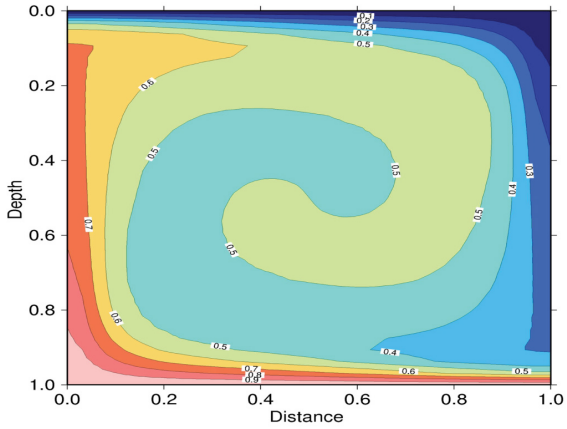
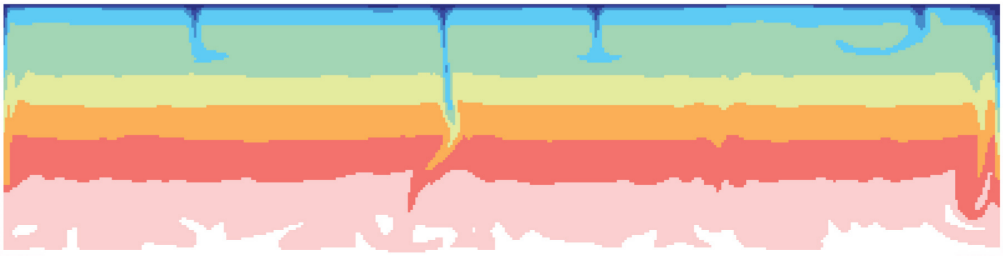
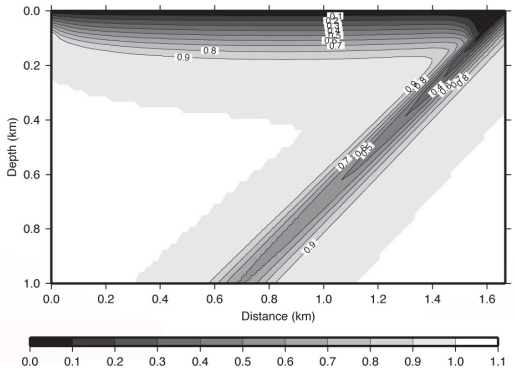
COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

ConMan

User Manual
Version 2.0







www.geodynamics.org

Scott King

ConMan

Scott King
Version 2.0

July 17, 2008

Contents

List of Figures

Chapter 1

Preface

1.1 Abstract

This manual serves as a user guide for ConMan, a vectorized finite element program for the solution of the equations of incompressible, infinite-Prandtl number convection in two dimensions originally written by Arthur Raefsky, Scott King, and Brad Hager. ConMan is a public domain program and is distributed free of charge to anyone who wishes to use it and may be freely copied and modified. ConMan is written in Standard Fortran 77 with cray pointers and runs on most unix systems with many fortran comilers (see Section 3.4 [TODO – 3.4 is placeholder section. Is this correct? Need label to put cross ref here]). Porting it to other systems should be straightforward. As with anything free it comes with no guarantees, but it has been benchmarked against other existing codes (see Chapter 7). The authors would appreciate any information regarding bugs or potential problems but make no promises regarding the timeliness of changes or fixes; see Section 3.5 for instructions on how to report problems.

1.2 Introduction

This manual contains all of the necessary information for setting up input and running ConMan. It assumes some familiarity with the finite element method and Fortran. An excellent reference book for more detail on the finite element method is *The Finite Element Method* by T.J.R. Hughes. All of the data structures and bookkeeping arrays in ConMan follow the conventions in Hughes so for the person who wishes to make extensive use of ConMan, this book is a worthwhile investment.

This manual is broken up into several parts: it begins with a brief introduction to the finite element method and the notation that is used throughout the manual and ConMan. There is a discussion of the equations solved and the material properties including how and where to modify the code. There is also discussion of some key points concerning the implementation and finally a description of all the input variables. Within this document the following convention will be followed: subroutine names from ConMan will be given in **bold** type, variables from ConMan will be given in *italicized* type.

1.3 Citation

Computational Infrastructure for Geodynamics (CIG) is making this source code available to you in the hope that the software will enhance your research in geophysics. The ConMan code was donated to CIG in June 2008. A number of individuals have contributed a significant portion of their careers toward the development of ConMan. It is essential that you recognize these individuals in the normal scientific practice by making appropriate acknowledgements.

The code is based on the method described in

- King, S.D., A. Raefsky, and B.H. Hager, ConMan: Vectorizing a finite element code for incompressible two-dimensional convection in the Earth's mantle,Phys. Earth Planet. Int., 59, 195-208, 1990.

The code was originally developed by Scott King, Arthur Raefsky and Brad Hager, although many people have contributed improvements to ConMan over the past 15 years. The ConMan team requests that in your oral presentations and in your papers that you indicate your use of this code and acknowledge the author of the code and CIG (`www.geodynamics.org`).

1.4 Support

ConMan maintenance is supported by a grant from the National Science Foundation to CIG, managed by the California Institute of Technology, under Grant No. EAR-0406751.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Chapter 2

Computational Approach and Governing Equations

2.1 The Finite Element Method

I am going to follow a similar form to Hughes (The Finite Element Method) Chapter 1 sections 1-4. There are two forms we can write the equation, the strong and the weak form. You are used to seeing the strong form, but not used to seeing the weak form. The finite element method, is cast in the weak form. In elasticity, for example, the weak form comes from a variational principal, such as the principal of virtual displacements. For viscous flow, there is also a variational form, but we will not go into that.

In general, the finite element method takes a differential equation (strong form) and transforms it into an integral equation (weak form).

2.1.1 The Strong Form

For example, the strong form of this simple equation is stated as follows:

Given $f(x) : [0,1] \rightarrow \Re$ and constants g and h, find $u : [0,1] \rightarrow \Re$, such that

$$u_{,xx}(x) + f(x) = 0 \tag{2.1}$$

$$u(1) = g \tag{2.2}$$

$$-u_{,x}(0) = h \tag{2.3}$$

This choice of initial conditions allows us to examine both kinds of boundary conditions. The solution is trivial, but that does not matter. For completeness, it is

$$u(x) = g + (1-x)h + \int_x^1 \left(\int_0^y f(z)dz \right) dy \tag{2.4}$$

2.1.2 The Weak Form

The weak form of the corresponding boundary value problem is stated:

Given f, g and h, as before. Find $u(x) \epsilon \mathcal{L}$ such that for all $w(x) \epsilon \nu$

$$\int_0^1 w_{,x}(x)\, u_{,x}(x) = \int_0^1 w(x)\, f(x)\, dx + w(0)\, h \tag{2.5}$$

ν is the set of weighting functions defined by

$$\nu = \left\{ w\left(x\right) | w\left(x\right) \epsilon H^1,\, w\left(1\right) = 0 \right\} \tag{2.6}$$

and $\mathcal{L}$ is a set of trial solutions defined by

$$\mathcal{L} = \left\{ u\left(x\right) | u\left(x\right) \epsilon \mathrm{H}^1,\, u\left(1\right) = g \right\} \tag{2.7}$$

H^1 is the set of all functions whose first derivatives are square integrable on [0, 1]. The integral equation is then solved by integrating over each element in the domain and adding the result. The result is a large sparse matrix equation of the form

$$[K]\, x = b \tag{2.8}$$

where *[K]* is referred to as the element stiffness matrix. There will be more to say about the implementation in Section 4.

2.1.3 Galerkin's Approximation

Now we have a start on the finite element method. I want to continue to follow Hughes; however his notation becomes quite difficult to keep up with. Now, lets begin to think about putting a solution on the computer. Because we will have a finite approximation, related to how fine we space our grid, our solution will only approximate the real solution. Following Hughes' notation, the solution on the grid will be denoted as u^h where h is some measure of the spacing at the grid. Then,

$$\int_0^1 w^h{}_{,x}\, u^h{}_{,x}\, dx \;=\; \int_0^1 w^h\, f^h\, dx + w^h(0)h. \tag{2.9}$$

approximates our exact solution u.

On a computer, we don't have a continuous solution. We have a solution at descrete points. We need to approximate the solution between the points (in order to integrate over the function). We will do this with **shape functions** as they are usually called in the finite element language. Hughes uses $N_A A = 1, 2, \cdots, n$ to denote the shape functions. You can also think of these as basis functions or interpolation functions. We require $N_A(1) = 0, A = 1, 2, \cdots, n$. In order to specify our boundary condition, we need another shape function which has the property

$$N_{n+1}(1) = 1. \tag{2.10}$$

Then, g^h is given by,

$$g^h = g N_{n+1} \tag{2.11}$$

and thus,

$$g^h(1) = g. \tag{2.12}$$

With these definitions, we can write our solution u^h as

$$u^h = \sum_{A=1}^{n} d_A\, N_A + g N_{n+1} \tag{2.13}$$

where the d_A's are unknown constants to be solved for.

In the next section we will make the shape functions more concrete. I want you to see how general this is, because in principle there is a great deal of flexibility in how we choose the shape functions.

We have not said anything more about this function w^h and how we are going to choose it. If our shape functions form a basis set for the grid, then we can represent **any** function as a sum of the basis functions times some arbitrary coefficients c_i,

$$w^h = \sum_{A=1}^{n} c_A\, N_A = c_1\, N_1 \;+\; c_2\, N_2 \;+\; \cdots \;+\; c_n\, N_n \tag{2.14}$$

If you don't remember this part of your mathematics background think of Fourier series. Any function one-dimensional function can be represented as an infinite series of sines and cosines times some unique set of coefficients. The shape functions form a similar kind of basis set.

Notice that because we required that $N_A(1) = 0, A = 1, 2, \cdots, n$, Equation 2.14 satisfies the requirement that $w^h(1) = 0$, as necessary.

