ELSEVIER Finite Elements in Analysis and Design 16 (1994) 175–189

FINITE ELEMENTS IN ANALYSIS AND DESIGN

Implementing and using high-order finite element methods[1]

Babak Bagheri[a], L. Ridgway Scott[b,*], Shangyou Zhang[c]

[a] *Computer Science Department, Penn State University, USA*
[b] *Department of Mathematics, University of Houston, University Park, Houston, TX 77204, USA*
[c] *Department of Mathematics, University of Delaware, USA*

Abstract

We Present theoretical analyses of and detailed timings for two programs which use high-order finite element methods to solve the Navier Stokes equations in two and three dimensions. The analyses show that algorithms popular in low-order finite element implementations are not always appropriate for high-order methods. The timings show that with the proper algorithms high-order finite element methods are viable for solving the Navier Stokes equations. We show that it is more efficient, both in time and storage, *not* to precompute element matrices as the degree of approximating functions increases. We also study the cost of assembling the stiffness matrix versus directly evaluating bilinear forms in two and three dimensions. We show that it is more efficient *not* to assemble the full stiffness matrix for high-order methods in some cases. We consider the computational issues with regard to both Euclidean and isoparametric elements. We show that isoparametric elements may be used with higher-order elements without increasing the order of computational complexity.

1. Introduction

High-order methods provide efficient ways to model complex physical phenomena such as incompressible fluid flows. However, computer codes for high-order finite elements are significantly more challenging to develop. We present some results obtained with two codes, fec and nmg, that use high-order finite element methods, and we discuss some of the design issues that affected the development of the codes, focusing on those issues related to high-order finite elements. We illustrate the use of fec applied to the Navier–Stokes equations in two dimensions and give some results obtained with nmg, which implements nonnested multilevel iterative techniques in three dimensions for high-order elements.

* Corresponding author.
[1] This work was supported in part by the National Science Foundation through award number DMS-9105437.

fec is written in the C++ programming language and uses its facilities to simplify implementation and to minimize the cost of maintaining and extending the code [1]. The data abstraction mechanisms of C++ make using fec essentially like using a separate language for finite elements, in that objects like interpolants, bilinear forms, and norms are explicitly available.

nmg is an implementation (in Fortran) of the algorithms discussed in [2]. It is an interactive code that allows one to experiment with different multilevel iterative methods for solving the linear system associated with finite element methods for elliptic problems in three dimensions.

The use of higher-order elements affects implementation strategies for computer codes in several ways. Key issues include the generation of basis functions and quadrature rules and the evaluation of bilinear forms. Whereas different approaches have essentially the same order of complexity for low-order elements, they become strongly differentiated as the degree of approximation increases. We present some analysis of the different algorithms that can be used to evaluate bilinear forms that was influential in the development of fec and nmg.

One key to the success of the finite element method as developed in engineering practice was the systematic way that computer codes for it could be implemented. One important step in this process is the *assembly* of integrated differential forms by summing their constituent parts over each *element*, which are computed separately. There are two issues to consider. First, the element-level forms must be computed efficiently. Second, one must decide, say for a bilinear form, whether to compute and store a global stiffness matrix or to compute global forms just by summing element-level forms at each occurrence.

We study two different algorithms for evaluating element-level multilinear forms arising in the approximation of the Navier–Stokes equations. This system can be viewed as a prototype as it involves both bilinear and trilinear forms. (In general, κ-linear forms are considered for arbitrary κ.) We show that it can be more efficient, both in time and storage, *not* to use element matrices as the degree of approximating functions increases.

In evaluating bilinear forms, one has the choice of forming the full stiffness matrix, or not. Once the stiffness matrix is formed, evaluating bilinear forms becomes more efficient than using element-level forms. We study the cost of assembly of the stiffness matrix versus direct evaluation of bilinear forms in two and three dimensions. We show that it can be more efficient *not* to assemble the full stiffness matrix for high-order methods in some cases.

We consider the computational issues with regard to both Euclidean and isoparametric elements. We show that isoparametric elements may be used with higher-order elements without increasing the order of computational complexity.

2. Evaluation of local bilinear forms

The *assembly* of integrated differential forms by summing their constituent parts over each *element*, which are computed separately, is facilitated through the use of a numbering scheme called the *local-to-global* index. This index, $\iota(e,\lambda)$, relates the local (or element) node number, $\lambda\in\mathscr{L}$, on a particular element, indexed by e, to its position in the global data structure.

We may write the Lagrange interpolant of a continuous function for the space of all piecewise polynomial (of some degree) functions (no boundary conditions imposed) via

$$f_I := \sum_{e} \sum_{\lambda \in \mathscr{L}} f(x_{n(e,\lambda)})\phi_\lambda^e, \tag{2.1}$$

where $\{\phi_\lambda^e : \lambda \in \mathscr{L}\}$ denotes the set of basis functions for polynomials of the given degree on T_e. We can relate all of the "element" basis functions ϕ_λ^e to a fixed set of basis functions on a "reference" element, $\mathscr{T}$, via an affine mapping, $\xi \to J\xi + x_e$, of $\mathscr{T}$ to T_e:

$$\phi_\lambda^e(x) = \phi_\lambda(J^{-1}(x - x_e)).$$

(By definition, the element basis functions, ϕ_λ^e, are extended by zero outside T_e.) The inverse mapping, $x \to \xi = J^{-1}(x - x_e)$ has as its Jacobian

$$J_{mj}^{-1} = \frac{\partial \xi_m}{\partial x_j}$$

and this is the quantity which appears in the evaluation of the bilinear forms. Of course, $\det J = 1/\det J^{-1}$.

