P-157
111-34

NASA Contractor Report 182290

Manifest: A Computer Program for 2-D Flow Modeling in Stirling Machines

David Gedeon
Gedeon Associates
Athens, Ohio

May 1989

Prepared for
Lewis Research Center
Under Contract NAS3-25195

Date for general release May 1991

Natio
Spac

Contents

Summary

This report is about a computer program named Manifest, after *manifold* and *estimate* which more or less describe its intended purpose. That is, Manifest is a program one might want to use to model the fluid dynamics in the manifolds commonly found between the heat exchangers and regenerators of stirling machines. But not just in the manifolds — in the regenerators as well. And in all sorts of other places too, such as: in heaters or coolers, or perhaps even in cylinder spaces. There are probably non-stirling uses for Manifest also. In broad strokes, Manifest will:

- Model oscillating internal compressible laminar fluid flow in a wide range of two-dimensional regions — either filled with porous materials or empty.
- Present a graphics-based user-friendly interface, allowing easy selection and modification of region shape and boundary condition specification.
- Run on a personal computer, or optionally (in the case of its number-crunching module) on a supercomputer.
- Allow interactive examination of the solution output so the user can view vector plots of flow velocity, contour plots of pressure and temperature at various locations and tabulate energy-related integrals of interest.

Part I

Practice

Chapter 1

Introduction

Manifest can be traced to a 1985 IECEC paper [6] in which I presented some thoughts on possible scenarios for flow non-uniformities in stirling machines as well as some preliminary ideas on two-dimensional modeling. Shortly thereafter I was able to obtain funding from NASA Lewis to pursue the matter in earnest. The first contract, completed in 1986, showed that two-dimensional flow modeling on a personal computer was feasible. In the second contract, I completely rewrote (or wrote for the first time) most of the software, adding many new features and generally polishing it into its present form. Somewhere during the second contract the name Manifest came to me on a bolt of lightning.

Copies of the software are generally available. For further information, NASA contractors should contact the Stirling Technology Branch at NASA Lewis. Others may contact me directly at Gedeon Associates.

1.1 Emerging From the Void

Manifest, like most large software projects, started out as only an idea in the primordial mist. Gradually the program took shape, but the path to the final product was not always straight forward. In fact, the going was a bit steep at times. This report would not be complete without at least mentioning some of the more notable events, decisions and milestones encountered along the way.

The first decision was the choice of programming language. Long a fan of the Microsoft Pascal language, I decided to stay with it for this project too. Besides the usual features of standard Pascal such as readability, strong typing, powerful data structures, pointers and recursion; Microsoft Pascal supports a number of language extensions I have come to depend on, like:

- Separately compiled modules or units
- Dynamic array allocation

- Procedures, functions and variable-length arrays passable as arguments
- Flexible string handling
- Linkabililty with special purpose libraries

In many ways, Microsoft Pascal is more like Modula-2 than standard Pascal. It eliminates most of the usual objections to using Pascal for scientific computation.

Another early decision was: what to use for a numerical solution algorithm? My requirements for an algorithm were that it be:

- Extendable to non-standard equations sets such as those describing flow through porous materials.
- Numerically stable and forgiving of large time steps. (In jargon, this means an *implicit* algorithm rather than an *explicit* one.)
- Suitable for low Reynolds number and Mach number flows typical of Stirling regenerators.
- Usable with boundary-fitted coordinate transformations.

After reading over the literature, I selected an algorithm designed by Beam and Warming [2,3] as most likely to succeed. This proved to be a good choice as it met all the above requirements and was relatively easy to customize as needs arose. And customize I did. By the time I was finished, Manifest's algorithmic heart had undergone surgery many times and survived one major transplant. The end result bears only a superficial resemblance to the original Beam and Warming algorithm. Full details are in chapter 9.

Another major decision was the choice of user interface. The problem was how to enter graphical data — things like the size and shape of the computational domain, etc. It was evident that the best way would have the user interacting with a graphical representation of the problem on the display screen. Anything else would have been unthinkable by today's standards. Consequently, a lot of the programming effort of Manifest went into its graphical user interface. Fortunately, I was able to find a library of low-level graphics procedures (MetaWindow by Metagraphics Corp.) that spared me a lot of work with the nuts and bolts of PC's and allowed me to focus on features specific to Manifest. I am happy with the result. Examples of Manifest's user interface are found in the User's Guide part of this report.

Manifest relies on boundary-fitted coordinates, and the theory behind them turned out to be more difficult than at first I'd thought. This is the rubber-sheet geometry that makes it possible to stretch irregular-shaped physical regions into nice rectangular coordinated grids so that finite-difference solution methods can be implemented. The problem is that the governing equations *stretch* along with the physical domain. That is, they change in form — in a non-trivial way. Worse yet, I had a set of non-standard (porous flow) equations to begin with and could find no good recipe books that would tell me how to proceed. I had to work most of it out out on my own. But, I learned a lot and have recorded it for posterity in chapter 7.

And about those porous flow equations. Although one can argue that the usual Navier-Stokes equations are sufficient for modeling flow anywhere, the pore size in stirling regenerators is very much smaller than the smallest computational grid feasible in Manifest. Therefore I had to add some empirical terms to the gasdynamic equations to cover heat transfer and flow resistance in porous materials. I did what I thought were the obvious things, and reported them in a verbal presentation on Manifest at NASA Lewis. Sometimes, it's amazing what riles people up. Anyway, the result of this encounter was that I felt obliged to derive my porous flow equations from first principles. Now this really cannot be done. Like turbulence modeling, it requires a great many leaps of faith along the way. So while the resulting equations are pretty much the same as before, at least now I can point to exactly what assumptions I made to get them. You can see for yourself in chapter 8.

After Manifest was all written, it was time to test it. The idea here was to *port* Manifest's number crunching module to the NASA Lewis Cray X-MP computer so that I could run some *big* problems. Port is computerese for installing a computer program on a different computer. In the case of a program written in a high-level language like Pascal, this should be mostly a matter of re-compiling. So I thought — but I was wrong. It took me a month to get Manifest working right on the Cray. At first there were the problems of features in Microsoft Pascal not available in Cray Pascal. Yes, that's right: Cray Pascal is not quite up to microcomputer software standards. Only the hardware is fast. That was not so bad, I had expected it. What I didn't expect was that Cray Pascal would also have bugs. Specifically, problems with efficiently allocating dynamic memory (memory allocated at run time) or reading more than one Boolean-type variable per line in an input file. But, after I worked around all the bugs, the Cray X-MP did turn out to be fast and turn-around time was not too bad. Then along came Telenet and CMS-Kermit.

Telenet was my way to access the Cray from my PC via modem and telephone link. CMS-Kermit was the program (running on the IBM front-end to the Cray) that enabled me to transfer input and output files back and forth. My personal visual image of this communications link involves signals bounced off Mars, ricocheting off at least a dozen switching networks then finally arriving (in some cases) in an ancient vacuum-tube based machine, stuck off in some forgotten corner of the Lewis Research Center.

As you might have suspected, communications was not always reliable. When I could get connected (and stay connected) and Kermit was feeling OK, I was able to transfer files back and forth — at thirty bytes per second. Now computational fluid dynamics typically involves big files. Many of mine were 100 Kbytes or so. While my modem was squirting information over the telephone lines at 2400 baud (300 bytes per second), somehow Kermit was slowing things down by a factor of 10. But, we do not question these things. At any rate, I suppose it's comforting to know that in these times, when we hear so much about communications in millions of bits per second, there is still a place where life is more relaxed.

So you see, developing Manifest was a bit frustrating at times. Major hurdles — one after another — kept popping up. Even though I had some inklings in the beginning that this would be the case, I was continually amazed at the depth of things that could go wrong. But I learned a lot. I suppose that's why I went ahead with it. Three years later: It was worth it.

Chapter 2

Sample Problems

The purpose of computing is insight, not numbers
— R.W. Hamming

It is always fun — and sometimes educational — to look at pictures and this chapter is full of them. All the computational plots here were produced by Manifest. In many cases, the NASA Lewis Cray X-MP computer did the number crunching using the mainframe version of Manifest's MF module (see chapter 6). Manifest's input and output modules MFSETUP and MFOUT, running on an IBM-XT compatible PC, took care of problem setup and produced the actual graphics images. These were printed on a laser printer from the display-screen image using a separate screen-dump utility. In some cases, the PC performed number crunching duties as well.

The problems here are mostly designed to illustrate the various ways that stirling flows can be non-uniform and non-steady. Jets, manifolds and oscillating flow are all covered.

Also illustrated are some of the fine points of how Manifest is used in practice. Manifestors must always be on guard for too-large Reynolds numbers (leading to flow instabilities), too-large or too-small computational time steps (causing numerical instability or incomplete flow development) and computational grids too coarse to resolve small-scale structures in the flow or that are too *curved* (causing reduced accuracy). I have tried to point these things out as they arise naturally in the context of modeling specific problems.

2.1 Creep

Creeping flow past a square block on a plate is a well documented phenomenon. Figure 2.1 shows the situation for a Reynolds number of 0.02 in an experimental photograph taken from reference [18]. At this Reynolds number, fluid inertial forces are nil and the flow is dominated by viscosity. The key feature to note is

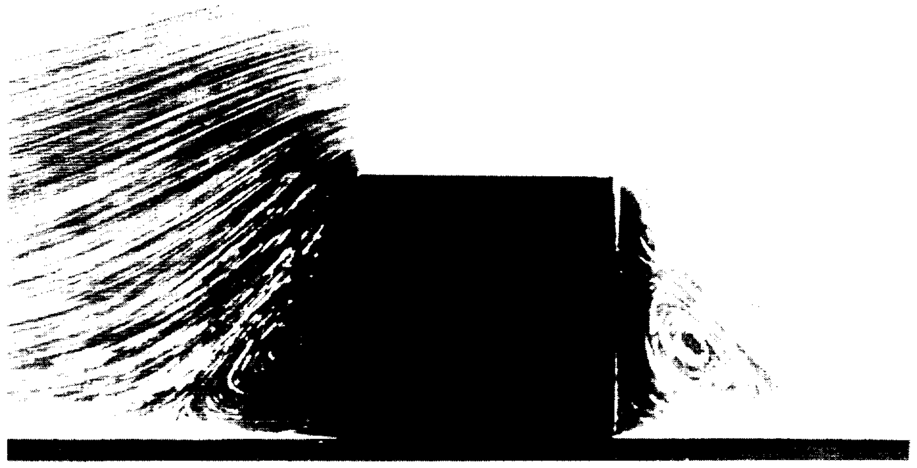


Figure 2.1: Creeping plane flow past a square block on a plate. Reynolds number is 0.02. Flow visualization by Glass beads in glycerin. From reference [18].

that the flow separates symmetrically ahead and behind the block, forming two recirculating eddies.

Set up in Manifest, this problem is good for validation purposes because it tests two important aspects of the program, namely:

1. That Manifest correctly evaluates the viscous stress tensor.
2. That Manifest correctly transforms the problem into curvilinear coordinates.