Using our definitions of the w^h's and our approximation for u^h, we can get the messy expression for Equation 2.9

$$\int_0^1 \left(\frac{\partial}{\partial x} \left(\sum_{A=1}^{n} c_A \, N_A \right) \frac{\partial}{\partial x} \left(\sum_{B=1}^{n} d_B \, N_B + g N_{n+1} \right) \right) dx \;\; = \;\; \int_0^1 \sum_{A=1}^{n} c_A \, N_A \, f^h \, dx + \sum_{A=1}^{n} c_A N_A(0) h. \tag{2.15}$$

By rearranging, we can write

$$\sum_{A=1}^{n} G_A c_A = 0 \tag{2.16}$$

where

$$G_A \;\; = \;\; \int_0^1 \left(\frac{\partial N_A}{\partial x} \right) \left(\sum_{B=1}^{n} d_B \, \frac{\partial N_B}{\partial x} \right) dx - \int_0^1 N_A \, f^h \, dx - N_A(0) h + \int_0^1 \frac{\partial N_A}{\partial x} \frac{\partial N_{n+1}}{\partial x} \, g \, dx \tag{2.17}$$

Now I use the fact that the shape functions are basis functions, so $N_A \times N_B$ is zero except when $A = B$. We could equally well use the fact that the c_A's are arbitrary. Both of these force us to conclude that each G_A must be identically zero and we get

$$\sum_{B=1}^{n} \left(\int_0^1 \frac{\partial N_A}{\partial x} \frac{\partial N_B}{\partial x} \, dx \right) d_B = \int_0^1 N_A \, f^h \, dx + N_A(0) h - g \int_0^1 \frac{\partial N_A}{\partial x} \frac{\partial N_{n+1}}{\partial x} \, dx \tag{2.18}$$

Everything in Equation 2.18 is known except the d_B's. This constitutes a system of n equations and n unknowns. We can think of the left hand side as a matrix, K_{AB} whose entries are

$$\int_0^1 \frac{\partial N_A}{\partial x} \frac{\partial N_B}{\partial x} \, dx \tag{2.19}$$

We can write

$$\sum_{B=1}^{n} K_{AB} d_B = F_A, \qquad A = 1, 2, \cdots, n \tag{2.20}$$

or as a matrix equation

$$[K] \; \{d\} = \{f\} \tag{2.21}$$

where

$$[k] = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & \cdots & K_{2n} \\ \vdots & \vdots & & \vdots \\ K_{n1} & K_{n2} & \cdots & K_{nn} \end{bmatrix} \tag{2.22}$$

By tradition, $[K]$ is the stiffness matrix, $\{f\}$ is the force vector, and $\{d\}$ is the displacement vector. When the problem under consideration pertains to a mechanical system, this makes the most sense, but even in heat conduction problems, or fluid flow problems, the terminology is still (often) retained.

2.1.4 Shape Functions

At this point, I will narrow the focus to deal with specifically the elements in my code, ConMan. It is possible to think very general shape functions, but in practice, people use triangles or quadralaterals (in 2-D). We will extend our finite element formulation to a 2-D equation next time. In terms of the level of approximation, there are also a lot of possibilities. We will stick to the simplest form, bi-linear elements, but you should be aware that higher order elements (bi-quadratic or bi-cubic spline elements) are also popular with some people. I am condensing a lot of very useful material from Chapter 3 of Hughes' book into one lecture. If you want to see more complete derivations, proofs of convergence, etc., of how to go about using higher order elements, look at Hughes book, Chapter 3.

Lets start by thinking of a rectangle that is 2a by 2b in length centered at (0,0). There are two properties we would like the shape functions to have

$$\sum_{A=1}^{4} N_A(X,Y) = 1 \tag{2.23}$$

$$\sum_{A=1}^{4} N_A(X,Y)\, X_A = X \tag{2.24}$$

$$\sum_{A=1}^{4} N_A(X,Y)\, Y_A = Y \tag{2.25}$$

Equation 2.23 says that they are normalized, so that they sum to one (everywhere on X,Y). Equations 2.24 and 2.25 state that the shape functions are also interpolation functions. Without doing a lot of derivation, I will claim that for the rectangle described above,

$$N_1 = \frac{(a-x)(b-y)}{4ab} \tag{2.26}$$

$$N_2 = \frac{(a+x)(b-y)}{4ab} \tag{2.27}$$

$$N_3 = \frac{(a+x)(b+y)}{4ab} \tag{2.28}$$

$$N_4 = \frac{(a-x)(b+y)}{4ab} \tag{2.29}$$

these shape functions satisfy the conditions in Equation 2.23 and Equations 2.24 and 2.25. A good exercise would be to show this is true.

> **Problem 7** Show that the shape functions defined above satisfy Equation 2.23 and Equations 2.24 and 2.25.

Notice that by convention, I start numbering my element nodes in the lower left hand corner and work counter-clockwise. *This is a very important point. It is followed through out all my finite element codes.* There is no magic reason, you just have to choose a starting place.

In ConMan, as in Hughes, we further choose to normalize this by setting $a = 1$ and $b = 1$. This choice gives us an element whose area is 1, which is a convenient way to think about things. (This is because we left the factor of $1/4$ in the denominator). In my code, to make one less set of computations, I in effect set $a = 0.5$ and $b = 0.5$ so that the denominator goes to 1.

Notice it is pretty easy to take derivatives of these shape functions

$$N_{1,x} = \frac{-(1-y)}{4} \tag{2.30}$$

$$N_{2,x} = \frac{(1-y)}{4} \tag{2.31}$$

$$N_{3,x} = \frac{(1+y)}{4} \tag{2.32}$$

$$N_{4,x} = \frac{-(1+y)}{4} \tag{2.33}$$

$$N_{1,y} = \frac{-(1-x)}{4} \tag{2.34}$$

$$N_{2,y} = \frac{-(1+x)}{4} \tag{2.35}$$

$$N_{3,y} = \frac{(1+x)}{4} \tag{2.36}$$

$$N_{4,y} = \frac{(1-x)}{4} \tag{2.37}$$

What do we do if we want to solve a problem on a domain that is not convenient to split into a grid of 1 by 1 unit elements? We use an important principle of mathematics, the jacobian of the transformation

$$K_{11} = \int_A^B N_{1,x} N_{1,x}\, dx = \int_0^1 N_{1,x} N_{1,x}\, J dx \tag{2.38}$$

where J is the Jacobian of the transformation. This is a very powerful point. When we are thinking of solving a regular Cartesian domain, this just corresponds to a stretching or a shrinking (notice we set $a = b = 1$ above. However, if we are thinking about a cylindrical geometry, for example, we can use the Jacobian of the transformation between the geometries. Lets look at two examples:

Converting an element 0.05 by 0.10 centered at (0.1,0.2) to the 'parent element' centered at (0,0). Hughes also uses ξ, η for the X, Y coordinate pair in the 'parent element' So we could write

$$x = 0.1 + 0.05\xi + 0.0\eta \tag{2.39}$$

$$y = 0.2 + 0.0\xi + 0.10\eta \tag{2.40}$$

or in matrix form we could write

$$\left\{ \begin{array}{c} x \\ y \end{array} \right\} = \left[\begin{array}{cc} 0.05 & 0.0 \\ 0.0 & 0.10 \end{array} \right] \left\{ \begin{array}{c} \xi \\ \eta \end{array} \right\} + \left\{ \begin{array}{c} 0.1 \\ 0.2 \end{array} \right\} \tag{2.41}$$

Where $[J]$ is the Jacobian of the transformation. If the transformation were from an arbitratry shaped quadralateral to the parent element, then the off diagonal terms will not be zero. It is easy enough to show that

$$\int_{x_1}^{x_2} \int_{y_1}^{y_2} f(x,y)\, dx\, dy = \int_{-1}^{1} \int_{-1}^{1} f(\xi,\eta) \det[J]\, d\xi\, d\eta \tag{2.42}$$

It turns out, and it is also easy to show, that $\det[J]$ is the ratio of the areas when going from one rectangle to another (in fact any Cartesian to Cartesian transformation).

Advanced Topic: Now suppose we want to map a cylindrical domain to our 'parent element.' We can use the same principle in this case:

$$x = r\cos\theta = \cos\theta\,\xi - r\sin\theta\,\eta \tag{2.43}$$

$$y = r\sin\theta = \sin\theta\,\xi + r\cos\theta\,\eta \tag{2.44}$$

so

$$\det[J_{geometry}] = r\cos^2\theta + r\sin^2\theta = r. \tag{2.45}$$

of we would get

$$\int_{r_1}^{r_2} \int_{\theta_1}^{\theta_2} f(r\cos\theta, r\sin\theta)\, r\, dr\, d\theta = \int_{-1}^{1} \int_{-1}^{1} f(\xi,\eta) \det[J_{area}]\, d\xi\, d\eta \tag{2.46}$$

2.1.4.1 Gauss Quadrature

An amazing fact, that makes the idea of finite elements easy and powerful is Gauss Quadrature. Gauss Quadrature is a way to turn an integral into a summation. Lets begin with a 1-D example, f(x) = c

$$\int_{-1}^{1} c\, dx = cx|_{-1}^{1} = 2c \tag{2.47}$$

Gauss noted that for any linear function

$$\int_{-1}^{1} f(x)\,dx = 2.0 \times f(0) = 2c \tag{2.48}$$

For a linear function, f(x) = a x + b

$$\int_{-1}^{1} f(x)\,dx = f(\frac{-1}{\sqrt{3}}) + f(\frac{-1}{\sqrt{3}}) \tag{2.49}$$

The direct way,

$$\int_{-1}^{1} (ax+b)\,dx = (\frac{ax^2}{2} + bx)|_{-1}^{1} = \frac{a}{2} + b - (\frac{a}{2} - b) = 2b \tag{2.50}$$

Gauss' way

$$\int_{-1}^{1} (ax+b)\,dx = a\frac{-1}{\sqrt{3}} + b + a\frac{1}{\sqrt{3}} + b = 2b \tag{2.51}$$

It turns out that $\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$ are exact for a linear equation, but from what I showed, so would any $-x, x$ combination, but what Gauss showed was more powerful, that if the function is of higher order, the $\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$ choice is the best approximation you can make with only two terms. If we go to three terms, the choice would be $-\sqrt{\frac{3}{5}}, 0, \sqrt{\frac{3}{5}}$.

