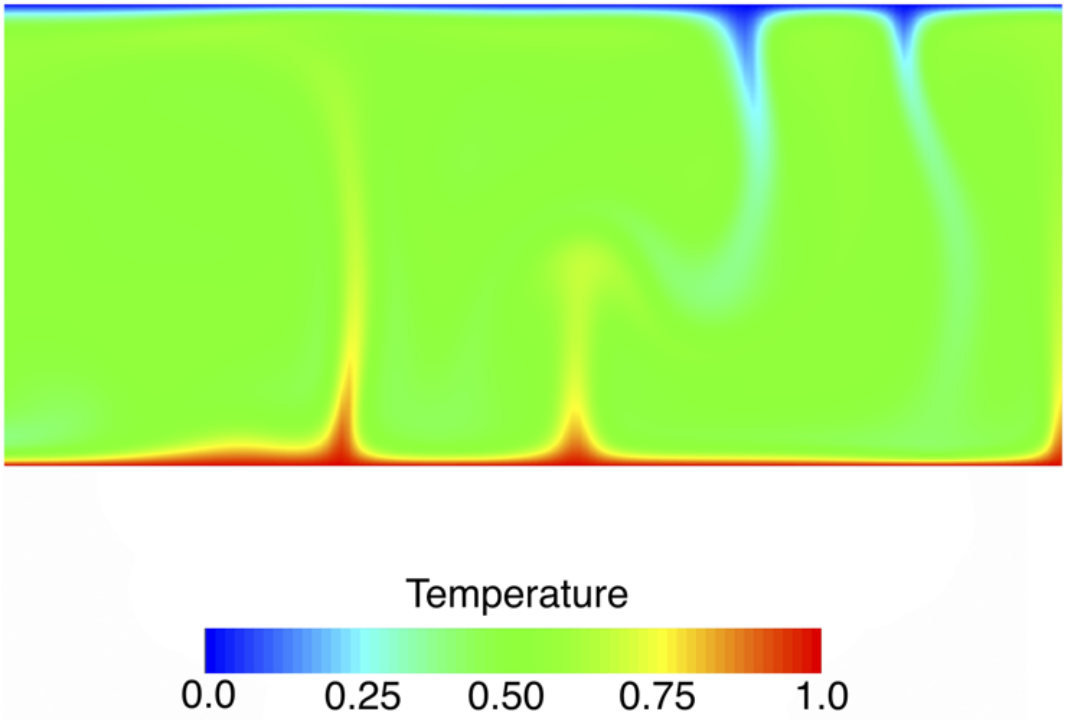
COMPUTATIONAL INFRASTRUCTURE FOR GEODYNAMICS (CIG)

ConMan

User Manual

Version 3.0



www.geodynamics.org

Scott King

ConMan

Version 3.0

Scott King
John Naliboff ???
Harsha Lokavarapu ???

May 15, 2020

Contents

List of Figures

Chapter 1

Preface

1.1 Abstract

This manual serves as a user guide for ConMan, a finite element program for the solution of the equations of compressible and incompressible, infinite-Prandtl number convection in two dimensions originally written by Arthur Raefsky, Scott King, and Brad Hager [King et al., 1990]. Changyeol Lee contributed significantly to the compressible formulation in this distribution [Lee and King, 2009]. The 3.0 version of ConMan solves the Bousinessq (BA), Extended Bousinessq (EBA), Trucated Anelastic Liquid (TALA), and Anelastic Liquid (ALA) approximations [King et al., 2010].

ConMan has always been a public domain program and is maintained and distributed by the Computational Infrastructure for Geodynamics (CIG) geodynamics.org and is made available under the GNU General Pubic License either version 2 or later.

ConMan is written in FORTRAN making use of FORTRAN's memory allocation and has been tested on a variety of linux systems with gfortran and intel fortan compilers. It has been benchmarked against other existing codes (see Chapter 9). Yet as with anything free, it comes with no guarantees. The authors would appreciate any information regarding bugs or potential problems but make no promises regarding the timeliness of changes or fixes; see Section 5.2 for instructions on how to report problems.

1.2 Changes Since Version 2.0

There are several significant differences that the user familiar with past versions of ConMan will find in the 3.0 version. First, we removed the clunky memory manager library (a set of routines wrapped around the c function malloc) and replaced them with FORTRAN 90's allocate and deallocate functions. This eliminates many of the compilation problems people experienced with the 2.0 version. Most of these routines were in files subroutines input and elminp.

As part of a general clean up, we replaced the separate input and elminp (both input subroutines) and created a new input subroutine. As part of this we removed the 'element library' function (eglib.F and eg2.F) which was a structure originally designed for different formulations 2D Cartesian, 3D Cartesian, spherical axysymmetric, ...). Because these were never fully developed, it made on sense to retain the cumbersome structure. We also moved all out the subroutines associated with output, into new output subroutine. The user does not need to hunt through the time_driver subroutine to find out where the specific output subroutines are called. Thus, subroutines geoid, fluxke, masflx, print, output_rheol, print_compbm_data, and stress are all in subroutine output.

We also changed the names of many of the subroutines to take advantage of longer subroutine names allowed by modern FORTRAN. Thus f_tlhs has become form_temp_matrix, f_vres has become form_velocity_rhs, f_vstf has become form_velocity_stiffness_matrix. Similarly, subroutine timdrv has become time_driver. As you look through the code there are examples where this could have been carried further.

Second, Picard iteration for the temperature equation is now a runtime option as opposed to a compiler option. This necessitated specifying both implicit and explicit subroutines for the temperature right hand

side, form_temp_rhs_implicit and form_temp_rhs_explicit As well as a form_temp_matrix.F for the implicit temperature matrix and form_temp_mass_matrix for the lumped mass matrix that has traditionally been used for the explicit version of the temperature solver.

Third, we added the EBA, TALA, and ALA formulations as described in King et al. [2010]. The compressible formulation is described in Chapter 3. This required a number of changes throughout the form_temperature and form_velocity subroutines. We provide a test suite that runs a subset of the problems from King et al. [2010] that can be used to verify the installation version 3.0.

Finally, we have added a cookbook of subduction wedge problems based on problems from the subduction zone benchmark paper [van Keken et al., 2008] and one based on the compressible convection benchmark paper [King et al., 2010]. This required adding a new 'fault' subroutine and a subroutine to implement the Batchelor corner flow boundary conditions. These can be found in the subduct.src directory.

1.3 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 Hughes [1987]. 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.

1.4 Contributors

ConMan was originally developed by Scott King, Arthur Raefsky, and Brad Hager [King and Hager, 1990]. The grid generation routines were adapted from DLEARN, a code distributed with Hughes [1987]. Numerous people have contributed to ConMan and related codes over the past 25 years, including: Louise H. Kellogg and Walter Kiefer (SCAM - Spherical Convection in an Axisymmetric Mantle), Cinzia G. Farnetani (cylindical version for plumes), Junan Chen, Steve S. Shapiro (marker chain and field methods), Mark Simons (geoid calculation), Peter Puster (cylindical version), Don E. Koglin (rheology and plates), Hannah L. Redmond (SCAM - Spherical Convection in an Axisymmetric Mantle), Peter van Keken (Picard iteration and benchmarking), Changyeol Lee and Ikuko Wada (compressible formulations and subduction problems). This distribution only includes the Cartesian version and does not include markers, marker chains or compositional fields. These could easily be added and I have the older routines if someone it interested in merging those features with the current code.

1.5 Citation

The ConMan team requests that in your oral presentations and in your papers that you indicate your use of this code by citing these two papers which describe the methods the code is based on:

- King, S.D., A. Raefsky, and B.H. Hager (1990), ConMan: Vectorizing a finite element code for incompressible two-dimensional convection in the Earth's mantle, *Phys. Earth Planet. Int., 59,* 195-208.
- King S. D., C. Lee, P. E. van Keken, W. Leng, S. Zhong, E. Tan, E., M. Gurnis, N. Tosi, and M. C. Kameyama (2010) A community benchmark for 2D Cartesian compressible convection in the Earth's mantle, *Geophys. J. Int., 180,* 73-87, 2010. doi:10.1111/j.1365-246X.2009.04413.x
- King, S. D., A. Raefsky, and B.H. Hager (2020), ConMan v3.0.0 [software], doi:10.5281/zenodo.3633152.

1.6 Support

ConMan is freely available from the Computational Infrastructure for Geodynamics (CIG) (`geodynamics.org`) under the GPL 2.0 or later license (Appendix A) in the hope that the software will enhance your research in geophysics. Maintenance is supported by a grant from the National Science Foundation to CIG, managed by the University of California at Davis.

Please acknowledge CIG as follows:

- ConMan is hosted by the Computational Infrastructure for Geodynamics (CIG) which is supported by the National Science Foundation under awards EAR-0949446 and EAR-1550901.

ConMan code was donated to CIG in June 2008.

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

2.1 The Finite Element Method

This section closely follows Hughes [1987, Chapter 1, sections 1-4]. There are two ways we can write the equation, the strong and the weak form. More readers are probably more familiar with the strong form, and less familiar with 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 in elasticity. For viscous flow, there is also a variational form, but we will not discuss that here.

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

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

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 6.

2.1.3 Galerkin's Approximation

Now we have a start on the finite element method. We continue to follow Hughes; however, his notation becomes quite difficult to keep up with. Now, let's 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 discrete 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. It is useful 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 about Fourier series. Any 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, we narrow the focus to deal with specifically the elements in ConMan. It is possible to think very general shape functions but, in practice people use triangles or quadralaterals (in 2D). In terms of the level of approximation, there are also a lot of possibilities. We will stick to the simplest form, bilinear elements, but you should be aware that higher-order elements (biquadratic or bicubic spline elements) are also popular with some people. This section is condensing a lot of very useful material from Chapter 3 of Hughes' book into a short overview. 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.

Let's 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. A visual representation of these shape functions is shown in Figure2.1.

In ConMan we further choose to normalize this by setting $a = 0.5$ and $b = 0.5$. This choice gives us an element whose area is 1, a convenient way to scale elements.