All of the multilinear forms arising in the variational formulation of differential equations are easily evaluated (assembled) using this representation. We consider the Navier–Stokes equations (in d dimensions) as they involve both bilinear and trilinear forms. For example,

$$a(\boldsymbol{v}, \boldsymbol{w}) = \sum_e a_e(\boldsymbol{v}, \boldsymbol{w}),$$

where the "element" bilinear form is defined via

$$\begin{aligned} a_e(\boldsymbol{v}, \boldsymbol{w}) &:= \int_{T_e} \nabla \boldsymbol{v}(x) : \nabla \boldsymbol{w}(x)\,\mathrm{d}x \\ &= \int_{\mathscr{T}} \sum_{j,k=1}^{d} \frac{\partial}{\partial x_j} v_k(J\xi + x_e) \frac{\partial}{\partial x_j} w_k(J\xi + x_e) \det(J)\,\mathrm{d}\xi \\ &= \sum_{k=1}^{d} \begin{pmatrix} v_k^{n(e,1)} \\ \vdots \\ v_k^{n(e,|\mathscr{L}|)} \end{pmatrix}^{t} \boldsymbol{K}^e \begin{pmatrix} w_k^{n(e,1)} \\ \vdots \\ w_k^{n(e,|\mathscr{L}|)} \end{pmatrix}, \end{aligned}$$

where the *element stiffness matrix*, $\boldsymbol{K}^e$, is given by

$$K_{\lambda,\mu}^e := \sum_{j,m,m'=1}^{d} \frac{\partial \xi_m}{\partial x_j} \frac{\partial \xi_{m'}}{\partial x_j} \det(J) \int_{\mathscr{T}} \frac{\partial}{\partial \xi_m} \phi_\lambda(\xi) \frac{\partial}{\partial \xi_{m'}} \phi_\mu(\xi)\,\mathrm{d}\xi$$

for $\lambda, \mu \in \mathscr{L}$. Note that we have identified the space of piecewise polynomials functions, $\boldsymbol{v}$, with the vector space of values, $(\boldsymbol{v}^i)$, at the nodes. The first (and usual) local-matrix method for computing a_e is to use the above formula directly and precomputing the integral in the last formula once and for all.

The nonlinear term in the Navier–Stokes equation can also be computed with this first method using "element" trilinear forms.

$$
\begin{aligned}
c_e(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) &:= \int_{T_e} \boldsymbol{u}\cdot\nabla\boldsymbol{v}(x)\cdot\boldsymbol{w}(x)\,\mathrm{d}x \\
&= \sum_{j,k=1}^{d} \sum_{\lambda,\mu,\rho\in\mathcal{S}} u_j^{\iota(e,\lambda)} v_k^{\iota(e,\mu)} w_k^{\iota(e,\rho)} \sum_{m=1}^{d} \frac{\partial\xi_m}{\partial x_j}\det(J)\,N_{\lambda,\mu,\rho,m} \\
&= \sum_{j,k=1}^{d} \sum_{\lambda,\mu,\rho\in\mathcal{S}} u_j^{\iota(e,\lambda)} v_k^{\iota(e,\mu)} w_k^{\iota(e,\rho)} N^e_{\lambda,\mu,\rho,j},
\end{aligned}
$$

where

$$
N_{\lambda,\mu,\rho,m} := \int_{\mathcal{T}} \phi_\lambda(\xi)\frac{\partial}{\partial\xi_m}\phi_\mu(\xi)\phi_\rho(\xi)\,\mathrm{d}\xi, \qquad N^e_{\lambda,\mu,\rho,j} := \sum_{m=1}^{d}\frac{\partial\xi_m}{\partial x_j}\det(J)\,N_{\lambda,\mu,\rho,m}.
$$

There is another way to evaluate multilinear forms that can be more efficient both in time and storage than the above approach using element matrices. Let us return to the example involving $a(\cdot,\cdot)$. For this second method, we simple replace integration by quadrature as follows.

$$
\begin{aligned}
a_e(\boldsymbol{v},\boldsymbol{w}) &= \int_{\mathcal{T}} \sum_{j,k=1}^{d}\left((J^{-1})^{t}\sum_{\lambda\in\mathcal{S}} v_k^{\iota(e,\lambda)}\nabla\phi_\lambda(\xi)\right)_j\left((J^{-1})^{t}\sum_{\lambda\in\mathcal{S}} w_k^{\iota(e,\lambda)}\nabla\phi_\lambda(\xi)\right)_j\det(J)\,\mathrm{d}\xi \\
&= \sum_{\xi\in\Xi}\omega_\xi\sum_{j,k=1}^{d}\left((J^{-1})^{t}\sum_{\lambda\in\mathcal{S}} v_k^{\iota(e,\lambda)}\nabla\phi_\lambda(\xi)\right)_j\left((J^{-1})^{t}\sum_{\lambda\in\mathcal{S}} w_k^{\iota(e,\lambda)}\nabla\phi_\lambda(\xi)\right)_j\det(J).
\end{aligned}
$$

The work estimate for this second method is of the order $|\Xi|\times|\mathcal{S}|$ whereas the first method is of the order $|\mathcal{S}|^2$. Depending on the number of quadrature points versus the number of basis functions, this approach may be more or less efficient. However, when we extend it to apply to the trilinear form $c(\cdot,\cdot)$ it becomes clearly superior. In this case we have

$$
c_e(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) = \sum_{\xi\in\Xi}\omega_\xi\sum_{j,k=1}^{2}\left(\sum_{\lambda\in\mathcal{S}} u_j^{\iota(e,\lambda)}\phi_\lambda(\xi)\right)\left((J^{-1})^{t}\sum_{\lambda\in\mathcal{S}} v_k^{\iota(e,\lambda)}\nabla\phi_\lambda(\xi)\right)_j\left(\sum_{\lambda\in\mathcal{S}} w_k^{\iota(e,\lambda)}\nabla\phi_\lambda(\xi)\right)\det(J)
$$

The work estimate for the second method remains of the order $|\Xi|\times|\mathcal{S}|$ whereas the first method is of the order $|\mathcal{S}|^3$. In general, the first method applied to a κ-linear form will have a work estimate of the form $\kappa\times|\Xi|\times|\mathcal{S}|$ whereas the first method would be of the order $|\mathcal{S}|^\kappa$.

