FINITE ELEMENT ANALYSIS OF NONISOTHERMAL REACTIVE FLOWS

David Roylance
Department of Materials Science and Engineering
Massachusetts Institute of Technology
Cambridge, MA 02139

December 2, 2003

Introduction

This document will outline the theoretical background and operating principles of a general finite element code which has been developed for research and teaching of materials processing operations, and present examples of its use. Some of this information is rather condensed, and the reader will probably wish to consult additional sources such as those listed in the References section for more thorough treatment of topics in transport theory and finite element methodology.

The finite element code to be discussed here has been developed for plane or axisymmetric problems in viscous fluid flow, such as might occur in polymer melt processing. Many other flow types can be handled as well, but polymer processing provides a convenient means of outlining the code's features and underlying numerical algorithms. Polymer processing problems typically involve the flow of incompressible viscous liquids at low Reynolds' numbers ("creeping" flow), in irregular geometries. It is common in such problems to have nonisothermal conditions prevail, and for the polymer to experience chemical reaction during processing. To model such problems satisfactorily, the code has been developed to solve for velocities, temperatures, and extent of chemical reaction simultaneously.

The finite element method consists of recasting the governing differential equations of engineering boundary value problems as a sequence of linear or nonlinear algebraic equations:

$$K_{ij}a_j = f_i \tag{1}$$

Here a_j and f_i denote column vectors which in the case of stress analysis problems are the displacements and externally applied forces at discrete points, or "nodes," which have been placed in the solution domain. The K_{ij} is a square matrix array which relates the displacement at node j to the force at node i. Finite element codes assemble the K_{ij} matrix from contributions of "elements" which have been placed in the solution domain so as to encompass the nodes. Once K_{ij} has been created, the unknown values of a_j can be computed by Gaussian reduction or other well-known techniques for solving sets of simultaneous algebraic equations.

The flow code follows this same approach, but in a somewhat more general way. The displacement vector a_j is now regarded as generalized entity, which can include nodal values of flow velocity, temperature, concentration of reactive species, or streamfunction. The generalized force vector f_i contains those entities which correspond to the generalized displacements, as follows:

Generalized displacement	Generalized Force
velocity	force
temperature	heat flux
species concentration	species flux
stream function	velocity gradients

Since velocity is a vector with two components while the other quantities are scalars, there can be up to five degrees of freedom at each node. It is not necessary to include variables which are not needed for a particular problem, and the user is able to define which degree of freedom numbers apply to which variables. For instance, a problem in which only velocities and stream functions are needed might use degree of freedom numbers 1, 2, and 3 for x-velocity (u), y-velocity (v), and streamfunction (ψ), respectively. Temperature and species concentration would not be computed, and memory would not be allocated for them.

1 Theoretical Background

1.1 Governing Equations

Finite element formulations for linear stress analysis problems are often derived by direct reasoning approaches. Fluid flow problems, however, are often viewed more easily in terms of their governing differential equations, and this is the approach used in the development of the processing code. The equations which govern the nonisothermal flow of a reactive fluid are derived in several texts on transport phenomena and polymer processing (e.g. References 1,2). These are the familiar conservation equations for transport of momentum, energy, and species:

$$\rho\left[\frac{\partial u}{\partial t} + u\nabla u\right] = -\nabla p + \nabla(\eta\nabla u) \tag{2}$$

$$\rho c\left[\frac{\partial T}{\partial t} + u\nabla T\right] = Q + \nabla(k\nabla T) \tag{3}$$

$$\left[\frac{\partial C}{\partial t} + u\nabla C\right] = R + \nabla(D\nabla C) \tag{4}$$

Here u, T, and C are fluid velocity (a vector), temperature, and concentration of reactive species; these are the principal variables in our formulation. Other parameters are density (ρ), pressure (p), viscosity (η), specific heat (c), thermal conductivity (k), and species diffusivity (D). The ∇ operator is defined as $\nabla = (\partial/\partial x, \partial/\partial y)$. The similarity of these equations is evident, and leads to considerable efficiency in the coding of their numerical solution. In all cases, the time rate of change of the transported variable (u, T, or C) is balanced by the convective or flow transport terms (e.g. $u\nabla T$), the diffusive transport (e.g. $\nabla[k\nabla T]$) and a generation term (e.g. Q).

In conventional closed-form analysis, one generally seeks to simplify the governing equations by dropping those terms whose numerical magnitudes are small relative to the others, and then proceeding with a formal solution. In contrast, all the terms (except $u\nabla u$, for now) are present in the processing code and the particularization to specific problems is done entirely by the selection of appropriate numerical parameters in the input dataset.

The units must be given special attention in these equations, especially as materials properties obtained from various handbooks or experimental tests will usually be reported in units which must be converted to obtain consistency when used in the above equations. The governing equations are volumetric rate equations. For instance, the heat generation rate Q is energy per unit volume per unit time, such as `N-m/m`^{3}`-s` if using SI units. The user must select units for all parameters so that each term in the energy equation will have these same units.

Q and R are generation terms for heat and chemical species respectively, while the pressure gradient ∇p plays an analogous role for momentum generation. The heat generation arises from viscous dissipation and from reaction heating:

$$Q = \tau : \dot{\gamma} + R(\Delta H) \tag{5}$$

where τ and $\dot{\gamma}$ are the deviatoric components of stress and strain rate, R is the rate of chemical reaction, and ΔH is the heat of reaction. R in turn is given by a kinetic chemical equation; in our model we have implemented an m-th order Arrhenius expression:

$$R = k_0 \exp\left(\frac{-E^\dagger}{R_g T}\right) C^m \tag{6}$$

where k is a preexponential constant, $E^\dagger$ is an activation energy, and $R_g = 8.31$ `J/mole-`$^\circ$`K` is the Gas Constant.