Notice that it is easy to take the derivatives of these shape functions. Below we write the x and y derivatives of the shape functions,

$$N_{1,x} = \frac{-(0.5-y)}{1} \tag{2.30}$$

$$N_{2,x} = \frac{(0.5-y)}{1} \tag{2.31}$$

$$N_{3,x} = \frac{(0.5+y)}{1} \tag{2.32}$$

$$N_{4,x} = \frac{-(0.5+y)}{1} \tag{2.33}$$

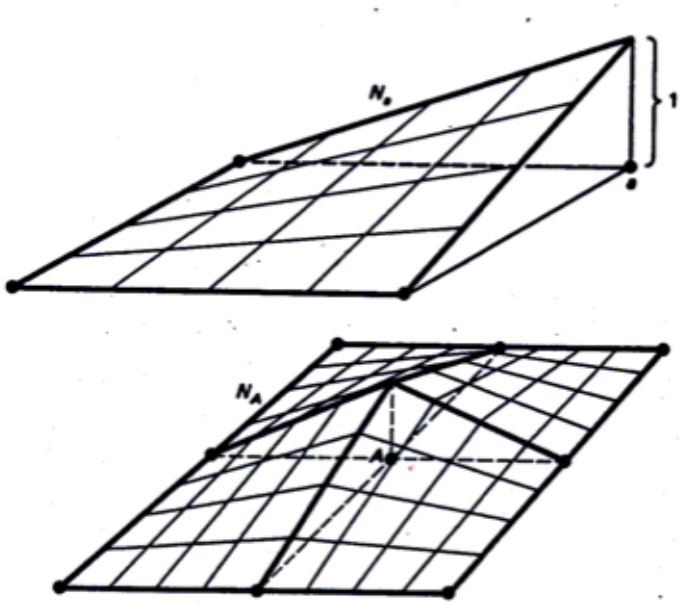


Figure 2.1: The bilinear shape function for a single element (top) and the four elements whose shape functions combine to form the global shape function for node A (bottom). Figure taken from Hughes, Section 3.2.

$$N_{1,y} = \frac{-(0.5-x)}{1} \tag{2.34}$$

$$N_{2,y} = \frac{-(0.5+x)}{1} \tag{2.35}$$

$$N_{3,y} = \frac{(0.5+x)}{1} \tag{2.36}$$

$$N_{4,y} = \frac{(0.5-x)}{1} \tag{2.37}$$

. 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 the shape functions above (see Figure 2.2).

However, if we are thinking about a cylindrical geometry, for example, we can use the Jacobian of the transformation between the geometries. Let's 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

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

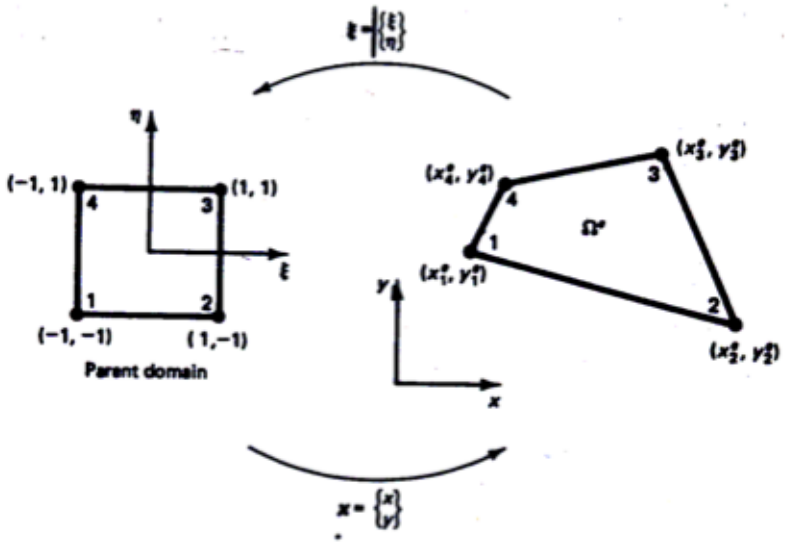


Figure 2.2: The mapping between the global domain (right) and the parent element domain (left) using the shape functions. Figure taken from Hughes, Section 3.2.

If the transformation was from an arbitrarily-shaped quadrilateral to the parent element, then the off diagonal terms in the matrix in equation 2.41 will not be zero. It is easy enough to show that

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

where $[J]$ is the Jacobian of the transformation. 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).

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

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

and putting these together, we find

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

2.1.4.1 Gauss Quadrature

Gauss Quadrature is a way to turn an integral into a summation. Let's begin with several 1D examples. The easiest case is that where our function is a constant on the interval -1, to 1, $f(x) = c$

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

. Any constant c integrated from -1 to 1 gives a value of $2c$. Note that if you take two times the value of the function evaluated at zero, you also get $2c$,

$$2.0 \times f(0) = 2c. \tag{2.48}$$

Gauss went on to show that for any higher-order function, the best value you could get with one point was to take the value of the function at the mid-point and multiply by a weighting function of 2.

Gauss showed that for any linear function, if one evaluated the function at two points and summed the result, and one wanted the sum of those two points to give you the best possible approximation to the integral over the range -1 to 1, those two points would be $\frac{-1}{\sqrt{3}}$ and $\frac{1}{\sqrt{3}}$. Consider an arbitrary linear function, $f(x) = ax + b$. Direct integration gives,

$$\int_{-1}^{1} (ax + b)\, dx = (\frac{ax^2}{2} + bx)|_{-1}^{1} = \frac{a}{2} + b - (\frac{a}{2} - b) = 2b. \tag{2.49}$$

Now let's evaluate the function at the two points, $\frac{-1}{\sqrt{3}}$ and $\frac{1}{\sqrt{3}}$, and sum the results,

$$f(\frac{-1}{\sqrt{3}}) + f(\frac{-1}{\sqrt{3}}) = a\frac{-1}{\sqrt{3}} + b + a\frac{1}{\sqrt{3}} + b = 2b, \tag{2.50}$$

This shows that $\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$ will always give you an exact result for a linear equation but, what Gauss showed was more powerful, that for any higher-order function, choosing the points $\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$ will give you the best approximation possible with only two function evaluations. If we go to three terms, it turns out that the best choice for the points to evaluate the function are $-\sqrt{\frac{3}{5}}, 0, \sqrt{\frac{3}{5}}$.

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

$$\xi = \frac{-1}{\sqrt{3}} \qquad \eta = \frac{-1}{\sqrt{3}} \tag{2.51}$$

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

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

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

are sufficient to exactly integrate our bilinear shape functions over the -1,-1 to 1,1 domain. Note this is not an approximation, because the function is linear, the integral is exact. (If we used higher-order elements we would need more function evaluations, hence the number of operations needed to calculate the integral would increase.)

The shape functions are generated in ConMan in the subroutine **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. Because we only evaluate the shape functions at the Gauss quadrature points and because the parent element is on the domain -0.5 to 0.5 in x and in y, we can precompute these values. The second part of this subroutine calculates the length of the sides of the elements in the physical domain (i.e., not the parent element domain). This is used in the SUPG elements (i.e., upwinding) described in a later section.

The subroutine **genshg** calculates the global shape functions. This uses the mapping and Jacobian of the transformation described above. We call this routine once and store all the shape functions because the grid remains fixed through out the computation, hence the loop over *numel*. This requires more memory to store the shape functions and their derivatives but, it reduces the number of computations needed. Again, because the shape functions and the derivatives of the shape functions are only needed at the Gauss points, we only store five values of these quantities, the four Gauss points listed in Equations 2.51–2.54, and a fifth Gauss point at (0,0) which is used when we use 'reduced intergation' for the pressure. It will become necessary to continuously update the shape functions if one wishes to modify ConMan 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 2.2). All calculations are done in the parent element domain and the results are assembled into the global equations. This means all calculations can be pre-computed for a single parent element **genshp**. Elements of different sizes or shapes filling an irregular global domain

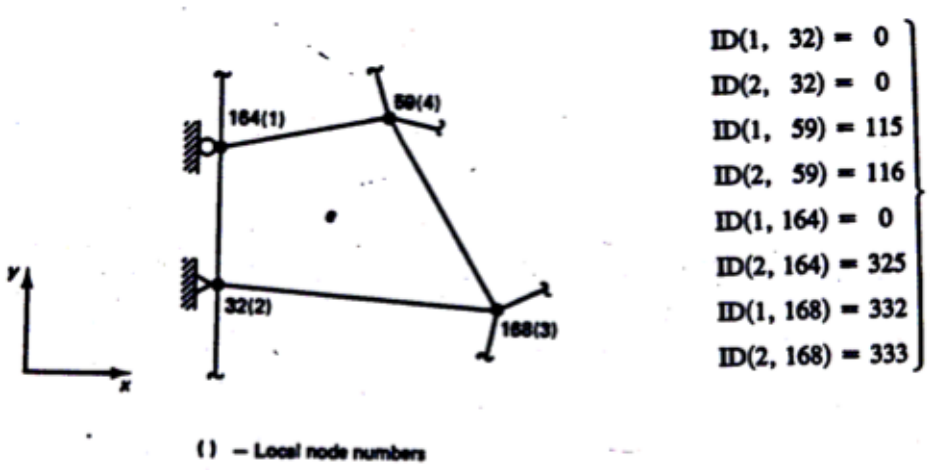


Figure 2.3: An example illustrating the relationship between global nodes and equation numbers for a 2 degree of freedom problem using the id array. An equation number of zero denotes a boundary condition. Figure taken from Hughes, Section 3.2.

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, calculated in subroutine **genshg**.

For ConMan the choice was made to use bilinear quadrilaterals as the parent elements (Figure 2.1). Higher-order elements (i.e., biquadratic or bicubic-spline) require more computational work per element. There is more computations necessary to evaluate the higher-order function itself and more integration points are needed to calculate the integral exactly using Gauss quadrature. It has been our experience that using a grid with more linear elements, rather than using high-order elements, is the best strategy for an efficient, accurate code for incompressible, advection-diffusion problems. Other codes have made different choices.

Because in finite element routines it is natural to loop over the elements, whereas for matrices or graphical output, it is useful to loop over the nodes, there is a need to map back and forth between the node numbering and element numbering. To do this we define several bookkeeping arrays to identify nodes and elements in each of the domains. In ConMan, those arrays are called:

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.

It is worth noting that the numbering of local elements always begins with 1 in the lower left-hand corner. There is no special reason; you just have to choose a convention.

With these mapping arrays, we are able to switch back and forth between looping over elements and looping over nodes. 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 because there may be more than one degree of freedom per node. Boundary conditions are specified with a zero equation number. Because the matrix is sparse, 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. Within ConMan, 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 its value is the global node number. It is easiest to look at an example.

Consider a 2-element by 2-element grid 2.4 with elements numbers consecutively in the horizontal direction and nodes numbered consecutively in the vertical direction, starting with the lower left hand corner. For each element, the local nodes are numbered starting with the lower left-hand corner. The ien array for this small grid is given in Table 2.1.