Theorem 2.1. *The local κ-linear forms can be computed in an amount of work proportional to the product of κ, the number of quadrature points, and the number of degrees of freedom of the local basis functions.*

3. Evaluation of bilinear forms for isoparametrics elements

When using high-order elements for problems with curved boundaries, it is essential to include some way of approximating the boundary conditions accurately. One of the most effective in engineering practice is to use isoparametric elements. In this approach, the element basis functions ϕ^e_λ are related to the reference basis functions via a polynomial mapping, $\xi \to F(\xi)$, of $\mathcal{T}$ to T_e:

$$\phi^e_\lambda(x) = \phi_\lambda(F^{-1}(x)).$$

The use of element matrices becomes quite complicated with isoparametric elements. Let us return to the example involving $a(\cdot,\cdot)$. Replacing integration with quadrature, we have

$$a_e(\mathbf{v},\mathbf{w}) = \sum_{\xi\in\Xi} \omega_\xi \sum_{j,k=1}^{d} \left(J_F^{-1}(\xi)^t \sum_{\lambda\in\mathscr{L}} v_k^{\iota(e,\lambda)} \nabla\phi_\lambda(\xi)\right)_j \left(J_F^{-1}(\xi)^t \sum_{\lambda\in\mathscr{L}} w_k^{\iota(e,\lambda)} \nabla\phi_\lambda(\xi)\right)_j \det(J_F(\xi)).$$

Writing this in terms of element matrices is quite similar to using trilinear forms, i.e., $a_e(u,v) = a_e(F;u,v)$. Typically, we would write

$$F_k(\xi) = \sum_{\rho\in\mathscr{L}} F_k^{\iota(e,\rho)} \phi_\rho(\xi)$$

so that

$$J_F(\xi)_{k,m} = \frac{\partial}{\partial\xi_m} F_k(\xi) = \sum_{\rho\in\mathscr{L}} F_k^{\iota(e,\rho)} \frac{\partial}{\partial\xi_m} \phi_\rho(\xi).$$

The above expression for a_e is then a trilinear expression in v_k, w_k and F_k. The cost of evaluating this in terms of element matrices would thus be of the order of $|\mathscr{L}|^3$.

The second method for evaluating multilinear forms is more efficient than the first method (using element matrices) for isoparametric elements. The cost of evaluating $J_F(\xi)$ as above is only of the order of $|\Xi|\cdot|\mathscr{L}|$. The amount of work involved in the inversion and transpose of J is independent of $|\mathscr{L}|$. Therefore, the work estimate for the first method remains of the order $|\Xi|\cdot|\mathscr{L}|$. Applying this method to the trilinear form $c(\cdot,\cdot,\cdot)$, we again have a work estimate of order $|\Xi|\cdot|\mathscr{L}|$. Thus, the asymptotic work estimate does not change with the introduction of isoparametric elements.

Theorem 3.1. *The local κ-linear forms for isoparametric elements can be computed in an amount of work proportional to the product of κ, the number of quadrature points, and the number of degrees of freedom of the local basis functions.*

4. Evaluation of bilinear forms using a global matrix

The evaluation of a global bilinear form such as $a(\mathbf{v},\mathbf{w})$ can be done via a global stiffness matrix, A, as

$$a(\mathbf{v},\mathbf{w}) = \sum_{i,j,k,l} A^{k,l}_{i,j} v^i_k w^j_k,$$

where $A^{k,l}_{i,j}$ is defined in the obvious way. Here $k,l = 1,\ldots,d$ and i and j range over the set, $\mathscr{I}$, of global indices $\iota(e,\lambda)$. Thus, it is a $|\mathscr{I}|\times|\mathscr{I}|$ block matrix, with $d\times d$ blocks. (Since there are no nonzero cross terms, $v^i_k w^j_l$, for $k \neq l$ these blocks are all diagonal, in fact, a constant times the identity.)

The first method involved an amount of work proportional to $|\mathcal{L}|^2 \times |\mathcal{E}|$ where $\mathcal{E}$ is the set of all elements, independent of whether a local element matrix is used or not.

Due to the sparseness of A, the cost of using a global matrix is more difficult to estimate. It can be done in $|A|$ operations, where $|A|$ denotes the number of nonzeros. However, it is easy to see that this is also proportional to $|\mathcal{L}|^2 \times |\mathcal{E}|$ since the entries corresponding to the interior nodes in each element are of this order. Therefore, both approaches offer asymptotically the same efficiency for higher-order methods.

4.1. *The two-dimensional case*

In two dimensions, the work to evaluate $a(v, w)$ using element matrices is precisely $d \times |\mathcal{L}|^2 \times |\mathcal{E}|$. If we are dealing with polynomials of degree p, then $|\mathcal{L}| = \frac{1}{2}(p+1)(p+2)$. Thus, the total work is precisely

$$d \times \tfrac{1}{4}(p+2)^2(p+1)^2|\mathcal{E}| = d \times |\mathcal{E}|(\tfrac{1}{4}p^4 + 6p^3 + \mathcal{O}(p^2)).$$

The number of nonzeros in A can be estimated using standard counting techniques which relate the number of triangles, edges and vertices in a triangulation (cf. [3]). Here we quote the result and refer the reader to the report [4] for a proof.