The viscosity η is a strong function of the temperature and the shear rate for many fluids, and the flow code has been written to include a Carreau power-law formulation for shear thinning and an Arrhenius expression for thermal thinning. The formal equation is:

$$\eta = \eta_0 \exp\left(\frac{-E_\eta}{R_g T}\right) \left[1 + (\lambda\dot{\gamma})^2\right]^{\frac{n-1}{2}} \tag{7}$$

Here η_0 is the "zero-shear" viscosity limit, E_η is an activation energy for thermal thinning, λ is a shape parameter, and n is the power-law exponent. This formulation is admittedly not suitable for all cases, such as liquids exhibiting strong elastic effects, but it is commonly used in much of the literature for viscous flow rheology.

The boundary conditions for engineering problems usually include some surfaces on which values of the problem unknowns are specified, for instance points of known temperature or initial species concentration. Some other surfaces may have constraints on the gradients of these variables, as on convective thermal boundaries where the rate of heat transport by convection away from the surface must match the rate of conductive transport to the surface from within the body. Such a temperature constraint might be written:

$$h(T - T_a) = -k\nabla T \cdot n \qquad \text{on} \quad \Gamma_h \tag{8}$$

Here h is the convective heat transfer coefficient, T_a is the ambient temperature, and n is the unit normal to the convective boundary Γ_h.

1.2 The Finite Element Formulation

Of course, it is usually impossible to solve the above set of equations in closed form, especially in light of the irregular boundary conditions often encountered in engineering practice. However, the equations are amenable to discretization and solution by numerical techniques such as finite differences or finite elements. A full treatment of the finite element method is beyond the scope

of this document, and the reader is referred to standard texts (e.g. [5,6]) for a more complete description. However, we will outline briefly the approaches used by the flow code, so the reader can see the overall scope of the method. This introduction will help in selecting various code options when setting up problems, and should provide an introduction to more extensive readings.

As an illustrative example, consider the specialization of the thermal transport equation to a two-dimensional problem in steady conductive heat transfer with internal heat generation and constant conductivity:

$$0 = Q + k\nabla^2 T \tag{9}$$

If a closed-form solution were being attempted, we would use successive integration or other mathematical techniques to determine a function $T(x, y)$ which satisfies this equation and also the boundary conditions of the problem. This can be done when the boundary conditions are sufficiently simple.

Considering the important case when no closed-form solution can be found, let us postulate a function $\tilde{T}(x, y)$ as an approximation to T:

$$\tilde{T}(x, y) \approx T(x, y) \tag{10}$$

Many different forms might be adopted for the approximation $\tilde{T}$. The finite element method discretizes the solution domain into an assemblage of subregions, or "elements," each of which have their own approximating functions. Specifically, the approximation for the temperature $\tilde{T}(x, y)$ within an element is written as a combination of the (as yet unknown) temperatures at the nodes belonging to that element:

$$\tilde{T}(x, y) = N_j(x, y)T_j \tag{11}$$

Here the index j ranges over the element's nodes, T_j are the nodal temperatures, and the N_j are "interpolation functions." These interpolation functions are usually simple polynomials (generally linear, quadratic, or occasionally cubic polynomials) which are chosen to become unity at node j and zero at the other element nodes. The interpolation functions can be evaluated at any position within the element by means of standard subroutines, so the approximate temperature at any position within the element can be obtained in terms of the nodal temperatures directly from Equation (11).

Since $\tilde{T}$ is an approximation rather than the true solution, we would expect that for a given set of approximate nodal temperatures Equation (8) would not be satisfied exactly:

$$Q + k\nabla^2 \tilde{T} \neq 0 \tag{12}$$

One powerful method for selecting the nodal temperatures so as to achieve a form of global accuracy is to ask not that the governing equation be satisfied identically everywhere within the element, but only that its integral over the element volume be as small as possible:

$$\int_V (Q + k\nabla^2 \tilde{T})\, dV = \mathcal{R} \approx 0 \tag{13}$$

Here $\mathcal{R}$ is the "residual" of the approximation; it would clearly be zero if $\tilde{T}$ happened to equal the true solution T.

Equation (13) provides only a single equation for each element, which would not be sufficient to determine all of the nodal temperatures in the approximation. However, we can obtain a

number of such residual equations by premultiplying the integrand by a "weighting function" which might, for instance, be chosen to enforce accuracy at a number of different points in the solution domain. This might involve choosing a weighting function which is unity in the vicinity of a point at which the approximation should be accurate (i.e. have a zero residual), and zero elsewhere. This is just what the interpolation functions do, and the "Galerkin" weighted residual method takes the weighting functions and the interpolation functions to be the same. The set of weighted residual equations then becomes:

$$\int_V N_i(Q + k\nabla^2\tilde{T})\,dV = \mathcal{R} \approx 0 \tag{14}$$

It is convenient to integrate Equation (14) by parts to reduce the order of differentiation; this also introduces the thermal boundary conditions in a natural way. The second-order term is expanded as:

$$\int_V N_i k\nabla^2\tilde{T}\,dV = \oint_\Gamma N_i k\nabla\tilde{T}\cdot n\,d\Gamma - \int_V \nabla N_i k\nabla\tilde{T}\,dV \tag{15}$$

