ConMan Users Guide

Version 2.0 July 1989

Scott King

§1 Abstract

This documents serves as a users' guide for ConMan, a vectorized finite element program for the solution of the equations of incompressible, infinite-Prandtl number convection in two-dimensions written by Arthur Raefsky and Scott King at Caltech. ConMan is a public domain program and is distributed free of charge to anyone who wishes to use it and may be freely copied and modified. ConMan is written in Standard Fortran 77 and runs under Unix and Unicos operating systems. Porting it to other systems should be straightforward. As with anything free it comes with no guarantees, but it has been benchmarked against other existing codes. The authors would appreciate any information regarding bugs or potential problems but make no promises regarding the timeliness of changes or fixes.

§2 Introduction

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

This manual is broken up into several parts: it begins with a brief introduction to the finite element method and the notation used through out 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 the vectorization strategy 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 and important points or potential traps will be noted with

♣ **This is an important point**

Questions and problems can be addressed to Scott King at
e-mail: scott@jsun.ucsd.edu
mail: IGPP 0225
Scripps Inst. Oceanography
La Jolla, CA 92093
e-mail probably has the quickest response.

§3 The Finite Element Method

In general, the finite element method takes a differential equation (strong form) and transforms it into an integral equation (weak form). For example, the strong form of this simple equation is stated as follows:

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

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

$$u(1) \ = \ g$$

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

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

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

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

$\mathcal{V}$ is the set of weighting functions defined by

$$\mathcal{V} \ = \ \{w(x) \mid w(x) \in H^1, \ w(1) \ = \ 0\}$$

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

$$\mathcal{L} \ = \ \{u(x) \mid u(x) \in H^1, \ u(1) \ = \ g\}$$

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

$$[K]x = b$$

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

There are two domains to keep in mind when thinking about the finite element method: the global domain and the parent element domain (figure 1). All calculations are done in the parent element domain and the results are assembled into the global equations. This means all calculations can be done for a single parent element. Elements of different sizes or shapes filling an irregular global domain geometry (i.e. non-rectangular) can be solved by the same program. The only difference between these elements is the Jacobian of the transformation between the input domain and the parent element domain, which is calculated in routine **genshg**.

For ConMan the choice was made to use bilinear quadrilaterals as the parent elements (figure 2). Higher order elements (i.e. biquadratic or bicubic-spline) require more computational work per element. It has been our experience that using grid refinement, rather than using high-order elements, is the best strategy for an efficient, accurate code for incompressible, advection-diffusion problems.

Figure 1. The mapping between the global domain (right) and the parent element domain (left) using the shape functions. Figure taken from Hughes, Sec 3.2.

Figure 2. The bilinear shape function for a single element (top) and the four elements whose shape functions combine to form the global shape function for node A (bottom). Figure taken from Hughes, Sec 3.2.

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

> id: transforms global nodes to equation numbers (figure 3).
> ien: transforms element local node numbers to global node numbers (figure 4).
> lm: transforms element local node numbers to global equation numbers.

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

In the code the data structures for these two arrays are

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

Figure 3. Example relationship between global nodes and equation numbers for a 2 degree of freedom problem using the id array. An equation number of zero denotes a boundary condition. Figure taken from Hughes, Sec 3.2.

Figure 4. Example relationship between global node numbers and local element numbers using the ien array. Local nodes are numbered counterclockwise from the bottom left hand corner. Figure taken from Hughes, Sec 3.2.

§4 Equations

The equations for incompressible convection (in dimensionless form) are the equations of momentum

$$\nabla^2 u = -\nabla p + Ra\ \theta\hat{k}$$

continuity

$$\nabla \cdot u \ = \ 0$$

and energy

$$\frac{\partial\theta}{\partial t} \ = \ u \cdot \nabla\theta \ + \ \nabla^2\theta + H$$

where u is the dimensionless velocity, θ is the dimensionless temperature, p is the dimensionless pressure, $\hat{k}$ is the unit vector in the vertical direction, H is the heat source term and t is the dimensionless time. In this form all the material properties are combined into one dimensionless parameter, the Rayleigh number, given by

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

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.

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 $\Delta t \rightarrow 0$.

