Combinatorial structure in evaluating finite element operators

Robert C. Kirby[*]

Combinatorial issues related to finite element computation include sparse matrices (representation, factorization, bandwidth, probing and structure for preconditioning) and meshing (generation, partitioning, parallelization). These techniques are not specific to finite elements and widely applicable to many different numerical methods. Beyond these fairly standard applications, however, it turns out that combinatorial structure can also be exploited to dramatically reduce the arithmetic cost of evaluting the algebraic operators associated with finite element methods.

What is the inherent arithmetic cost of constructing the local stiffness matrix? For a large class of PDE, the computation of a local element matrix can be viewed as a fixed linear transformation acting on a vector that encodes the element's geometry and material properties. For affine finite elements, this linear transformation is fixed for all elements in the mesh once the variational form and finite element basis are fixed. Moreover, the matrix is rich in structure that allows it to be applied to any vector in fewer than n^2 operations. For example, some rows of the matrix may be zero or only contain a few nonzero entries each. Sometimes two more rows will share certain properties, such as sharing many nonzero entries or being colinear in Euclidean space. Each of these cases allows the inner product of one row of the matrix with a target vector to be computed efficiently using the result of the other one. In some cases, three or more rows of the matrix may lie in some low-dimensional Euclidean plane. This linear dependence also can be used to apply the linear transformation more efficiently.

These various situations are modeled by appropriate combinatorial structures. Dependencies between rows i and j of the matrix, called *complexity-*

[*]Department of Mathematics and Statistics, Texas Tech University; MS 1042; Lubbock, TX 79410

reducing relations are encoded by a graph in which the edge weights $d(i, j)$ indicate the cost of computing the inner product of row i with an arbitrary vector once the inner product of j with that vector is known. A minimum spanning tree of this graph provides a low-arithmetic algorithm for applying the matrix to an arbitrary vector.

Geometric dependencies between three or more vectors are not naturally modeled by a graph. In fact, the incidence relation between the rows of the matrix and Eucliden subspaces can be viewed as a *finite linear space*. When many of the rows lie in such linear dependences, it is desirable to locate some small set of rows that can be used to *generate* the remaining rows. Both detection of linear dependence among the rows of the matrix and constructing such a generator are interesting algorithmic problems. While searching for d-dimensional linear dependence among n vectors by performing Gaussian elimination on $(d+1)$-tuples requires $O(n^{d+1})$ work, it is possible to do this for most data sets in $O(n^d)$ by a randomized algorithm. While the complexity of determining a geometric generator for a finite linear space is yet unknown, an algorithm that seems at least to provide suitable approximations is presented.

All of the effort in locating and finding an algorithm for applying the matrix to a vector is naturally considered as a compile-time process, performed once per combination of variational form and finite element basis. In fact, it is possible to set up an offline process that searches for an efficient algorithm based on this combinatorial information and generates relatively low-level code. For example, these algorithms have been implemented in a code called `FErari`, which serves an optimizing backend to the `FEniCS Form Compiler`. This toolchain will be described at the end of the talk.