Here Γ is the element boundary, and n is the unit normal to the boundary. Using Equation (8) for the boundary convection condition, Equation (14) becomes:

$$\begin{aligned}\int_V \nabla N_i k\nabla\tilde{T}\,dV \quad &= \textstyle\int_V N_i Q\,dV + \oint_\Gamma N_i k\nabla\tilde{T}\cdot n\,d\Gamma \\ &= \textstyle\int_V N_i Q\,dV - \oint_\Gamma N_i h(\tilde{T} - T_a)d\Gamma\end{aligned} \tag{16}$$

Now using expression for $\tilde{T}$ from Equation (11) and factoring out the nodal temperatures which are not functions of x and y, we obtain a relation in which the nodal temperatures are related to the nodal heat fluxes:

$$k_{ij}T_j = q_i \tag{17}$$

where

$$k_{ij} = \int_V \nabla N_i k\nabla N_j\,dV + \oint_\Gamma N_i h N_j\,d\Gamma \tag{18}$$

and

$$q_i = \int_V N_i Q\,dV + \oint_\Gamma N_i h T_a\,d\Gamma \tag{19}$$

Of course, the integrals in the above equations must be replaced by a numerical equivalent acceptable to the computer. Gauss-Legendre numerical integration is commonly used in finite element codes for this purpose, since that technique provides a high ratio of accuracy to computing effort. Stated briefly, the integration consists of evaluating the integrand at optimally selected integration points within the element, and forming a weighted summation of the integrand values at these points. In the case of integration over two-dimensional element areas, this can be written:

$$\int_A f(x,y)\,dA \approx \sum_l f(x_l, y_l)w_l \tag{20}$$

The location of the sampling points x_l, y_l and the associated weights w_l are provided by standard subroutines. In most modern codes, these routines map the element into a convenient shape, determine the integration points and weights in the transformed coordinate frame, and then map the results back to the original frame. The functions used earlier both for interpolation and residual weighting can be used for the mapping as well, achieving a significant economy in coding. This yields what are known as "numerically integrated isoparametric elements," and these are a mainstay of the finite element industry.

Equations (17)–(19), with the integrals replaced by numerical integrations of the form in Equation (20), are the finite element counterparts of Equation (9), the differential governing equation. The computer will use these by looping over each element, and over each integration point within the element. At each integration point, the integrands for the various terms, such as k_{ij} as given in Equation (18) must be computed. A simplified flow chart for the formation of the k_{ij} thermal stiffness matrix is shown below:

```
begin loop over elements
  obtain integration points and weights for element
  loop over element integration points (l subscript)

    obtain interpolation functions at integration point
    loop over nodes (i subscript)

      loop over nodes (j subscript)

        compute integrand
        add to thermal stiffness matrix

      end inner node loop
    end outer node loop
  end loop over integration points
end loop over elements
```

It can be appreciated that a good deal of computation is involved just in forming the terms of the stiffness matrix, and that the finite element method could never have been developed without convenient and inexpensive access to a computer.

The description above treats only two terms of one of our governing equations; similar procedures were used in the flow code to treat all of the terms in Equations (2)–(4), with some special techniques needed occasionally as will be discussed below. The contributions of all needed terms are computed by a single element routine, and are added to the locations in the stiffness matrix associated with the degree of freedom number for the variable under consideration. One additional element type is used for boundaries on which convective heat transfer occurs.

1.3 The Penalty Method for Incompressible Fluids

The momentum equation requires special treatment, because here we seek a solution for the nodal velocities which satisfy an incompressibility constraint in addition to minimizing the residual of the Galerkin equations. At present, there are two principal methods available for enforcing incompressibility: one introduces the pressure as an independent nodal variable, and the other employs a "penalty" formulation which does not require additional variables. The flow code uses the penalty formulation, although future developments may include the velocity-pressure formulation as an option.

As described more completely elsewhere [5,11] the penalty formulation dissociates the stresses and deformation gradients into dilatational (hydrostatic) and distortional (shearing) components, and treats these separately. This is useful in describing polymer molecular response,

since the dilatational terms are influential in controlling molecular mobility and such phenomena as glass transition temperature and viscosity which depend on mobility. The distortional terms create almost all the dissipation for heating effects, and also are likely to be responsible for mechanical degradation of the molecular structure.

Incompressibility is enforced essentially by taking the bulk compressibility to be a large number (a "penalty coefficient") in comparison with the viscosity, and using this in the relation between dilatational stresses and flow rates. However, this tends to make the system of equations ill-conditioned, and substantial research has been aimed at achieving stable and accurate solutions in such cases. A similar problem arises in the deformation of incompressible solids, such as rubber. It has been found, however, that the problem of ill conditioning is alleviated by integrating the volumetric terms at a lower numerical order than the viscous terms, a process known as "selective reduced integration."

The user is asked to make several numerical selections in using the penalty method; these include the integration orders for both the penalty and regular terms, and the magnitude of the numerical factor which is used to "penalize" incompressibility. When using four-noded quadrilateral elements, good results have been obtained by taking the penalty integration order as 1, the regular order as 2, and the penalty coefficient as 10^7 times the viscosity. The user should consider experimenting with these values, and consulting the technical literature for ongoing research with regard to this technique.

1.4 Algorithms for Coupled and Nonlinear Problems

The governing equations contain a number of factors which couple the equations together. For instance, the flow velocity u appears in the convective terms of the heat and species equations, and the heat generation term in the energy equation contains a contribution from the heat of reaction as governed by the reaction rate in the species equation. Further, the various material constants (viscosity, diffusivity, etc.) are in general functions of the solution variables themselves, rendering the problem nonlinear in the material sense. The flow code is capable of a variety of iterative techniques for solving such problems.