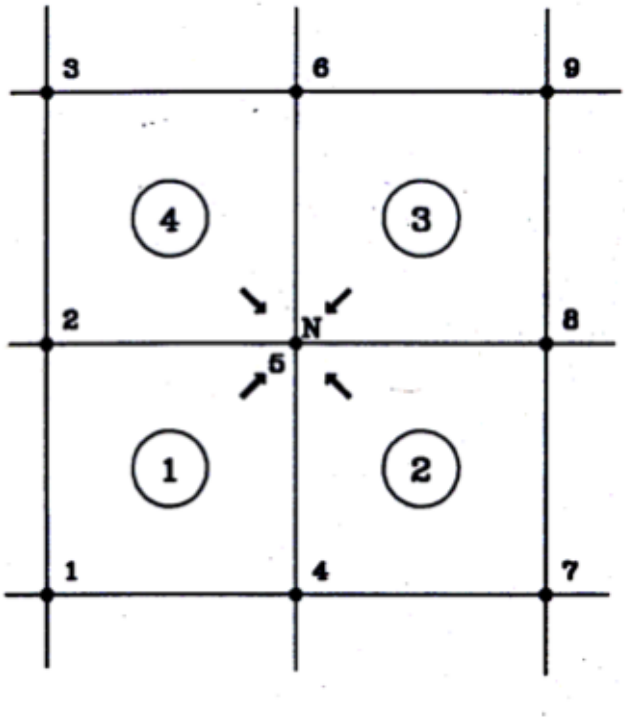


Figure 2.4: An example illustrating the relationship between global node numbers and local element numbers using the ien array. Local nodes are numbered counterclockwise from the bottom left-hand corner.

element	local node 1	local node 2	local node 3	local node 4
element 1:	ien (1, 1) = 1	ien (1, 2) = 4	ien (1, 3) = 5	ien (1, 4) = 2
element 2:	ien (2, 1) = 4	ien (2, 2) = 7	ien (2, 3) = 8	ien (2, 4) = 5
element 3:	ien (3, 1) = 5	ien (3, 2) = 8	ien (3, 3) = 9	ien (3, 4) = 6
element 4:	ien (4, 1) = 2	ien (4, 2) = 5	ien (4, 3) = 6	ien (4, 4) = 2

Table 2.1: ien array for the 2 element by 2 element grid shown in 2.4.

Nodes and elements can be numbered with either the horizontal or vertical direction increasing fastest. This is controlled by the users choice of increment in the x and y directions for the nodes and the ien array

in the **geom** file. For problems where flow out the left-hand side is matched by flow in the right-hand side (i.e., wrap around boundary conditions), it is necessary to increment the nodes fastest in the vertical (y) direction. Not only does this reduce the bandwidth of the matrix, the logic of the wrap-around boundary condition assumes this ordering and will likely fail if you increment the nodes fastest in the horizontal (x) direction.

2.1.6 Bousinessq Equations

ConMan was originally written to solve the equations of creeping thermal convection using the Bousinessq approximation. In this version we have adapted it to include the Extended Bousinessq (EBA), Truncated Anelastic Liquid (TALA), and Anelastic Liquid (ALA) approximations. For brevity, we present the description of the finite element method applied to the Bousinessq approximation.

$$\tau_{ij,j} + f_i = 0 \tag{2.55}$$
$$u_{i,i} = 0 \tag{2.56}$$

where

$$\tau_{ij} = -p\delta_{ij} + 2\mu u_{(i,j)} \tag{2.57}$$

where

$$u_{(i,j)} = (u_{i,j} + u_{j,i})/2 \tag{2.58}$$

We replace Equation 2.57 with the following relationships

$$\tau_{ij} = -p^\lambda \delta_{ij} + 2\mu u_{(i,j)} \tag{2.59}$$
$$0 = u_{i,i} + p^\lambda/\lambda. \tag{2.60}$$

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 2.60 into 2.59 we get

$$\tau_{ij} = \lambda u_{i,i}\delta_{ij} + 2\mu u_{(i,j)} \tag{2.61}$$

or

$$\tau_{ii} = 3\lambda u_{i,i} + 2\mu u_{i,i} \tag{2.62}$$

or

$$\tau_{ii}/3 = -p = (\lambda + 2/3\mu)u_{i,i} \tag{2.63}$$

but we also have

$$-p^\lambda = \lambda u_{i,i} \tag{2.64}$$

from Equation 2.60. 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 2.61 into Equation 2.55 we have

$$\{\lambda u_{i,i}\delta_{ij} + 2\mu u_{(i,j)}\}, j + f_i = 0 \tag{2.65}$$

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

$$\frac{\partial}{\partial x}\{\lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) + 2\mu(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial x})/2\} + \frac{\partial}{\partial y}\{2\mu(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y})/2\} + f_x = 0 \tag{2.66}$$
$$\frac{\partial}{\partial x}\{2\mu(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x})/2\} + \frac{\partial}{\partial y}\{\lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) + 2\mu(\frac{\partial v}{\partial y} + \frac{\partial v}{\partial y})/2\} + f_y = 0 \tag{2.67}$$

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

$$\lambda(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial y}) + 2\mu\frac{\partial^2 u}{\partial x^2} + \mu(\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 v}{\partial y \partial x}) + f_x = 0 \tag{2.68}$$

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

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 carefully by Hughes in *The Finite Element Method* on pages 197-200, but he has left out a number of intermediate steps. Nothing about this step is hard, just tedious. There is, however, a clever shortcut. 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. To see this for yourself, simply take the equations directly above this paragraph, multiply by a weighting function w and integrate over the 2D domain Ω, then use integration by parts. You will find (after a little algebra)

$$\int\int_\Omega \frac{\partial w}{\partial x}\{\lambda(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}) + 2\mu\frac{\partial u}{\partial x}\} + \frac{\partial w}{\partial y}\{2\mu(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y})/2\}\, d\Omega + \int\int_\Omega f_x\, w\, d\Omega = b.c.\ terms \tag{2.70}$$

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

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

$$\int_a^b w\, dv = w\, v|_a^b - \int_a^b v\, dw \tag{2.72}$$

where 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

$$\frac{\partial w}{\partial x} = N_x \tag{2.73}$$

$$\frac{\partial w}{\partial y} = N_y \tag{2.74}$$

$$\frac{\partial u}{\partial x} = u\, N_x \tag{2.75}$$

$$\frac{\partial u}{\partial y} = u\, N_y \tag{2.76}$$

$$\frac{\partial v}{\partial x} = v\, N_x \tag{2.77}$$

$$\frac{\partial v}{\partial y} = v\, N_y \tag{2.78}$$

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

$$\int\int_\Omega N_x\{\lambda(u\,N_x + v\,N_y) + 2\mu u\,N_x\} + N_y\{\mu(v\,N_x + u\,N_y)\}\,d\Omega + \int\int_\Omega f_x\,w\,d\Omega = b.c.\ terms \tag{2.79}$$

$$\int\int_\Omega N_x\{\mu(u\,N_y + v\,N_x)\} + N_y\{\lambda(u\,N_x + v\,N_y) + 2\mu v\,N_y\}\,d\Omega + \int\int_\Omega f_y\,w\,d\Omega = b.c.\ terms \tag{2.80}$$

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 2D matrix equation

$$[K_\lambda] = \begin{bmatrix} N_x\lambda N_x & N_x\lambda N_z \\ N_z\lambda N_x & N_z\lambda N_z \end{bmatrix} \tag{2.81}$$

and

$$[K_\mu] = \begin{bmatrix} N_x 2\mu N_x + N_z\mu N_z & N_z\mu N_x \\ N_x\mu N_z & N_z 2\mu N_z + N_x\mu N_x \end{bmatrix}. \tag{2.82}$$

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:

$$[K_\lambda] + [K_\mu] = [B]^T[D][B] \tag{2.83}$$

$$[D_\lambda] + [D_\mu] = [D] \tag{2.84}$$

where

$$[D_\mu] = \mu \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.85}$$

and

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

and

$$[B] = \begin{bmatrix} N_x & 0 \\ 0 & N_y \\ N_y & N_x \end{bmatrix}. \tag{2.87}$$

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 (Equation 2.83) is made up of the two terms from the left hand side of the integral equation. The full element stiffness matrix for the quadralilateral element is an 8 by 8 matrix made up of 16 of the 2 by 2 matrices. In older versions of ConMan we only stored the upper triangular part of the matrix. **New to Version 3.0:** In the original version of ConMan, we took advantage of this by considering only the upper triangular part of the stiffness matrix and saving both storage and operations using Cholesky factorization. In the TALA and ALA the stiffness matrix is no longer symmetric, so we now use sparse matrix factor **unfact** and back substitution **unback** routines for all cases. 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, 1987, p. 84-89]. More details of the method and a formal error analysis can be found in Hughes et al. [1979]. The stiffness matrix is formed in routine **f_vstf** and the right-hand side is formed in routine **f_tres**.

The energy equation is an advection-diffusion equation. The formal statement is

Find $T : \Omega \rightarrow R$ such that

$$\dot{T} + u_i T_{,i} = \kappa T_{,ii} + H \quad on\ \Omega \tag{2.88}$$

$$T = b \quad on\ \Gamma_b \tag{2.89}$$

$$T_{,j} n_j = q \quad on\ \Gamma_q \tag{2.90}$$

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

$$\int_\Omega (w+p)\,\dot{T} d\Omega = -\int_\Omega (w+p)\,(u_i T_{,i})\,d\Omega \tag{2.91}$$

$$-\kappa \int_\Omega w_{,i} T_{,i} d\Omega + \int_{\Gamma_q} w T_{,j} n_j d\Gamma_q \tag{2.92}$$

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

$$p = \tau u \nabla T = \tilde{k} \frac{u_i w_{,i}}{||u||^2} \tag{2.93}$$

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 et al., 1979]. The Petrov-Galerkin function can be thought of as a standard Galerkin method in which we counterbalance the numerical under diffusion by adding an artificial diffusivity of the form

$$(\xi u_\xi h_\xi + \eta u_\eta h_\eta)/2 \tag{2.94}$$

with

$$\xi = 1 - \frac{2\kappa}{u_\xi h_\xi} \tag{2.95}$$

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

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 [1979]. It has since 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.

For the 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., 1990].

For the explicit method, the time stepping in the energy equation is done using a second-order predictor-corrector algorithm. The form of the predictor-corrector algorithm is
Predict:

$$T_{n+1}^{(0)} = T_n + \Delta t \left(1 - \alpha\right) \dot{T}_n \tag{2.97}$$

$$\dot{T}_{n=1}^{(0)} = 0 \tag{2.98}$$

Solve:

$$M^* \Delta \dot{T}_{n+1}^{(i)} = R_{n+1}^{(i)} \tag{2.99}$$

$$R_{n+1}^{(i)} = -\left[\dot{T}_{n+1}^{(i)} + u \cdot \left(T_{n+1}^{(i)}\right), x\right](w+p) - \tilde{k} w_{,x}\left(T_{n+1}^{(i)}\right), x + \quad (\textit{boundary condition terms}) \tag{2.100}$$

Correct:

$$T_{n+1}^{(i+1)} = T_{n+1}^{(i)} + \Delta t \alpha \dot{T}_{n+1}^{(i)} \tag{2.101}$$

$$\dot{T}_{n+1}^{(i+1)} = \dot{T}_{n+1}^{(i)} + \Delta \dot{T}_{n+1}^{(i)} \tag{2.102}$$

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, 1987, p. 562-566].

