Appendix to Article “A Functional Estimate of Covariation”

J. O. Ramsay

June 23, 2015

Finite element bases for surfaces owe their importance in numerical analysis to the sparsity of matrices containing inner products, but the price to pay for this is the complexity of program code that computes the nonzero entries. Fortunately, the price is paid by the programmer, so that the user is spared this effort and can expect relatively fast computation. The computational effort of computing the matrices $\mathbf{R}(s_{ij}, t_{ij})$ is only required once, rather than at each step of an optimization algorithm. In this appendix the essential computational details are discussed as well as the design of code in Matlab and R that can be used to estimate FE- Σ λ and σ surfaces.

The computation of $\mathbf{R}(s, t)$ can be implemented as follows, assuming for simplicity of notation that $T = I$ and therefore that $\delta = 1$. For given s and t , define i_s and i_t as the largest integers satisfying $i_s \leq s$ and $i_t \leq t$, respectively; with the exception that if $s = T$ then $i_s = I - 1$, and correspondingly for i_t . The natural way to assemble the integral is to integrate the product $\phi_{k_1\ell_1}\phi_{k_2\ell_2}$ of two basis functions over each of the vertical strips of width δ in turn, and then sum these integrals. Of course, a diagonal boundary of the parallelogram may also be within a vertical strip, as is the case for the upper hexagon over $w \in [1, 2]$ in Figure 1, and this also affects the integration.

The hexagonal support of the basis functions implies that the necessary conditions for $r_{k_1\ell_1 k_1\ell_1}(s, t) \neq 0$ are

- s is contained within the vertical extent of the support of $\phi_{k_1\ell_1}$,
- t is contained within the vertical extent of the support of $\phi_{k_2\ell_2}$, and
- $\phi_{k_1\ell_1}$ and $\phi_{k_2\ell_2}$ have overlapping support at levels s and t , respectively.

These conditions are illustrated in Figure 1 where the lower dashed s -line passes through the lower hexagon, the upper dashed t -line passes through upper hexagon, and both hexagons are nonzero only over $w \in [0, 2]$. However, we see, because of the values of s and t in this case, that the product of the basis function values will only be nonzero over $w \in [0.5, 1.5]$; and, moreover, the integrand for this inner product will be piecewise quadratic with a break in slope at $w = 1$.

Let w_j be equal to the left boundary of the j th vertical strip, counting from the right. We call the point $i_s + 1, j + 1$ and $i_t + 1, j + 1$ the *base points* for the basis functions $\phi_{k_1\ell_1}$

Figure 1: The mesh associated with $I = T = 4$ and $J = 3$. The dashed horizontal lines indicate the path of integration over w at $s = 1.5$ and $t = 2.5$. Each of the two hexagons, plotted in thick solid lines, contain one of the dashed lines, and therefore the product of the two hexagons when evaluated at the corresponding values of s and t are nonzero, in this case over the interval $w \in [0.5, 1.5]$.

and $\phi_{k_2\ell_2}$, respectively. These two hexagons satisfy the overlap conditions, and so do those resulting from shifting either to the right by δ , and/or by shifting either vertically by δ . In Figure 1, for example, the lower hexagon is centred on its base point for the interval $[1,2]$, but the upper triangle is shifted up off its base point by 1. These shifts imply that there are 16 basis function pairs in total for each vertical strip for which the integration has to be computed. For each such integration, one sees that the integrand will be positive for from 0 to 3 segments within any vertical strip, depending on the centres of the two hexagons and whether or not their domains are intersected by one or the other diagonal boundary of the parallelogram. Finally, there are two values of δ_s and δ_t , so that there are $16 \times 3 \times 2 = 96$ possible integrands. Although each of these integrations are relatively elementary, the large number involved argues for the use of symbolic computation software, and Matlab code for both the algebraic computations are distributed with the packages in Matlab and R whose releases will accompany this paper.

When the sampling points $t_i, i = 1, \dots, n$ are shared across records, it is feasible to store the $n(n+1)/2$ matrices $\mathbf{R}$ in a cell array (Matlab) or a list (R), and for n up to 50 or so on computers equipped with larger amounts of RAM by current standards. Beyond this, however, and almost certainly in the case where the sampling points are unbalanced, these will have to be lodged in disk storage and retrieved each time needed. This was the case, for example, for the Fels growth data analyses in Section ??, where the matrices for each record were stored in a single cell array. The time required to retrieve these from disk storage was much less than the time required to re-compute them each on each occasion that they were needed. An analyses of the Fels data required a few hours on a fairly

Figure 2: The $\mathbf{R}$ matrices for the mesh defined by $I = T = 2$ and $J = 1$ as plotted by the `spy` function in Matlab. Each dot is in the position of a nonzero entry in the matrix resulting from evaluation points 0, 0.5, ..., 2.

powerful desktop computer.

However, in the special case where the evaluation points resonate with the I intervals in the sense that they are equally spaced and $n = mI + 1$ for some m , great savings are possible when $J < I$, since it is only required that the matrices be computed over the J by J mesh. Figure 2 shows the result of using the `spy` function in Matlab, which plots a symbol at the location of each nonzero value, so that the upper left corner corresponds to matrix index pair (1,1). The mesh is the simple case $I = T = 2$ and $J = 1$, and the five evaluation points are 0.5 units apart. The 25 $\mathbf{R}$ matrices corresponding to the possible pairs (s, t) can all be assembled from the three matrices in the (1,1), (2,1) and (2,2) positions in the array since the values as well as patterns in all other arrays can be found in one of these. What differs from matrix to matrix is only the positioning of the pattern, a variation that is obvious in the diagonal matrices each of which is shifted by one unit along the diagonal from its predecessor. This process is especially useful if a high resolution surface rendering of a strongly band-structured covariance is needed.