The momentum equation is solved using the penalty method to enforce incompressibility. The formal statement of the problem is as follows:

Given:

$$
\begin{aligned}
f\colon \Omega \rightarrow \Re^n &\qquad \textit{body force vector} \\
g\colon \Gamma_g \rightarrow \Re^n &\qquad \textit{imposed velocity vector} \\
h\colon \Gamma_h \rightarrow \Re^n &\qquad \textit{imposed traction vector}
\end{aligned}
$$

$$\Gamma_g \bar{\cup} \Gamma_h = \Gamma$$

$$\Gamma_g \cap \Gamma_h = \Phi$$

where Γ is the boundary of the domain Ω. Γ_g and Γ_h are the parts of the boundary where velocities and tractions are specified.

Find $u : \Omega \rightarrow \Re^n$ and $p : \Omega \rightarrow \Re$

$$
\begin{aligned}
t_{ij,j} + f_i &= 0 \qquad on\ \Omega \\
u_{i,i} &= 0 \qquad on\ \Omega \\
u_i &= g_i \qquad on\ \Gamma_g \\
t_{ij} n_j &= h_i \qquad on\ \Gamma_h
\end{aligned}
$$

with the constitutive equation for a Newtonian fluid

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

where t_{ij} denotes the Cauchy stress tensor, p is the pressure, δ_{ij} is the Kronecker delta and $u_{(i,j)} = (u_{i,j} + u_{j,i})/2$.

In the penalty formulation, the above is replaced by

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

where

$$p^{(\lambda)} = -\lambda u_{i,i}^{(\lambda)}$$

and λ is the penalty parameter (repeated subscripts means summation over all indices).

This formulation automatically enforces incompressibility since the solution converges to the incompressible stokes equation as λ approaches infinity (Temam, 1977). Also, the unknown pressure field is eliminated. This is quite useful not only because the amount of computational work is decreased because no pressure equation is solved, but also because it eliminates the need to create artificial boundary conditions for the pressure equation. There are no pressure boundary conditions in the formal specification of the problem. By examining the equation we see that the role of pressure is to balance the system, so physically the penalty formulation makes sense.

The equation is cast in the weak form and the Galerkin formulation (i.e. the weighting functions are the same as the basis functions) is used to solve the weak form of the equation.

$$V \;=\; \{w \,\epsilon\, H^1 \mid w = 0 \quad on \; \Gamma_g\}$$

V is the set of all weighting functions w which vanish on the boundary. Similarly V^h is a subset of V parameterized by h, the mesh parameter. Let g^h denote an approximation of g which converges to g as $h \;\rightarrow\; 0$.

Find $u^h \;=\; w^h \;+\; g^h$, $w^h \,\epsilon\, V^h$, such that for all $\bar{w}^h \,\epsilon\, V^h$

$$\int_\Omega (\lambda w_{j,j}\bar{w}_{i,i} \;+\; 2\mu w_{i,j}\bar{w}_{i,j})\, d\Omega \;=\; \int_\Omega f_i \bar{w}_i^h d\Omega \;+\; \int_{\Gamma_h} h_i \bar{w}_i^h d\Omega$$

$$-\int_\Omega \left(\lambda g_{j,j}^h \bar{w}_{i,i} \;+\; 2\mu g_{(i,j)}^h \bar{w}_{(i,j)}^h\right) d\Omega$$

$$w_i^h \;=\; \sum N_A u_{iA}$$

$$u_{iA} \;=\; u_i\,(x_A)$$

where N_A is the shape function for node A for the element.

The element stiffness matrix (Figure 5) is made up of the two terms from the left hand side of the integral equation. The integration is done using two by two gauss quadrature, which is exact when the elements are rectangular and bilinear shape functions are used. The λ term is under-integrated (one point rule) to keep the large penalty value from effectively locking the element (Malkus and Hughes, 1978). The right hand side is made up of three known parts, the body force term (f_i), the applied tractions (h_i) and the applied velocities (g_i). The momentum equation is equivalent to an incompressible elastic problem, and the resulting stiffness matrix will always be positive definite (Hughes, 1986 p. 84-89). This allows us to consider only the upper triangular part of the stiffness matrix and save both storage and operations using Cholesky factorization. More details of the method and a formal error analysis can be found in Hughes, Liu and Brooks (1979).