The flexibility of the flow code in solving these varied problems types arises from its basic control strategy, which is taken from that suggested by Prof. R.L. Taylor in the Zienkiewicz textbook, *The Finite Element Method* [5]. Rather than solving the matrix equation $Ka = f$ directly, the flow code uses an "unbalanced force vector" approach in which the right hand side is the difference between the imposed value of f and the product Ka using the current estimate for a:

$$K\Delta a = f - Ka_0 \tag{21}$$

Solution of this system than gives the change $\Delta a = a - a_0$ relative to the initial estimate a_0 which will be necessary in a to eliminate the unbalanced force.

Also following the Taylor approach, the flow code controls code execution by means of a series of macro commands which are appended to the input dataset. For a simple linear problem, four keywords would be used:

`form` — form the unbalanced force vector (the right hand side of the equation set).

`tang` — assemble the stiffness matrix K_{ij} (also known as the "tangent" stiffness matrix).

`solv` — solve the assembled set of equations, i.e. solve for the change vector Δa, and add it to the previous estimate for displacements a_0. In this case, the a_0 will be just the specified boundary values for a, with the unconstrained displacements taken initially as zero.

`disp` — send the computed displacements to the output datafile.

To illustrate the capability offered by this macro-controlled code logic, consider the case of a partially coupled problem, such as one in which the streamfunctions $\psi(x, y)$ are desired in addition to the flow velocities. (The streamfunctions coincide with the fluid particle pathlines in steady flow problems, and are very useful in plotting the results of flow simulations.) The right hand side of a suitable governing equation for the streamfunctions depends on the velocities according to the following equation:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} \tag{22}$$

Using the Galerkin treatment, the finite element counterpart of this equation is:

$$\left(\int_V \nabla N_i \nabla N_j \, dV \right) \psi_j = \int_V N_i \left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} \right) dV \tag{23}$$

It is seen that the unbalanced force vector cannot be computed correctly until the velocity gradients are known; although the left hand side (the streamfunction stiffness matrix) does not depend on the velocities. (Neither does the velocity depend on the streamfunction, so the problem is only partially coupled.)

This situation can be handled by performing the `form-solv` macros twice. After the first pass, the velocities will be correct, but the stream functions will not, because they were computed before the correct velocities were available. In the second pass, the streamfunction equation will use the corrected velocities and the streamfunctions will then be computed correctly. The `tang` macro need not be recomputed, since it has no terms which depend on the velocities. The macro keyword list for this might be:

```
tang
form
solv
form
solv
disp
```

The macro set includes a `loop` command, so the above list might also be written:

```
tang
loop          2
form
solv
next
disp
```

All macro commands between `loop` and `next` will be repeated a number of times given by the argument in column 15 of the `loop` line.

Next, consider a fully nonlinear problem in which the material parameters may depend on the solution variables and the equations are fully coupled. An example might be a nonisothermal flow problem in which the viscosity is allowed to vary with temperature and the convective transport of heat is significant. More complicated iterative approaches must be used in such problems. In "Newton-Raphson" iteration, the stiffness matrix is recomputed at each step and an updated solution for the nodal unknowns is obtained from the current unbalanced force vector. The macro keywords for this would be:

```
loop        15
tang
form
solv
next
disp
```

This will direct the code to perform 15 iterations or until a preset tolerance is reached. Techniques for nonlinear problems are diverse and often complicated, and the flow code is able to carry out several strategies depending on the macro keywords. The reader is directed to Reference [5] for a more complete discussion of these macros and their use in setting up a variety of solution modes.

1.5 Streamline Upwinding for Convective Transport

Convective transport of heat or chemical species in flow problems deserves special note. A Galerkin treatment of the heat convection term $\rho c\, u \nabla T$, for instance, gives:

$$k_{ij,conv} = \int_V N_i \rho c\, u \nabla N_j \, dV \tag{24}$$

Apart from the fact that this expression gives an unsymmetric stiffness (the `tang` macro must be replaced by `utan` to solve the system satisfactorily), no unusual features are apparent. However, the stability criteria governing this first-order term are different than those for the other (second order) terms, and it is common to find that incorporation of convective terms leads to oscillatory or unstable results.

Simulation of convection-dominated flow is accomplished in the flow code by means of a "streamline upwinding" formulation [12]. Briefly stated, this involves increasing the viscosity artificially along the streamline direction, which improves stability without distorting the velocity predictions in the "crosswind" direction excessively. This technique is relatively well accepted as a valid means of obtaining stable results from equations having strong first order terms, although it is not completely without controversy. Almost certainly, the increased stability is gained at some expense in accuracy.

The user has three choices for treating convection, controlled by the selection of the "convection integration order" parameter. If this parameter is set to 0, the convective terms are not included at all. If set to 1, streamline upwinding is used. If set to 2, conventional Galerkin handling is used (likely with unsatisfactory results unless very fine meshes are used, but feel free to experiment).

1.6 Algorithm for Transient Problems

The finite element method can handle transient problems by using a variety of time-stepping algorithms adapted from finite-difference technique. The governing equations for transport involve only the first derivative of time, and this renders the extension to transient problems simple and efficient. When the time derivative terms are added, the finite element matrix equations take the form:

$$C\left(\frac{da}{dt}\right) + Ka = f \tag{25}$$

where the C matrix stores the inertial influences and da/dt is the derivative vector of the nodal variables. For instance, the $\rho c(\partial T/\partial t)$ inertial term for the energy equation is obtained from the Galerkin procedure as:

$$C_{ij} = \int_V N_i \rho c N_j dV \tag{26}$$

Equation (25) can be written in finite difference form as:

$$C\left(\frac{a_{n+1} - a_n}{\Delta t}\right) + K[\theta a_{n+1} + (1-\theta)a_n] = f \tag{27}$$

Here the a_n and a_{n+1} indicate the solution variables at time n and $n+1$, respectively. The forcing terms f are assumed constant over a small time increment Δt, and θ is a parameter between 0 and 1 which allows the time stepping scheme to be adjusted between forward and backward differencing. The method is unconditionally stable for $\theta \geq (1/2)$, and $\theta = (2/3)$ corresponds to a Galerkin-like treatment of the time derivative terms.

