\; f_6 \,) &= (\, \phi_1 \;\; \phi_2 \;\; \phi_3 \;\; \phi_4 \;\; \phi_5 \;\; \phi_6 \,) \begin{pmatrix} 2h & & & \frac{1}{3}h^3 & & \frac{1}{3}h^3 \\ & \frac{2}{\sqrt{3}}h^2 & & & & \\ & & \frac{2}{\sqrt{3}}h^2 & & & \\ & & & \frac{2}{3\sqrt{5}}h^3 & & \\ & & & & \frac{2}{3}h^3 & \\ & & & & & \frac{2}{3\sqrt{5}}h^3 \end{pmatrix} \\ &= \Phi\, C \end{aligned}$$

The product $R_h\tilde{\beta}$ may now be expressed as

$$\begin{aligned}
R_h\tilde{\beta} &= C^{-1}\sum_{i=1}^{4}\left(\int_{\Omega_i}\Phi^T\begin{pmatrix} f_1 & f_2 & f_3 & f_4 & f_5 & f_6\end{pmatrix}\mathrm{dx}\right)\beta_i \\
&= C^{-1}\sum_{i=1}^{4}\begin{pmatrix} \langle\phi_1,f_1\rangle_{h/2} & \langle\phi_1,f_2\rangle_{h/2} & \cdots & \langle\phi_1,f_6\rangle_{h/2} \\ \langle\phi_2,f_1\rangle_{h/2} & \langle\phi_2,f_2\rangle_{h/2} & & \vdots \\ \vdots & \vdots & & \langle\phi_5,f_6\rangle_{h/2} \\ \langle\phi_6,f_1\rangle_{h/2} & \langle\phi_6,f_2\rangle_{h/2} & \cdots & \langle\phi_6,f_6\rangle_{h/2}\end{pmatrix}\beta_i \\
&= C^{-1}\begin{pmatrix}\tilde{R}_{h,1} & \tilde{R}_{h,2} & \tilde{R}_{h,3} & \tilde{R}_{h,4}\end{pmatrix}\tilde{\beta},
\end{aligned}$$

hence $R_h = \begin{pmatrix} C^{-1}\tilde{R}_{h,1} & C^{-1}\tilde{R}_{h,2} & C^{-1}\tilde{R}_{h,3} & C^{-1}\tilde{R}_{h,4}\end{pmatrix}$. Evaluating the integrals in $\tilde{R}_{h,i}$ we get

$$\tilde{R}_{h,1} = \begin{pmatrix}
\frac{1}{2}h & 0 & 0 & \frac{1}{48}h^3 & 0 & \frac{1}{48}h^3 \\
-\frac{\sqrt{3}}{4}h & \frac{\sqrt{3}}{24}h^2 & 0 & -\frac{\sqrt{3}}{96}h^3 & 0 & -\frac{\sqrt{3}}{96}h^3 \\
-\frac{\sqrt{3}}{4}h & 0 & \frac{\sqrt{3}}{24}h^2 & -\frac{\sqrt{3}}{96}h^3 & 0 & -\frac{\sqrt{3}}{96}h^3 \\
0 & -\frac{\sqrt{5}}{16}h^2 & 0 & \frac{\sqrt{5}}{480}h^3 & 0 & 0 \\
\frac{3}{8}h & -\frac{1}{16}h^2 & -\frac{1}{16}h^2 & \frac{1}{64}h^3 & \frac{1}{96}h^3 & \frac{1}{64}h^3 \\
0 & 0 & -\frac{\sqrt{5}}{16}h^2 & 0 & 0 & \frac{\sqrt{5}}{480}h^3
\end{pmatrix}
\quad
\tilde{R}_{h,2} = \begin{pmatrix}
\frac{1}{2}h & 0 & 0 & \frac{1}{48}h^3 & 0 & \frac{1}{48}h^3 \\
\frac{\sqrt{3}}{4}h & \frac{\sqrt{3}}{24}h^2 & 0 & \frac{\sqrt{3}}{96}h^3 & 0 & \frac{\sqrt{3}}{96}h^3 \\
-\frac{\sqrt{3}}{4}h & 0 & \frac{\sqrt{3}}{24}h^2 & -\frac{\sqrt{3}}{96}h^3 & 0 & -\frac{\sqrt{3}}{96}h^3 \\
0 & \frac{\sqrt{5}}{16}h^2 & 0 & \frac{\sqrt{5}}{480}h^3 & 0 & 0 \\
-\frac{3}{8}h & -\frac{1}{16}h^2 & \frac{1}{16}h^2 & -\frac{1}{64}h^3 & \frac{1}{96}h^3 & -\frac{1}{64}h^3 \\
0 & 0 & -\frac{\sqrt{5}}{16}h^2 & 0 & 0 & \frac{\sqrt{5}}{480}h^3
\end{pmatrix}$$