Proposition 4.1. *In two dimensions, the number of nonzero entries in A is $d \times |\mathcal{E}| \times G(p)$ where*

$$G(p) = \tfrac{1}{2} + \tfrac{3}{2}p(p+1) + \tfrac{3}{2}(p-1)(p+1)^2 + \tfrac{1}{4}(p-1)(p-2)(p+1)(p+2).$$

Therefore the work estimates for a form evaluation for the two approaches agree exactly to order p^4. However, in computing the global matrix, one must do an extra evaluation using element matrices just to compute the global matrix. Let $L(p)$ denote the coefficient of $d \times |\mathcal{E}|$ in the work estimates for the technique using element matrices only:

$$L(p) = \tfrac{1}{4}(p+1)^2(p+2)^2.$$

Thus, the cost of r evaluations of $a(u, v)$ using the global approach exceeds that of the local approach when

$$rG(p) + L(p) > rL(p),$$

or equivalently when

$$r < \frac{L(p)}{L(p) - G(p)} = 1 \Big/ \left(1 - \frac{G(p)}{L(p)}\right). \tag{4.1}$$

Note that $G(p)/L(p)$ tends to one as p increases.

In the piecewise linear case, we have $G(1)/L(1) = 0.389$, so the global approach is cheaper as soon as $r \geqslant 2$. However, for $p = 4$, the global approach is more expensive for $r \leqslant 6$. For $p = 10$, it is more expensive for $r \leqslant 24$. If one is using a multilevel iterative scheme to solve the linear systems related to the bilinear form, then it can be expected that only a small number of evaluations of $a(\cdot,\cdot)$ will ever be done.

Simplifying the above estimates yields the following.

Theorem 4.2. *Suppose that $d = 2$. If the number of bilinear form evaluations, r, satisfies*

$$r \leqslant (p+2)^2/6,$$

where p is the polynomial degree of finite elements, then it is cheaper not *to form the global stiffness matrix, but rather to compute the forms directly by summing the local forms at each occurrence.*

4.2. *The three-dimensional case*

In the three-dimensional case, the techniques are more involved but similar in spirit. The work to evaluate $a(v, w)$ using element matrices is, for polynomials of degree p, precisely $d \times D_p^2 \times T$, where T is the number of tetrahedra in the triangulation and $D_p = (p+3)(p+2)(p+1)/6$ is the dimension of the space of polynomials of degree p in three dimensions. (We drop the common factor of $d = 3$ in all of the computations here.) Let F, E and V denote the number of faces, edges and vertices, respectively, in the triangulation.

The major difference from the two-dimensional case is that we no longer have a simple relation between V and T. As in the two-dimensional case, one relation is provided by Euler's formula:

$$T - F + E - V = c,$$

where c depends only on the topology of the domain. Putting two marbles on each face and then moving each marble into opposing tetrahedra shows that

$$F \approx 2T.$$

Combining these two expressions shows that

$$E \approx T + V.$$

We can no longer find a simple relation between V and T. On a regular mesh based on boxes having six tetrahedra per box, we would have $T \approx 6V$. On the other hand, we could equally well have a regular mesh based on boxes having five tetrahedra per box, with $T \approx 5V$. In either case, the contribution from the vertices in the expression above for E is small.

For each vertex node, v, there will be a nonzero for every node within the *star* of the vertex, i.e., the union of the tetrahedra meeting at v. Let the number of such tetrahedra be denoted by $N_T(v)$. Moving four marbles from each tetrahedron to its vertices shows that

$$\sum_v N_T(v) \approx 4T,$$

where T is the number of tetrahedra. Let $N_F(v)$ and $N_E(v)$ be the numbers of faces and edges meeting at v. Note that the number, $N_V(v)$, of vertices in the star of v (not including v) is equal to $N_E(v)$.

For v in the interior of Ω, the boundary of the star of v is topologically a sphere. Thus, we can count the numbers of tetrahedra, faces and edges meeting at v as follows. Each tetrahedron in the star of v corresponds to a face on the boundary of the star, each face meeting at v corresponds to an edge in the corresponding triangulation of the boundary of the star, and each edge corresponds to a boundary vertex. Thus using Euler's formula on the boundary of the star, we have

$$N_T(v) - N_F(v) + N_E(v) = 2.$$

Using marbles again, on the boundary of the star only, we find

$$2N_F(v) = 3N_T(v).$$

Therefore,

$$N_F(v) = \tfrac{3}{2}N_T(v), \qquad N_V(v) = N_E(v) = \tfrac{1}{2}N_T(v) + 2.$$

Summing the expressions above, we find

$$\sum_v N_F(v) \approx 6T, \qquad \sum_v N_V(v) = \sum_v N_E(v) \approx 2T + 2V.$$

Suppose each edge, e, is shared by $N_T(e)$ tetrahedra. Again by using marbles on the boundary of the star $\sigma(v)$ of v, we find

$$\sum_{e\in\sigma(v)} N_T(e) = 3N_T(v),$$

where $\sigma(v)$ is the star of v. Moreover, the sum of $N_T(e)$ over all e in the entire triangulation is $6T$, i.e.,

$$\sum_e N_T(e) \approx 6T,$$

since

$$12T \approx 3\sum_v N_T(v) = \sum_v \sum_{e\in\sigma(v)} N_T(e) = 2\sum_e N_T(e).$$

The latter equality just says that every edge has two vertices.

The nonzero matrix entries associated with a vertex v are all those contained in the star of the vertex. To count these, we take the union of all the local nodes (excluding v) over all the tetrahedra in the star of v, then account for duplications. Nodes on each edge e will be counted $N_T(e)$ times, so we must reduce the count by $(N_T(e) - 1)p$ for each e. Summing this over all edges in the star gives a reduction of $(3N_T(v) - N_E(v))p$. The number of redundant nodes on any face and not on the edges already considered is simply $\frac{1}{2}(p-1)(p-2) + (p-1)$ which counts the nodes in the interior of the face plus the nodes on the interior of the edge lying on the boundary of the star. Accounting for duplicate representation for the vertex, edge and face nodes in the star of v, the number of nonzeros is seen to be

$$1 + N_T(v)\times(D_p - 1) - (3N_T(v) - N_E(v))\times p - N_F(v)\times\tfrac{1}{2}(p-1)^2.$$