The second assertion follows because of the way the computational grid is stretched and bent. Figure 2.2 shows how I built up the computational domain by wrapping three mitered quadrilaterals around the sides of the block. Figure 2.3 illustrates the actual computational grid. It is very curved. Since, evaluating the viscous stress tensor in curvilinear coordinates is one of the most complicated operations in Manifest, getting it right goes a long way toward validating the program.

Figures 2.4 and 2.5 show the computational solution (obtained on a PC). I used one-atmosphere helium as the working fluid, but closely approximated incompressible conditions by keeping velocity low ($\approx$ 6 m/s). The computational Reynolds number is about 0.03 based on the dimensions of the block, a very low value achieved by artificially increasing viscosity by a factor of 10^5 — sticky helium. I set up linear velocity-profile boundary conditions along the left (entrance) and right (exit) edges of the domain and impermeable no-slip

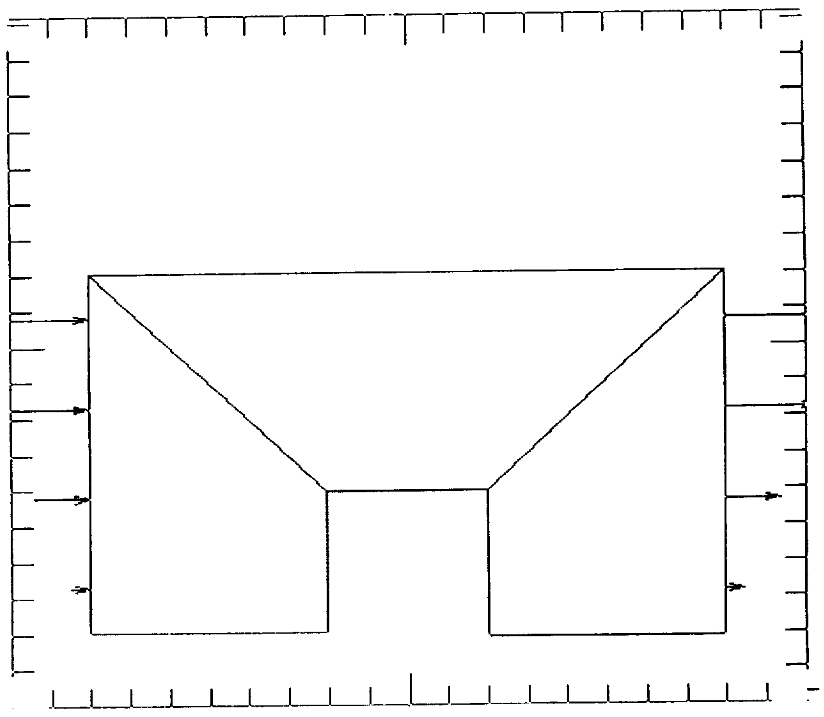


Figure 2.2: The solution domain for Creeping plane flow past a square block.

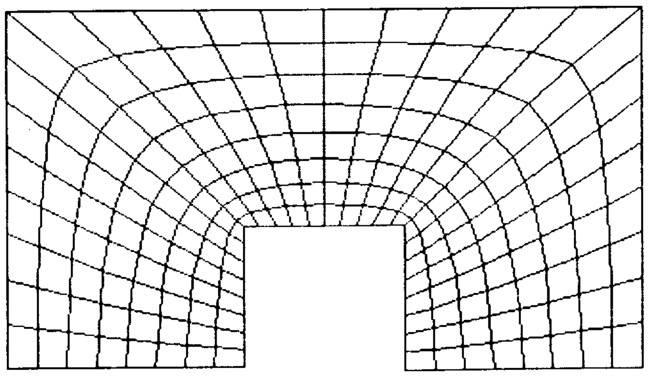


Figure 2.3: The computational grid.

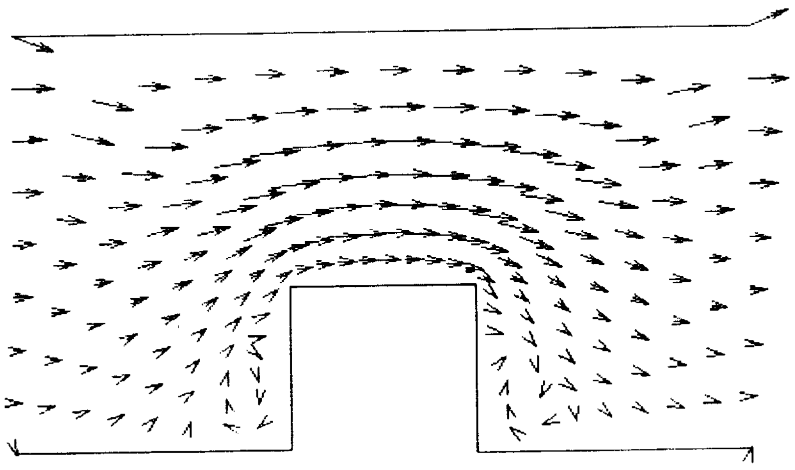


Figure 2.4: The computational solution for Creeping plane flow past a square block.

boundary conditions for the remaining exterior edges. These conditions mimic a very deep boundary layer for established flow along a plate, although there is somewhat of an inconsistency because the upper-wall boundary is not present in a free stream. I ran the problem in steady-flow mode for several time steps until the solution steadied down to the results shown. Even though the computational mesh is a bit coarse, the detail-enlargement of figure 2.5 clearly shows the required flow separation and recirculating eddy beyond the lee face of the block.

2.2 Jets

Flow leaving stirling coolers or heaters is often traveling in the form of high velocity jets when it enters the regenerator canister. The question is: to what extent do these jets penetrate the regenerator matrix? If the answer is, a significant amount, then we are likely to see the regenerator pumping loss and thermal performance degrade over that predicted by models based on uniform flow throughout.

The problems in this section address this issue in a visual way. Two dimensional laminar jets are shown impinging on regenerator samples of variable permeability — ranging all the way from no matrix at all to matrices dense enough to stop the jet dead on. However, these preliminary results are not the last word. One would do well to keep in mind the following restrictions.

- Simulated flows are two-dimensional plane flows, whereas in reality, most stirling jets emanate from tubes or rectangular channels and are three-dimensional.

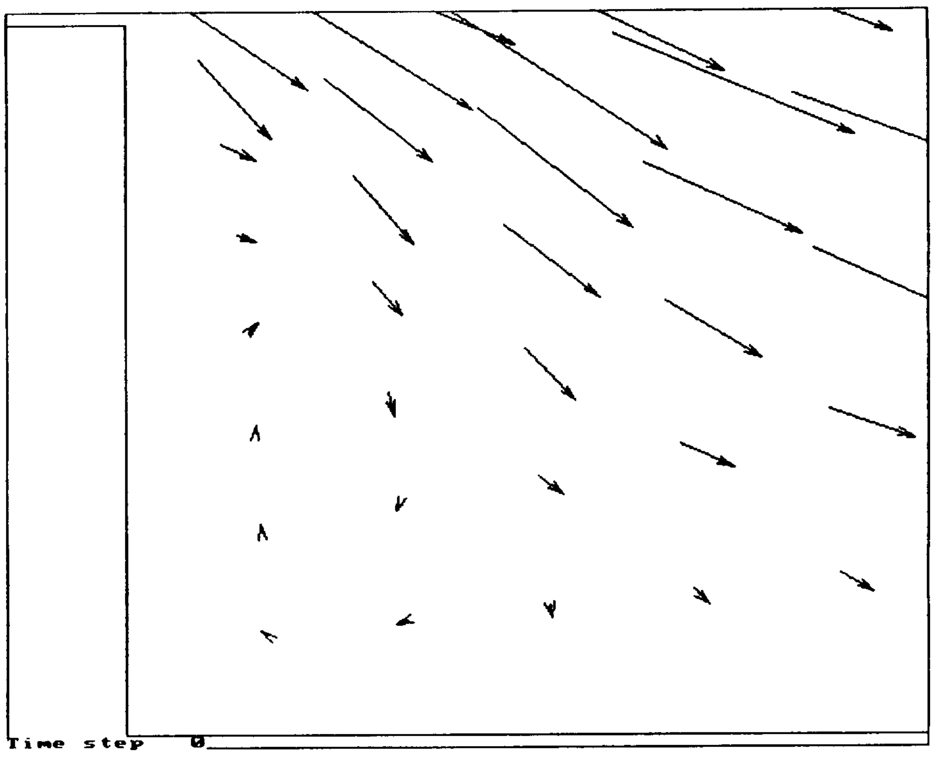


Figure 2.5: Flow detail beyond the lee face of the block.

Working Fluid	helium
Pressure	15 mPa
Temperature	650K
Jet velocity	40 m/s
Computational time step	1.25E–5 s
Computational Courant Number	$c\,\Delta t/\Delta x \approx 100$

Table 2.1: Values common to all jet simulations.

- Simulated flows are steady, not oscillating. I did this to save computational time, but, as a consequence, was unable to measure the thermal performance of the regenerator over a complete cycle, even though it is, theoretically, within the capability of Manifest.

- Reynolds numbers upstream of the matrix are restricted to values of about 100 by artificially increasing viscosity there. This has no effect on the pressure gradient within the matrix, but tends to diffuse the jets upstream of the matrix more than in reality. The reason for this restriction is that Manifest cannot properly resolve higher Reynolds number jets in its computation grid, nor can it model turbulence.

The geometry for all the problems in this section corresponds, more-or-less, to the heater-regenerator interface of the Space Power Research Engine (SPRE) on test at NASA Lewis. At the heater end of the SPRE, the regenerator canister frontal area per tube is about 13 mm^2. The square root of this number gives a rough estimate of the tube spacing — about 4 mm. In the two-dimensional equivalent problem, one can argue that an axis of symmetry occurs every 4 mm, so I chose 4 mm to be the width of the computational domain and modeled only a single jet. The length of the computational domain is 10 mm, which is somewhat shorter than the actual regenerator length of 25 mm, but I wanted to focus only on the region of interest. In the accompanying figures only about the first 8 mm of the domain is pictured. To give the jet opportunity to develop, I left a gap two computational cells wide between the jet inlet and the matrix face in all cases. At the discretization I used, this gap amounts to 2.0 mm. The actual hardware gap is 0.55 mm, somewhat less. The mass flow rate, velocity and temperature of the jets is always the same and corresponds closely to the peak influx condition of the actual SPRE. Some of the other properties common to all jet simulations are in table 2.1.

To increase resolution, I ran all the jet simulations on the Cray.