Rearranging, this relation becomes:

$$[(C/\Delta t) + \theta K]\Delta a = f - K a_n \tag{28}$$

where $\Delta a = a_{n+1} - a_n$. This implicit relation permits the values a_{n+1} at the end of the next time increment Δt to be computed from the current values a_n.

The right hand side is treated just as in steady problems, since the unbalanced force vector approach already subtracts the Ka_n term from f. The only difference is that the stiffness matrix K must be multiplied by θ and the $(C/\Delta t)$ term must be added; this is done when the user sets the "time-stepping flag" in the material property portion of the input dataset (the section following the `mate` keyword) to be 1 rather than 0. A macro list for a transient problem might look like the following:

```
init
dt           10
disp
tang
loop         20
time
form
solv
disp          2
next
end
    1   15       300.
   16   30       400.

stop
```

The keyword `init` instructs the computer to read the data following the `end` keyword for setting the nodal initial values. In this cases, the first degree of freedom for nodes 1 through 15 are set to 300.0, and nodes 16 through 30 are set to 400.0. A blank line is used to end these initializing data. The keyword `dt` sets the time step to 10 (the units, as always, are up to the user). The `disp` which follows next spools the initial values to the output file. The `tang`

command here appears outside the loop; this will save a good deal of computing time, and is permissible if nothing in the stiffness matrix changes with time. Next, we ask for 20 time steps; each begins with the `time` command, which adds Δt to the current time. The displacements will be spooled for output at every second time step, as indicated by the '2' in column 15 of the `disp` command.

2 Example Problems

The numerical algorithms outlined above have been coded in Fortran and implemented on a number of computer systems of varying size. An implementation for the IBM-PC and compatible family of microcomputers was accomplished with no special difficulties, although some extra caution was needed to insure that the Fortran compiler computed array addresses beyond 64 kilobytes correctly. The PC version requires the presence of a math coprocessor and 450K of available memory, and is accompanied by a graphics postprocessing module which requires an EGA display system.

The PC distribution disk has a number of demonstration datasets, named `demo1` through `demo6`. These can be run by giving the name of the datafile to be used as input on the command line; for instance to run `demo1`, simply type `flow demo1`. The run log will appear on the screen, and when the job is complete the files `demo1.plt` and `demo1.out` will be stored in the current directory. The `.out` file is a formatted listing of numerical values produced by the code; it can be displayed on the terminal monitor or sent to the printer as desired. The `.plt` file contains the same numerical results, but without quite so much formatting; it is intended to be read in by a postprocessing code in order to generate graphical views of the output. Once `flow demo1` has been run, the results can be viewed graphically by typing `post demo1`.

As inspection of the demonstration datasets will indicate, the preparation of input datasets for finite element analysis can be a tedious and error-prone chore. For any but the most simple problems, it is essential that a computerized "preprocessor" be employed to generate the dataset. Discussion of available preprocessors is beyond the scope of this document, but the reader should be aware that several such codes are available both commercially and from finite element literature sources. For instance, Reference [14] contains a very useful Fortran listing which can be used to generate the lists of nodal coordinates and element connectivities.

A brief description of these demonstration problems follows:

2.1 demo1 – simple pressure/drag flow.

Here we treat a standard textbook problem in fluid mechanics, that of an incompressible viscous fluid constrained between two boundaries of infinite lateral extent. Although idealized, this problem is often used to model the down-channel flow in a melt extruder [4]. A positive pressure gradient is applied in the x-direction, and the upper boundary surface at $y = H$ is displaced to the right at a velocity of $u(H) = U$. The y-velocities are all set to zero; the problem is underconstrained otherwise. This simple problem was solved by a 10x3 mesh of 4-node quadrilateral elements, as shown in Figure 1. Numerical parameters such as η and H are set to unity in this example problem. When the problem is linear, numerical results for other problem parameters can be obtained by suitable scaling; nondimensionalized units are used in Figure 1 to emphasize this point.

The user will find it instructive to compare the resulting velocities $u(y)$ with the theoretical values. In this case the theoretical solution requires only the x-direction momentum equation;

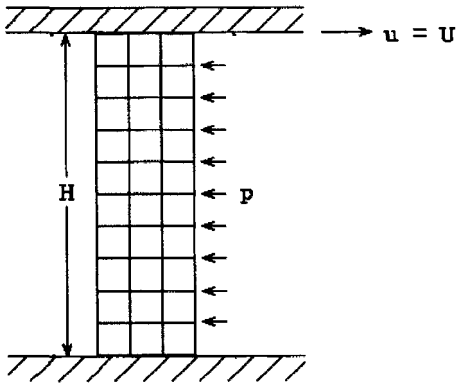

Figure 1: Simulation of drag/pressure plane flow.

after dropping terms which are identically zero, this becomes:

$$\frac{d^2u}{dy^2} = \frac{1}{\eta}\left(\frac{\Delta p}{\Delta x}\right) \tag{29}$$

Integrating this twice, and applying the boundary conditions $u(0) = 0, u(H) = U$, the velocity is given as

$$u(y) = \frac{1}{2\eta}\left(\frac{\Delta p}{\Delta x}\right)\left(Hy - y^2\right) + U\left(\frac{y}{H}\right) \tag{30}$$

The first term above is the "Poiseuille" parabolic distribution produced by the applied pressure; the second term is the linear "Couette" distribution caused by the drag. Figure 2 shows the finite-element prediction of this velocity profile for two cases: a Newtonian fluid (power-law exponent = 1) and a shear-thinning fluid (power-law exponent = 0.3). The shear-thinning analysis is nonlinear, and was accomplished by Newton-Raphson iteration as described earlier; the dataset for this case is named `demo1a` on the PC distribution disk.