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

Figure 5. The element stiffness matrix for the 2-D Cartesian stokes equation. The 8 by 8 matrix is make up of 16 2 by 2 submatrices of the form shown below. The λ and μ parts are shown separately for clarity.

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

$$\begin{aligned} \dot{T} \ + \ u_i T_{,i} \ &= \ \kappa T_{,ii} \ + \ H \qquad on \ \Omega \\ T \ &= \ b \qquad on \ \Gamma_b \\ T_{,j} n_j \ &= \ q \qquad on \ \Gamma_q \end{aligned}$$

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

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

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

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

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

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

Galerkin method in which we counterbalance the numerical underdiffusion by adding an artificial diffusivity of the form

$$(\xi u_\xi h_\xi \;+\; \eta u_\eta h_\eta)/2$$

with

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

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

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

The resulting matrix equation is not symmetric, but since the energy equation only has one degree of freedom per node, while the momentum equation has two or three, the storage for the energy equation is small compared to the momentum equation. Since we use an explicit time stepping method, the energy equation is not implemented in matrix form. The added cost of calculating the Petrov-Galerkin weighting functions is much less than the cost of using a refined grid with the Galerkin method. The Galerkin method requires a finer grid then the Petrov-Galerkin method to achieve stable solutions (Travis et al., 1989).

Time stepping in the energy equation is done using an explicit predictor-corrector algorithm. The form of the predictor-corrector algorithm is

Predict:

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

$$\dot{T}_{n+1}^{(0)} \;=\; 0$$

Solve:

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

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

$$\tilde{k}\; w_{,x}(T_{n+1}^{(i)})_{,x} \;+\; (\mathit{boundary\ condition\ terms})$$

Correct:

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

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

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

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

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

§5 Implementation

ConMan was designed to take advantage of machines capable of vector instructions. Hence thoughout the code, operations which would be performed on an individual element on a scalar machine are grouped together so that they can be performed on a group of elements. This is transparent to the user and requires only a small modification to the program style and data structure. The only unfortunate side effect of this operation is that the arrays of element quantities (e.g. *ien, lm, evisc*) are now shuffled to avoid vector recurrences. Consider Figure 6, a small rectangular grid with one degree of freedom per node. When assembling the equation for global degree of freedom 5 (marked N), there is a contribution from elements 1, 2, 3 and 4 (circled). This is updated as follows:

```
DO E=1,NUMEL
.
.
(localize data)
.
.
(form local stiffness matrix in LOCAL)
.
.
DO N=1,4
GLOBAL( LM(E,N) ) = GLOBAL( LM(E,N) ) + LOCAL(N)
ENDDO
ENDDO
```

where `E` is the element number, `N` is the local node number, `LOCAL(N)` is the value being assembled for element `E` and local node `N` and `GLOBAL` is the equation for the global node. Now the task is to assemble m elements, $e_1, \cdots, e_m$, at one time, making sure that no global node is updated twice. This can be accomplished if for all elements, $e_1, \cdots, e_m$, lm(e_a,n) is not equal to lm(e_b,n) for e_a and e_b in $e_1, \cdots, e_m$. Also since the innermost loops are the vectorizable loops, we unroll the `N` loop over the local nodes.

```
.
.
(loop over all blocks)
.
.
.
(localize a block of data)
.
.
(form a block of local stiffness matrices in LOCAL)
.
.
DO IV=1,NVEC
IVEL=IV+IEL-1
GLOBAL( LM(IVEL,1) ) = GLOBAL( LM(IVEL,1) ) + LOCAL(IV,1)
GLOBAL( LM(IVEL,2) ) = GLOBAL( LM(IVEL,2) ) + LOCAL(IV,2)
GLOBAL( LM(IVEL,3) ) = GLOBAL( LM(IVEL,3) ) + LOCAL(IV,3)
GLOBAL( LM(IVEL,4) ) = GLOBAL( LM(IVEL,4) ) + LOCAL(IV,4)
ENDDO
```

`NVEC` elements at a time can be processed, and the array `LOCAL` must be of length `NVEC` by 4. Notice that the `NUMEL` elements have been broken up into groups `NVEC` long with `IEL` marking the first element in the group (see Figure 7).