2.2.1 Free Jets

I started out by modeling free jets with no matrix present. This sounds easy, but one must be on guard for high Reynolds numbers in Manifest lest the solution

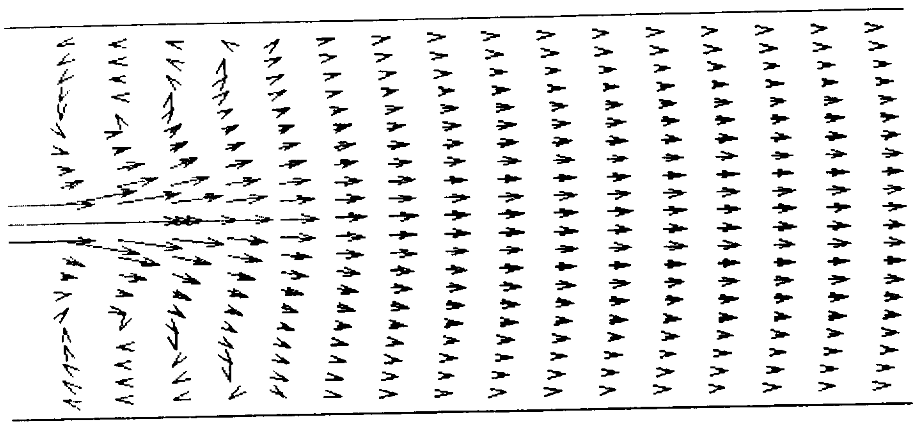


Figure 2.6: Free jet, VisMultiplier = 1000, $R_e = 10$.

go unstable or, at least, fail to adequately resolve boundary layers. Based on the inlet hydraulic diameter (twice the gap), the Reynolds number for jets in the SPRE is about 10^4 — highly turbulent and beyond the means of Manifest. The obvious question was: Just how high a Reynolds number could Manifest model accurately?

So it was that I embarked on a series of jet problems with increasing Reynolds number. I held all input variables to SPRE values except for one: viscosity. By arbitrarily setting Manifest's VisMultiplier input variable, I was able to adjust viscosity — and hence, Reynolds number — to any value I wished. Figures 2.6 through 2.9 show jets with Reynolds numbers R_e ranging from 10 to 350. At $R_e = 10$, a jet of sorts emerges. Note the flow circulations that are a necessary consequence of boundary-layer separation at the inlet. But, viscous forces quickly overwhelm the jet, resulting in rapid diffusion and re-attachment of the boundary layer to the side walls. This would not do as a basis for evaluating jet penetration into the matrix. At $R_e = 35$ the jet is much more pronounced, extending about 7 mm before wall re-attachment. Looking better. At $R_e = 100$ the jet is very pronounced but shows signs of instability. Hmmm... Finally, at $R_e = 350$ flow instability is pronounced and the jet is beginning to tax Manifest's resolving abilities in the chosen computational grid.

2.2.2 Jets Impinging on a Porous Matrix

I decided to use the $R_e = 100$ jet as the basis for simulations with a porous matrix present. This seems to be roughly the upper limit of Reynolds number for which an accurate solution is available, and instability does not set in until beyond the point at which the matrix was to begin. Figure 2.10 shows the $R_e = 100$ jet, now impinging upon the baseline SPRE regenerator. This regenerator consists of woven stainless steel screens with porosity 0.84 and a wire diameter of

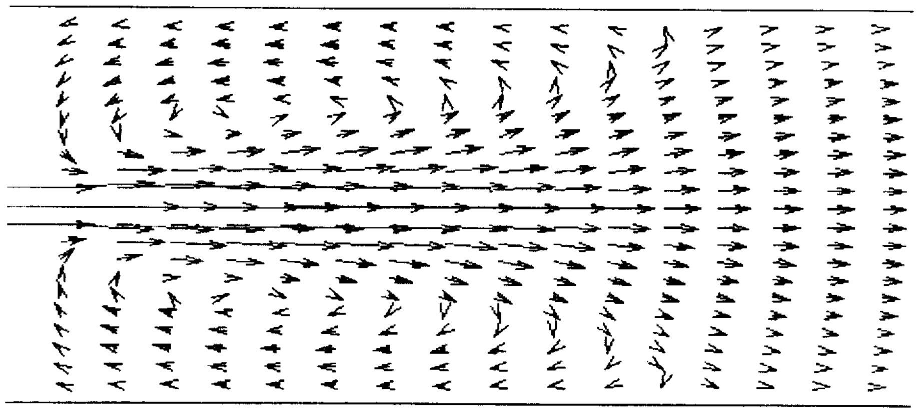


Figure 2.7: Free jet, VisMultiplier = 300, $R_e = 35$.

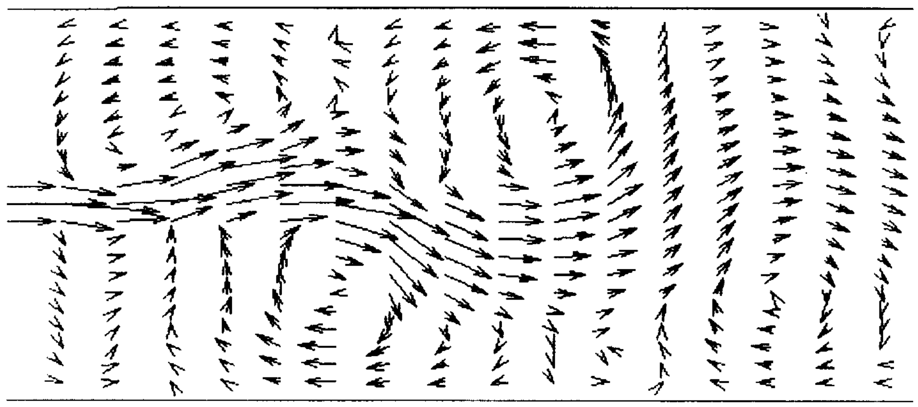


Figure 2.8: Free jet, VisMultiplier = 100, $R_e = 100$.

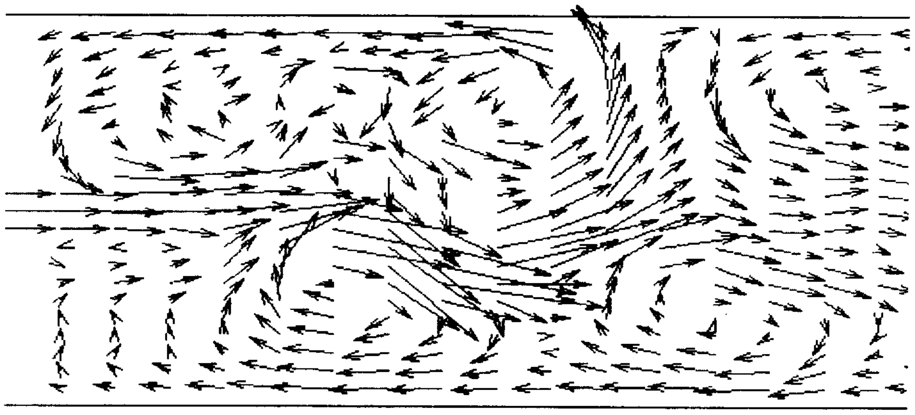


Figure 2.9: Free jet, VisMultiplier = 30, $R_e = 350$.

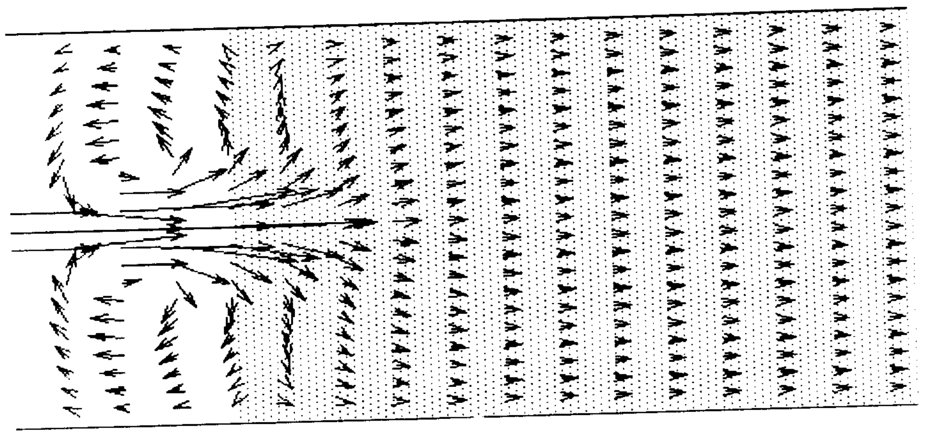


Figure 2.10: A laminar jet ($R_e = 100$) impinging on the baseline SPRE regenerator (shaded).

25 microns. Manifest knows it only as a porous material with hydraulic diameter of 133 microns. (For screens, hydraulic diameter is given by $d_h = d_w\beta/(1-\beta)$, where d_w is wire diameter and β is porosity.) Note that somewhere between the second and third millimeter into the matrix (the columns of arrows are spaced 0.5mm apart) the flow becomes essentially uniform. The reader can get the best idea of the advancing velocity profile shape by tilting the page and looking at the columns of arrows (tips) obliquely.

In figure 2.10, the fluid viscosity was only adjusted between the jet inlet and the matrix face, not in the matrix itself. So to the extent that the pre-matrix dispersion and dynamic pressure head of the jet remain intact (and I think they do), the matrix velocity field of figure 2.10 is representative of the actual flow field of the SPRE regenerator.

To get a better idea of the transitional region between the point of jet impingement and the development of uniform flow, I next simulated two cases using more permeable matrices. Figure 2.11 and 2.12 have, respectively, 1/4 and 1/16 the friction factors of the baseline SPRE regenerator. At 1/4 friction factor the flow does not become completely uniform until about 5 or 6 mm penetration, and at 1/16 friction factor the flow is noticeably non-uniform over the entire picture. At this permeability the effects of the uniform-flow exit boundary condition are no longer negligible. Note that in both cases, but especially in figure 2.12, the flow circulations on either side of the jet persist into the matrix. This means that flow is actually the wrong way at some points of the transition layer denying the regenerator the opportunity to *re-charge* thermally. As a consequence, one expects that the regenerator temperature profile will be noticeably non-uniform in the transition layer and, perhaps, for a good bit further on as well.

One might wonder if the apparent velocity boundary layers near the side

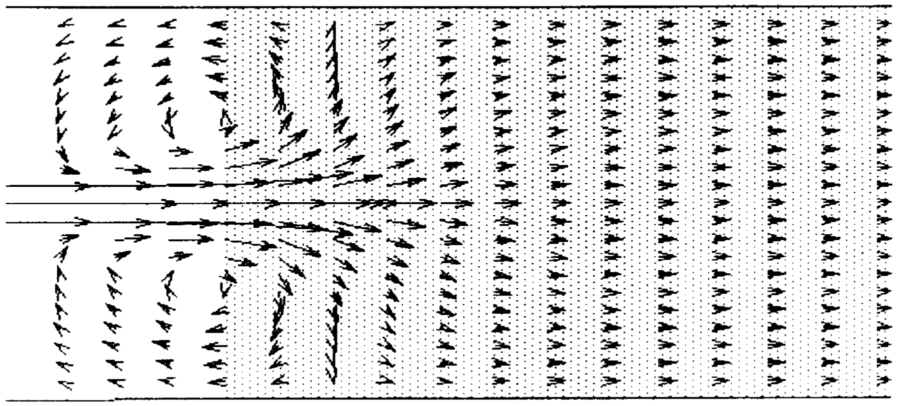


Figure 2.11: A laminar jet ($R_e = 100$) impinging on a regenerator with 1/4 the SPRE friction factor.