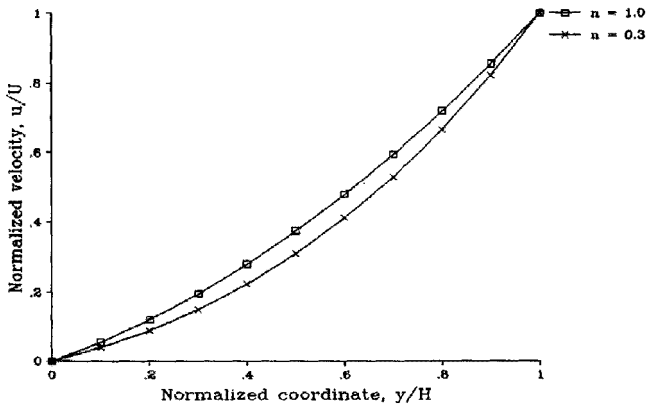

Figure 2: Finite element prediction of plane flow velocity profile.

2.2 demo2 – Couette flow with heat generation

This problem uses the same grid as `demo1`, and illustrates some additional capability of the flow code. Here no pressure gradient is imposed (this is then drag or "Couette" flow only), but we also compute the temperatures resulting from internal viscous dissipation. Note that the degree of freedom number for temperature is set at 3, and the code is looped twice. (After the first loop the velocities are known, but another pass is needed so the heat transfer equations are given the correct velocities from which the dissipation can be computed.)

The shear rate in this case is just $\dot{\gamma} = (\partial u/\partial y) = U/H$. The associated stress is $\tau = \eta\dot{\gamma} = \eta(U/H)$, and the thermal dissipation is then $Q = \tau\dot{\gamma} = \eta(U/H)^2$. The energy equation then becomes:

$$\frac{d^2T}{dy^2} = -\left(\frac{\eta}{k}\right)\left(\frac{U}{H}\right)^2 \tag{31}$$

This can be solved easily by integrating twice and imposing fixed thermal boundary conditions at the top and bottom of the flow channel; here we have set $T(0) = T(H) = 0$. The temperature profile predicted by the code is shown in Figure 3; the reader should compare this with the theoretical values.

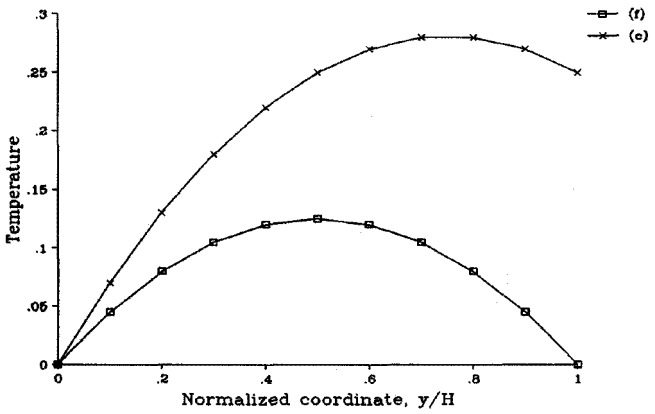

Figure 3: Finite element simulation of plane Couette flow with thermal dissipation and conductive heat transfer. (f) – fixed temperature condition; (c) – convective boundary condition.

The specific numerical values, of course, reflect the values used as input parameters; the unit values used in this simulation give a "Brinkman Number" $Br = (QH^2)/(kT)$ of unity, if the characteristic temperature in this definition is also taken as unity. This dimensionless parameter, which represents the relative importance of internal heat gneration rate to conductive heat transfer rate, can be used in scaling the numerical results for other problem parameters.

Figure 3 also shows the temperature profile which is obtained if the upper boundary exhibits a convective rather than fixed condition; the dataset for this run is `demo2a`, which makes use of the convective boundary element discussed earlier. The convective heat transfer coefficient h was set to unity; this corresponds to a "Nusselt Number" $Nu = (hH/k) = 1$.

2.3 demo3 – 4:1 entry flow

Here flow passes from a reservoir into a capillary with a 4:1 constriction ratio. The mesh takes advantage of symmetry, so the lower boundary is actually the centerline of a plane capillary

entry flow. This is a moderately large problem, one often used to assess code performance by the fluid modeling community. However, the mesh is too coarse to resolve certain fine features of the flow, such as a recirculation which appears in the stagnation region of the reservoir. A fully developed parabolic flow profile is imposed at the left hand boundary, and the streamfunction is set to zero along the centerline. The code loops twice to compute both the temperatures and the streamfunction (note the streamfunction degree of freedom is set to 4).

In this dataset, convective heat transfer is neglected and only conductive thermal transport is considered (the temperature convection integration order is set to 0 in `demo3`). The relative importance of convective versus conductive heat transfer is indicated by the "Peclet Number," defined as $Pe = \rho c U H/k$, where U and H are a characteristic velocity and length for the system. (The upstream centerline velocity and the reservoir half-height are convenient choices, and were both taken as unity in `demo3`). The Peclet number is usually large for polymer processing operations due to the low thermal conductivity of those materials. For higher Peclet numbers, try setting the temperature convection order to 1 (for streamline upwinding) or 2 (for Galerkin treatment). Reference [9] contains a further discussion of the finite element analysis of this problem. Figure 4 shows the graphical display from `post` for the mesh and primary variables as computed by the code.