The predictor step is computed in the subroutine **time_driver**, the residual vector, R, is formed in the subroutine **form_temp_rhs_explicit**, the lumped mass matrix , M^*, (which is a vector) is formed in the subroutine **form_temp_mass_matrix**, and the corrector step is also computed in the subroutine **form_temp_rhs_explicit**.

For the implicit energy equation solver, the equations are cast in a matrix form. The big advantage here is the use of Picard iteration, which allows the solution to quickly advance to the steady-state solution in far fewer steps. This version of the energy equation uses the routines **form_temp_matrix** and **form_temp_rhs_implicit**.

Chapter 3

Incompressible and Compressible Equations

Version 3.0 of ConMan includes the Extended Bousinessq Approximation (EBA), Truncated Alelastic Liquid Approximation (TALA), and Alelastic Liquid Approximation (ALA) forms of the compressible equations [King et al., 2010]. The formulation can be changed by setting the flag (itype) on the second line of the input file. These forms follow the derivations in King et al. [2010] including the use of the Adams-Williamson equation of state. This was a useful analytic form for a benchmark but probably not the best form for research. This is provides so that users can see how to extend ConMan for their own research. Here we repeat section 2 of King et al. [2010] with comments so that the reader can follow the implementation in the code. The references can be found in King et al. [2010].

Mass conservation is given by

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) \ = \ 0, \tag{3.1}$$

where ρ is the density and $\vec{u}$ is the velocity The conservation of momentum is given by

$$\frac{D\rho\vec{u}}{Dt} \ = \ -\nabla P \ + \ \nabla \cdot \tau + \rho\vec{g} \tag{3.2}$$

where P is the pressure, $\vec{g}$ is the gravity, D/Dt is the material derivative, and τ is the deviatoric stress tensor given by

$$\tau = 2\eta\dot{\epsilon} = \eta(\nabla\vec{u} + \nabla\vec{u}^{\ T}) - \frac{2}{3}\eta\nabla\cdot\vec{u}\delta_{ij} \tag{3.3}$$

where η is the dynamic viscosity, $\dot{\epsilon}$ is the strain-rate tensor, and δ_{ij} is the Kroneker delta. Equation 3.3 assumes that the bulk viscosity of the fluid is zero. Finally, the equation of energy conservation is given by,

$$\rho c_p \frac{DT}{Dt} \ - \ \alpha T \frac{DP}{Dt} \ = \ \nabla \cdot (k\nabla T) \ + \ \rho H \ + \ \phi \tag{3.4}$$

where T is the temperature, c_p is the heat capacity at constant pressure, α is the coefficient of thermal expansion, k is the thermal conductivity, H is the volumetric heat production and ϕ is the viscous dissipation given by

$$\phi = \frac{1}{2}\ \tau : \dot{\epsilon} = \tau_{ij}\frac{\partial u_i}{\partial x_j}. \tag{3.5}$$

In compressible convection, there is the additional required assumption–the reference state,

$$T = \bar{T} + T' \tag{3.6}$$

$$P = \bar{p} + p' \tag{3.7}$$

$$\rho = \bar{\rho}(\bar{T}, \bar{p}) + \rho' \tag{3.8}$$

where the over-barred quantities are time-independent and functions of depth only. The reference pressure is given by the hydrostatic approximation

$$\nabla \bar{p} = \bar{\rho} g. \tag{3.9}$$

Using the assumption that $p' \ll \bar{p}$, we can eliminate pressure from the energy equation (3.4), yielding

$$\begin{aligned} \rho c_p \frac{DT'}{Dt} \;&=\; \nabla \cdot (k \nabla (T' + \bar{T})) \;+\; \rho H \;+\; \phi - \rho c_p \vec{u} \cdot \nabla \bar{T} \\ &\quad - \alpha (\bar{T} + T') \bar{\rho} g w \end{aligned} \tag{3.10}$$

where $\vec{u} \cdot \vec{g} = -wg$, where w is the upward component of velocity.

For the reference state $(\bar{\rho}, \bar{T})$, we assume an adiabatic Adams-Williamson equation of state (Birch, 1952), where

$$\bar{\rho}(z) = \rho_r \exp(\frac{\alpha_r g_r}{\gamma_r c_{p_r}} z) \quad , \quad \bar{T}(z) = T_{surf} \exp(\frac{\alpha_r g_r}{c_{p_r}} z) \tag{3.11}$$

where z is the depth coordinate (parallel to the direction of gravity), γ_r is the reference value for the Grüneisen parameter, T_{surf} is the surface temperature, and variables with the sub-script r are constant values used in defining the reference state. From this reference state, we note that $\nabla \bar{T} = (0, -\alpha_r g_r \bar{T} / c_{p_r})$, which along with dropping terms with ρ' and that $c_p \approx c_{p_r}$, allows us to further simplify the energy equation (3.10),

$$\bar{\rho} \bar{c}_{p_r} \frac{DT'}{Dt} \;=\; \nabla \cdot (k \nabla (T' + \bar{T})) \;+\; \bar{\rho} H \;+\; \phi - \bar{\rho} \bar{\alpha} g w T'. \tag{3.12}$$

The expansivity $\bar{\alpha}$ is $\frac{\alpha}{\alpha_r}$ and formally dependent on the reference state. For the purposes of the benchmark, we will assume that $\bar{\alpha} = 1$.

3.1 Equations under the Anelastic Liquid Approximation (ALA)

We nondimensionalize the equations using the reference values for density, ρ_r, thermal expansivity, α_r, temperature contrast, ΔT_r, thermal conductivity, k_r, heat capacity, c_p, once again assuming that $c_p \approx c_{p_r}$ depth of the fluid layer, L, and viscosity, η_r. The non-dimensionalization for velocity, u_r, pressure, p_r, and time, t_r, become

$$u_r = \frac{k_r}{\rho_r c_p L} \quad , \quad p_r = \frac{\eta_r k_r}{\rho_r c_p L^2} \quad , \quad t_r = \frac{\rho_r c_p L^2}{k_r}. \tag{3.13}$$

The non-dimensionalization introduces four non-dimensional numbers, the Prandtl number, Pr, the Mach number, M, the dissipation number, Di, and the Rayleigh number, Ra. If we assume that the relative volume change due to temperature, $\alpha_r \Delta T_r \ll 1$, $M^2 Pr \ll 1$ and $Pr \to \infty$, we arrive at the Anelastic Liquid Approximation (ALA).

Under the ALA, the conservation of mass becomes

$$\nabla \cdot (\bar{\rho} \vec{u}) \;=\; 0 \tag{3.14}$$

the conservation of momentum becomes

$$0 \;=\; -\nabla p' \;+\; \nabla \cdot \tau + Di \frac{\bar{\rho} c_p}{K_s \gamma_r c_v} p' - Ra \bar{\rho} \bar{\alpha} \hat{g} T' / \Delta T_r \tag{3.15}$$

where $\hat{g}$ is the unit vector in the direction of gravity, c_v is the specific heat at constant volume, $\bar{\rho}$ is now dimensionless (i.e, equation (3.13) divided by ρ_r) and the Rayleigh number and dissipation number are given by

$$Ra = \frac{\alpha_r \Delta T_r {\rho_r}^2 g_r L^3 c_p}{\eta_r k_r}, \qquad Di = \frac{\alpha_r g_r L}{c_p} \tag{3.16}$$

With the assumption of constant thermal conductivity, and using the dimensionless reference states for $\bar{\rho}$ and $\bar{T}$ given by

$$\bar{\rho} = \rho_r \exp(z' Di / \gamma_r), \qquad \text{and} \qquad \bar{T} = \frac{T_{surf}}{\Delta T_r} \exp(z' Di), \tag{3.17}$$

where z' is the dimensionless vertical coordinate. The conservation of energy (3.12) under the ALA becomes

$$\bar{\rho}c_p\frac{DT'}{Dt} + Di\bar{\rho}\bar{\alpha}wT' \;=\; \nabla^2 T' \;+\; \bar{\rho}H \;+\; \phi\frac{Di}{Ra} + Di^2\bar{T}. \tag{3.18}$$

Equation (3.18) resembles the familiar energy equation in Bousinessq convection with the addition of three terms: the viscous dissipation, $\phi\frac{Di}{Ra}$, the work done against gravity

$$W = Di\bar{\rho}\bar{\alpha}wT' \tag{3.19}$$

and a second-order term in dissipation number, $Di^2\bar{T}$ that arises from substituting equation (17) into the $\nabla\cdot(k\nabla\bar{T})$ term in equation 3.12. Because $\bar{T}$ is independent of time and only depends on depth, this terms acts like a depth-dependent internal heat source. The volume-averaged work done against gravity $< W >$ exactly balances the volume-averaged viscous dissipation $< \phi >$ [Hewitt et al., 1975, Zhang and Yuen, 1996, Leng and Zhong, 2008]. This is one of the measures that we use to assess the energy conservation of the codes.

It is also worth pointing out that the reference state used here fails when $T_o \to 0$, leading to the non-sensible reference temperature state $\bar{T} = 0$. For the ALA benchmark, equations 3.14, 3.15 and 3.18 are solved.

3.1.1 Equations under the Truncated Anelastic Liquid Approximation (TALA)

For the TALA, the pressure term in the buoyancy force is ignored, in which case equation 3.15 becomes

$$0 \;=\; -\nabla p' \;+\; \nabla\cdot\tau - Ra\bar{\rho}\bar{\alpha}\hat{g}T' \tag{3.20}$$

Some numerical methods have difficulty with equation 3.15 and the TALA (equation 3.20) has often been used in compressible studies [Jarvis and McKenzie, 1980, Christensen and Yuen, 1985, Ita and King, 1994]. Jarvis and McKenzie [1980] show that there is an imbalance between viscous dissipation and gravitational potential energy with the TALA. Leng and Zhong [2008] demonstrate that the imbalance is caused by ignoring dynamic pressure's effects on buoyancy and can be removed using the ALA. We further compare the difference between these formulations below.

3.2 Equations under the Extended Bousinessq Approximation (EBA)

For the EBA, the reference state changes to $\bar{\rho} = 1$ and $\bar{T} = 0$. This leads to the first step beyond the Bousinessq approximation and is a useful check that the additional terms in the energy equation that scale with dissipation number, Di, have been implemented and scale properly. With the further assumption that $\bar{\alpha} = 1, k = 1$, and $\bar{c_{p_r}} = 1$, the conservation and constitutive equations become

$$\nabla\cdot\vec{u} \;=\; 0, \tag{3.21}$$

the conservation of momentum becomes

$$0 \;=\; -\nabla p \;+\; \nabla\cdot\tau - Ra\hat{g}T' \tag{3.22}$$

$$\frac{DT'}{Dt} + Diw(T' + T_o) \;=\; \nabla^2 T' \;+\; \bar{\rho}H \;+\; \phi\frac{Di}{Ra}. \tag{3.23}$$

and

$$\tau = 2\eta\dot{\epsilon} = \eta(\nabla\vec{u} + \nabla\vec{u}^{\,T}). \tag{3.24}$$

Note that because $\bar{T} = 0$, the boundary condition at the base of the fluid layer remains unmodified in the EBA (i.e., $T' = 1$).