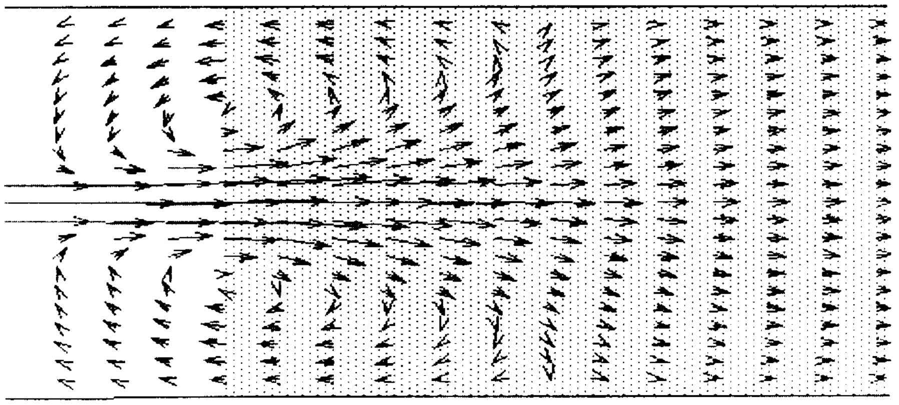


Figure 2.12: A laminar jet ($R_e = 100$) impinging on a regenerator with 1/16 the SPRE friction factor.

walls of the matrices in figures 2.11 and 2.12 are due to viscous interactions at the solid wall or just to the fact that the jet has not yet spread out sufficiently. Evidence points to the latter reason. I was able to check this out because Manifest has two ways to model the momentum equation in porous materials. The first way includes a variant of the usual viscous stress tensor along with an empirical term based on the well known Ergun equation for flow through porous materials. This is explained in full detail in chapter 8. The second way turns off the stress-tensor term under the justification that it is overwhelmed by the Ergun term in almost all cases. This way, the simulation runs faster. I simulated the 1/16 friction factor case both with and without the stress-tensor present and was unable to notice any difference between the two. From this I conclude that for all matrices simulated, the stress-tensor is indeed negligible and the no-slip wall boundary condition has no effect on the flow — even as close as one grid point away.

2.2.3 An Approximation for Jet Penetration Depth

Jet penetration depth can be understood in dimensionless form in terms of the dynamic pressure head at the jet inlet and the pressure gradient in the matrix. The idea is that by the time a jet encounters about one dynamic-head worth of frictional pressure drop, it will more-or-less run out of steam.

An approximation of the steady-flow matrix momentum equation at the jet center line is

$$\frac{\partial}{\partial x}\left(\rho v^2 + P\right) = F \tag{2.1}$$

where v is the center-line axial velocity, P is pressure and F represents the drag force per unit volume arising from viscous stress between the gas and the matrix. F can be expressed as $-(f/d_h)(\rho v^2/2)$, where f is the matrix friction factor and d_h is the matrix hydraulic diameter. The left side of (2.1) can be expanded to $v\frac{\partial}{\partial x}(\rho v) + \rho v\frac{\partial v}{\partial x} + \frac{\partial P}{\partial x}$. From the steady-flow continuity equation $\frac{\partial}{\partial x}(\rho v) = -\frac{\partial}{\partial y}(\rho u)$, where u is the radial velocity component. If we are willing to neglect $\frac{\partial}{\partial y}(\rho u)$ then it follows that $\frac{\partial}{\partial x}(\rho v)$ is negligible. Furthermore, I make the assumption that $\frac{\partial P}{\partial x} \propto F$. This assumption is more convenient than rigorous but it simplifies things. It is borne out somewhat by inspection of pressure contour plots for figures 2.10 through 2.12 (not shown). The approximate jet penetration equation then reduces to

$$\frac{\partial v}{\partial x} \propto -\frac{f}{d_h}v \tag{2.2}$$

Now, (2.2) is not quite as simple as it looks since f is itself a function of velocity. But it doesn't matter. I am only using it to lend credibility to the following guess at the solution form:

$$\frac{v(x) - v_\infty}{v_0 - v_\infty} = e^{-\left(K\frac{f_0}{d_h}x\right)} \tag{2.3}$$

Here, $v(x)$ is the axial velocity at a depth x measured from the face of the matrix, v_0 is the jet inlet velocity, v_∞ is the deep matrix velocity, f_0 is the friction factor just inside the matrix at velocity v_0 and K is some constant to be determined by computational experiment. One nice feature of the above functional form is that it allows an un-ambiguous definition of a characteristic penetration depth D as the depth for which

$$\frac{v(D) - v_\infty}{v_0 - v_\infty} = e^{-1} \tag{2.4}$$

In other words, velocity peters out by a factor $1/e \approx 0.37$ for successive layers D thick. In terms of D, the axial velocity function reduces to

$$\frac{v(x) - v_\infty}{v_0 - v_\infty} = e^{-x/D} \tag{2.5}$$

By careful examination of figures 2.10 through 2.12, one can evaluate the constant K in equation (2.3). I have done this but, unfortunately, the results don't seem to fit equation (2.5) as closely as I had hoped. For the low-permeability baseline matrix, $K = 0.2$ is a good fit. For the high-permeability matrix $K = 0.7$ seems better. The intermediate-permeability matrix is somewhere in between. Nonetheless, until something better comes along, the conservative thing to do would be to use

$$K \approx 0.2 \tag{2.6}$$

in which case the penetration depth works out to

$$D \approx 5\frac{d_h}{f_0} \tag{2.7}$$

We can take this even further. For most fibrous porous materials, the friction factor is represented in the well-known Ergun form [11] as

$$f = (c_1/R_e + c_2)\beta^{-0.6} \tag{2.8}$$

where β is porosity and for fibrous materials c_1 is about 150 and c_2 is about 1.5. Using this approximation, the penetration depth for fibrous materials becomes

$$D \approx 5\frac{\beta^{0.6} d_h}{150/R_e + 1.5} \tag{2.9}$$

In applying this formula, one must evaluate Reynolds number in the matrix just beyond the point of jet impingement — that is, as

$$R_e = \frac{\rho v_0 d_h}{\mu} \tag{2.10}$$

In the SPRE case, for example, $R_e \approx 1.7 \times 10^3$ so that the first term in the Ergun friction factor is fairly small. Assuming that this will also be the case for most jets of practical interest, one can neglect the first term in the Ergun friction factor and wind up with a particularly simple expression for the characteristic penetration depth

$$D \approx 3\beta^{0.6} d_h \tag{2.11}$$

Or, since $d_h = d_w \beta/(1-\beta)$ for fibrous materials, the characteristic penetration depth in terms of fiber diameter d_w is

$$D \approx \frac{3\beta^{1.6}}{1-\beta} d_w \tag{2.12}$$

2.2.4 A Condition for Uniform Flow

The characteristic depth D is a handy measure of jet penetration, but it is only part of the picture. Usually one is interested in the depth x required for flow to achieve some prescribed level of uniformity, namely:

$$\frac{v(x) - v_\infty}{v_\infty} \leq \epsilon \tag{2.13}$$

where ϵ is some prescribe tolerance. Now, the left side of (2.13) can be unfactored into

$$\left(\frac{v(x) - v_\infty}{v_0 - v_\infty}\right)\left(\frac{v_0 - v_\infty}{v_\infty}\right)$$

and the second factor can be approximated

$$\frac{v_0 - v_\infty}{v_\infty} \approx \sigma - 1 \tag{2.14}$$

where σ is the ratio of matrix free-flow area to total tube flow area. Replacing the first factor with equation (2.5) gives the flow-uniformity criterion as

$$e^{-(x/D)}(\sigma - 1) \leq \epsilon \tag{2.15}$$

which after a little algebra becomes

$$x/D \geq \ln\left(\frac{\sigma - 1}{\epsilon}\right) \tag{2.16}$$

Inequality (2.16) is a potentially useful result that engineers can keep in mind when designing regenerators. In the SPRE baseline case, the free-flow area ratio σ is about 8.9. The depth required for flow to achieve uniformity to within 10% ($\epsilon = 0.10$) is, according to (2.16) $x \geq 4.4D$. In terms of the D approximation given in equation (2.12) this amounts to $x \geq 62d_w = 1.6$mm at the SPRE porosity of 0.84 and wire diameter of 25 microns.

One cautionary remark: Jet penetration depth is really just a measure of the distance over which the dynamic pressure head in the jet has an appreciable affect on the regenerator velocity field. Even with a small D, one must take care to provide proper distribution manifolds at the regenerator faces to insure uniform flow. Without such manifolds, the regenerator velocity field will just fan out from the sources — more, or less, depending on the source spacing — even in the limiting case of pure Darcy (viscous dominated) flow.

2.3 Side-Inlet Manifolds

Another fundamental problem in regenerator theory is branching flow from distribution manifolds into (or out of) the faces of the regenerator matrix. To separate this phenomenon from the previously considered jet phenomenon, I am focusing on what I call ***side-inlet*** manifolds. In these, the flow at the manifold inlets is parallel to the regenerator faces, so no jets impinge upon the matrix. Roughly speaking, most regenerator manifold problems can be decomposed into the sum of a jet impingement followed by a side-inlet problem. That is, after the matrix has turned the jet aside, the flow must then branch-off into the matrix.

There are three distinct factors that cause static pressure variations in manifolds and thus prevent uniform flow distribution into the regenerator faces. They are: dynamic pressure changes, wall-friction effects and inertial pressure gradients. Each of these factors can be made arbitrarily small by increasing manifold size, but this is not an option in stirling design where minimal void volume is important. A good stirling manifold achieves uniform flow distribution with the manifold volume as small as possible.

2.3.1 Dynamic Pressure

According to Bernoulli's law, a change in dynamic pressure head occurs as flow progresses up or down the manifolds and changes velocity. Static pressure increases as velocity decreases. The velocity change depends on the rate at which flow branches off (or in) and the degree of manifold taper. The change in dynamic pressure along the manifold is roughly given by

$$\frac{\partial P_h}{\partial x} = -\rho \frac{\partial}{\partial x}\left(\frac{v^2}{2}\right) \tag{2.17}$$

where ρ is the gas density and v is manifold velocity. Typically, in a supply manifold, static pressure increases in the direction of flow due to decreasing manifold velocity as flow branches off. In a discharge manifold, the reverse happens and static pressure tends to decrease in the flow direction. Clearly, the problem is worst at times of peak velocity.