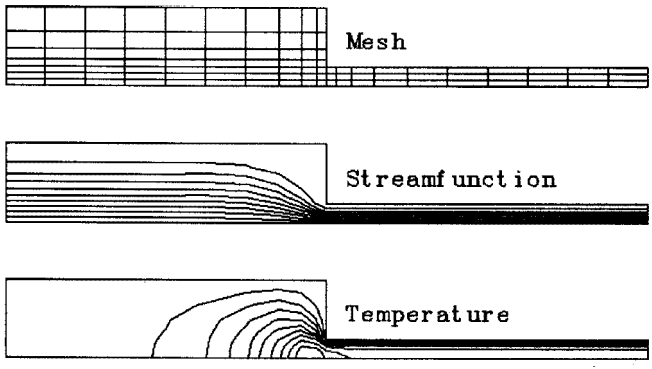

Figure 4: Finite element simulation of 4:1 entry flow with coupled conductive heat transfer.

When the macro `stre` is included in the input dataset, the code loops again over the elements and computes auxiliary variables which may be related to the gradients of the primary variables. These can then be contoured by `post`, using the `stress` option. Stress components 1 through 5 refer to vorticity ω, pressure p, and the stresses $\tau_{xx}, \tau_{yy}, \tau_{xy}$. Some of these, as generated by `post demo3`, are displayed in Figure 5.

The contours of Figure 5 illustrate that the pressure drop occurs largely in the capillary, where it is fairly uniform away from the entrance and exit regions. It should be noted that penalty flow formulations are sometimes afflicted with an anomolous "checkerboarding" of the computed pressures, in which the pressures oscillate from element to element. The reader is referred to Hughes' paper [11] for a further discussion of this numerical artifact, and how it can be avoided or smoothed by suitable postprocessing.

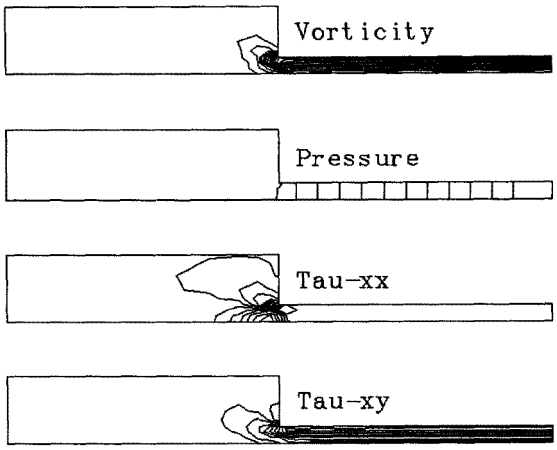

Figure 5: Stresses and associated auxiliary variables in 4:1 entry flow

2.4 demo4 – transient heat conduction

This is another textbook problem, the famous and challenging differential equation $(\partial T/\partial x) = \alpha(\partial^2 T/\partial x^2)$. We apply it here to a one-dimensional transient heat conduction problem, so $\alpha = (k/\rho c)$ is the thermal diffusivity. However, this analysis is identical to transient species diffusion or flow near a suddenly accelerated flat plate, if α is suitably interpreted. A single strip of quadrilateral elements is placed along the x-axis, in which all temperatures are initially set to zero. The right-hand boundary is then subjected to a step increase in temperature $(T(H,t) = T_H)$, and we want to compute the transient temperature variation $T(x,t)$. This is done by setting the time-stepping flag to 1 and looping the code repeatedly as described earlier.

The temperature profiles along the x-axis at various times are shown in Figure 6. These values should be compared with the theoretical solution $T = \mathrm{erfc}[(1-x)/2\sqrt{\alpha t}]$. Some numerical oscillations are noted at the heated boundary at short times due to the inability of the rather coarse mesh and time increment to capture the thermal boundary layer which forms there. However, this can easily be avoided if desired by using a finer mesh in that region, and also by stepping with shorter time increments initially.

2.5 demo5 – 10x10 grid, convection check $Pe = 10^6$

This problem illustrates the handling of convective thermal transport, and is taken from Reference [12]. As shown in Figure 7, a uniform velocity is imposed on a square mesh in a direction skewed to the coordinate axes, and two different temperatures are imposed along the lefthand and bottom boundaries. Normalized temperatures are used as shown, and the lower-left-hand corner is a compromise at $T = 0.5$. The temperature of the left boundary should be carried along the skew direction by the flow. Observation of the temperature contours, in particular the

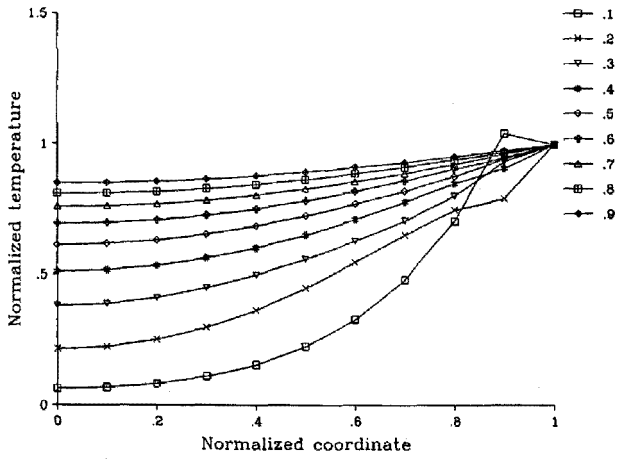

Figure 6: Temperature profiles in transient heat conduction.

lack of spreading along the stream direction, shows that streamline upwinding is able to model this difficult problem quite well.