To integrate a 2D Cartesian region, like our parent element, it turns out that 2 by 2 quadrature, or the four points

$$\xi = \frac{-1}{\sqrt{3}} \qquad \eta = \frac{-1}{\sqrt{3}} \tag{2.52}$$

$$\xi = \frac{1}{\sqrt{3}} \qquad \eta = \frac{-1}{\sqrt{3}} \tag{2.53}$$

$$\xi = \frac{1}{\sqrt{3}} \qquad \eta = \frac{1}{\sqrt{3}} \tag{2.54}$$

$$\xi = \frac{-1}{\sqrt{3}} \qquad \eta = \frac{1}{\sqrt{3}} \tag{2.55}$$

are sufficient to exactly integrate our bilinear shape functions over the -1,-1 to 1,1 domain.

At this point, it would be worth talking about the code ConMan for a minute. The shape functions are generated in ConMan in the routine **genshp** for GENerate SHape functions Parent domain. If you look at the routine you will find the first part of it is pretty easy to follow from the discussion above. Some of the second part is a little tricky in the details, but generally it is also pretty easy to follow.

The subroutine **genshg** deals with the global shape functions (i.e., deals with the geometry and size). Originally, we called this routine once and stored all the shape functions. As problem sizes have grown, this took a lot of storage, so now we call it on the fly for each element when needed. It is computationally more expensive but cuts storage. This strategy will also be necessary for a Lagrangian formulation or adaptive gridding.

There are two domains to keep in mind when thinking about the finite element method: the global domain and the parent element domain (Figure 2.1). All calculations are done in the parent element domain and the results are assembled into the global equations. This means all calculations can be done for a single parent element. Elements of different sizes or shapes filling an irregular global domain geometry (i.e., non-rectangular) can be solved by the same program. The only difference between these elements is the Jacobian of the transformation between the input domain and the parent element domain, which is calculated in routine **genshg**.

For ConMan the choice was made to use bilinear quadrilaterals as the parent elements (Figure 2.2). Higher order elements (i.e., biquadratic or bicubic-spline) require more computational work per element. It has been our experience that using grid refinement, rather than using high-order elements, is the best strategy for an efficient, accurate code for incompressible, advection-diffusion problems.

Figure 2.1: The mapping between the global domain (right) and the parent element domain (left) using the shape functions. Figure taken from Hughes, Sec 3.2.

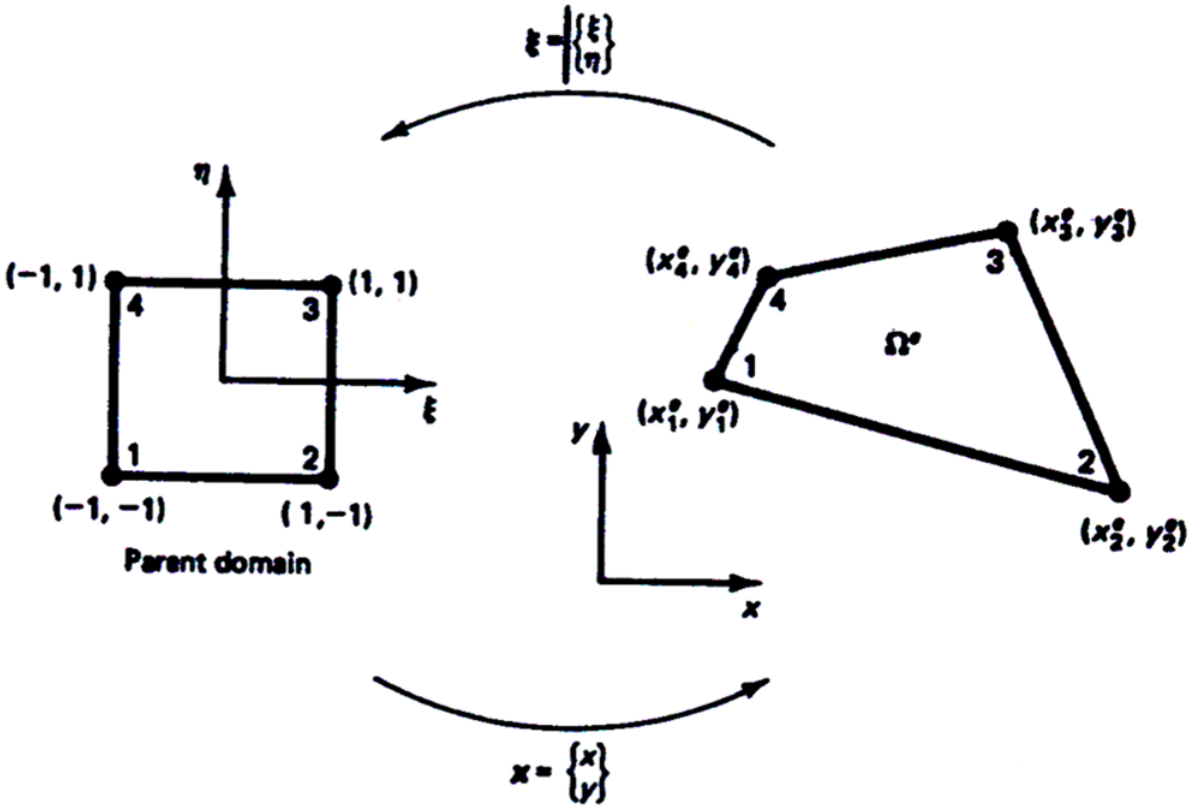


Figure 2.2: The bilinear shape function for a single element (top) and the four elements whose shape functions combine to form the global shape function for node A (bottom). Figure taken from Hughes, Sec 3.2.

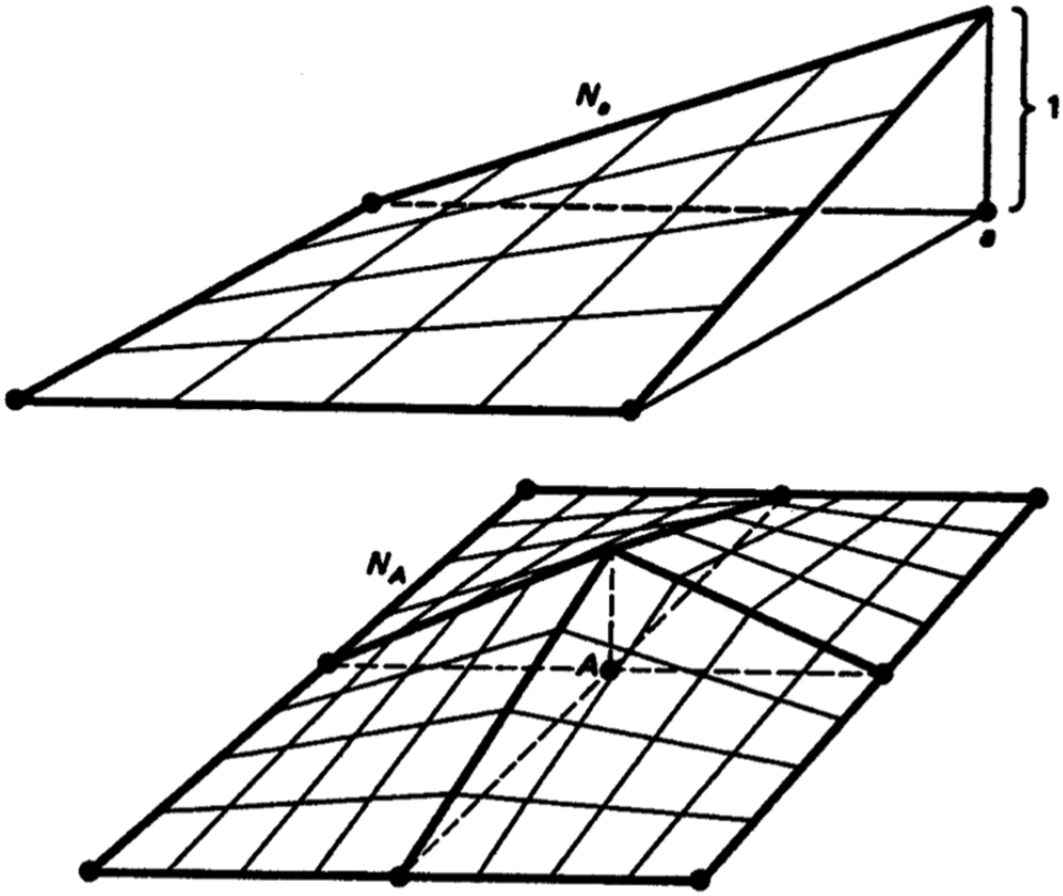
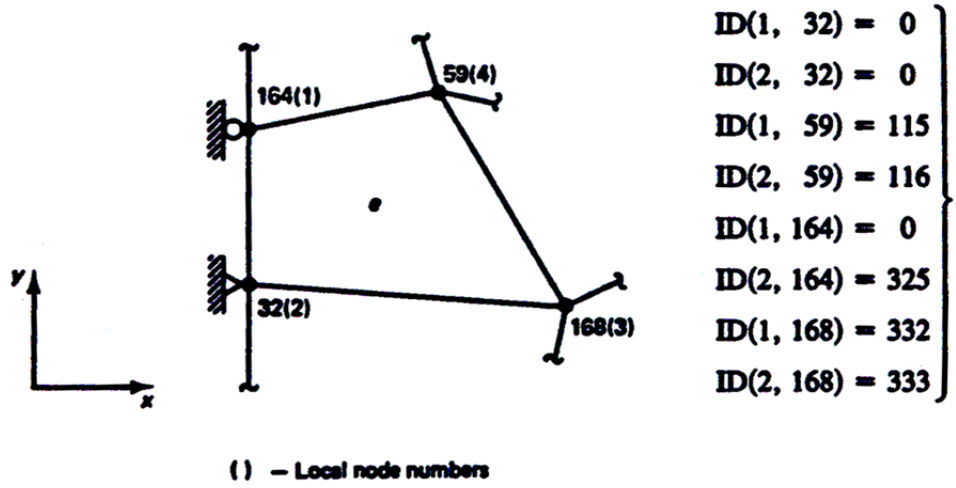


Figure 2.3: Example relationship between global nodes and equation numbers for a 2 degree of freedom problem using the id array. An equation number of zero denotes a boundary condition. Figure taken from Hughes, Sec 3.2.