2.3.2 Wall Friction

Just as for flow in any channel, flow in manifolds must pay the price of viscous shear stress at the walls. In a supply manifold, frictional pressure drop tends to counteract the dynamic pressure change. In a discharge manifold, the reverse happens and frictional and dynamic pressure changes add. Since manifolds switch between supply and discharge roles every cycle, designing balanced stirling manifolds is especially tricky. Frictional pressure gradient $\frac{\partial P_f}{\partial x}$ can be estimated by the same means as for ordinary channel flow

$$\frac{\partial P_f}{\partial x} = -\frac{f}{d_h}\frac{\rho v^2}{2} \tag{2.18}$$

where f is a Darcy friction factor taken from some standard mechanical engineering reference. However, one should be careful not to expect great accuracy here. First of all, velocity v is changing along the manifold and flow is not fully developed. Second, velocity profiles are apt to be significantly different from textbook channel flow because branching flow at the manifold/matrix interface is nothing like the boundary condition at an impermeable no-slip wall.

2.3.3 Inertial Pressure Gradient

As a result of Newton's second law, gas inertia can induce a pressure gradient along a manifold whenever the density and time rate-of-change of gas velocity are significant. One can approximate inertial pressure gradient $\frac{\partial P_i}{\partial x}$ from

$$\frac{\partial P_i}{\partial x} = -\rho\frac{\partial v}{\partial t} \tag{2.19}$$

Note that, unlike the previous two, this effect is largest at the times of flow reversal when net velocity is zero.

2.3.4 Some Examples

By properly balancing the manifold shape, inlet velocity, matrix pressure drop, and so forth; one can wind up with reasonably uniform flow in the regenerator. But, obviously, there are many variables to play with. Too many, in fact, for there to be much hope of obtaining a closed-form general theory of manifold design. But that is why we have Manifest. By careful computational study, one can, in principle, optimize the manifold design for any given machine.

To illustrate just how one might proceed, I've chosen to model a problem that illustrates all the major manifold phenomena. I started out with a regenerator geometry that is similar to several actual regenerators I've met in the past, but identical to none of them. The basic geometry consists of a square-section regenerator with tapered side-inlet manifolds at either face. Table 2.2 gives the essential dimensions. I felt that tapered manifolds with inlet height equal to 1/10 the total length, and with height tapering linearly to 1/4 the inlet value, were about right. I chose the oscillating-flow frequency low enough to avoid any organ-pipe resonances but high enough to keep the computational Courant number in the stable range. (More on Courant-number stability later.) The

Working Fluid	helium
Pressure	10 mPa
Temperature	300K
Oscillating flow frequency	100 Hz
Regenerator length	40 mm
Regenerator width	40 mm
Matrix hydraulic diameter	39 microns
Matrix porosity	0.90
Manifold height at inlet	4 mm
Manifold height at end	1 mm
Peak inlet velocity	25 m/s
Computational time step	6.25E–4 s
Computational Courant Number	$c\Delta t/\Delta x \approx 300$

Table 2.2: Input data for a representative side-inlet manifold problem.

inlet mass-flux amplitude gives a matrix-flow tidal amplitude of about 1/10 the matrix length. To keep the manifold Reynolds numbers in the laminar range, I set the manifold viscosity multiplier to 160, which brought the peak manifold Reynolds number (evaluated at the inlet) to about 1000. Later on I increased the multiplier to illustrate the effects of increased frictional pressure gradient.

Because of the potential significance of inertial pressure gradients, all the simulations in this section used the oscillating-flow mode of Manifest. Flow is sinusoidal at the inlets and exactly in phase (or out of phase, depending on how you look at it), so that the total gas mass within the system remains constant. Running the simulations for a bit more than one cycle seemed to be enough to establish flow equilibrium.

All simulations were isothermal with zero heat transfer between the matrix and the gas. I wanted to avoid the complications of thermal effects and the necessity to consume huge amounts of computer time waiting to establish thermal equilibrium. However, given enough time, it would be possible for manifest to directly evaluate regenerator thermal performance (net enthalpy flux across the regenerator faces). In this event, to speed-along thermal equilibrium, it would probably be a good idea to artificially reduce the matrix solid thermal capacity to only a few times higher than that of the gas.

By using a relatively coarse computational mesh, I was able to run all the problems shown in this section on a PC. The solutions lack the resolution of their Cray counterparts, but are sufficiently accurate to portray the principle phenomena of interest.

The first illustration, figure 2.13, shows the velocity field in the above problem at the point of peak flow velocity. Flow distribution is worst at this time. Note that the velocity profiles in the two manifolds are quite different. Tapering of the supply manifold causes velocity to remain high throughout even though

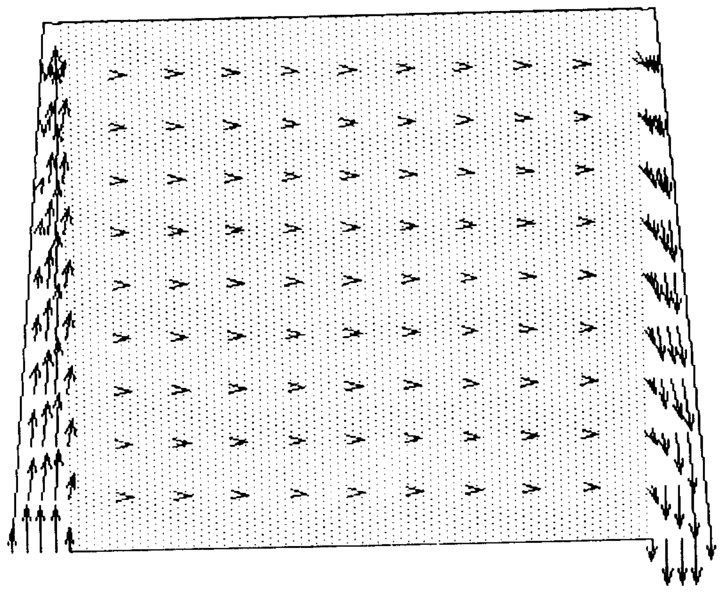


Figure 2.13: Velocity field at the instant of peak flow. Inlet Reynolds number about 1000.

flow is branching off. Also, in the supply manifold the velocity remains high near the matrix interface while in the discharge manifold it is low there. The latter effect is due to the low-velocity fluid constantly entering the fluid stream from the matrix. Consequently, the outer-wall velocity gradient, and hence wall shear stress, is higher in the discharge manifold than in the supply manifold.

The matrix interior velocity field is fairly uniform but difficult to see at the magnification factor shown. A better idea of the matrix flow can be had by looking at the pressure contour lines shown in figure 2.14. Since matrix flow is approximately proportional to the local pressure gradient, it is easy to visualize the flow field. Uniform flow implies columns of vertical pressure contours — not quite the case in this figure. For the various reasons cited above, the pressure change along the supply manifold is about 5% of the total matrix pressure drop and the pressure change along the discharge manifold about 10%. The dynamic pressure head accounts for about 1.6 contour lines in the manifolds, the rest is due to fluid friction. Apparently, in this example, the dynamic pressure head overwhelms friction in the supply manifold — pressure is highest at the top. However, In the discharge manifold, frictional pressure drop is significant.

Figure 2.15 show the situation one quarter period later at the instant of flow reversal. All velocities are small compared to the peak-flow case but there is an unmistakable pattern in the matrix interior which is driven by the inertial component of manifold pressure drop. The reason for this can be seen most

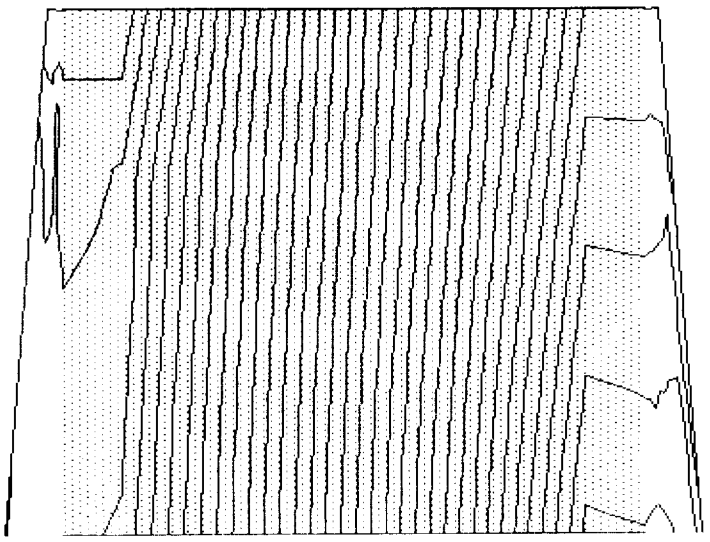


Figure 2.14: Pressure Contours at the instant of peak flow. Contour interval = 3.16 E3 N/m^2.

clearly in figure 2.16 which shows the pressure contours at the instant of flow reversal. The pressure gradients have opposite sign in the two manifolds and the mid-point pressures are about the same. The contour interval is the same as before.

The situation at the instant of peak velocity in the opposite direction is not shown. Because of the problem's symmetry, it is exactly like the previous peak-velocity illustration except reversed. It follows that in either direction, the net pressure drop across the matrix is somewhat greater near the bottom than the top (due mainly to the discharge manifold pressure gradient), resulting in increased flow there. In the present example, the effect is not too pronounced and the regenerator flow is probably good enough for most stirling purposes.

The final two illustration, figures 2.17 and 2.18 show what happens when frictional pressure drop becomes the overwhelming factor in both manifolds. I arbitrarily increased the manifold viscosity multiplier by a factor of 10. Note that the pressure gradient in the supply manifold is now reversed. This time, frictional pressure drop overwhelms the dynamic head. Also, the velocity profile in the discharge manifold now shows a much thicker boundary layer. The pressure gradient in the discharge manifold is the same sign as before but the the total pressure drop is higher.

The matter of arbitrarily setting manifold viscosity multipliers deserves some discussion. In modeling a real manifold for which flow is always laminar, one

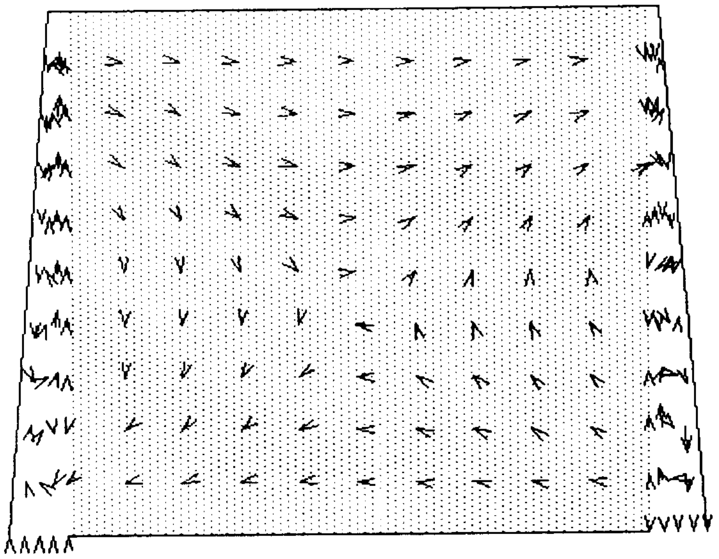


Figure 2.15: Velocity field at the instant of flow reversal.