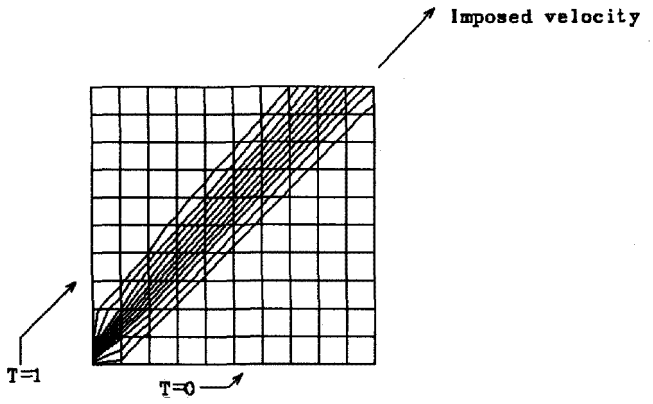

Figure 7: Temperature contours in convection-dominated flow

2.6 demo6 – thermally-driven buoyancy flow

This problem uses the same 10x10 mesh of `demo5`, modified to solve another problem with the same geometry. Here the vertical boundaries are held at fixed temperatures, the left hotter than the right, while the horizontal boundaries are left unconstrained. A linear temperature gradient is thus set up between the left and right boundaries, as can be seen by contouring the third degree of freedom in `post`. A body force term is present in the momentum equation, giving a vertical force of $\rho\alpha(T - T_a)$, where here α is a coefficient of volumetric thermal expansion. The cooler and denser fluid at the right will tend to move down and displace the warm fluid at the left, setting up a clockwise circulation as seen in the streamline contour plot of Figure 8. The code loops three times, finding the temperatures in the first pass, the velocities which depend

on the temperatures in the second pass, and the streamlines which depend on the velocities in the third pass.

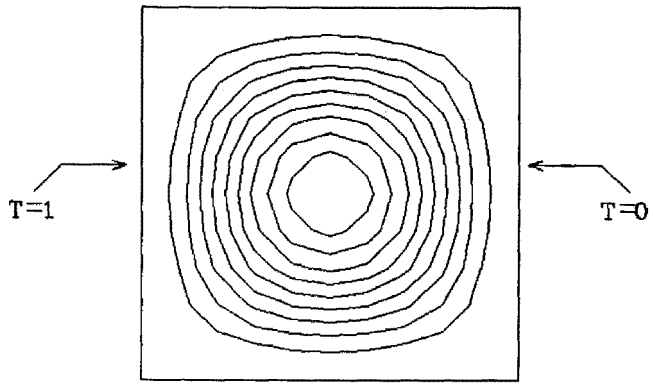

Figure 8: Streamlines for thermally-driven buoyancy flow.

3 References

[1] May, C.A., ed., *Chemorheology of Thermosetting Polymers,* American Chemical Society Symposium Series, No. 227, Washington, D.C., 1983.

[2] Ryan, M.E., "Rheological and Heat Transfer Considerations for the Processing of Reactive Systems," *Polymer Engineering and Science,* Vol. 24, pp. 698-706, June 1984.

[3] Bird, R.B., W.E. Stewart, and E.N. Lightfoot, *Transport Phenomena,* John Wiley & Sons, New York, 1960.

[4] Middleman, S., *Fundamentals of Polymer Processing,* McGraw-Hill Co., New York, 1977.

[5] Zienkiewicz, O.C., *The Finite Element Method,* McGraw-Hill Co., London, 1977.

[6] Baker, A.J., *Finite Element Computational Fluid Mechanics,* McGraw-Hill Co., New York, 1983.

[7] Roylance, D.K., "Use of 'Penalty' Finite Elements in Analysis of Polymer Melt Processing," *Polymer Engineering and Science,* vol. 20, pp. 1029-1034, 1980.

[8] Douglas, C. and D. Roylance, "Finite Element Analysis of Nonisothermal Polymer Processing Operations," *Finite Element Flow Analysis,* Elsevier North-Holland Inc., 1982. (Proceedings of the Fourth International Symposium on Finite Element Methods in Flow Problems, Tokyo, July 1982.)

[9] Roylance, D.K., "Finite Element Modeling of Nonisothermal Polymer Flows," *Computer Applications in Applied Polymer Science,* American Chemical Society Symposium Series, No. 197, pp. 265 - 276, 1982.

[10] Douglas, C. and D. Roylance, "Chemorheology of Reactive Systems: Finite Element Analysis," *Chemorheology of Thermosetting Polymers,"* ACS Symposium Series, No. 227, pp. 251-262, 1983.

[11] Hughes, T.R.J., W.L. Liu, and A. Brooks, "Finite Element Analysis of Incompressible Viscous Flows by the Penalty Function Formulation," *Journal of Computational Physics,* Vol. 30, pp. 1-60, 1979.

[12] Hughes, T.R.J. and A. Brooks, "A Multidimensional Upwind Scheme With No Crosswind Diffusion," *Finite Element Methods in Convection Dominated Flows,* American Society of Mechanical Engineers, pp. 19-36, 1979.

[13] Aylward, L., Douglas, C., and Roylance, D. "A Transient Finite Element Model for Pultrusion Processing," *Polymer Process Engineering,* vol. 3, pp. 247-261, 1985.

[14] Segerland, L.J., *Applied Finite Element Analysis,* John Wiley & Sons, New York, 1976.