Because of the changing between domains, it is necessary to define several bookkeeping arrays to identify nodes and elements in each of the domains.

id transforms global nodes to equation numbers (Figure 2.3).

ien transforms element local node numbers to global node numbers (Figure 2.4).

lm transforms element local node numbers to global equation numbers.

With these, the code is able to go back and forth between the parent element domain and the global domain. Global node numbering is specified by the user, and equation numbers are assigned by the code to denote the row in the stiffness matrix corresponding to the degree(s) of freedom for that node. One global node may have more than one equation number (since there may be more than one degree of freedom per node). Boundary conditions are specified with a zero equation number. Since it is a sparse matrix, it is desirable to permute the stiffness matrix for computational efficiency. These arrays spare the user from dealing with the transformations, while making the code efficient.

In the code, the data structures for these two arrays are

id (degree-of-freedom , global-node-number) = equation-number

ien (local-node-number, element-number) = global-node-number

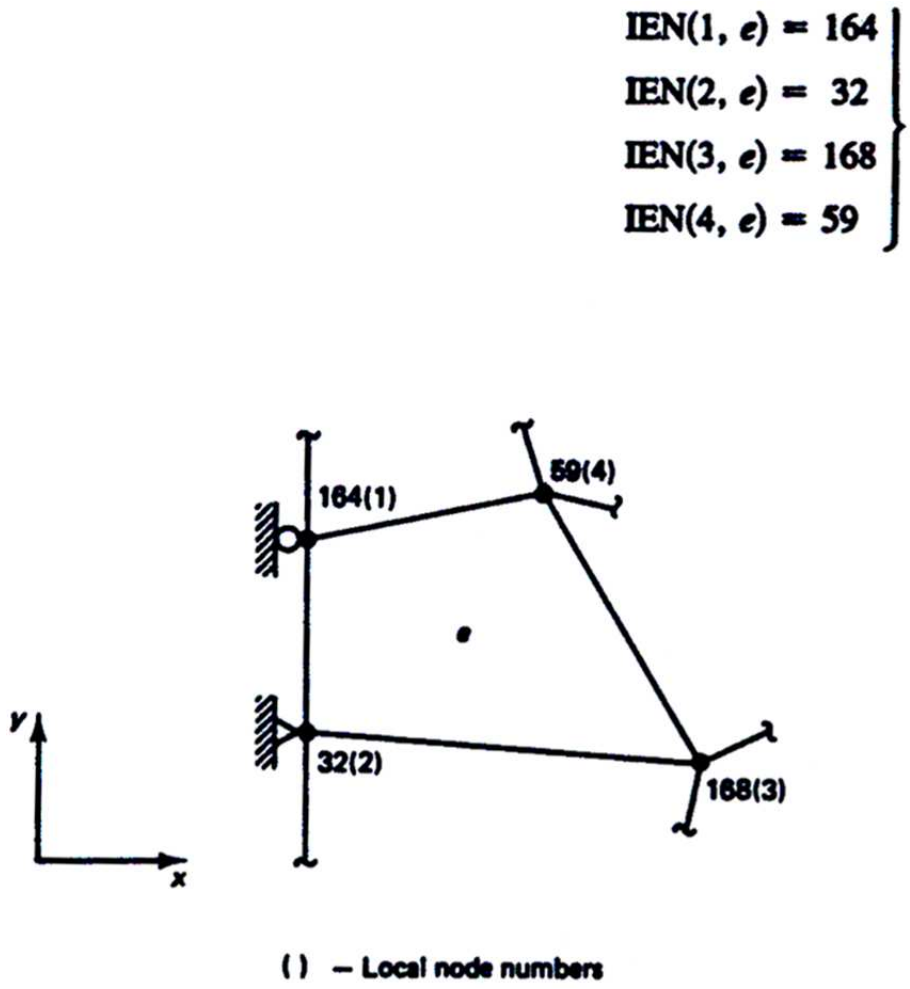
lm (degree-of-freedom, local-node-number, element-number) = global-equation-number

2.1.5 The Element Point of View

Transforming between the element and global points of view is done with the data structure called the IEN array for Element to Node transformation. The IEN array takes an element number and a local node number and it's value is the global node number. It is easiest to look at some examples:

Consider a 3 element by 3 element grid. I will number my elements and node starting in the lower left hand corner, and working up fastest.

Figure 2.4: Example relationship between global node numbers and local element numbers using the ien array. Local nodes are numbered counterclockwise from the bottom left hand corner. Figure taken from Hughes, Sec 3.2.



```
 For element 3: ien (3, 1) = 3 ien (3, 2) = 7 ien
(3, 3) = 8 ien (3, 4) = 4 For element 5: ien (5, 1) = 6 ien (5, 2)
= 10 ien (5, 3) = 11 ien (5, 4) = 7
```

There are three kinds of opperations we might think of wanting. The first is taking values of some function (coordinates, velocities, temperatures, stresses, etc.) defined on the global grid and getting the values for a single element, this is called a *gather* operation. The next is taking values in an element and spreading them out to the global array, this is called a *scatter* operation. The third operation is to take the value at a local node and add it to the global value for that node, an *assembly* step.

All three operations, gather, scatter, and assemble are done by the routine local. Because it is called by the genshp routine above, it is a good example to look at.

2.1.6 Equations

$$\tau_{ij,j} + f_i = 0 \tag{2.56}$$

$$u_{i,i} = 0 \tag{2.57}$$

where

$$\tau_{ij} = -p\delta_{ij} + 2\mu u_{(i,j)} \tag{2.58}$$

where

$$u_{(i,j)} = (u_{i,j} + u_{j,i})/2 \tag{2.59}$$

We replace equation 2.58 with the following relationships

$$\tau_{ij} = -p^\lambda \delta_{ij} + 2\mu u_{(i,j)} \tag{2.60}$$

$$0 = u_{i,i} + p^\lambda/\lambda. \tag{2.61}$$

As λ approaches infinity, these relations approach the incompressible solution. Also, as λ approaches infinity, p^λ approaches the hydrostatic pressure in the incompressible case. In general, the hydrostatic pressure is $-\tau_{ii}/3$. Substituting Equation 2.61 into 2.60 we get

$$\tau_{ij} = \lambda u_{i,i}\delta_{ij} + 2\mu u_{(i,j)} \tag{2.62}$$

or

$$\tau_{ii} = 3\lambda u_{i,i} + 2\mu u_{i,i} \tag{2.63}$$

or

$$\tau_{ii}/3 = -p = (\lambda + 2/3\mu)u_{i,i} \tag{2.64}$$

but we also have

$$-p^\lambda = \lambda u_{i,i} \tag{2.65}$$

from Equation 2.61. Clearly in the incompressible limit $\lambda \gg \mu$ then $\lambda + 2/3\mu \to \lambda$ and $p^\lambda \to p$. Also note that the continuity equation is satisfied.

Now, substituting Equation 2.62 into Equation 2.56 we have

$$\{\lambda u_{i,i}\delta_{ij} + 2\mu u_{(i,j)}\}, j + f_i = 0 \tag{2.66}$$

At this point, it is probably easier to switch to differential notation. I will also specialize to 2-D:

$$\frac{\partial}{\partial x}\{\lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial z}) + 2\mu(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial x})/2\} + \frac{\partial}{\partial z}\{2\mu(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial z})/2\} + f_x = 0 \tag{2.67}$$

$$\frac{\partial}{\partial x}\{2\mu(\frac{\partial u}{\partial z} + \frac{\partial v}{\partial x})/2\} + \frac{\partial}{\partial z}\{\lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial z}) + 2\mu(\frac{\partial v}{\partial z} + \frac{\partial v}{\partial z})/2\} + f_z = 0 \tag{2.68}$$

These are second order partial differential equations. Simplifying, I get

$$\lambda(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial z}) + 2\mu\frac{\partial^2 u}{\partial x^2} + \mu(\frac{\partial^2 u}{\partial z^2} + \frac{\partial^2 v}{\partial z \partial x}) + f_x = 0 \tag{2.69}$$

$$\lambda(\frac{\partial^2 u}{\partial z \partial x} + \frac{\partial^2 v}{\partial z^2}) + \mu(\frac{\partial^2 u}{\partial x \partial z} + \frac{\partial^2 v}{\partial x^2}) + 2\mu\frac{\partial^2 v}{\partial z^2} + f_z = 0 \tag{2.70}$$

Now we use the same technique (approach) as we used in Possion's equation to turn the differential form into an integral form. You can either look at it as we find the variational form of the stokes equation (which is what we are doing) or you can think of it as multiplying by a weighting function w and integrating over the domain. Then using integration by parts to convert the second derivatives to first derivatives. This is done in carefully by Hughes on pages 197-200, but he has left out a number of intermediate steps. Nothing about this step is hard, it is just tedious. There is, however, a cleaver short cut. If we return to the messy equations at the top of the page, multiply them by the weighting function w and integrate over the domain, then we do not have to use integration by parts. If you are confused, or don't believe me, then you should take the equations directly above this paragraph, multiply by a weighting function w and integrate over the 2-D domain Ω, then use integration by parts. You will find (after a little algebra)

$$\int\int_\Omega \frac{\partial w}{\partial x}\{\lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial z}) + 2\mu\frac{\partial u}{\partial x}\} + \frac{\partial w}{\partial z}\{2\mu(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial z})/2\}\, d\Omega + \int\int_\Omega f_x\, w\, d\Omega = b.c.\ terms \tag{2.71}$$

$$\int\int_\Omega \frac{\partial w}{\partial x}\{2\mu(\frac{\partial u}{\partial z} + \frac{\partial v}{\partial x})/2\} + \frac{\partial w}{\partial z}\{\lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial z}) + 2\mu\frac{\partial v}{\partial z}\}\, d\Omega + \int\int_\Omega f_z\, w\, d\Omega = b.c.\ terms \tag{2.72}$$

Note that we don't get something for nothing, this short cut does not give us the boundary condition terms (velocity or flux). These would fall out of the integration by parts. Recall,

$$\int_a^b w\, dv = w\, v|_a^b - \int_a^b v\, dw \tag{2.73}$$

were in our case w is the weighting function and v is the second derivative term. The first term gives us the flux (first derivative) boundary conditions. In the case of the momentum equations, that is the applied tractions (or stress boundary conditions).