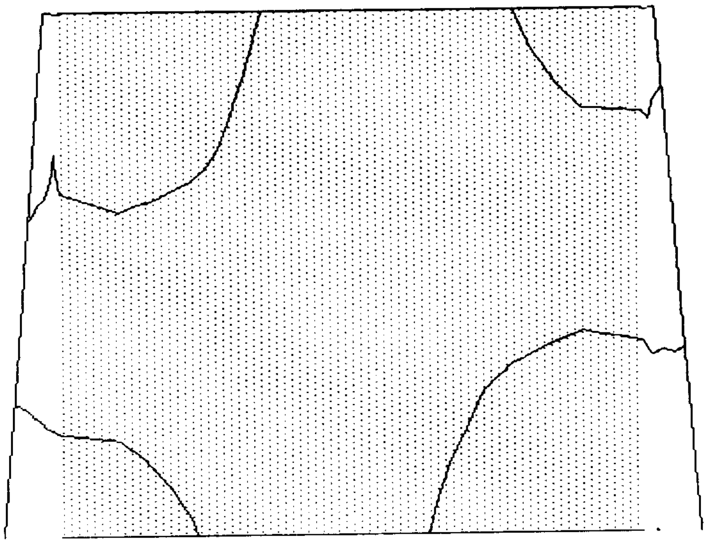


Figure 2.16: Pressure Contours at the instant of flow reversal. Contour interval = 3.16 E3 N/m^2.

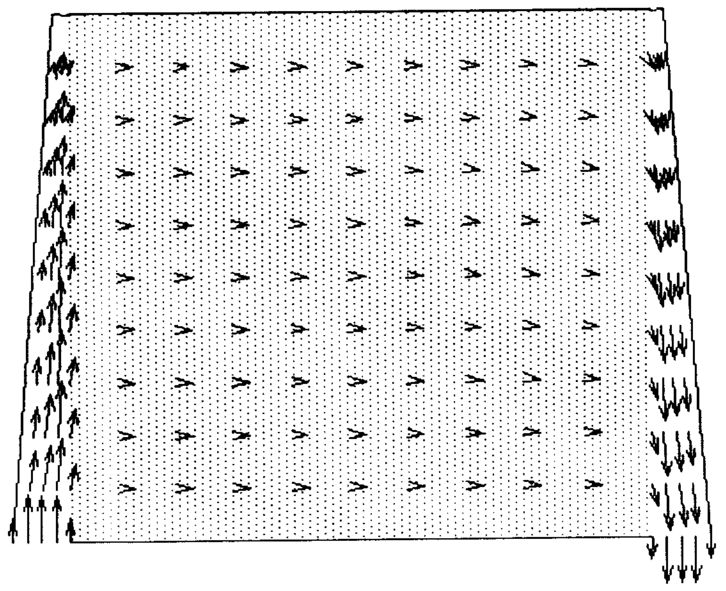


Figure 2.17: Velocity field at the instant of peak flow. Increased viscocity case. Inlet Reynolds number about 100.

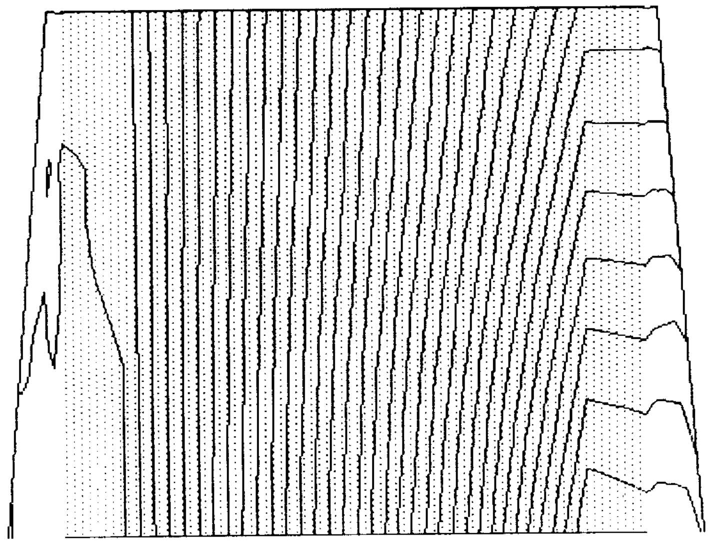


Figure 2.18: Pressure Contours at the instant of peak flow. Increased viscosity case Contour interval = 3.16 E3 N/m^2.

would certainly not do this. But in a turbulent case, a viscosity multiplier can serve as a crude turbulence model, approximating eddy viscosity in a solution devoid of the small-scale fluctuations of true turbulence. One way to estimate the viscosity multiplier is to look at a standard Moody-type friction factor diagram. By extending the laminar-flow branch slightly one can determine a Reynolds number — say R_e^* — for which the laminar friction factor equals the actual turbulent friction factor. Usually R_e^* is between 2000 and 3000. By introducing a viscosity multiplier so that the computational manifold Reynolds number will be R_e^*, the resulting non-turbulent solution in Manifest should have, roughly, the correct frictional characteristics. If the manifold is only a few computational cells thick then Manifest can model flows with Reynolds numbers in the range 2000 to 3000 without going unstable.

2.4 Oscillating Flow in Channels

Two more poorly understood areas of stirling machines are the heat transfer and pressure drop characteristics in the heat exchangers. The prevailing conditions there are oscillating flow which, depending on the dimensionless frequency, may or may not have much in common with steady flow. For better or for worse, engineers have traditionally applied steady-flow correlations for heat transfer and pressure drop to stirling machine design. And the machines have usually turned out all-right. On some occasions, though, oscillating-flow really *is* important, and steady-flow assumptions just don't work.

Until recently, the only information about oscillating flow has come from analytical studies of simple geometries. Closed-form solutions exist for incompressible, laminar, sinusoidally oscillating flow in tubes or between parallel plates. These solutions give velocity profiles under various flow conditions which, in turn, can be used to solve for wall shear stress (frictional pressure gradient). Heat transfer is more difficult. The only analytic solutions I know of assume linear axial temperature gradients for both the fluid and the wall.

However, in typical stirling heat exchangers, the flow conditions are far from ideal. Pressures are fluctuating and the flow is predominantly turbulent. The entry region, over which the velocity profile develops into its theoretical interior value, may be a significant part of the total heat exchanger length. Only in regenerators is the axial temperature profile nearly linear. In heaters and coolers the wall temperatures are roughly constant while the gas temperatures vary in a complex way along the axis. Recently, some experimental research programs have made progress in understanding realistic oscillating flows. However, the number of variables and boundary conditions may prove overwhelming for any hope of finding a concise easy-to-apply set of engineering correlations — especially for heat transfer.

Enter Manifest. By modeling actual stirling heat exchangers in 2-D and on a case-by-case basis, one can come to terms with entrance effects, nonlinear temperature profiles and weird boundary conditions. Of course, with Manifest there are still questions about the effects of turbulence and 3-D flows, but the addition of a turbulence model, along with a bit of confirmation from the wind tunnel, might enable Manifest to sort this out eventually. Anyway, even in its present form, Manifest can model some heat exchanger problems that border on realistic. What follows are some qualification problems, mostly comparing Manifest to what is already known (or might be guessed) but, perhaps, pointing out some things that are not. They were all run on the Cray.

Before getting started, here is a list of the dimensionless groups that characterize oscillating flow — at least for ideal laminar flow. These are the Valensi number (sometimes known as kinetic Reynolds number)

$$V_a = \frac{\rho\omega d_h^2}{4\mu} \tag{2.20}$$

the peak Reynolds number

$$R_e = \frac{\rho v d_h}{\mu} \tag{2.21}$$

and the tidal amplitude ratio

$$\frac{\delta}{L} = \frac{v}{\omega L} \tag{2.22}$$

Here, ρ is gas density, ω is angular frequency, μ is viscosity, v is the peak velocity (amplitude of first harmonic), d_h is hydraulic diameter, δ is the tidal amplitude (back and forth motion of a tracer particle) and L is the heat exchanger length. The Valensi number is a dimensionless measure of oscillation frequency. Flow with V_a below about 10 is essentially steady flow. For V_a above about 100 velocity profiles get weird because inertia in the core flow causes it to lag the changing pressure gradient more than the low-velocity flow near the wall. All this is complicated by (and complicates the creation of) turbulence.

2.4.1 Velocity Profiles

The obvious first step is to confirm that Manifest agrees with the exact analytic-solution velocity profiles for sinusoidal oscillating flow between parallel plates. To come close to incompressible conditions I modeled helium at 25 bar pressure and low flow velocities (Mach number less than 0.01 in all cases). I used a computational domain comprising parallel plates with 20 mm spacing and 240 mm length. To get Valensi number and peak Reynolds number into the range of interest, I had to introduce a viscosity multiplier of about 1000 on the helium. Spatially-uniform timewise-sinusoidal velocity boundary conditions were in effect at both ends of the computational domain.

Peak Velocity Profiles

The first set of plots, figures 2.19 through 2.22, show Manifest velocity profiles in the central region of the domain at the instant of peak section-average velocity.

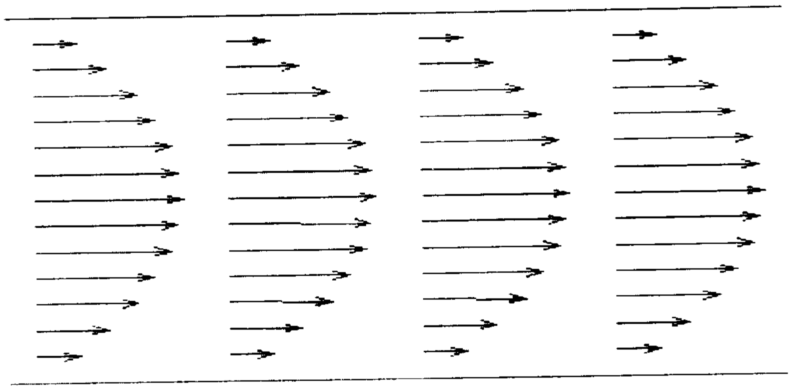


Figure 2.19: Manifest peak velocity profiles. $V_a = 10$, $R_e = 10$, $\delta/L = 4.2\text{E}{-2}$.

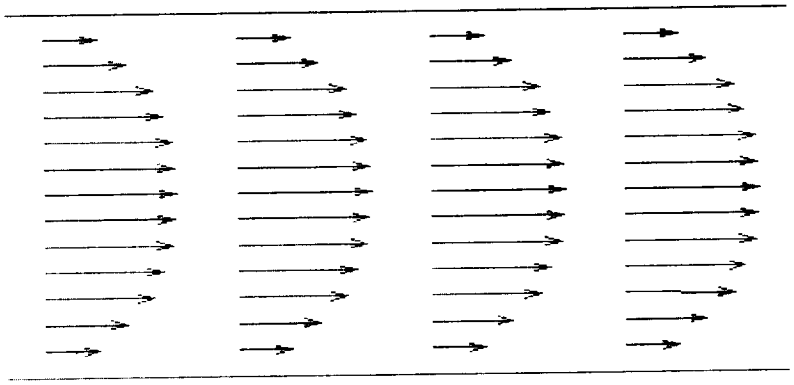


Figure 2.20: Manifest peak velocity profiles. $V_a = 40$, $R_e = 10$, $\delta/L = 1.0\text{E}{-2}$.