Thus, the total number of nonzero matrix entries in rows indexed by vertex nodes is equal to

$$V\times(2p+1) + T\times(4D_p - 3p^2 - 4p - 7).$$

The number of edge contributions for a given edge, e, is the number of edge nodes in the interior of the edge times the number of all nodes on the edge plus, for each tetrahedron meeting there, all local nodes for a tetrahedron excluding one full face:

$$(p-1)\times\{p+1+N_T(e)[D_p-\tfrac{1}{2}(p+1)(p+2)]\}.$$

Summing this over all edges, we find the total number of edge contributions is approximately

$$E(p^2-1)+(p-1)6T[D_p-\tfrac{1}{2}(p+1)(p+2)].$$

The number of contributions from face nodes is

$$T(p-1)(p-2)\times[2D_p-\tfrac{1}{2}(p+1)(p+2)]$$

(the total number of faces is about $2T$). The latter term accounts for the nodes in the two tetrahedra meeting at each face, excluding the duplications on the common face. Finally, the number of contributions from interior nodes is exactly

$$TD_{p-4}D_p.$$

The total number of all nonzeros in A is the sum of the above contributions and simplifies to

$$(p^2+2p)V+[(D_p)^2\tfrac{1}{2}(p^4+6p^3+13p^2+2p+6)].$$

On a regular mesh based on boxes having six tetrahedra per box, we would have $T\approx 6V$. On the other hand, we could equally well have a regular mesh based on boxes having five tetrahedra per box, with $T\approx 5V$. In either case, the contribution from the $(p^2+2p)V$ term is negligible.

The comparison of the direct approach versus the global stiffness matrix is dramatic in three dimensions. For $p=1$, it is $3V+2T$ for the latter versus $16T$ for the former, or nearly a factor of eight in favor of using a global stiffness matrix. However, for $p=2$, the ratio drops to less than three, and for $p=3$ one has to do three form evaluations before using a global stiffness matrix becomes more efficient. As in the two-dimensional case, the break-even point grows quadratically in p. We have ignored the work associated with counting the vertices, so this would only degrade the estimate of efficiency of using a global stiffness matrix. Using the techniques leading to (4.1), the following is proved.

Theorem 4.3. *Suppose that $d=3$. If the number of bilinear form evaluations, r, satisfies*

$$r\leqslant\frac{[(p+3)(p+2)(p+1)]^2}{18(p^4+6p^3+13p^2+2p+6)},$$

where p is the polynomial degree of finite elements, then it is cheaper not *to form the global stiffness matrix, but rather to compute the forms directly by summing the local forms at each occurrence.*

5. Computational experiments

Comparisons with exact solutions will be made using ℓ_p norms, $\|\cdot\|_p$, on the finite-dimensional space of nodal values of finite element functions defined on Ω. More precisely, let $f=\sum_j f(z_j)\phi_j$

where $\{z_j\}$ are the nodal points and $\{\phi_j\}$ denotes the standard Lagrange basis ($\phi_i(z_j) = \delta_{ij}$, the Kronecker delta). Then

$$\|f\|_p = \left(\sum_j |f(z_j)|^p\right)^{1/p}, \quad 1 \leqslant p < \infty, \qquad \|f\|_\infty = \max_j |f(z_j)|. \tag{5.1}$$

When f denotes a vector quantity, interpret $|f|$ as the Euclidean length of the vector.

To evaluate the effectiveness of higher-order elements in fec, we studied fluid flow in a converging channel (a.k.a. Jeffrey–Hamel flow). Such a flow is radial and after a change of variables satisfies the ordinary differential equation

$$u'' + 4u + u^2 = 6C, \quad u(0) = u(\alpha) = 0, \tag{5.2}$$

where differentiation is with respect to the polar angle ϕ and α is the angle of convergence of the channel.

Fig. 1 shows the solutions of (5.2) for $\alpha = 45^\circ$ and $C = 10^2, 10^3, 10^4, 10^5$. These constants correspond, roughly, to Reynolds numbers $\Re \simeq 20, 73, 238, 764$, respectively. Given a solution u of Eq. (5.2), we find that

$$\boldsymbol{u}_\nu(\boldsymbol{x}) := \nu \frac{u(\operatorname{atan}(y/x))}{x^2 + y^2} \boldsymbol{x}, \quad \boldsymbol{x} = (x, y) \in \Omega \tag{5.3}$$

solves the steady Navier–Stokes equations with kinematic viscosity ν over a wedge domain Ω (cf. [5]). In our experiments, Ω is the quadrilateral with vertices $\{(1,0), (2,0), (1,1), (2,2)\}$.

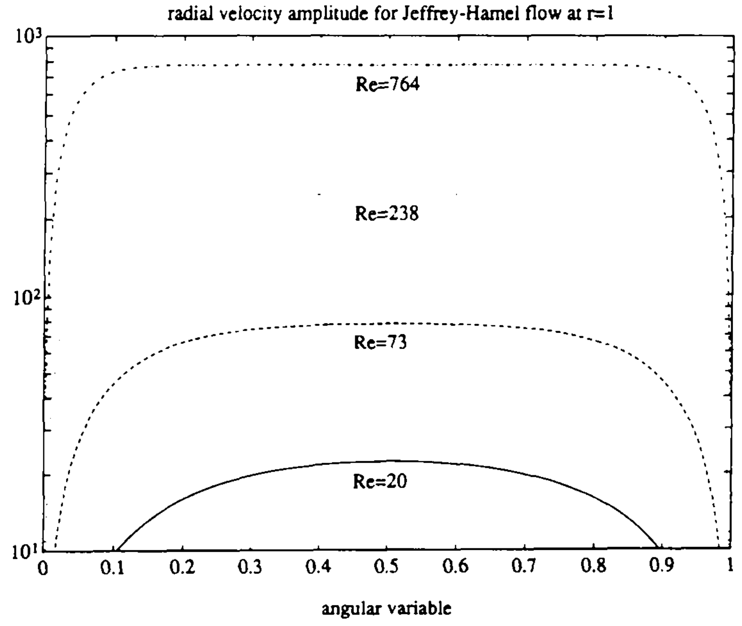


Fig. 1. Solutions of Eq. (5.2).