Now we make use of Galerkin's approximation, or more simply, we use the same weighting functions as we use for interpolation function, i.e., the shape functions, N. So we substitute

$$\frac{\partial w}{\partial x} = N_x \tag{2.74}$$

$$\frac{\partial w}{\partial z} = N_z \tag{2.75}$$

$$\frac{\partial u}{\partial x} = u\, N_x \tag{2.76}$$

$$\frac{\partial u}{\partial z} = u\, N_z \tag{2.77}$$

$$\frac{\partial v}{\partial x} = v\, N_x \tag{2.78}$$

$$\frac{\partial v}{\partial z} = v\, N_z \tag{2.79}$$

into our weak form equations. Although messy, that is straight-forward.

$$\int\int_\Omega N_x\{\lambda(u\, N_x + v\, N_z) + 2\mu u\, N_x\} + N_z\{\mu(v\, N_x + u\, N_z)\}\, d\Omega +$$

$$\int\int_{\Omega} f_x \, w \, d\Omega = b.c. \; terms \tag{2.80}$$

$$\int\int_{\Omega} N_x\{\mu(u\,N_z + v\,N_x)\} + N_z\{\lambda(u\,N_x + v\,N_z) + 2\mu v\,N_z\}\,d\Omega + \int\int_{\Omega} f_z \, w \, d\Omega = b.c. \; terms \tag{2.81}$$

At this point, it is useful to separate the equations into a λ part and a μ part. We can also write them as a 2-D matrix equation

$$[K_\lambda] = \begin{bmatrix} N_x\lambda N_x & N_x\lambda N_z \\ N_z\lambda N_x & N_z\lambda N_z \end{bmatrix} \tag{2.82}$$

and

$$[K_\mu] = \begin{bmatrix} N_x 2\mu N_x + N_z\mu N_z & N_z\mu N_x \\ N_x\mu N_z & N_z 2\mu N_z + N_x\mu N_x \end{bmatrix}. \tag{2.83}$$

Hughes makes use of an interesting, and important observation. This observation will greatly simplify constructing the stiffness matrix for arbitrary coordinate systems. We can rewrite the stiffness matrices above in the following form:

$$[K_\lambda] + [K_\mu] = [B]^T[D][B] \tag{2.84}$$

$$[D_\lambda] + [D_\mu] = [D] \tag{2.85}$$

where

$$[D_\mu] = \mu \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.86}$$

and

$$[D_\lambda] = \lambda \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.87}$$

and

$$[B] = \begin{bmatrix} N_x & 0 \\ 0 & N_z \\ N_z & N_x \end{bmatrix}. \tag{2.88}$$

The momentum and energy equations form a simple coupled system of differential equations. We treat the incompressibility equation as a constraint on the momentum equation and enforce incompressibility in the solution of the momentum equation using a penalty formulation described below. Since the temperatures provide the buoyancy (body force) to drive the momentum equation and since there is no time-dependence in the momentum equation, the algorithm to solve the system is a simple one: Given an initial temperature field, calculate the resulting velocity field. Use the velocities to advect the temperatures for the next time step and solve for a new temperature field. If the time stepping for the temperature equation is stable, then this method is stable and converges as $\Delta t \to 0$.

The element stiffness matrix (Figure 2.5) is made up of the two terms from the left hand side of the integral equation. The integration is done using two by two gauss quadrature, which is exact when the elements are rectangular and bilinear shape functions are used. The λ term is under-integrated (one point rule) to keep the large penalty value from effectively locking the element (Malkus and Hughes, 1978). The right hand side is made up of three known parts, the body force term (f_i), the applied tractions (h_i) and the applied velocities (g_i). The momentum equation is equivalent to an incompressible elastic problem, and the resulting stiffness matrix will always be positive definite (Hughes, 1986 p. 84-89). This allows us to consider only the upper triangular part of the stiffness matrix and save both storage and operations using Cholesky factorization. More details of the method and a formal error analysis can be found in Hughes, Liu and Brooks (1979).

The stiffness matrix is formed in routine **f_vstf** and the right hand side is formed in routine **f_tres**.

Figure 2.5: The element stiffness matrix for the 2D Cartesian stokes equation. The 8 by 8 matrix is make up of 16 2 by 2 submatrices of the form shown below. The λ and μ parts are shown separately for clarity.

$$[K]_\nu = \nu \begin{bmatrix} 2N_x(i)N_x(j) + N_y(i)N_y(j) & N_x(j)N_y(i) \\ N_x(i)N_y(j) & N_x(i)N_x(j) + 2N_y(i)N_y(j) \end{bmatrix}$$

$$[K]^e = [K]_\nu^e + [K]_\lambda^e$$

$$[K]_\lambda = \lambda \begin{bmatrix} N_x(i)N_x(j) & N_x(i)N_y(j) \\ N_x(j)N_y(i) & N_y(i)N_y(j) \end{bmatrix}$$

Figure 2.6: The storage for the stiffness matrix used in routine **f_vstf**.

$$[K]^e = \begin{array}{c|cccccccc|c} & \multicolumn{8}{c|}{i=} & j= \\ & 1 & 2 & 3 & 4 & & & & & \\ \hline 1 & 2 & 4 & 7 & 11 & 16 & 22 & 29 & & \\ & 3 & 5 & 8 & 12 & 17 & 23 & 30 & & 1 \\ \hline & & 6 & 9 & 13 & 18 & 24 & 31 & & \\ & & & 10 & 14 & 19 & 25 & 32 & & 2 \\ \hline & & & & 15 & 20 & 26 & 33 & & \\ & & & & & 21 & 27 & 34 & & 3 \\ \hline & & & & & & 28 & 35 & & \\ & & & & & & & 36 & & 4 \end{array}$$

I'm not sure if we need this too...

The energy equation is an advection-diffusion equation. The formal statement is
Find $T : \Omega \rightarrow R$ such that

$$\dot{T} + u_i T_{,i} = \kappa T_{,ii} + H \quad on\ \Omega \tag{2.89}$$

$$T = b \quad on\ \Gamma_b \tag{2.90}$$

$$T_{,j} n_j = q \quad on\ \Gamma_q \tag{2.91}$$

where T is the temperature, u_i is the velocity, κ is the thermal diffusivity and H is the internal heat source. The weak form of the energy equation is given by

$$\int_\Omega (w+p)\,\dot{T} d\Omega = -\int_\Omega (w+p)\,(u_i T_{,i})\,d\Omega \tag{2.92}$$

$$-\kappa \int_\Omega w_{,i} T_{,i} d\Omega + \int_{\Gamma_q} w T_{,j} n_j d\Gamma_q \tag{2.93}$$

where $\dot{T}$ is the time derivative of temperature, $T_{,i}$ is the gradient of temperature, w is the standard weighting function and $(w+p)$ is the Petrov-Galerkin weighting function with p, the discontinuous streamline upwind part of the Petrov-Galerkin weighting function, given by

$$p = \tau u \nabla T = \tilde{k} \frac{u_i w_{,i}}{||u||^2} \tag{2.94}$$

The energy equation is solved using Petrov-Galerkin weighting functions on the internal heat source and advective terms to correct for the under-diffusion and remove the oscillations which would result from the standard Galerkin method for an advection dominated problem (Hughes and Brooks, 1977). The Petrov-Galerkin function can be thought of as a standard Galerkin method in which we counterbalance the numerical underdiffusion by adding an artificial diffusivity of the form

$$(\xi u_\xi h_\xi + \eta u_\eta h_\eta)/2 \tag{2.95}$$

with

$$\xi = 1 - \frac{2\kappa}{u_\xi h_\xi} \tag{2.96}$$

$$\eta = 1 - \frac{2\kappa}{u_\eta h_\eta} \tag{2.97}$$

where h_ξ and h_η are the element lengths and u_ξ and u_η are the velocities in the local element coordinate system (ξ η system) evaluated at the element center. This form of discretization has no crosswind diffusion because the "artificial diffusion" acts only in the direction of the flow (i.e., it follows the streamline), hence the name Streamline Upwind Petrov-Galerkin (SUPG). This makes it a better approximation than straight upwinding and it has been demonstrated to be more accurate than Galerkin or straight upwinding in advection dominated problems (Hughes and Brooks, 1977). It has recently been shown that the SUPG method is one of a broader class of methods for advection-diffusion equations referred to as Galerkin/Least-Squares methods (Hughes et al., 1988).