In all cases peak Reynolds number is only 10 and, consequently, the velocity profiles are fully developed. The plots show Valensi number increasing by factors of four from $V_a = 10$ to $V_a = 640$. Note that the flow progresses from the familiar parabolic profile of laminar flow to a *slug* profile characteristic of inertial dominance in the core flow.

Flow-Reversal Velocity Profiles

Even more interesting are the velocity profiles at the instant of flow reversal — in the sense of the section-mean average. At these times the differences between the core flow and the boundary layer are most apparent. Figure 2.23 shows the exact analytic velocity profiles for several Valensi numbers at the instant of flow reversal. These are for incompressible parallel flow in an infinite domain so the peak Reynolds number and tidal amplitude are irrelevant. The profiles are normalized by U_0 the center-line velocity amplitude.

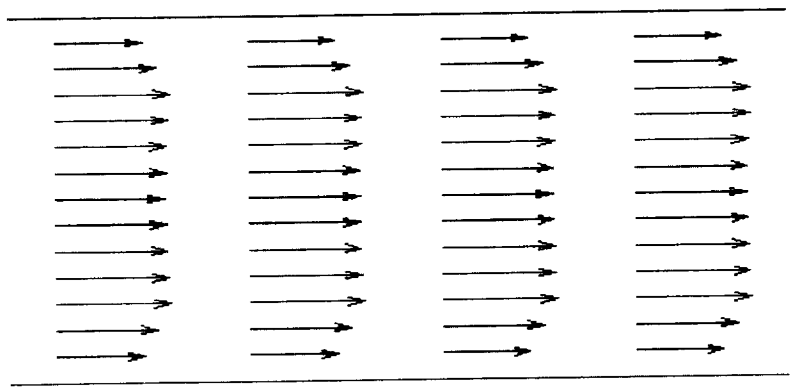


Figure 2.21: Manifest peak velocity profiles. $V_a = 160$, $R_e = 10$, $\delta/L = 2.6\text{E}{-3}$.

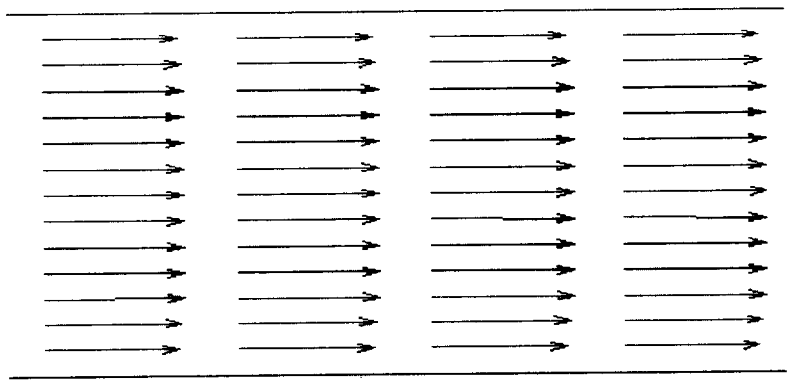


Figure 2.22: Manifest peak velocity profiles. $V_a = 640$, $R_e = 10$, $\delta/L = 6.6\text{E}{-4}$.

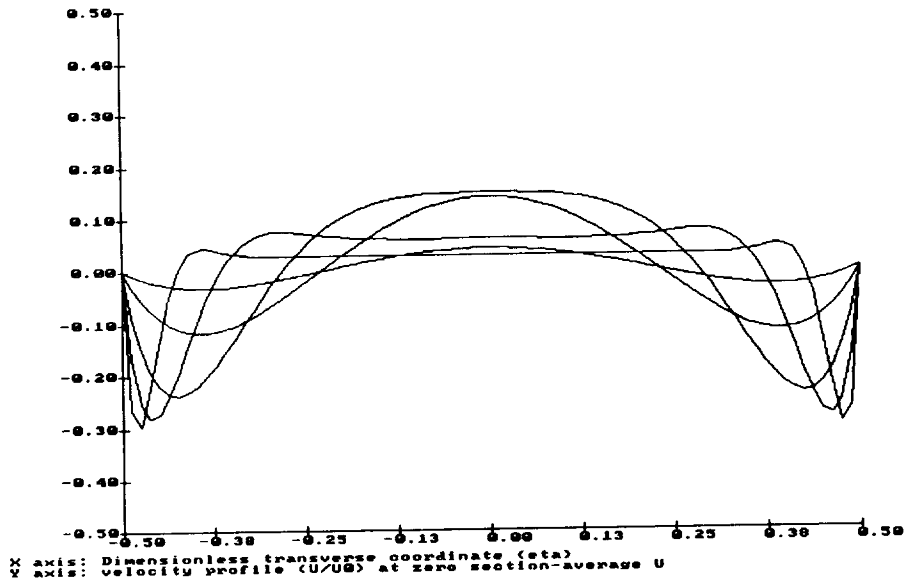


Figure 2.23: Theoretical velocity profiles at the instant of section-average flow reversal. For V_a = 10, 40, 160, 640 and 2056. The inertial core gets bigger and the boundary layer thinner with increasing V_a.

Figures 2.24 through 2.27 show the Manifest mid-domain solutions at the instant of section-average flow reversal. For better visibility, the velocities in these plots are scaled up by a factor of 7.5 over those in the previous peak-velocity plots. Therefore, any errors are also magnified. For all except the last case, small circles are superimposed on the middle column of arrows to indicate where the tip of the arrow would be for the exact solution. The $V_a = 640$ case is way off. I think Manifest is nearing an acoustic resonance on this one. The dimensional frequency is 1273 Hz while the organ-pipe half-wave frequency for the 240 mm domain length is 2080 Hz. Disregarding the last case, agreement between the exact solution and Manifest is not too bad. Even the last solution is not *wrong*, it is just that the incompressible-flow analytic solution knows nothing of acoustical resonance. Actually, it is good that Manifest is capable of spotting this sort of thing.

Velocity Profile Development

Not considered at all by most analytic solutions is the matter of entry length. For steady laminar tube flow, Kays and London ([10] Fig. 6-23, p. 138) present

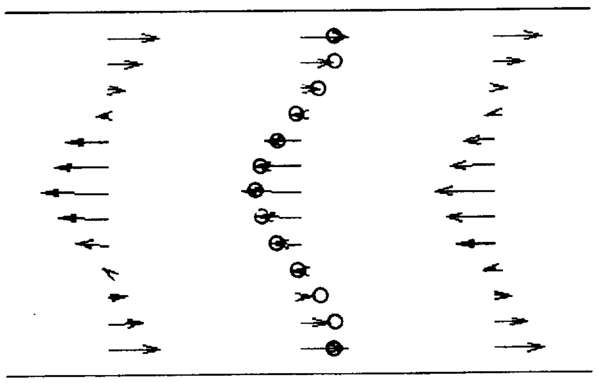


Figure 2.24: Manifest flow-reversal velocity profiles. Small circles indicate exact solution. $V_a = 10$, $R_e = 10$, $\delta/L = 4.2E{-}2$.

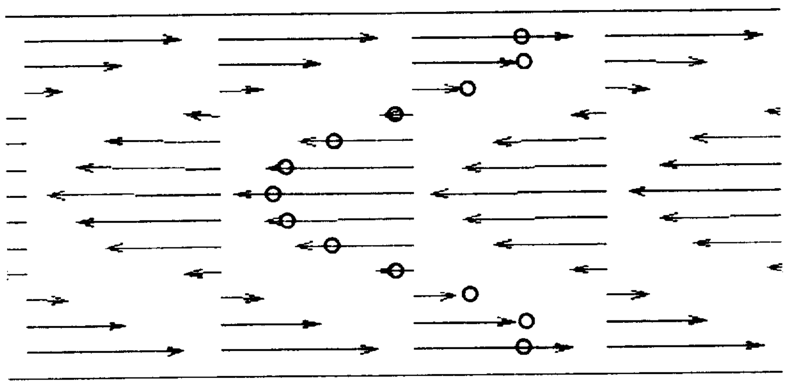


Figure 2.25: Manifest flow-reversal velocity profiles. Small circles indicate exact solution. $V_a = 40$, $R_e = 10$, $\delta/L = 1.0E{-}2$.

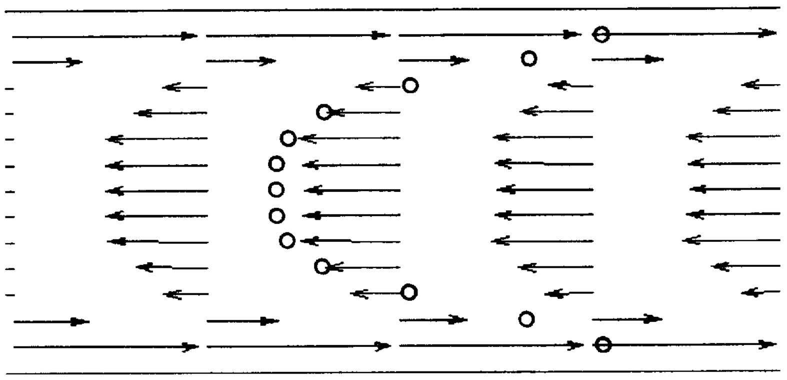


Figure 2.26: Manifest flow-reversal velocity profiles. Small circles indicate exact solution. $V_a = 160$, $R_e = 10$ $\delta/L = 2.6E{-}2$.

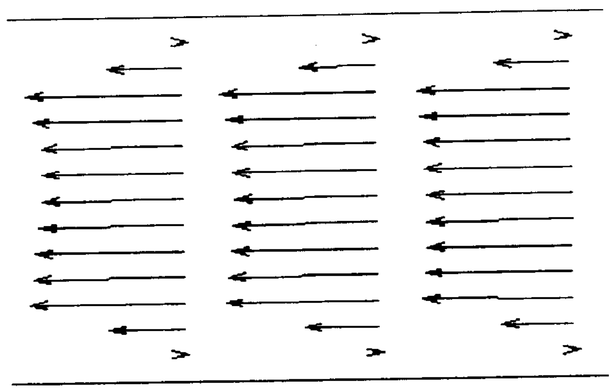


Figure 2.27: Manifest flow-reversal velocity profiles perturbed by acoustic resonance. $V_a = 640$, $R_e = 10$ $\delta/L = 6.6\text{E}{-4}$.

data suggesting that flow requires an entry length x_e of about

$$\frac{x_e}{d_h} \approx \frac{R_e}{100} \tag{2.23}$$

before the wall shear stress (friction factor) is within 25% of its fully developed value. Certainly we expect this result to hold in oscillating flow for low Valensi numbers. But, it is not so clear what we should expect for high Valensi numbers. Intuitively, the dimensionless groups that characterize the entry length are Valensi number, and Peak Reynolds number. The ratio of tidal amplitude to hydraulic diameter δ/d_h is, clearly, also relevant, but it is just proportional to the quotient of R_e and V_a.