3.3 Equations under the Bousinessq Approximation (BA)

By dropping the terms that scale with the dissipation number, Di, equations 3.21-3.23 reduce to the Bousinessq approximation. Under the BA, equations 3.21, 3.22 and 3.24 remain unchanged and equation 3.23 becomes

$$\frac{DT'}{Dt} \;=\; \nabla^2 T' \;+\; \bar{\rho} H. \tag{3.25}$$

Chapter 4

Implementation

4.1 Introduction

There are generally three phases to ConMan, input, time stepping, and output. 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 **time_driver.F**. All output subroutines have been gathered into a subroutine called **output**. The rather cumbersome structure of **eglib** calling **eg2.F** for element routines has been removed. This was originally in place to allow for different types of elements to be considered but has never been used.

In version 3.0 we have made a number of minor changes:

1. Many subroutines were renamed to take advantage of the relaxation of the eight character naming limit in FORTRAN77. The major elemenr subroutines are now named **form_velocity_stiffness_matrix**, **form_velocity_rhs**, **form_temperature_rhs_explicit**, **form_temperature_rhs_implicit** and the files are given the same names as the subroutines. This should make it easier for users to follow the code.

2. For compressible convection, the stiffness matrix is no longer symmetric, thus we now calculate the full local and global stiffness matrices and use subroutines **unfact.F** and **unback.F** to calculate the forward reduction and back substitution of the global stiffness matrix. This is not as efficient for the Bousinessq and Extended Bousinessq cases but it simplifies the code.

3. We have removed the memory manager and now use FORTRAN's allocate function to dynamically allocate memory. This greatly improves the portability of the code.

4. Picard iteration is now a runtime, as opposed to compile, option.

5. We have collected the output subroutines into a subroutine called **output.F** and this is called at the end of the time-stepping loop in subroutine **time_drive.F**.

4.2 Material Properties

4.2.1 Buoyancy

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

$$Ra = \frac{g\alpha\Delta T d^3}{\kappa\mu} \tag{4.1}$$

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,

$$Ra_{buoy} = g\alpha \tag{4.2}$$

The depth, d, and the temperature difference, ΔT are specified from the grid and the temperature boundary conditions, while 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. This is the way most users set up problems in ConMan.

4.2.2 Rheology

The viscosity can be a function of temperature, pressure, or strain-rate. This is done in the subroutine **rheol**. The subroutine **rheol** is called for each element and the values of the coordinates *xl*, velocity *vl*, temperature *tl*, and element number *iel*, are passed into subroutine **rheol**. The viscosity, calculated at each integration point, *evisc(intp)* is returned by **rheol**. The functional form of the viscosity law depends on the choice of the input parameter *ntimvs*. If *ntimvs* = 0, then a constant viscosity is used and *evisc(intp)*=1.0 for each integration point. If *ntimvs* = 1, then the viscosity is a function of temperature following

$$evisc(intp) = \mu_o \exp\left[-b\ tq(intp)\right] \tag{4.3}$$

where $b = 6.907755279$, $tq(intp)$, is the temperature at the integration point (calculated from tl), and μ_o is the value of the viscosity from the input file (usually chosen to be 1.0). The choice of b gives a factor of 1000 variation across the temperature range of 0 to 1. This is the rheology used in Blankenbach et al. [1989] case 2a. If *ntimvs* = 2, then the viscosity is a function of temperature following

$$evisc(intp) = \mu_o \exp\left[-b\ tq(intp) + c\,(1.0 - zq(intp))\right] \tag{4.4}$$

where $b = 9.704060528$, $c = 4.148883083$, $tq(intp)$, is the temperature at the integration point (calculated from tl), $(1.0 - zq(intp))$ is the depth of the domain and μ_o is the value of the viscosity from the input file (usually chosen to be 1.0). The choice of b gives a factor of 10000 variation across the temperature range of 0 to 1. This is the rheology used in Blankenbach et al. [1989] case 2b.

If *ntimvs* = 3, then the viscosity is a function of temperature and pressure following an Arrhenius formulation,

$$evisc(intp) = \mu_o \left\{ \exp\left\{ \frac{E^* * 1.0e3 + V^*(1.0 - z(intp))}{R * (tq(intp) + T_o)} \right\} - \exp\left\{ \frac{E^* * 1.0e3 + V^*(1.0 - z(intp)}{R * (1 + T_o)} \right\} \right\} \tag{4.5}$$

where μ_o is the preexponential viscosity, E^* is the activation energy (set in the input file), V^* is the activation volume (set in the input file), and T_o is the temperature offset. 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.F** to be 273. ΔT is hardwired to be 2000.0. The scaling in **rheol.F** is such that E^* can be input in kJ/mole and V^* can be input as cm^3.

If *ntimvs* = 4, the rheology is a combination dislocation/diffusion creep rheology following ...

If *ntimvs* = 5, the rheology is a yield-stress formulation consistent with Stein and Hansen [2008]. In this form, the effective viscosity *evisc(intp)*,

$$evisc(intp) = \frac{2}{\frac{1}{\eta_{PT}} + \frac{1}{\eta_E}} \tag{4.6}$$

where η_{pT} is the pressure and temperature dependent rheology,

$$\eta_{PT} = \exp\left[-b\ tl(intp) + c(1 - z(intp))\right] \tag{4.7}$$

and η_E, is the viscoplastic form

$$\eta_E = \eta^* + \frac{\sigma_y}{\dot{e}_2} \tag{4.8}$$

where η^* is the plastic rheology, σ_y is the yield stress rheology, and $\dot{e}_2$ is the second invarient of the strain-rate tensor. In this formulation, η^* is hard-wired to 10^{-5} and b is hard-wired to $11.512925 = \ln 10^5$. c is set from the input file (tcon(1)) and σ_y is set from the input file (tcon(2)).

The user can easily modify the existing or add new rheology options for specific problems of interest. Using *ntimvs* = 4 or 5 one should add an iteration loop within each time step in subroutine **time_drive.F** to allow the change in velocity each step to come to consistency with the rheology. Experience has shown that after the first step, which may require 10-20 iterations, 2-3 iterations per step is all that is needed to converge [King et al., 1990].

Users who are new to viscous flow should approach viscoplastic or composite rheologies with an abundance of caution, as small changes can cause convergence headaches and/or unexpected results.

4.2.3 Internal Heating

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

$$Ra = \frac{g\alpha H d^5}{k\kappa\mu} \tag{4.9}$$

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.

Chapter 5

Installation

5.1 Building from Source

5.1.1 System Requirements

ConMan has been tested on a variety of computational platforms including:

- Mac OS X El Capitan (10.11.6) (gfortan configured with gcc version 6.1.0)
- Scientific Linux 6.10 (gfortran configured with gcc version 4.4.7 20120313)
- ubuntu xxxx (?) *placeholder for CIG information*

It is likely to run on other platforms as well but has not been tested.

5.1.2 Dependencies

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

5.1.3 Downloading and Unpacking the ConMan Code

You can get the source for the latest release from the ConMan web page (`geodynamics.org/cig/software/packages/mc/conman/`). Download the source archive and unpack it using the `tar` command:

```
$ tar xzf ConMan-3.0.tar.gz
```

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 the git version control client installed on your machine, type:

```
git -version
```

You should get a response that looks something like this:

```
git version 2.8.4 (Apple Git-73
```

Otherwise, you will need to download and install git, available at the Git Website (`https://git-scm.com/downloads`). Then the code can be checked out with the following command:

```
git clone https://github.com/geodynamics/conman.git
```

The ConMan software package contains the follow directories:

~/src ConMan source code directory.

~/doc ConMan manual and other documentation.

~/cookbook1 input and geometry files for the Blankenbach benchmark cases.

~/cookbook2 input and geometry files for the driven slab problem in van Keken et al. [2008].

~/cookbook3 input and geometry files for the compressible benchmarks in King et al. [2010].

5.1.4 Compiling and Running ConMan

ConMan comes ready to run with a standard **Makefile** that contains the system calls for the compiler (FC) and the loader (LD). The distribution assumes this is gfortran, otherwise they will need to be modified for your local environment.

There are no external libraries or packages required. All source code is in the directories src, src/grid.src, src/utils.src, and src/solver.src

Typing 'make' in the src directory will produce the ConMan executable `conman` in the main directory. The warning messages *Branch at (1) may result in an infinite loop* are normal and expected. We take advantage of FORTRAN's ability to continue reading a file after a read error. This is done to allow the user to annotate files with comments, which has proven to be a useful and popular feature. If there is really and input error, the code will fail and ususally the problem can be diagonsed looking at the file **out.xxxx**.

To run ConMan type:

```
$ conman < runfile
```

where `runfile` is a text file that has a list of filenames that can be up to 80 characters long. For more details on `runfile`, see Chapter 8.