The resulting matrix equation is not symmetric, but since the energy equation only has one degree of freedom per node, while the momentum equation has two or three, the storage for the energy equation is small compared to the momentum equation. Since we use an explicit time stepping method, the energy equation is not implemented in matrix form. The added cost of calculating the Petrov-Galerkin weighting functions is much less than the cost of using a refined grid with the Galerkin method. The Galerkin method

requires a finer grid then the Petrov-Galerkin method to achieve stable solutions (Travis et al., 1989). Time stepping in the energy equation is done using an explicit predictor-corrector algorithm. The form of the predictor-corrector algorithm is

Predict:

$$T_{n+1}^{(0)} = T_n + \Delta t\,(1-\alpha)\,\dot{T}_n \tag{2.98}$$

$$\dot{T}_{n=1}^{(0)} = 0 \tag{2.99}$$

Solve:

$$M^* \Delta \dot{T}_{n+1}^{(i)} = R_{n+1}^{(i)} \tag{2.100}$$

$$R_{n+1}^{(i)} = -\left[\dot{T}_{n+1}^{(i)} + u \cdot \left(T_{n+1}^{(i)}\right), x\right](w+p) - \tag{2.101}$$

$$\tilde{k} w_{,x}\left(T_{n+1}^{(i)}\right), x + \quad (boundary\,condition\,terms) \tag{2.102}$$

Correct:

$$T_{n+1}^{(i+1)} = T_{n+1}^{(i)} + \Delta t \alpha \dot{T}_{n+1}^{(i)} \tag{2.103}$$

$$\dot{T}_{n+1}^{(i+1)} = \dot{T}_{n+1}^{(i)} + \Delta \dot{T}_{n+1}^{(i)} \tag{2.104}$$

where i is the iteration number (for the corrector), n is the time step number, T is the temperature, $\dot{T}$ is the derivative of temperature with time, $\Delta\dot{T}$ is the correction to the temperature derivative for the iteration, M^* is the lumped mass matrix, $R_{n+1}^{(i)}$ is the residual term, Δt is the time step and α is a convergence parameter. Note that in the explicit formulation M^* is diagonal.

The time step is dynamically chosen, and corresponds to the Courant time step (the largest step that can be taken explicitly and maintain stability). With the appropriate choice of variables, $\alpha = 0.5$ and two iterations, the method is second order accurate (Hughes, 1986, p. 562-566).

The predict step is done in routine **timdrv**, the residual R is formed in routine **f_tres**, M^* is formed in routine **tmass**, and the correct step is also done in **f_tres**.

Chapter 3

Implementation

3.1 Introduction

There are generally two phases to ConMan, input and time stepping. The main program is found in **ConMan.F**. The input is read in the files **input.F and elminp.F**. Time stepping is doing in **timdrv.F**. For legacy reasons, there is a rather complex structure of **eglib** calling **eg2.F** which calls the assembly and solve routines.

There are three significant differences that the user who has seen versions of ConMan in the past will find in this version. The original version of ConMan, distributed by King and/or Hager, was designed to take advantage of machines with vector registers, such as the Cray X-MP or Y-MP. Hence thoughout the code, operations that would be performed on an individual element on a scalar machine were grouped together so that they could be performed on a group of elements.

With this version of ConMan, the reordering of elements into block with independent degrees of freedom and the reorganization of routines into loops over element groups with inner do-loops having lengths equal to the length of vector registers has been removed. This means that the structure of the routines that form the element stiffness matrix (**f_vstf.F**) and right-hand side, or residual, (**f_vres.F**) for the Stokes equation, and the routines for the calculation of the right-hand side of the energy eqation (**f_tmres.F**) are now loops over elements with short inner loops over local element nodes and or integration points. One could argue that because modern CPU's relying on fast cache to keep the arithmatic units busy, the kind of grouping we did to take advantage of vector registers is still useful for modern processors. However, the element reordering made algorithms such as particle tracking (not implemented in this version) more challenging and was difficult for users new to the finite element method to understand. Therefore, we have removed the element reordering and block vectorization.

The second major change is that we have implemented a Picard iteration algorithm for steady-state problems (c.f. van Keken - thesis). Currently, this is implemented by changing compiler flags in the Makefile. Picard iteration is an implicit solution to the energy equation (as opposed to the predictor-corrector method described above); hence, non-symmetric factor and back-solve routines have been added as well as a routine to calculate the energy equation matrix and right-hand side vectors (**f_trhsimp.F**). To use Picard iteration you change the variable α in the Time Sequence card (second line of the input file) from 0.5 to 1.0 and change the iterations from 2 to 1 (same line). You need to "make clean" and remake with the target picard, to generate a new executable (conman.pic). A future revision may merge these energy equation solvers into a single code and hide the need to change the iteration steps from the user. When you have a steady-state solution, Picard iteration usually converges in 10-100 steps, as opposed to several thousand steps. When there is no steady solution, the Picard results are not useful and the explicit solver should be used.

The third significant change is the replacement of the **mpoint** function which allocated memory in the blank common array 'a' with the call **mmgetblk**. The 'mm' routines are a fancy wrapper around a c *malloc* call. The reason for this change is fairly technical, and I will spare the users with the details. The memory manager source can be found in the directory mm.src. Care has to be taken when compiling on 32-bit or 64-bit machines (change the header file in mm.src from mm2000_32.h to mm2000_64.h) and change the compile flags in the Makefile. It is now possible to use Fortran90's allocate routine for dynamic memory

allocation and the memory manager should be removed in a future revision. Making changes in the memory manager is tricky and this package is fragile. Users who have a problem with the memory manager should consult CIG or Scott King.

3.2 Material Properties

As discussed above, the equations in dimensionless form have one dimensionless parameter, the Rayleigh number.

$$Ra = \frac{g\alpha\Delta T d^3}{\kappa\mu} \tag{3.1}$$

where g is the acceleration due to gravity, α is the coefficient of thermal expansion, ΔT is the temperature drop across the box, d is the depth of the box, κ is the thermal diffusivity, and μ is the dynamic viscosity. In ConMan, the input parameter is the buoyancy part of the Rayleigh number.

$$Ra_{buoy} = g\alpha \tag{3.2}$$

The depth, d, and the temperature difference, ΔT are specified from the grid and the temperature boundary conditions. κ and μ are separate input parameters. If the depth, temperature difference, κ and μ are set to 1, then the buoyancy number, RA_{buoy}, and the Rayleigh number, Ra, are the same.

The viscosity can be a function of temperature and/or depth. This is done in routine **rheol**. The user can easily modify the functional form for specific problems. The default functional form is

$$\mu\left(T,Z\right) = \mu_o \left\{ \exp\left\{ \frac{E^* * 1.0e3 + V^* z}{R * (T + T_o)} \right\} - \exp\left\{ \frac{E^* * 1.0e3 + V^* z}{R * (1 + T_o)} \right\} \right\} \tag{3.3}$$

where μ_o is the preexponential viscosity, E^* is the activation energy, V^* is the activation volume, T_o is the temperature offset, T is the temperature and z is the depth. In the input files, μ_o, is input on the viscosity card, E^*, is input as Tcon(1) V^* is input as Tcon(2), and T_o is hardwired in **rheol.newt.F** to be 273. and ΔT is hardwired to be 2000.0.

The scaling in **rheol.newt.F** is such that E^* can be input in kJ/mole and V^* can be input as cm^3.

Internal heating can be specified through the internal heating parameter. If no bottom temperature is specified the Rayleigh number becomes

$$Ra = \frac{g\alpha H d^5}{k\kappa\mu} \tag{3.4}$$

where H is the internal heating parameter and k is the thermal conductivity. The grid can have multiple material groups, each with its own set of material properties.

3.3 Installation

ConMan comes ready to run with a Unicos makefile. The file **Makefile** contains the system calls for the compiler and the loader, FC and LD respectively. These need to be changed for your machine. Also the calls to **second**, a Cray timing routine, will have to be changed in routine **timer**.

3.4 Building from Source [WEI – thought you might need this section. Or else delete]

3.4.1 System Requirements

ConMan works on a variety of computational platforms and has been tested on workstations running

- Mac OS X 10.4.6 (G4, G5, and Intel)

- Windows 2000 and XP SP2
- RedHat Fedora Core 5 (x86)
- OpenSuse 10.0 (x86)
- Gentoo (x86)
- Debian stable (x86 and AMD64), testing (x86), and unstable (x86)

ConMan has also been tested on clusters running Redhat 7.2 (x86) and RedHat Enterprise Linux 3 (EM64T).

3.4.2 Dependencies

This version of ConMan is self-contained and requires no external libraries.

3.4.3 Downloading the Code

You can get the source for the latest release from the ConMan web page (`geodynamics.org/cig/software/packages/mc/conman/`). In that tarball is the file ???name???.

3.4.3.1 Source Code Repository (Experts Only)

Advanced users and software developers may be interested in downloading the latest ConMan source code directly from the CIG source code repository, instead of using the prepared source package. To check whether you have a subversion client installed on your machine, type:

```
svn
```

You should get a response that looks something like this:

```
Type 'svn help' for usage.
```

Otherwise, you will need to download and install a Subversion client, available at the Subversion Website (`subversion.tigris.org/project_packages.html`). Then the code can be checked out with the following command:

```
svn checkout http://geodynamics.org/svn/cig/mc/3D/ConMan/trunk ConMan
```

3.5 Support

The primary point of support for ConMan is the CIG Mantle Convection Mailing List (`cig-mc@geodynamics.org`). Feel free to send questions, comments, feature requests, and bugs to the list. The mailing list is archived at

```
cig-mc Archives (geodynamics.org/pipermail/cig-mc/)
```

You may also use the bug tracker

```
Roundup (geodynamics.org/roundup)
```

to submit bugs and requests for new features.

Chapter 4

Input Guide

To run ConMan a series of nine file names are needed, some for input and some for output. Usually these are read from a runfile. The first two files are input files **input** and **geom** and are described in this section. The third file is an output file showing all the input parameters in a verbose form. The fourth and fifth files are an input temperature file (optional) and an output temperature file. These are for starting a new run from a previous run. The sixth file is a time series file (see routine **fluxke**), the seventh file is the coordinates, velocities and temperatures, the eighth file is for stresses (see routine **stress.F**) and the ninth file is for geoid and topography (see routine **geoid.F**). These file names are read in routine **ConMan.F**.