The shuffling of the elements is done in routine **genshp**, and the variables in the code (iel, $nvec$, iv, $ivel$) are the same as described above.

Figure 6. The four surrounding elements (1, 2, 3 and 4 circled) all contribute to the global equation 5.

Figure 7. A representation of the storage of array LM, which is NUMEL by 4 long. The NUMEL elements are broken up into groups of length NVEC for vector processing

♣ **In** *common.h* **the parameter** *lvec* **sets the maximum length of the vectors. It is usually wise to set this to the length of the vector registers on the machine (64 for Cray, 128 for Convex).**

♣ **If a length greater than 128 is used, then the size of the commons** *temp1* **and** *tempx* **must be changed in routine flow.**

The element stiffness matrix for the stokes equation is always upper triangular. Since $NVEC$ matrices at a time are formed, to save storage only the upper triangular part is stored (36 entries instead of 64 for 8 by 8 matrix). The numbering scheme for the storage is shown in Figure 8.

Figure 8. The storage for the stiffness matrix used in routine **f_vstf**.

§6 Material Properties

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

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

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.

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

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

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

$$\mu(T,Z) \ = \ \mu_o\{exp\{\frac{E^* \ + \ V^*z}{T+T_o}\} \ - \ exp\{\frac{E^* \ + \ V^*z}{1+T_o}\}\}$$

where μ_o is the preexponential viscosity, E^* is the activation energy, V^* is the activation volume, T_o is the temperature offset, T is the temperature and z is the depth. μ_o, E^*, V^*, and T_o are all input in the material section.

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

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

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.

§7 Installation

ConMan comes ready to run with a Unicos makefile. The file **Makefile** contains the system calls for the compiler and the loader, FC and LD respectively. These need to be changed for your machine. Also the calls to **second**, a Cray timing routine, will have to be changed in routine **timer**. To run ConMan on a 32 bit machine, each routine needs to have the comment in front of the line with "implicit double precision" removed. In routine **intemp** the functions cos and sin need to be changed to $dcos$ and $dsin$. Also, in routine **pickdt** the functions abs and $amin$ need to be changed to $dabs$ and $dmin$. Some compilers have automatic options to force reals to be real*8, so check your manual.

§8 Input Guide

To run ConMan a series of eight 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** are described in this section. The third file is an output file showing all the input parameters in a nace fancy form. The forth 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 and the eighth file is for mean value calculations (see routine **mean**). These file names are read in routine **flow**.

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. For the UNIX and CRAY interfaces ConMan reads the file names to attach to these units from a file named '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 13 characters long). The input deck was broken up so that an automatic grid generating routine could be used to generate coordinates, boundary conditions and element connectivities separate from ConMan. The only automatic grid generation conman does is linear or bilinear interpolation which is described in the appropriate sections of this guide.

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

Title Card
Any descriptive character string up to 80 characters long

Global Constants Card

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

numnp	total number of nodal points
nsd	number of spatial dimensions
ndof	number of degrees of freedom
nelx	number of elements in the x1 (horizontal) direction
nelz	number of elements in the x2 (vertical) direction
mprec	precision flag
	1 - single
	2 - double
iflow	data check flag
	0 - check data only
	1 - execute code
necho	echo data flag
	0 - minimum data echo (terse)
	1 - echo data to output file (verbose)
inrstr	read restart file flag
	0 - use default start (conductive)
	1 - read restart file from unit 16
iorstr	write restart file flag
	0 - don't write restart file
	1 - write restart file to unit 17
nodebn	number of edge nodes for nusselt smoother
ntimvs	temperature dependent viscosity flag
	0 - stiffness matrix factored once
	1 - stiffness matrix factored every time step
ntseq	number of time sequences
	currently only one supported
numeg	number of element groups
	currently only one supported
isky	flag for skyline factor
	0 - dmf solver (if supported)
	1 - regular skyline
nwrap	number of nodes to wrap
	equal to number of elements in vertical
	to use nodes must be numbered increasing
	fastest in vertical direction