In our experiments with fec, we solve the time-dependent Navier–Stokes equations with Dirichlet boundary conditions given by the restriction of $\boldsymbol{u}_e$ to the boundary of Ω. We use operator splitting and modified fixed-point iterations to reduce the time-dependent Navier Stokes equations to a sequence of linear Stokes-like problems. Representing the actions of the mass, stiffness, and convection (nonlinear) operators of the Navier–Stokes equations by M, K, and C, respectively, we can write the time-stepping scheme as follows. Given $\boldsymbol{u}^{1,1}$, solve the following equation for $\boldsymbol{u}^{n+1,k+1}$, $k \in \{1, \ldots, F\}$, $n \in \{1, \ldots, T\}$:

$$\left(\frac{1}{\Delta t} M + K\right)(\boldsymbol{u}^{n+1,k+1}) = \frac{1}{\Delta t} M(\boldsymbol{u}^n) + C(\boldsymbol{u}^{n+1,k}), \qquad \boldsymbol{u}^{n+1,1} = \boldsymbol{u}^n.$$

After F such iterations, we set $\boldsymbol{u}^{n+1} = \boldsymbol{u}^{n+1,F+1}$. At each of these steps, we solve F Stokes problems using the iterated penalty method (cf. [1]) with enough penalty iterations to ensure that the ℓ_2-norm of the divergence is no more than 10^{-10} times the ℓ_2-norm of the gradient of the velocity. For our experiments, F is fixed at 2 and T is chosen large enough to reach steady state, which was for us defined to be when $\|u^{n+1} - u^n\|_2/\Delta t < 10^{-10}$.

The main step of the iterated penalty method involves solving a symmetric, positive-definite system, which we do by a sparse matrix routine based on Markowitz' method which reduces to the minimum-degree algorithm in the symmetric case (cf. [6]). The system is factored only once, and from then on only forward and backward substitutions are performed.

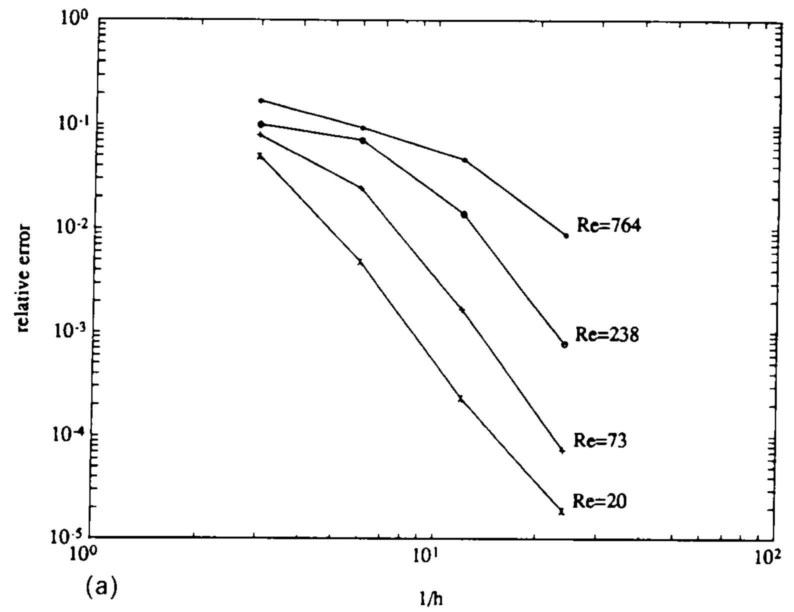
We define the relative velocity error E to be

$$E := \frac{\|\boldsymbol{u}_c - \boldsymbol{u}_e\|_\infty}{\|\boldsymbol{u}_e\|_\infty},$$

where $\boldsymbol{u}_c$ is computed using fec and $\|\cdot\|_\infty$ is the norm defined in (5.1). We identify each experiment with three numbers: the degree of approximating polynomials p, the number of regular subdivisions s of the mesh, and the Reynolds number $\mathfrak{R}$. The mesh $s = 0$ consists of five triangles based on the set of vertices $\{(1,0), (2,0), (1,1), (2,2), (1.5,1)\}$. We are interested in values of E for fixed p and varying s ("the h version") and for fixed s and varying p ("the p version"), as well as situations when both are varied ("the h–p version").

The main conclusion of the results in Figs. 2 and 3 are that both the h method (with degree four or more, see [7]) and the p method are extremely accurate. It is notable that increasing the degree by only two in the p method achieves the same sort of accuracy improvement as a mesh-halving in the h method. The number of degrees of freedom is less for the p method in this case, but the associated matrix is less sparse, leading to more work in solving the linear system in the penalty iteration.

To compare the h and p versions more closely, Fig. 3(a) plots E against the time a workstation (Sparcstation 2 with 64MB of RAM running SunOS 4.1.1 using cfront 2.1 and the native SunOS C compiler) takes to carry out one step of the time-stepping scheme for both the p and h versions. The times given correspond to the second method for computing element-level forms. The time required for the first method was much larger. With $s = 1$, each computation of the nonlinear term of the Navier–Stokes equations required an amount of work equivalent to 30 backsolves for $p = 4$ and 60 backsolves for $p = 6$.