The input for ConMan is read from two different FORTRAN units. The first unit, **iin**, contains the time stepping, output, and material parameters as well as element type information while the second unit, **igeom**, contains the coordinates, boundary values and connectivity information. ConMan reads the file names to attach to these units from standard input. The typical way to run conman is to create a file with nine lines, one file name per line and redirect this into the executable (i.e., % conman.pic < runfile &). **iin** is attached to the file named on the first line and **igeom** is attached to the file named on the second line (names must be ASCII with a length less than 80 characters long).

The input deck was broken up so that an automatic grid generating routine could be used to generate coordinates, boundary conditions and element connectivities separate from ConMan. The only automatic grid generation conman does is linear or bilinear interpolation which is described in the appropriate sections of this guide.

The following sixteen cards or groups of cards are read from the **iin** unit (throughout this guide a "card" will mean one line of an ASCII text file). These constitute the parameter part of the input "deck" for the program ConMan. The format for this guide is a **bold** title line giving the card title followed by an *italicized* line showing the order of the parameters and a listing of the parameters (with a brief explanation).

Title Card *Any descriptive character string up to 80 characters long*

Global Constants Card *numnp nsd ndof nelx nelz mprec iflow necho inrsts iorstr nodebn ntimvs ntseq numeg isky nwrap*

numnp . . . . . total number of nodal points

nsd . . . . . number of spatial dimensions (always 2)

ndof . . . . . number of degrees of freedom (always 2)

nelx . . . . . number of elements in the x1 (horizontal) direction

nelz . . . . . number of elements in the x2 (vertical) direction

mprec . . . . . precision flag (always use double)

1 - single

2 - double

iflow . . . . . data check flag

0 - check data only

1 - execute code

necho . . . . . echo data flag
0 - minimum data echo (terse)
1 - echo data to output file (verbose)

inrstr . . . . . read restart file flag
0 - use default start (conductive)
1 - read restart file from unit 16

iorstr . . . . . write restart file flag
0 - don't write restart file
1 - write restart file to unit 17

nodebn . . . . . number of edge nodes for nusselt smoother

ntimvs . . . . . temperature dependent viscosity flag
0 - stiffness matrix factored once
1 - stiffness matrix factored every time step

ntseq . . . . . number of time sequences (always 1)
currently only one supported

numeg . . . . . number of element groups (always 1)
currently only one supported

isky . . . . . flag for skyline factor
0 - regular skyline
1 - vectorized skyline

nwrap . . . . . number of nodes to wrap
equal to number of elements in vertical
to use nodes must be numbered increasing
fastest in vertical direction

Time Sequence Cards - ntseq cards
nstep niter alpha delt epstol

nstep . . . . . number of time steps

niter . . . . . number of multicorrector iterations
2 - second-order expicit
1 - picard

alpha . . . . . multicorrector parameter
0.5 for explicit 2nd order
1.0 for picard

delt . . . . . time step (not used)

epstol . . . . . tolerance for hybrid method (not used)

Output Step Card *nsdprt nsvprt nstprt nsmprt*

nsdprt . . . . . steps between disk output

nsvprt . . . . . steps between velocity output (not used)

nstprt . . . . . steps between temperature, velocity & stress output

nsmprt . . . . . steps between stress field output (not used)

Velocity Boundary Condition Flag Cards *bnode enode incr (bcf(i), i=1,ndof)*

bnode . . . . . beginning node

enode . . . . . ending node

incr . . . . . node increment

bcf(i) . . . . . boundary condition flag for ith degree of freedom

0 - free slip

1 - pinned degree of freedom

0 0 0 0 0 to end VBCF cards

Temperature Boundary Condition Flag Cards *bnode enode incr bcf*

bnode . . . . . beginning node

enode . . . . . ending node

incr . . . . . node increment

bcf . . . . . boundary condition flag for temperature

1- fixed temperature

0 0 0 0 to end TBCF cards

Nusselt Number Boundary Condition Flag Cards - Edge Nodes top and bottom rows of nodes *bnode enode incr*

bnode . . . . . beginning node

enode . . . . . ending node

incr . . . . . node increment

0 0 0 to end NNBCF (type a) cards

Nusselt Number Boundary Condition Flag Cards - Second Row Nodes second from top and bottom rows of nodes *bnode enode incr*

bnode . . . . . beginning node

enode . . . . . ending node

incr . . . . . node increment

0 0 0 to end NNBCF (type b) cards

Initial Temperature Card *pert xsize zsize*

pert . . . . . perturbation from conductive state

xsize . . . . . nondimensional length (x1 direction) of box

zsize . . . . . nondimensional height (x2 direction) of box

Element Parameter Cards - numeg cards *ntype numel nen nenl numat nedof numsuf nipt implv implt*

ntype . . . . . element type

2 - two dimensional elements

numel . . . . . total number of elements

nen . . . . . number of element nodes (always 4)

nenl . . . . . number of local element nodes (always 4)

numat . . . . . number of material groups

nedof . . . . . number of element degrees of freedom (always 2)

numsuf . . . . . number of imposed stress/flux cards

nipt . . . . . number of integration points per element (always 5)

implv . . . . . currently unused

implt . . . . . currently unused

Viscosity Card *visc(i), i=1,numat*

visc(i) . . . . . preexponential viscosity coefficient for ith element

Penalty Card *alam(i), i=1,numat*

alam(i) . . . . . penalty parameter for ith element

Diffusivity Card *diff(i), i=1,numat*

diff(i) . . . . . thermal diffusivity for ith element

Buoyancy Rayleigh Number Card *Ra(i), i=1,numat*

Ra(i) . . . . . bouyancy part of Rayleigh number for ith element

Internal Heating Parameter Card *dmhu(i), i=1,numat*

dmhu(i) . . . . . internal heat source for ith material group

Activation Energy Card *tcon(1,i), i=1,numat*

tcon(1,i) . . . . . nondimensional activation energy for ith material group for temperature dependent viscosity

Temperature Dependent Viscosity Temperature Offset Card *tcon(2,i), i=1,numat*

tcon(2,i) . . . . . nondimen temperature offset for ith material group for temperature dependent viscosity

Surface Force/Flux Cards - numsuf cards *nel side fnorm ftan flux*

nel . . . . . element number

side . . . . . side to apply force and flux

1 - bottom

2 - right side

3 - top

4 - left side

fnorm . . . . . normal surface force

ftan . . . . . tangential surface force

flux . . . . . heat flux

The following four groups of cards are read from the **igeom** unit. These constitute the geometry part of the input "deck" for the program `conman`. The format of this section is the same as above.

4.1 Coordinate Group

Absolute Coordinate Card *node gp (x(i,node) i=1,nsd)*

node . . . . . the node whose coordinates are to be specified

gp . . . . . generation parameter for automatic generation
- 0 - no autogeneration
- 2 - generate a line using node as a starting point
- 4 - generate a box using node as the lower left corner

x(i,node) . . . . . coordinate value in the ith spatial dimension

Corner Generation Cards - gp-1 cards *node mgen (x(i,node) i=1,nsd)*

node . . . . . node number

mgen . . . . . generation parameter
- 0 - don't use this as the start of a generation sequence
- 1 - use this as the start of a generation sequence

x(i,node) . . . . . coordinate value in the ith spatial dimension

Generation Increment Card *ninc1 inc1 ninc2 inc2*

ninc1 . . . . . number of additional nodes to generate in x1 direction

inc1 . . . . . increment of nodes in x1 direction

ninc2 . . . . . number of additional nodes to generate in x2 direction
- 0 - if gp equals 2

inc2 . . . . . increment of nodes in x2 direction
- 0 - if gp equals 2

0 0 0 0 to end coordinate group

4.2 Velocity Boundary Condition Group

Absolute Velocity Card *node gp (v(i,node) i=1,nsd)*

node . . . . . the node whose velocities are to be specified

gp . . . . . generation parameter for automatic generation
- 0 - no autogeneration
- 2 - generate a line using node as a starting point
- 4 - generate a box using node as the lower left corner

v(i,node) . . . . . velocity value in the ith spatial dimension

Corner Generation Cards - gp-1 cards *node mgen (v(i,node) i=1,nsd)*

node . . . . . node number

mgen . . . . . generation parameter
- 0 - don't use this as the start of a generation sequence
- 1 - use this as the start of a generation sequence

v(i,node) . . . . . velocity value in the ith spatial dimension

Generation Increment Card *ninc1 inc1 ninc2 inc2*

ninc1 . . . . . number of additional nodes to generate in x1 direction

inc1 . . . . . increment of nodes in x1 direction

ninc2 . . . . . number of additional nodes to generate in x2 direction

0 - if gp equals 2

inc2 . . . . . increment of nodes in x2 direction

0 - if gp equals 2

0 0 0 0 to end velocity group

4.3 Temperature Boundary Condition Group

Absolute Temperature Card *node gp t(node)*

node . . . . . the node whose velocities are to be specified

gp . . . . . generation parameter for automatic generation

0 - no autogeneration

2 - generate a line using node as a starting point

t(node) . . . . . temperature value

Corner Generation Cards - gp-1 cards *node mgen t(node)*

node . . . . . node number

mgen . . . . . generation parameter

0 - don't use this as the start of a generation sequence

1 - use this as the start of a generation sequence

t(node) . . . . . temperature value

Generation Increment Card *ninc1 inc1 ninc2 inc2*

ninc1 . . . . . number of additional nodes to generate in x1 direction

inc1 . . . . . increment of nodes in x1 direction

ninc2 . . . . . number of additional nodes to generate in x2 direction

0 - if gp equals 2

inc2 . . . . . increment of nodes in x2 direction

0 - if gp equals 2

0 0 to end temperature group

4.4 Element Connectivity (ien) Generation Group

Absolution Element Card *elnu ng mat no (ien(elnu,i) i=1,nen)*

elnu . . . . . element number

ng . . . . . generation parameter

0 - no generation

1 - generate using increments from increment card

mat no . . . . . material number for this element

ien(elnu,i) . . . . . global node number for the ith local node of element counterclockwise from lower left corner