$$\tilde{R}_{h,4} = \begin{pmatrix}
\frac{1}{2}h & 0 & 0 & \frac{1}{48}h^3 & 0 & \frac{1}{48}h^3 \\
-\frac{\sqrt{3}}{4}h & \frac{\sqrt{3}}{24}h^2 & 0 & -\frac{\sqrt{3}}{96}h^3 & 0 & -\frac{\sqrt{3}}{96}h^3 \\
\frac{\sqrt{3}}{4}h & 0 & \frac{\sqrt{3}}{24}h^2 & \frac{\sqrt{3}}{96}h^3 & 0 & \frac{\sqrt{3}}{96}h^3 \\
0 & -\frac{\sqrt{5}}{16}h^2 & 0 & \frac{\sqrt{5}}{480}h^3 & 0 & 0 \\
-\frac{3}{8}h & \frac{1}{16}h^2 & -\frac{1}{16}h^2 & -\frac{1}{64}h^3 & \frac{1}{96}h^3 & -\frac{1}{64}h^3 \\
0 & 0 & \frac{\sqrt{5}}{16}h^2 & 0 & 0 & \frac{\sqrt{5}}{480}h^3
\end{pmatrix}
\quad
\tilde{R}_{h,3} = \begin{pmatrix}
\frac{1}{2}h & 0 & 0 & \frac{1}{48}h^3 & 0 & \frac{1}{48}h^3 \\
\frac{\sqrt{3}}{4}h & \frac{\sqrt{3}}{24}h^2 & 0 & \frac{\sqrt{3}}{96}h^3 & 0 & \frac{\sqrt{3}}{96}h^3 \\
\frac{\sqrt{3}}{4}h & 0 & \frac{\sqrt{3}}{24}h^2 & \frac{\sqrt{3}}{96}h^3 & 0 & \frac{\sqrt{3}}{96}h^3 \\
0 & \frac{\sqrt{5}}{16}h^2 & 0 & \frac{\sqrt{5}}{480}h^3 & 0 & 0 \\
\frac{3}{8}h & \frac{1}{16}h^2 & \frac{1}{16}h^2 & \frac{1}{64}h^3 & \frac{1}{96}h^3 & \frac{1}{64}h^3 \\
0 & 0 & \frac{\sqrt{5}}{16}h^2 & 0 & 0 & \frac{\sqrt{5}}{480}h^3
\end{pmatrix}.$$

Multiplying $\tilde{R}_{h,i}$ by C^{-1} we finally arrive at the four components of R_h,