Time Sequence Cards - ntseq cards

nstep niter alpha delt epstol

nstep	number of time steps
niter	number of multicorrector iterations 2 for explicit 2nd order
alpha	multicorrector parameter 0.5 for explicit 2nd order
delt	time step (not used)
epstol	tolerance for hybrid method (not used)

Output Step Card

nsdprt nsvprt nstprt nsmprt

nsdprt	steps between disk output
nsvprt	steps between velocity output (not used)
nstprt	steps between temperature output
nsmprt	steps between mean field output (not used)

Velocity Boundary Condition Flag Cards

bnode enode incr (bcf(i), i=1,ndof)

bnode	beginning node
enode	ending node
incr	node increment
bcf(i)	boundary condition flag for ith degree of freedom 0 - free slip 1 - pinned degree of freedom

0 0 0 0 0 to end VBCF cards

Temperature Boundary Condition Flag Cards

bnode enode incr bcf

bnode	beginning node
enode	ending node
incr	node increment
bcf	boundary condition flag for temperature 1 - fixed temperature

0 0 0 0 to end TBCF cards

Nusselt Number Boundary Condition Flag Cards - Edge Nodes

top and bottom rows of nodes

bnode enode incr

bnode	beginning node
enode	ending node
incr	node increment

0 0 0 to end NNBCF (type a) cards

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

bnode	beginning node
enode	ending node
incr	node increment

0 0 0 to end NNBCF (type b) cards

Initial Temperature Card
pert xsize zsize

pert	perturbation from conductive state
xsize	nondimensional length (x1 direction) of box
zsize	nondimensional height (x2 direction) of box

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

ntype	element type
	2 - two dimensional elements
	3 - three dimensional elements
numel	total number of elements
nen	number of element nodes
nenl	number of local element nodes
numat	number of material groups
nedof	number of element degrees of freedom
numsuf	number of surface force flux cards
nipt	number of integration points per element
implv	currently unused
implt	currently unused

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

visc(i)	preexponential viscosity coefficient for ith element

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

alam(i)	penalty parameter for ith element

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

diff(i)	thermal diffusivity for ith element

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

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

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

dmhu(i)	internal heat source for ith material group

Activation Energy Card

tcon(1,i), i=1,numat

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

Temperature Dependent Viscosity Temperature Offset Card

tcon(2,i), i=1,numat

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

Surface Force / Flux Cards - numsuf cards

nel side fnorm ftan flux

nel	element number
side	side to apply force and flux 1 - bottom 2 - right side 3 - top 4 - left side
fnorm	normal surface force
ftan	tangential surface force
flux	heat flux

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

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

§9 Sample Input Files

The lins below are a sample 32 element by 32 element input deck for a 1 by 1 square.

```
32 by 32 element input deck
1089 2 2 32 32 1 1 0 0 1 66 1 1 1 1 0
50 2 0.5 0.0005 0.000001
50 50 50 50
1 1057 33 0 1
1057 1089 1 1 0
1 33 1 1 0
33 1089 33 0 1
1 1 1 1 1
33 33 1 1 1
1057 1057 1 1 1
1089 1089 1 1 1
0 0 0 0 0
1 1057 33 1
33 1089 33 1
0 0 0 0
1 1057 33
33 1089 33
0 0 0
2 1058 33
32 1088 33
0 0 0
0.001 1.0 1.0
2 1024 4 4 1 2 0 5 0 0
1.0
1.0e07
1.0
77927.0
0.0
0.0
0.0
0.0
```

The lines below are a sample geometry file for the 32 by 32 element problem.

```
1 4 0.0 0.0
1057 1 1.0 0.0
1089 1 1.0 1.0
33 1 0.0 1.0
32 33 32 1
0 0 0.0 0.0
0 0 0.0 0.0
1 2 1.0
1057 0 1.0
32 33
0 0 0.0 0.0
1 1 1 1 34 35 2
32 1 33 32 32 1
0 0 0 0 0 0 0
```

§10 References

Brooks, A., 1981. A Petrov-Galerkin finite-element formulation for convection dominated flows. Ph.d. Thesis, California Institute of Technology, Pasadena, Ca.

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

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

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

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

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

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

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

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