Increment Card *nel1 incel1 incn1 nel2 incel2 incn2*

nel1 . . . . . number of elements in x1 (horizontal) direction
incel1 . . . . . increment of elements in x1 (horizontal) direction
incn1 . . . . . increment of nodes in x1 (horizontal) direction
nel2 . . . . . number of elements in x2 (vertical) direction
incel2 . . . . . increment of elements in x2 (vertical) direction
incn2 . . . . . increment of nodes in x2 (vertical) direction

0 0 0 0 0 0 to end element connectivity group

Chapter 5

Sample Input Files

The lines below are a sample 50 element by 50 element input deck for a 1 by 1 square, constant viscosity with the Picard method. This is Blankenbach 1a

```
50 x 50 el. plate problem from Blankenbach et al. 1989 #Nds sdm dof X Z prc ck echo rrst wrst nus t
```

The lines below are a sample geometry file for the 50 by 50 element problem.

```
coordinates 1 4 0.0 0.0 2551 1 1.0 0.0 2601 1 1.0 1.0 51 1 0.0 1.0 50 51 50 1 0 0 0.0 0.0 velocity
```

Chapter 6

Output Guide

6.1 The Output Files

Chapter 7

The Benchmark Cases

7.1 Constant Viscosity Benchmark for ConMan

Here we reproduce the results from Blankenbach et al. (1989) for constant viscosity in a unit-aspect ratio domain, with free-slip boundary conditions, heated from below and cooled from above. The user should note that to run this problem, you will need to modify the routines **rheol.newt.F** and **geoid.F** following the comments in those routines. The constant viscosity calculations (1a, 1b, and 1c) use a Rayleigh number of 10^4, 10^5, and 10^6 respectively and the dimensional parameters are listed in Table 7.1. The time is the run time in seconds for the Picard version of the code using 100 iterations on an intel MacPro with two 3 GHz processors using the intel fortran compiler with -O2 optimization. While the problems in Blankenbach et al. are specified dimensionally, the equations are solved non-dimensionally within ConMan for numerical stability and the scaling factors are applied to the results.

The results are computed on uniformly spaced grids and the global properties of Nusselt number and root-mean-square velocity, as well as the topography and geoid at the left and right hand side of the domain are reported, along with the extrapolated value from Christensen's results (see Blankenbach et al., 1989 for discussion) in Table 7.2 (Rayleigh number 10^4), Table 7.3 (Rayleigh number 10^5), and Table 7.4 (Rayleigh number 10^6). For the globally properties of Nusselt number and root-mean-square velocity, the 50 by 50 element grid are within 1% of Christensen's extrapolated results even for the Rayleigh number 10^6 calculations. The agreement for the point values of topography and geoid are also 1% error on the 50 by 50 grid for the topography and geoid for the 50 by 50 element grid for the Rayleigh number 10^6 calculations (Table 7.4). For the 200 by 200 grid, all values converge to Christensen's extrapolated results.

Parameter	Symbol	Value
depth of domain	d	10^6 m
gravitational acceleration	g	10 m/s^2
temperature difference	ΔT	1000 K
density	ρ	4000 kg m^{-3}
thermal diffusivity	κ	1.0×10^{-6} m^2 s^{-1}
coefficient of thermal expansion	α	2.5×10^{-5}
kinematic viscosity	η	2.5×10^{19} Pa s (1a)
		2.5×10^{18} Pa s (1b)
		2.5×10^{17} Pa s (1c)
gravitational constant	G	6.673×10^{-11}

Table 7.1: Mantle parameters for Blankenbach constant viscosity benchmarks.

Grid	V_{rms}	Nusselt No.	$Topo_L$	$Topo_R$	$Geoid_L$	$Geoid_R$	Run Time (sec)
50	42.906	4.887	2261.956	-2911.473	55.346	-63.178	1.98
100	42.875	4.885	2256.094	-2905.356	54.957	-62.765	18.04
200	42.867	4.885	2254.541	-2903.763	54.856	-62.658	187.43
†C_{ext}	42.865	4.884	2254.021	-2903.221	54.822	-62.622	
†Christensen′s extrapolated values.							

Table 7.2: Blankenbach (1989) Benchmark 1a: Steady State, 2D, constant viscosity convection in a 1 by 1 box with Rayleigh number 10^4 using ConMan

Grid	V_{rms}	Nusselt No.	$Topo_L$	$Topo_R$	$Geoid_L$	$Geoid_R$	Run Time (sec)
50	193.592	10.546	1482.778	-2014.228	28.846	-33.104	1.82
100	193.297	10.539	1467.169	-2008.138	28.034	-32.327	17.66
200	193.248	10.536	1462.487	-2005.473	27.789	-32.099	185.22
†C_{ext}	193.214	10.534	1460.986	-2004.205	27.703	-32.016	
†Christensen′s extrapolated values.							

Table 7.3: Blankenbach (1989) Benchmark 1b: Steady State, 2D, constant viscosity convection in a 1 by 1 box with Rayleigh number 10^5 using ConMan

Grid	V_{rms}	Nusselt No.	$Topo_L$	$Topo_R$	$Geoid_L$	$Geoid_R$	Run Time (sec)
50	840.524	21.864	941.607	-1301.980	14.958	-16.678	1.71
100	835.606	22.023	945.108	-1290.926	14.109	-15.632	17.38
200	834.353	21.981	936.439	-1285.756	13.654	-15.204	184.27
†C_{ext}	833.989	21.997	931.962	-1283.813	13.452	-15.034	
†Christensen′s extrapolated values.							

Table 7.4: Blankenbach (1989) Benchmark 1c: Steady State, 2D, constant viscosity convection in a 1 by 1 box with Rayleigh number 10^6 using ConMan.

Grid	V_{rms}	Nusselt No.	$Topo_L$	$Topo_R$	$Geoid_L$	$Geoid_R$	Run Time (sec)
50	488.950	10.080	1041.464	-4012.790	18.584	-55.084	6.17
100	482.583	10.070	1017.502	-4081.259	17.657	-54.790	60.85
200	480.879	10.067	1012.217	-4094.520	17.417	-54.654	739.20
400	480.493	10.066	1011.109	-4097.093	17.360	-54.610	10826.89
†C_{ext}	480.433	10.066	1010.925	-4098.073	17.343	-54.598	
†Christensen′s extrapolated values.							

Table 7.5: Blankenbach (1989) Benchmark 2a: Steady State, 2D, temperature-dependent viscosity convection (b=6.907755279) in a 1 by 1 box with Rayleigh number 10^4 using ConMan.

7.2 Temperature-Dependent Viscosity Benchmark for ConMan

Here we reproduce the results from Blankenbach et al. (1989) for temperature-dependent viscosity in a unit-aspect ratio domain, with free-slip boundary conditions, heated from below and cooled from above (case 2a). For this problem, the temperature-dependence of viscosity is given by

$$\eta(T) = \eta_o \exp\left[-\ln\{1000\}\frac{T}{\Delta T}\right] \tag{7.1}$$

where $\eta_o = 2.9 \times 10^{19}$ and $\Delta T = 1000.0$. The other scaling parameters are the same as Table 7.1. The results for a Rayleigh number of 10^4 are presented in Table 7.5. Here once again the global properties of Nusselt number and root-mean-square velocity for the 50 by 50 grid are within 1-2% of Christensen's extrapolated results; however, in contrast to the constant viscosity cases, the values of topography and geoid in the corners differ from Christensen's extrapolated results by as much as 3% for topography and 7% for geoid on the 50 element by 50 element grid. Again, by the 400x400 grid, the values are well within 0.5%.

Appendix A

License

GNU GENERAL PUBLIC LICENSE Version 2, June 1991. Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. Copyright the software, and

2. Offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program" below refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification.") Each licensee is addressed as "you."

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version," you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found. For example:

> One line to give the program's name and a brief idea of what it does. Copyright © (year) (name of author)
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright © year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items – whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> (signature of Ty Coon)
> 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliography

[1] Brooks, A., 1981. A Petrov-Galerkin finite-element formulation for convection dominated flows. Ph.D. Thesis, California Institute of Technology, Pasadena, CA.

[2] Brooks, A.N. and Hughes, T.J.R., 1982. Streamline upwind/Petrov-Galerkin formulatins for convection dominated flows with particular emphasis on the inpompressible Navier-Stokes equations. *Comp. Meth. in Appl. Mech. and Eng., 32,* 199-259.

[3] Hughes, T.J.R., 1987. *The finite element method.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey

[4] Hughes, T.J.R. and Brookes, A.N., 1987. A multi-dimensional upwind scheme with no crosswind diffusion. In: Finite element methods for convection dominated flows. *ASME*, New York, 34:19-35.

[5] Hughes, T.J.R., Franca, L.P., Hulbert, G.M., Johan, Z., and Shakib, F., 1988. The Galerkin/least-squares method for advective-diffusive equations. In: Recent developments in computational fluid dynamics. T.E. Tezduyar (Editor), *ASME,* New York, 95:75-99.

[6] Hughes, T.J.R., Liu, W.K., and Brooks, A.N., 1979. Finite element analysis of incompressible viscous flows by the penalty function formulation. *J. Comput. Phys.,* 30: 19-35.

[7] Malkus, D.S. and Hughes, T.J.R., 1978. Mixed finite element methods reduced and selective integration techniques: a unification of concepts. *Comp. Meth. in Appl. Mech. and Eng.,* 15, 63-81.

[8] Temam, R., 1977. Navier-Stokes equations: therory and numerical analysis. North-Holland. Amsterdam.

[9] Travis, B.J., C. Anderson, J. Baumgardner, C. Gable, B.H. Hager, P. Olson, R.J. O'Connell, A. Raefsky, and G. Schubert, 1991. A benchmark comparison of numerical methods for infinite Prandtl number convection in two-dimensional Cartesian geometry, *Geophys. Astrophys. Fluid Dynamics,* in press.