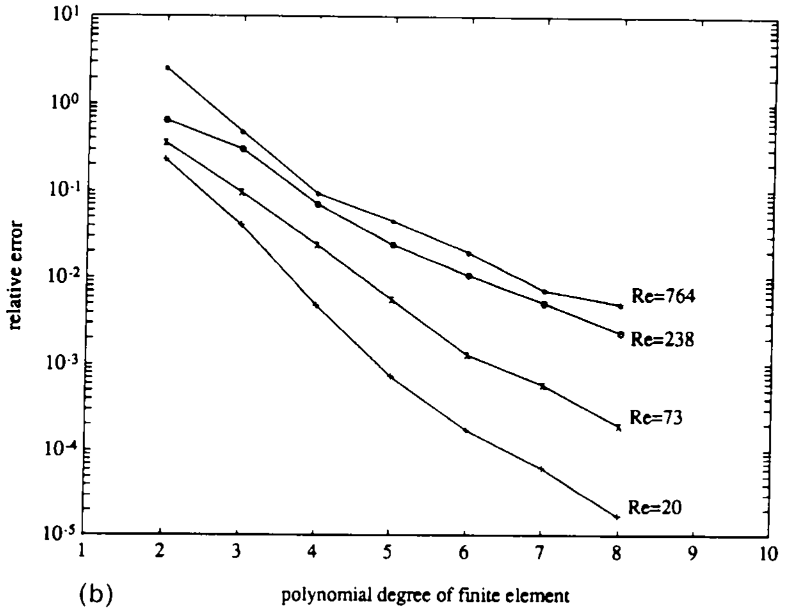
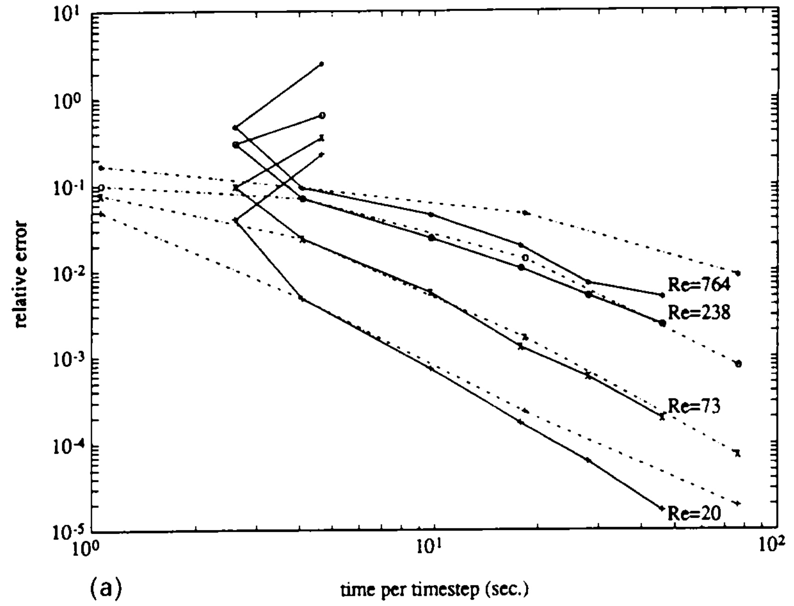


Fig. 2. The h(a) and p (b) version velocity errors: (a) each curve plots E against s varying from 0 to 3 for $p = 4$, (b) each curve plots E against p varying from 2 to 8 for $s = 1$. There is one curve for each $\Re \in \{20, 73, 238, 764\}$ annotated by $\{\times, +, \circ, \star\}$.

The determining factor for the work required at each time step is the number of penalty iterations needed to solve each Stokes problem. In each time step, three penalty solves are done, and each penalty solve required about ten iterations to achieve a relative accuracy of 10^{-10}, or about one penalty iteration per digit except for $p < 4$. The lack of divergence-stability (cf. [7])



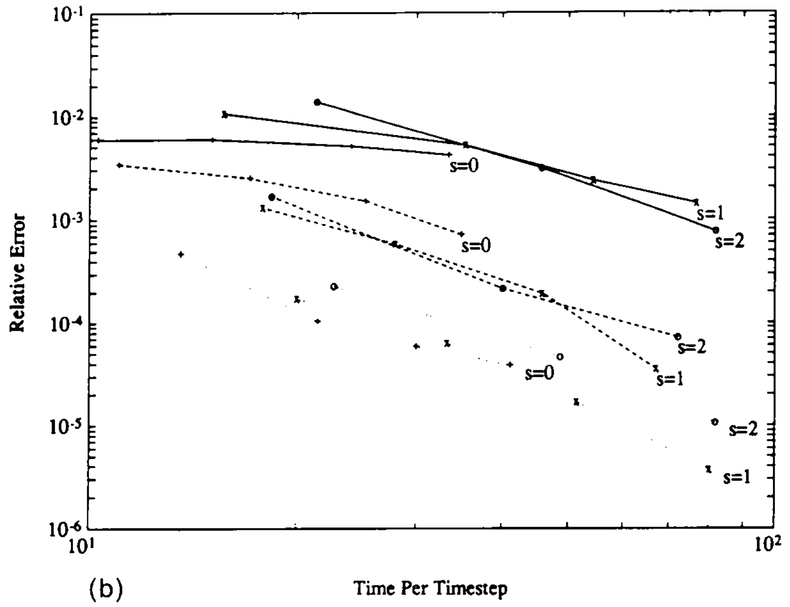
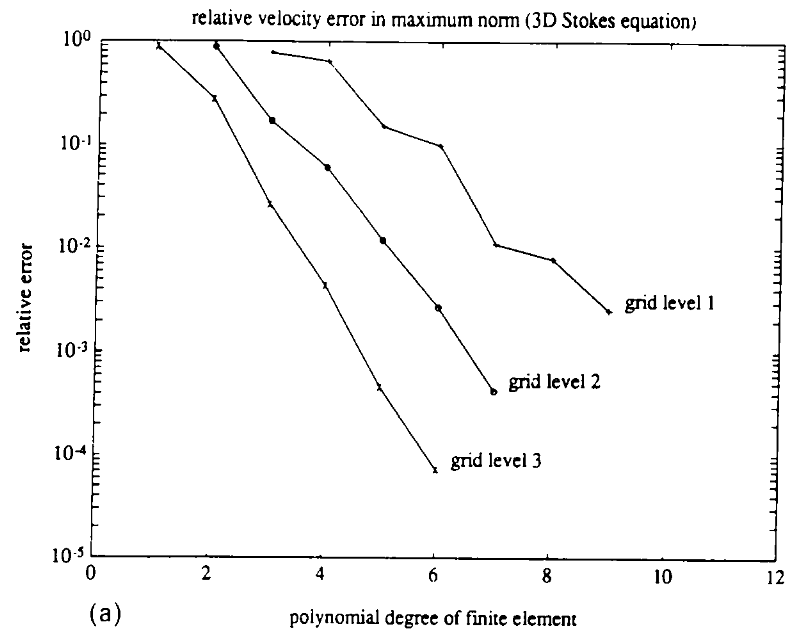