5.2 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/)
```

WE DO NOT USE ROUNDUP ANY LONGER
You may also use the bug tracker

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

to submit bugs and requests for new features.

FEATURE REQUESTS SHOULD GO TO ISSUES. BUGS TOO???

Chapter 6

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 subroutine **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).

6.1 General Input File

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 (integer)

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

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

iflow . . . . . data check flag (integer)

0 - check data only

1 - execute code

necho . . . . . echo data flag (integer)

0 - minimum data echo (terse)

1 - echo data to output file (verbose)

inrstr . . . . . read restart file flag (integer)

0 - use default start (conductive)

1 - read restart file from unit 16

iorstr . . . . . write restart file flag (integer)

0 - don't write restart file

1 - write restart file to unit 17

nodebn . . . . . number of edge nodes for nusselt (integer) smoother

ntimvs . . . . . temperature dependent viscosity (integer) flag

0 - stiffness matrix factored once – constant viscosity

1 - Blankenbach 2a case (hardwired constants)

2 - Blankenbach 2b case (hardwired constants)

3 - temperature dependent Arrhenius law (diffusion creep) using activation energy and volume below

4 - composite dislocation/diffusion creep (code does not currently support non-Newtonian dislocation creep)

5 - rheology based on Stein and Hansen (20xx?) (currently code does not currently support non-Newtonian dislocation creep)

nwrap . . . . . number of nodes to wrap (integer)

equal to number of elements in vertical (this requires nodes must be

numbered increasing fastest in vertical direction)

itype . . . . . compressible formulation (integer)

1 - ALA

2 - TALA

3 - EBA

4 - BA

isolve . . . . . temperature solver (integer)

1 - explicit

2 - implicit

3 - Picard

Time Sequence Card *nstep accel*

nstep . . . . . number of time steps (integer)

accel . . . . . a factor that multiplies the time step (real number)

Output Step Card *nstprt tmax datasv tsave tmovie*

nstprt . . . . . steps between output (integer)

tmax . . . . . maximum time, usually diffusion scaling (real)

datasv . . . . . time interval for time series output (real)

tsave . . . . . time interval for temperature/velocity output (real)

tmovis . . . . . time interval for movie output (real) - not supported

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

Equation of state Card *Di T0 diff_ T cgamma rho0*

Di . . . . . Dissipation number

T0 . . . . . surface temperature (dimensional)

diff_T . . . . . temperature difference across the box (dimensional)

cgamma . . . . . Gruneisen parameter (non-dimensional)

rho0 . . . . . reference density (non-dimensional)

Element Parameter Card - *numat numsuf*

numat . . . . . number of material groups

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

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

visc(i) . . . . . preexponential viscosity coefficient for ith material group

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

alam(i) . . . . . penalty parameter for ith material group

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

diff(i) . . . . . thermal diffusivity for ith material group

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

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

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) . . . . . activation energy for ith material group for temperature dependent viscosity (kJ/mole)

Activation Volume Card *tcon(2,i), i=1,numat*

tcon(2,i) . . . . . activation volume for ith material group for temperature dependent viscosity (cm^3/mole)

Viscosity Cutoff Card *tcon(3,i), i=1,numat*

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.

6.2 Geometry Input File

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

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

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

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 7

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    X   Z  ck echo rrst wrst nus tdvf  iwrap  itype isolve
  2601  50  50   1    0    0    1 102    0      0      4      3
 time step information
    100   1  1.0  1.0  0.50000
 output information
    100    100    100    100
 velocity boundary condition flags: IFCMT,DELNXTLN
 bnode   enode   incr bcf1 bcf2
     1  2551     51    0    1
  2551  2601      1    1    0
      1   51      1    1    0
    51  2601     51    0    1
     1     1      1    1    1
    51    51      1    1    1
  2551  2551      1    1    1
  2601  2601      1    1    1
      0      0      0    0    0
 temperature boundary condition flags
     1  2551    51    1
    51  2601    51    1
      0      0      0    0
 bndy info (top - bottom rows)
     1  2551    51
    51  2601    51
      0      0      0
 bndy info (2nd from top - 2nd from bottom rows)
      2  2552    51
     50  2600    51
      0      0      0
 initial condition information
    0.1    1.0    1.0    1.0
 element information
 2  2500 4 4 1 2 0 5 0 0
 viscosity
  1.0e0
 penalty number
```

```
 0.1E+08
diffusivity (always one)
   1.0
Rayleigh number
 1.0e+04
internal heating parameter
   0.0
   0.0
   0.0
   1.0e7
```

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 boundary conditions (non-zero)
      0    0  0.0  0.0
 temperature boundary conditions (non-zero)
      1    2  1.0
   2551    0  1.0
     50   51
      0    0  0.0
 element connectivity and material groups
      1      1      1      1     52     53      2
     50     50     51     50      1      1
      0      0      0      0      0      0      0
```

Chapter 8

Output Guide

8.1 The Output Files

The 10 required output files names are taken from the names in the runfile. The execution of `conman` proceeds by % conman < runfile where the runfile has a list of filenames that can be up to 80 characters long.
The names in the runfile are attached to the following input or output files.