$$R_{h,1} = \begin{pmatrix}
\frac{1}{4} & \frac{5}{64}h & \frac{5}{64}h & \frac{1}{128}h^2 & 0 & \frac{1}{128}h^2 \\
-\frac{3}{8}\frac{1}{h} & \frac{1}{16} & 0 & -\frac{1}{64}h & 0 & -\frac{1}{64}h \\
-\frac{3}{8}\frac{1}{h} & 0 & \frac{1}{16} & -\frac{1}{64}h & 0 & -\frac{1}{64}h \\
0 & -\frac{15}{32}\frac{1}{h} & 0 & \frac{1}{64} & 0 & 0 \\
\frac{9}{16}\frac{1}{h^2} & -\frac{3}{32}\frac{1}{h} & -\frac{3}{32}\frac{1}{h} & \frac{3}{128} & \frac{1}{64} & \frac{3}{128} \\
0 & 0 & -\frac{15}{32}\frac{1}{h} & 0 & 0 & \frac{1}{64}
\end{pmatrix}
\quad
R_{h,2} = \begin{pmatrix}
\frac{1}{4} & -\frac{5}{64}h & \frac{5}{64}h & \frac{1}{128}h^2 & 0 & \frac{1}{128}h^2 \\
\frac{3}{8}\frac{1}{h} & \frac{1}{16} & 0 & \frac{1}{64}h & 0 & \frac{1}{64}h \\
-\frac{3}{8}\frac{1}{h} & 0 & \frac{1}{16} & -\frac{1}{64}h & 0 & -\frac{1}{64}h \\
0 & \frac{15}{32}\frac{1}{h} & 0 & \frac{1}{64} & 0 & 0 \\
-\frac{9}{16}\frac{1}{h^2} & -\frac{3}{32}\frac{1}{h} & \frac{3}{32}\frac{1}{h} & -\frac{3}{128} & \frac{1}{64} & -\frac{3}{128} \\
0 & 0 & -\frac{15}{32}\frac{1}{h} & 0 & 0 & \frac{1}{64}
\end{pmatrix}$$

$$R_{h,4} = \begin{pmatrix}
\frac{1}{4} & \frac{5}{64}h & -\frac{5}{64}h & \frac{1}{128}h^2 & 0 & \frac{1}{128}h^2 \\
-\frac{3}{8}\frac{1}{h} & \frac{1}{16} & 0 & -\frac{1}{64}h & 0 & -\frac{1}{64}h \\
\frac{3}{8}\frac{1}{h} & 0 & \frac{1}{16} & \frac{1}{64}h & 0 & \frac{1}{64}h \\
0 & -\frac{15}{32}\frac{1}{h} & 0 & \frac{1}{64} & 0 & 0 \\
-\frac{9}{16}\frac{1}{h^2} & \frac{3}{32}\frac{1}{h} & -\frac{3}{32}\frac{1}{h} & -\frac{3}{128} & \frac{1}{64} & -\frac{3}{128} \\
0 & 0 & \frac{15}{32}\frac{1}{h} & 0 & 0 & \frac{1}{64}
\end{pmatrix}
\quad
R_{h,3} = \begin{pmatrix}
\frac{1}{4} & -\frac{5}{64}h & -\frac{5}{64}h & \frac{1}{128}h^2 & 0 & \frac{1}{128}h^2 \\
\frac{3}{8}\frac{1}{h} & \frac{1}{16} & 0 & \frac{1}{64}h & 0 & \frac{1}{64}h \\
\frac{3}{8}\frac{1}{h} & 0 & \frac{1}{16} & \frac{1}{64}h & 0 & \frac{1}{64}h \\
0 & \frac{15}{32}\frac{1}{h} & 0 & \frac{1}{64} & 0 & 0 \\
\frac{9}{16}\frac{1}{h^2} & \frac{3}{32}\frac{1}{h} & \frac{3}{32}\frac{1}{h} & \frac{3}{128} & \frac{1}{64} & \frac{3}{128} \\
0 & 0 & \frac{15}{32}\frac{1}{h} & 0 & 0 & \frac{1}{64}
\end{pmatrix}.$$

Bibliography

[1] P. J. Besl and R. C. Jain. Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2):167–192, March 1988.

[2] T. Kanungo, B. Dom, W. Niblack, and D. Steele. A fast algorithm for MDL-based multi-band image segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 609–616, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[3] Y. G. Leclerc. Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3:73–102, May 1989.

[4] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, PA, 1997.