Fig. 3. E versus time per time-step: (a) h and p versions; the solid curves trace the p version ($s = 1$) results for $p = 2, 3, \ldots$, and the dot-dashed curves trace the h version ($p = 4$) results for $s = 0, 1, 2$, (b) h–p version; the dotted, dashed and solid curves trace increasing values of p for Reynolds numbers $\Re \simeq 20, 73, 238$, respectively. The values of p for $s = 0, 1$ and 2 are $\{8, 9, 10, 11\}$, $\{6, 7, 8, 9\}$ and $\{4, 5, 6\}$, respectively.

for $p < 4$ leads to an almost ten-fold increase in the number of penalty iterations which is reflected in Fig. 3(a).

Fig. 3(b) presents a different view of the timing data. It allow us to compare various h and p version combinations directly for each Reynolds number. Each set of curves corresponds to one



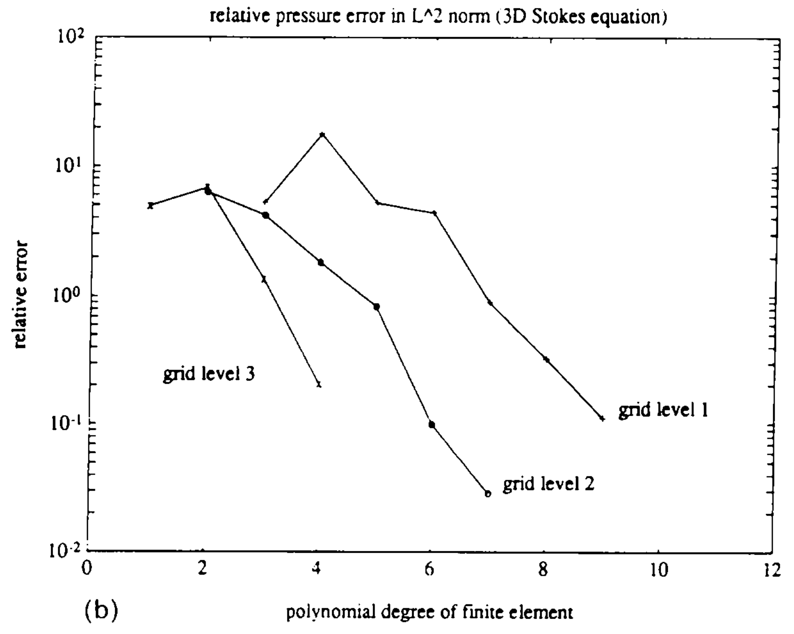


Fig. 4. Velocity (a) and pressure (b) errors for 3D Stokes equations.

approximate Reynolds number. Each set of curves contains three curves, one for each $s = 0, 1, 2$. The curves for subdivision $s = 0$ start at $p = 8$; the curves for $s = 1$ start at $p = 6$, and the curves for $s = 2$ start at $p = 4$.

These curves show that in many cases increasing p is often preferable to subdividing the mesh. This happens whenever a curve for a higher number of subdivisions falls below a curve for a lower number of subdivisions. For example, the set of curves for $\Re = 20$ says that the p version is preferable in every case.

Another important observation based on Fig. 3(b) is that varying the degree of polynomials gives us finer control over the time complexity of our finite element approximation than regularly subdividing the mesh. Indeed, if we want to increase the accuracy of our approximation by a relatively small factor while paying a relatively small penalty in time, increasing p may be the simplest solution.

Several tests were done with the code nmg solving the Stokes equations on a unit cube $(0, 1)^3$ in three dimensions. The exact solution was taken to be

$$\boldsymbol{u} = 1000 * (w_y - w_z, w_z - w_x, w_x - w_y), \qquad p = 10xyz,$$

where $w = [x(1-x)y(1-y)(z(1-z)]^2$. The appropriate right-hand side was computed in order that this yields an exact solution to the Stokes equations.

Relative ℓ_2 errors for the velocity as defined above were computed for a range of values of s and p. These are depicted in Fig. 4(a).

In Fig. 4(b), relative ℓ_2 errors for the pressure are depicted for a range of values of s and p. We again see that the p method is quite effective, requiring only an increase in degree of two or three to match the effect of a mesh-having in the h method.

References

[1] L.R. Scott and B. Bagheri, "Software environments for the parallel solution of partial differential equations", in: R. Glowinski and A. Lichnewsky (eds.), *Computing Methods in Applied Sciences and Engineering IX*, SIAM, Philadelphia, 378-392, 1990.

[2] L.R. Scott and S. Zhang, "Higher dimensional non-nested multigrid methods", *Math. Comp.* **58**, pp. 457-466, 1992.

[3] G. Strang and G.J. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[4] B. Bagheri, L.R. Scott and S. Zhang, "Details regarding the implementation of high order finite element methods", Technical Report, Department of Mathematics, University of Houston, TX, USA.

[5] L.D. Landau and E.M. Lifshitz, *Fluid Mechanics*, Pergamon, Oxford, 1959.

[6] S. Pissanetzky, *Sparse Matrix Technology*, Academic Press, New York, 1984.

[7] L.R. Scott and M. Vogelius, "Norm estimates for a maximal right inverse of the divergence operator in spaces of piecewise polynomials", *M^2AN* **19**, pp. 111-143, 1985.