```
input
geometry
output
input restart
output restart
time series output
temperature and velocity field output
stress and viscosity field output
compressible benchmark output
geoid output
```

All files are ASCII files. The *input* and *geometry* files are as described in the previous section.

The *output* file is a formatted record of the input. If the variable *necho* is set to one, then values of the coordinates and boundary conditions are output and the file can become large. Near the end of the *output* file execution time is listed for various subparts of the code.

The *restart input* file is an input file that is used if the variable *inrstr* is set to one. The file is ASCII but formatted and the format statement can be found in the file **input.F**. The first line contains the initial timestep and time, the second line is a header, and the third through *numnp* lines contains the node, temperature and time derivative of temperature.

The *restart output* file is an output file that is used if the variable *iorstr* is set to one. We recommend this always be set to run. This file is overwritten every *nsdprt* steps and the output is written from the file **timdrv.F**

The *time series output* file is written every time step. The values are ASCII and are heat flux at the bottom, heat flux at the top, kinetic energy, and time. All values are dimensionless. This file is written from the routine **fluxke.F**

The *Temperature and velocity output* file is an ASCII file written from the routine **print.F**. There are two header lines that contain *nsd, nelx, nelz, numnp, nstep, time*. The second line labels the output and the third through numnp lines list the node, x, z, vx, vz, and temperature values at each node. This file is output every *nsvprt* steps and each successive set of values is appended to the end of the file. The unix command *split -numnp+2 tempfile* can be used to split the file into files that are *numnp+2* lines long.

The *stress output* file is an ASCII file written from the routine **prtstr** which can be found in the file **stress.F**. Like the temperature velocity file there are two header lines that contain *nsd, nelx, nelz, numnp,*

nstep, time. The second line labels the output and the third through `numnp` lines list the node, x, z, txx, tzz. txz. p and viscosity at each node. This file is output every *nsvprt* steps and each successive set of values is appended to the end of the file. The unix command *split -numnp+2 tempfile* can be used to split the file into files that are *numnp+2* lines long.

The *compressible benchmark output* file contains time series of output requested in the benchmark paper. The calculations are done in **print_compbm_data.F** Some of the calculations are repeated from **fluxke.F** but the duplication was left in place. Both subroutines are called in **output.F** and one or the other could be disabled if the user so chooses.

The *geoid output* file contains the horizontal coordinate, the dynamic topography, the geoid contribution from the temperature only and the total geoid (from surface and cmb topography, and internal (i.e., temperature) density contrasts). This is written from the file **geoid.F**. The dimensional values are hard wired into the code and can be found at the top of **geoid.F**. This version of the geoid code has not been tested with flow-through boundary conditions (i.e, *nwrap = nelz*). ***Warning.*** *As written, the code assumes the parameters from Blankenbach et al. [1989] and adjusts the viscosity as the Rayleigh number changes. It would be ideal to make the scaling parameters input. This is left to the user community.*

See Table 9.1. It is best to carefully check the ***geoid.F*** *source for the specific problem of interest.*

The *geoid output* file contains additional output for the compressible case. The columns are: Rayleigh number, Nusselt Number, Vrms, maximum surface velocity, average surface velocity, rms temperature, volume-average viscous dissipation, ϕ, and volume-average work against the gravitational potential, W. In the ALA approximation, ϕ and W should exactly balance. This is a measure of the energy conservation approximation.

Chapter 9

The Benchmark Cases

We provide input and geometry files for the benchmark cases described below in the directories `cookbook1`, `cookbook2` and `cookbook3`.

9.1 Cookbook 1: Blankenbach Benchmark Cases

Here we reproduce the results from Blankenbach et al. [1989] for constant viscosity and temperature-dependent viscosity in a unit-aspect ratio domain, with free-slip boundary conditions, heated from below and cooled from above. ConMan results were not a part of the Blankenbach et al. [1989] paper but results from the the grids used here were published in King [2009]. 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 9.1. The time is the CPU time in seconds for the Picard version of the code using 30 iterations (60 iterations for the temperature-dependent, 2a, problems) on a MacBook Air with a 2.2 GHz Intel Core i7 using the gfortran compiler (6.1.0) with -O3 optimization. While the problems in Blankenbach et al. [1989] 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. The global properties of Nusselt number and root-mean-square velocity, 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 9.2 (Rayleigh number 10^4), Table 9.3 (Rayleigh number 10^5), and Table 9.4 (Rayleigh number 10^6). For the global properties of Nusselt number and root-mean-square velocity, the 50 by 50 element grid are already 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 9.4). 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 9.1: Mantle parameters for Blankenbach constant viscosity benchmarks.

Grid	V_{rms}	Nusselt No.	$Topo_L$	$Topo_R$	$Geoid_L$	$Geoid_R$	Time (sec)
50	42.997	4.887	2261.956	-2911.473	55.346	-63.178	2.98
100	42.947	4.885	2256.094	-2905.356	54.957	-62.765	16.05
200	42.910	4.885	2254.541	-2903.764	54.856	-62.658	182.96
†C_{ext}	42.865	4.884	2254.021	-2903.221	54.822	-62.622	
†Christensen's extrapolated values.							

Table 9.2: Blankenbach et al. [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.	$Topo_L$	$Topo_R$	$Geoid_L$	$Geoid_R$	Time (sec)
50	195.165	10.546	1482.778	-2014.228	28.846	-33.104	2.95
100	194.253	10.539	1467.169	-2008.138	28.034	-32.327	14.74
200	193.767	10.536	1462.487	-2005.474	27.789	-32.099	182.83
†C_{ext}	193.214	10.534	1460.986	-2004.205	27.703	-32.016	
†Christensen's extrapolated values.							

Table 9.3: Blankenbach et al. [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.	$Topo_L$	$Topo_R$	$Geoid_L$	$Geoid_R$	Time (sec)
50	850.794	21.864	941.607	-1301.980	14.958	-16.678	4.04
100	841.722	22.023	945.108	-1290.926	14.109	-15.632	24.10
200	837.643	21.980	936.439	-1285.756	13.654	-15.204	289.96
†C_{ext}	833.989	21.997	931.962	-1283.813	13.452	-15.034	
†Christensen's extrapolated values.							

Table 9.4: Blankenbach et al. [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	497.239	9.993	1041.464	-4012.790	18.584	-55.084	15.56
100	487.755	10.083	1017.502	-4081.259	17.657	-54.790	173.33
200	483.714	10.069	1012.217	-4094.521	17.417	-54.654	2441.90
†C_{ext}	480.433	10.066	1010.925	-4098.073	17.343	-54.598	
†Christensen's extrapolated values.							

Table 9.5: Blankenbach et al. [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.

In addition, 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

$$\eta(T) = \eta_o \exp\left[-\ln\{1000\}\frac{T}{\Delta T}\right] \tag{9.1}$$

where $\eta_o = 2.9 \times 10^{19}$ and $\Delta T = 1000.0$. The other scaling parameters are the same as Table 9.1. The results for a Rayleigh number of 10^4 are presented in Table 9.5. 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%.

The benchmark calculations can be run by typing Run_Cookbook1.sh and the tables can then be created using Make_table.sh. You may need to edit Make_table.sh to specify the location of latex and dvips if they are not already in your path.

9.2 Cookbook 2: Constant Viscosity Driven Slab Problems from van Keken *et al.* [2008]

The geometry used here is based on the subduction zone benchmark described in van Keken et al. [2008]. The results presented here differ slightly from the published results using ConMan because in that work a deformed mesh was used. Here we use a regular mesh. While we have never fully investigated the reason behind this, we suspect that element locking [Hughes, 1987, pp. x-yy] is a significant problem because the element geometry was aligned so that for the element in the corner, one edge of the element was along the slab and the other edge of the element was aligned with the bottom of the lithosphere (at 50 km depth) which was a specified no-slip condition. Thus three of the four nodes of the element were boundary conditions. The benchmarks require using a discontinuous velocity field (split nodes along the fault), which turns out to be critical to match the benchmark results. It also uses a linear ramp in slab velocity from zero at the corner (50 km, 50 km) to vslab 6 km down the slab. This was done in order to minimize the impact of the pressure singularity at the corner of the wedge.

In the subdirectory Cookbook2 I include 66x60, 132x120, 198x180, 264x240, and 330x300 element grids for three types of constant viscosity problems. It is not possible to use a 6 km velocity ramp along the slab, as required by the benchmark, because the element diagonal lengths are 14 km for the 66x60 element grid, 7 km for the 132x120 element grid, 4.7 km for the 198x180 element grid, 3.5 km for the 264x240 element grid and 2.8 km for the 330x300 element grid. If a user wishes to explore these problems further, they might consider modifying the mesh to create a region near the tip of the wedge that is more highly refined with a spacing that would allow for a 6 km ramp. The **A** series of calculations use the analytic Batchelor solution in the mantle wedge. As such, these only test advection. The **B** series of calculations use the analytic Batchelor solution at the boundaries of the domain where mantle wedge material flows in or out. The **C** series of calculations use natural boundary conditions along the boundaries of the domain. The 330x300 grid computations take approximately 5 minutes each.

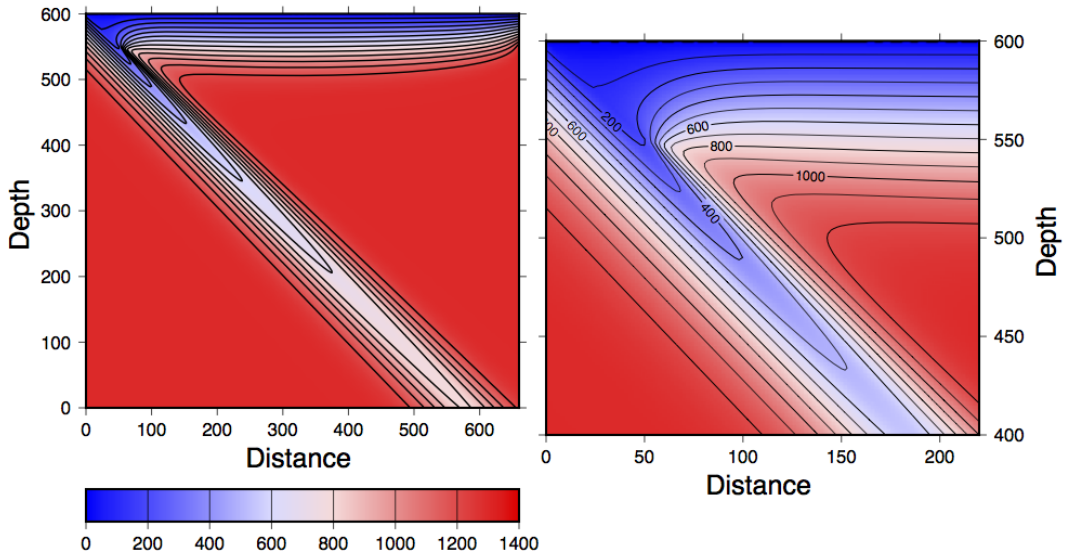


Figure 9.1: Thermal structure for the mantle wedge problem in Cookbook 2 example for 330 by 300 element grid with imposed Batchelor wedge inflow/outflow boundary conditions. Compare with van Keken et al. [2008] Figure 2.

For the grids provided, the resulting thermal structure should look like Figure 9.1. There are two key changes: 1) the value of the velocity boundary condition flag is set to 2, instead of 1 for nodes that the user wants the code to use the imposed Batchelor solution [Batchelor, 1967]; 2) the problem requires a fault, which is implemented here in **subduct.src/fault.F**. The fault comes into play in the advection of the temperature field. The fault is assumed to cross the element from the upper left hand corner to the lower right hand corner. This works because in the benchmark problem the fault is at 45 degrees and this problem uses square elements. For nodes or below on the fault (footwall) the velocities at the lower left, lower right and upper left Gauss points are set to vslab and the upper right Gauss point is set to zero. For the upper triangular part of the fault (hanging wall) the velocities of the lower right and upper left Gauss points, which are on the fault, are now set to zero (the hanging wall sees no fault movement). Subroutine **fault.F** is called in **form_temp_matrix.F**. The Batchelor solution is calculated in **subduct.src/batchelor.F** and assumes the specific grid geometry used in these input files. If the user wishes to modify the grid, they will need to hack the **batchelor.F** and **fault.F** files appropriately.

Imposing the Batchelor solution could also be accomplished by setting the velocity boundary condition flags to 1 and in the geometry file setting the values of the velocity boundary condition to be the velocities from the Batchelor solution.

The subduction zone thermal benchmark requests output values specified at specific grid points and the user would be well advised to carefully read van Keken et al. [2008] for details. From van Keken et al. [2008], *to compare model results each group contributed the temperature field as discreted values T_{ij} on an equidistant grid with 6 km spacing, which is a 111×101 matrix stored row-wise starting in the top left corner. From this grid we have extracted the following measurements for direct comparison: (1) the temperature $T_{11,11}$ which is at coordinates (60, 60 km) and just down-stream from the corner point.* Further, users are requested to provide the L2 norm of the slab-wedge interface temperature between 0 and 210 km depth defined by

$$||T_{slab}|| = \sqrt{\frac{\sum_{i=1}^{36} {T_{ii}}^2}{36}}, \tag{9.2}$$

Case	Grid	T_{11}	$\|\|T_{slab}\|\|$	$\|\|T_{wedge}\|\|$	Time (sec)
1a	66x60	226.61	463.77	804.32	2.21
1a	132x120	346.51	488.73	827.83	12.90
1a	198x180	364.19	499.35	835.75	38.74
1a	264x240	402.26	506.77	838.02	132.31
1a	330x300	386.77	505.58	837.93	300.57
1b	66x60	193.49	381.30	686.77	2.50
1b	132x120	262.84	438.60	770.27	10.22
1b	198x180	288.47	463.42	797.87	44.29
1b	264x240	415.82	511.40	840.80	135.43
1b	330x300	387.50	500.64	833.42	321.01
1c	66x60	192.10	372.87	666.34	2.45
1c	132x120	260.88	433.84	759.46	10.30
1c	198x180	286.81	459.99	790.22	44.88
1c	264x240	414.05	509.23	836.24	144.93
1c	330x300	385.94	498.59	828.96	324.21

Table 9.6: Slab thermal structure results from van Keken et al. [2008] benchmarks.

and the L2 norm of the temperature of the mantle wedge from 54 to 120 km depth,

$$||T_{wedge}|| = \sqrt{\frac{\sum_{i=10}^{21}\sum_{j=10}^{i} {T_{ij}}^2}{78}}. \tag{9.3}$$

These quantities are reported in Table 9.6. Comparing Table 9.6 with the results in van Keken et al. [2008], few observations can be made. First, the results for T_{11}, $||T_{slab}||$, $||T_{wedge}||$ approach the result reported in van Keken et al. [2008] for ConMan and are within the range of values reported by the other codes ($379.87 < T_{11} < 396.3$, $502.26 < ||T_{slab}|| < 520.14$, and $825.89 < ||T_{wedge}|| < 866.52$ and for 1a. For 1b and 1c, the wedge values are somewhat lower than most codes, yet are inline with the 1a result. Previously the VT result (i.e., ConMan) was the lowest in the range of reported values, thus the results here are actually an improvement over the results reported in the benchmark. The results reported in van Keken et al. [2008] used an irregular grid and the fault implementation was not as clean as the implementation here. The next thing to note is that the low-resolution grid (66x60) results are more than 100 degress lower than the most refined grid for T_{11} but the differences are less pronounced for $||T_{slab}||$ and $||T_{wedge}||$.

It is instructive to look at the temperature along the top of the slab, plotted in Figure 9.2. It is notable that for the coarse grids (red, greeen, and blue lines), the imposed Batchelor velocity solution (solid lines, 'a' cases), which is the true solution, and the calculations with the Batchelor boundary conditions (dotted lines,'b' cases) or natural boundary conditions (dashed lines, 'c' cases) are far apart. With increasing grid resolution, the difference between the imposed Batchelor velocity solution and the numerical velocity solution approach each other.

On the coarse grids, the under and over-shoot of the temperature (i.e., the saw-tooth pattern), is characteristic of an under-resolved solution. SUPG elements can only do so much. This behavior disappears as the grid is refined.

To run the benchmark calculations use the scripts run_66x60.sh, run_132x120.sh, run_198x180.sh, run_264x240.sh, and run_330x300.sh in the Cookbook2 subdirectory. The scripts will run the three problems on the grid described in the script name and create plots (postscript) of the temperature and velocity fields using GMT. (The plotting script is written for GMT5.) If you do not have GMT on your system you can comment out the lines that contain the script plot_temp_GMT Then the script Make_table.sh will create a latex table of the results which should be identical to Table 9.6. This script also uses GMT for the analysis. The resulting file with the table will be vanKeken08.ps You may need to edit Make_table.sh to specify the location of latex and dvips if they are not already in your path. Figure 9.1 can be reproduced by running the script run_330x300.sh and then Make_vanKeken08_fig2.sh. The resulting file will be vanKeken08_fig2.ps

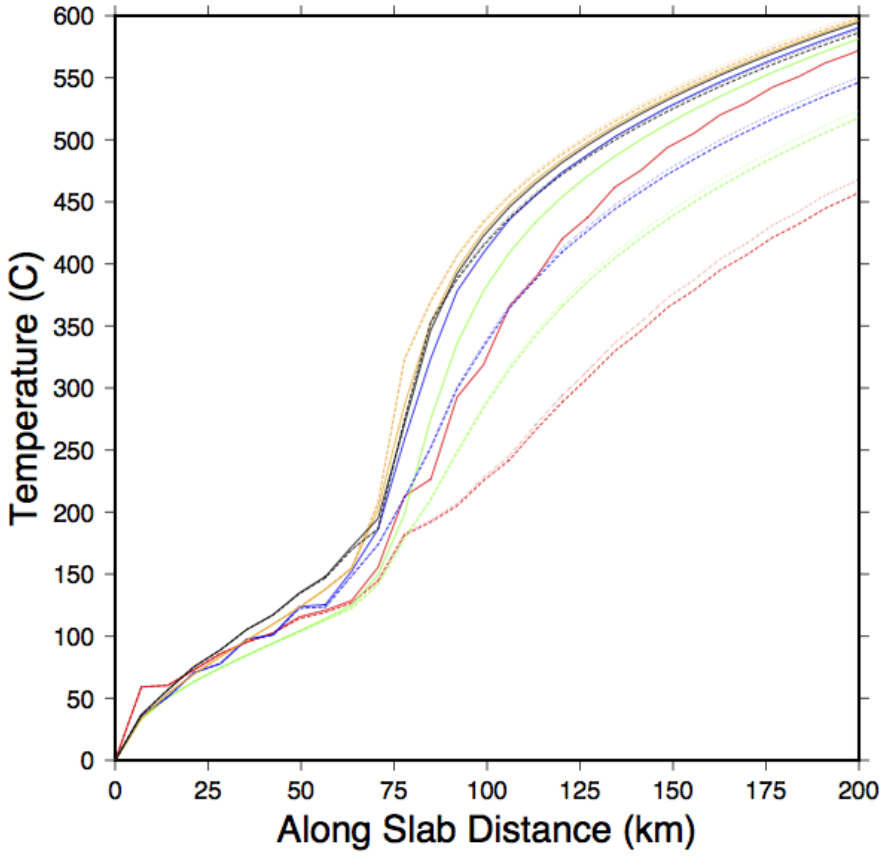


Figure 9.2: Temperature along the top of the slab for the calculations in Table 9.6. Solid lines are for 'a' cases, dotted lines are for 'b' cases and dashed lines are for 'c' cases. The red lines are 66x60 element grids, the green lines are 132x120 element grids, the blue lines are 198x180 element grids, the orange lines are 264x240 element grids, and the black lines are for 330x300 element grids.

To reproduce the plot of slab surface temperature, Figure 9.2, run Make_top_of_slab.sh. The resulting postscript file will be tos.ps.

9.3 Cookbook 3: Compressible Benchmark Cases from King *et al.* [2010]

The benchmark problems are in unit-aspect ratio domain, with free-slip top, bottom, and sides walls. The total temperature is fixed at the top $T(z=0)=0$ and bottom $T(z=1)=1$ and the side-walls have no flux boundary conditions. We consider Bousinessq (BA), extended Bousinessq (EBA), truncated anelastic liquid (TALA) and anelastic liquid (ALA) approximations. All thermodynamic properties are fixed constants. The Rayleigh number, Ra, ranges from $10^4 - 10^6$ and we vary the Dissipation number, Di, from 0 to 2. All of the cases in the test suite use 64 by 64 uniformly spaced elements. All grids used in the benchmark are in the range of 60-128 elements (or nodes) per side. We compared some solutions on more refined grids to check the convergence of the methods and error; however, the grids here are sufficient to resolve the problems in this study. The participants were asked to report surface heat flux (Nusselt number) and rms velocity for all cases and in addition, the viscous dissipation and work done for the compressible cases.

The Nusselt number is a ratio of the average surface heat flow from the convective solution to the heat flow due to conduction and is calculated by

$$Nu = -\frac{1}{\lambda \Delta T}\int_0^{\lambda} \frac{\partial T'(x, z=z_{top})}{\partial z} dx, \tag{9.4}$$

where 0 and λ are the left and right coordinates of the domain, respectively, z_{top} is the top of the domain and, ΔT is the temperature contrast across the domain. The rms velocity is given by

$$V_{rms} = \frac{1}{\lambda(z_{top}-z_{bot})}\int_{z_{bot}}^{z_{top}}\int_0^{\lambda} \left[(u^2+w^2)\right]^{\frac{1}{2}} dx \;\; dz. \tag{9.5}$$

In addition to the set of calculations described above, we added a series of temperature-dependent cases with a viscosity function

$$\eta(T) = \eta_o \exp\left[-\beta T\right], \tag{9.6}$$

where $\eta_o = 1$, T is normalized by $\Delta T = 3000.0$ and $\beta = \ln(1000)$, following problem 2a in Blankenbach et al. [1989]. These calculations are in a unit-aspect ratio domain, with free-slip top, bottom, and sides walls. Participants were asked to provide ALA results if possible and if not, TALA results.

In the directory Cookbook3 there is a shell script *RUN_TEST_SUITE* which runs a series of problems from the 2010 compressible benchmark paper. Each problem is in a subdirectory named in such a way that the problem is fairly obvious. Each of the constant viscosity problems runs in under 10 minutes and many run in under 2 minutes. The temperature-dependent problems run in under 2 hours. The entire test suite will run overnight on most systems (if you are impatient you can move the temperature-dependent problem directories to a name that starts with something other than 'test') and the script will avoid running these. After the test suite is finished you can run Make_table.bash (this assumes that you have latex and dvips installed). This will generate a table with your results and the expected values, and should look like the table below. You can view the resulting table on your system by typing <view> blankenbach.ps, where <view> is the software you use to view postscript files on your system.

9.3.1 Boundary Conditions for Compressible Convection

It is important to point out that boundary conditions for temperature require special care when compared with the more commonly used Bousinessq approximation. In the description of this problem, the total temperature jump across the model is ΔT_r which is comprised of both a contribution from the reference state $\bar{T}$ and the potential temperature T'. The non-dimensional temperature at the surface, T_o is given by $T_{surf}/\Delta T_r$. Because equation 3.18 is written in terms of the potential temperature, T' the boundary conditions for equation 3.18 are, $T'(z=0)=0$ and $T'(z=1)=1-\exp(Di)$. This requires care when defining the Nusselt number, as discussed above. The different codes used different formulations (e.g., CU formulation

is based on total temperature, while the VT and UM formulations are based on potential temperature). We do not internally modify the basal boundary condition in this formulation and the user must specify the boundary condition as $T_{bc} = 1 - \exp(Di)$ in the **geom.** file.

This table is a sample of the results from the test suite in Cookbook 3. For some of the cases, the results do not match the King et al. [2010] results exactly. Additionally, King et al. [2010] only report top Nusselt number and only report the temperature dependent results for the ALA case. Here we report both the top and bottom Nusselt numbers, which provides a measure of the energy conservation.

There are several potential reasons for the disagreement: 1) there may have been rounding of some of the values reported in the supplemental table; 2) the results were computed from four separate versions of ConMan, one for each approximation, in the merged version the order of some of the operations differ substantially from the separate versions; 3) the small differences may reflect the cutoff at which the solution was judged to have converged; 4) for the $Ra = 10^6$ case a refined grid was likely used. The results here are consistent with those in Cookbook 1 and demonstrate that a substantially more refined grid is needed to achieve the rms velocity reported in King et al. [2010] for this Rayleigh number.

The results here are consistent with the variations between codes reported in the benchmark paper and to the best of our knowledge are correct. There is a second script RUN_PARALLEL.sh which checks for the number of CPUs on a multi-core system and runs up to that number of jobs simultaneously. The steady-state diagnostics (v_{rms}, top and bottom Nusselt numbers, average temperature and surface velocity can be compared with the ConMan results in King et al. [2010] by running the script Make_table.sh. The script assumes that latex and dvips are in your path and will produce a file named benchmark.ps. An example of benchmark.ps is shown below. The test results in Table 9.7 were computed on a MacBook Air with a 2.2 GHz Intel Core i7 processor running 10.11.6 (El Capitan) using gfortran (6.1.0) with compiler flags -O3.

Problem	V_{rms}	Nu Bot	Nu Top	ave T	Vsurf
Blankenbach 1a explicit	42.979	4.886	4.886	0.500	41.431
Blankenbach 1a Picard	42.979	4.886	4.886	0.500	41.431
King et al. [2010]	42.900	-	4.890	0.500	41.400
Blankenbach 1b explicit	194.768	10.540	10.540	0.500	198.282
Blankenbach 1b picard	194.768	10.540	10.540	0.500	198.282
King et al. [2010]	195.080	-	10.540	0.500	198.280
Blankenbach 1c explicit	846.728	22.017	22.017	0.500	884.520
Blankenbach 1c picard	846.729	22.017	22.017	0.500	884.520
King et al. [2010]	838.024	-	22.020	0.500	884.520
Blankenbach 2a explicit	493.066	10.076	10.071	0.741	98.760
Blankenbach 2a Picard	493.067	10.076	10.071	0.741	98.760
King et al. [2010]	-	-	-	-	-
EBA cv Di=0.25 explicit	38.543	4.096	4.096	0.491	36.599
EBA cv Di=0.25 Picard	38.543	4.096	4.096	0.491	36.599
King et al. [2010]	38.476	-	4.097	0.491	36.598
EBA cv Di=1.0 explicit	24.242	2.194	2.194	0.467	22.242
EBA cv Di=1.0 Picard	24.243	2.194	2.194	0.467	22.243
King et al. [2010]	24.232	-	2.194	0.467	22.243
EBA td Di=0.25 explicit	374.401	7.629	7.616	0.692	81.793
EBA td Di=0.25 Picard	374.401	7.629	7.616	0.692	81.795
King et al. [2010]	-	-	-	-	-
TALA cv Di=0.25 explicit	40.163	4.426	4.424	0.513	39.301
TALA cv Di=0.25 Picard	40.163	4.426	4.425	0.513	39.301
King et al. [2010]	40.200	-	4.430	0.513	39.300
TALA cv Di=1.0 explicit	26.072	2.567	2.505	0.509	26.432
TALA cv Di=1.0 Picard	26.072	2.567	2.505	0.509	26.432
King et al. [2010]	26.100	-	2.570	0.509	26.400
TALA td Di=0.25 explicit	383.222	7.813	7.740	0.706	82.928
TALA td Di=0.25 Picard	383.222	7.813	7.740	0.706	82.928
King et al. [2010]	-	-	-	-	-
ALA cv Di=0.25 explicit	40.043	4.414	4.415	0.515	38.837
ALA cv Di=0.25 Picard	40.043	4.414	4.415	0.515	38.837
King et al. [2010]	40.095	-	4.414	0.515	38.837
ALA cv Di=1.0 explicit	24.983	2.472	2.473	0.510	24.401
ALA cv Di=1.0 Picard	24.983	2.472	2.473	0.510	24.402
King et al. [2010]	25.016	-	2.472	0.510	24.401
ALA td Di=0.25 explicit	380.059	7.735	7.683	0.707	81.095
ALA td Di=0.25 Picard	380.058	7.735	7.683	0.707	81.094
King et al. [2010]	381.690	-	7.710	0.707	81.090

Table 9.7: Steady State, 2D, convection in a 1 by 1 box King et al. [2010] benchmarks.

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 Â© (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 Â© 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.