The obvious way to get some insight is to conduct a series of computational experiments where R_e and V_a vary independently over suitable ranges. Figure 2.28 shows peak velocity profiles over, roughly, the first half of the computational domain where peak Reynolds number is 250 in all cases but Valensi number ranges from 10 to 160. It is difficult to say exactly, but it appears that, for all plots, flow is pretty much developed by about the fifth column of arrows (about twice the plate spacing or one hydraulic diameter). The important observation is that, to the resolution of the grid, entry length seems independent of Valensi number. The jury is still out on whether this conclusion will withstand closer scrutiny.

Figure 2.29 shows developing velocity profiles for peak Reynolds number ranging from 10 to 1000 with a Valensi number of 10 in all cases. The plot at R_e = 1000 shows a curious instability in the flow: a small oscillation superimposed on the main flow. I am not sure how much of this is real and how much is computational artifact. At any rate, at $R_e = 1000$, flow seems to be developing for most of the computational domain. A good guess for the entry length x_e in this case would be about 5 hydraulic diameters. Assuming entry length is

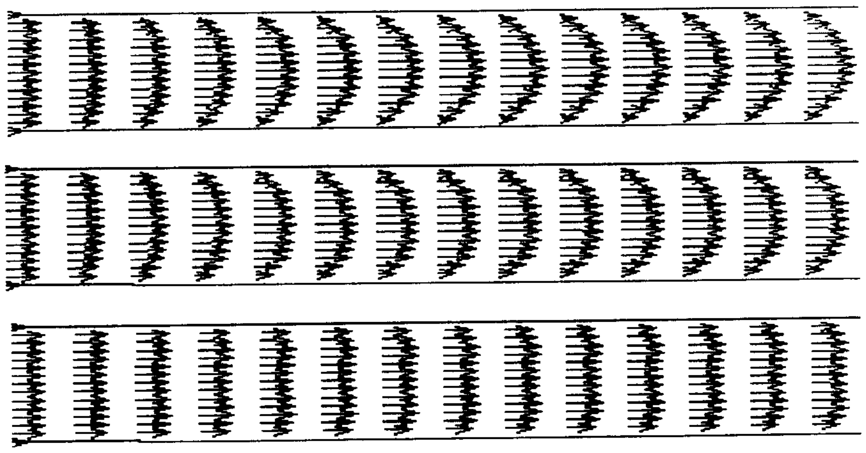


Figure 2.28: Developing peak velocity profiles. $R_e = 250$, $V_a = 10$, 40 and 160.

linear with peak Reynolds number, it looks as if a reasonable approximation to the entry length is

$$\frac{x_e}{d_h} \approx \frac{R_e}{200} \tag{2.24}$$

Not too much different from the Kays and London data for steady tube flow. At least part of the difference may lie with the fact that Manifest is simulating flow between parallel plates — not tube flow. But the major reason is, no doubt, due to my subjective interpretation of what constitutes a *developed* velocity profile.

2.4.2 Gas-to-Wall Heat Transfer

This last series of problems grew out of a desire to model gas-to-wall heat transfer in a situation similar to one that might be found in an actual stirling engine. According to Seume and Simon [14], a significant number of stirling heat exchangers operate with tidal amplitude ratios δ/L on the order of 1, peak Reynolds numbers on the order of 10,000 and Valensi numbers on the order of 100. Tidal amplitude and Valensi number were no problem for Manifest. But I had to limit peak Reynolds number to about 1000 to avoid flow instabilities. Also, I thought that realistic L/d_h values (typically about 100 in stirling heat exchangers) would make for a long skinny computational domain in which it would be difficult to see what was going on. So, I opted for a relatively short and fat L/d_h.

I set up Manifest to model a heat exchanger with a 300K wall subject to sinusoidally oscillating flow entering each end at 290K. The computational domain comprised parallel plates of length 10.0 mm spaced 1.0 mm apart. I divided the domain into three parts: an 8.0 mm long central region with two 1.0 mm regions on either side. I set up an isothermal 300K wall boundary condition

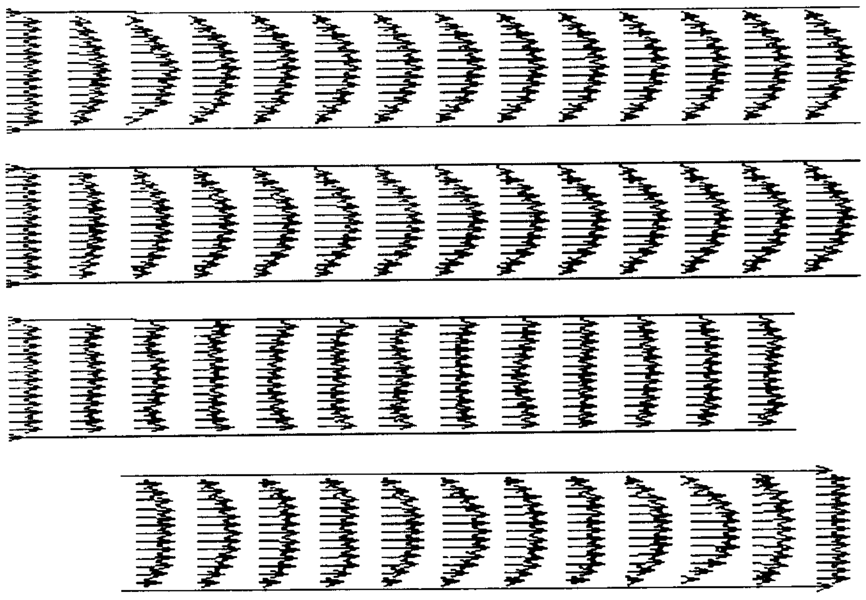


Figure 2.29: Developing peak velocity profiles. R_e = 10, 100, 1000, V_a = 10. The R_e = 1000 case is folded to show the entire computational region.

in the central region. Both outer regions had linear wall temperature profiles smoothly connecting the 290K inlets to the 300K central region. This was necessary because of the way Manifest determines its inflow temperature boundary conditions (from the wall temperatures). Inflow temperature was always 290K but outflow temperature was extrapolated from the interior solution in the usual manner of Manifest. Flow was prescribed as spatially-uniform at the two inlets. The expected result was that heat conduction across the thermal boundary layer near the wall would heat the gas. In equilibrium, this heat would be advected out the two ends by the flow (enthalpy transport). Agreement between the enthalpy advected out the ends and the net wall-to-gas heat transfer was to be a check on the degree of convergence in Manifest.

My goals was to compare these results with those produced by a one-dimensional model using a steady-flow correlation for heat transfer. I chose to use the GLIMPS computer simulation [7] for this because I had no exact analytic solutions available. I don't think they exist. Also, GLIMPS is commercially available software and comparisons with it and other models are always welcome.

Because of the planned GLIMPS comparison, I did not want to introduce any fictitious viscosity multipliers into Manifest to get the dimensionless groups to come out right. Instead, I decided to do any necessary tuning by adjusting either pressure or frequency. Initially I set pressure at 25 bar which I thought was a fairly typical value. But, this forced frequency to be relatively low — on the order of a few Hertz. This caused problems.

Like most solvers of partial differential equations, Manifest has stability limits. In the case of compressible fluid dynamics solvers, a classical measure of numerical instability is the Courant number $N_c = c\Delta t/\Delta x$ where c is the sonic velocity, Δt is the time step and Δx is the physical grid spacing. Explicit methods fare well with N_c below about one. Manifest does much better, allowing N_c to get as high as several hundred before running amok. But, there are limits. With Δx relatively small, in the present computational domain, and Δt relatively large, at the low frequency, Manifest went unstable. One fix would have been to just increase the number of time steps per cycle (presently about 16 to 18), but this would have been costly in terms of computer time. There was a better way. I reduced pressure instead.

As a result, the pressures of the simulations in this section are somewhat lower than normal (0.25 and 2.5 bar) and the frequencies somewhat higher (431 Hz). But, this is why we have dimensionless groups. Since R_e, V_a and δ/L are all where they should be, things like absolute pressure and frequency are not important. I expect that future researchers who attempt to use Manifest to model realistic stirling heat exchangers will encounter similar problems. They can solve them the same way.

Results and Comparisons

I chose to model two cases. The first case has a peak Reynolds number of 100 and a Valensi number of 5 (Pressure = 0.25 bar, frequency = 431 Hz). Under these conditions, the steady-flow assumptions in GLIMPS should be fairly accurate, and one would expect GLIMPS and Manifest to agree fairly well. The second case has a peak Reynolds number of 1000 and a Valensi number of 50 (Pressure = 2.5 bar, frequency = 431 Hz). According to thermal entry length tables for laminar parallel-plate flow, also in Kays and London ([10] Fig. 6-17, p. 134), the steady-flow Nusselt number is enhanced by a factor of about 1.8 over the fully-developed value under these conditions. Since GLIMPS does not have an entry length correction factor, the two programs should begin to differ for this case. Both cases have a tidal amplitude ratio δ/L of 1.0, very close to typical stirling practice. Figure 2.30 shows the simulated velocity profiles for the two cases. Note the blunter profile (higher V_a) and longer entry length (higher R_e) of the second case.

Tables 2.3 and 2.4 give the comparison between GLIMPS and Manifest. Viscous dissipation numbers are about the same in both cases and the two programs seem to agree fairly well. Apparently, Valensi number is not yet high enough to significantly perturb the velocity boundary layer. But even if it had, I would still expect good agreement because GLIMPS includes a Valensi-number adjustment (based on the exact solution for oscillating incompressible flow) for laminar friction factors. Heat transfer agreement is good in the first case but

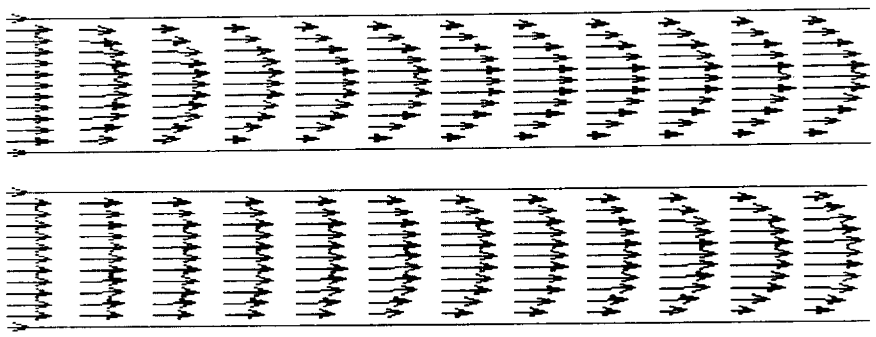


Figure 2.30: Developing peak velocity profiles for the two gas-to-wall heat transfer problems. Top case: $R_e = 100$, $V_a = 5$ and $\delta/L = 1$. Bottom case: $R_e = 1000$, $V_a = 50$ and $\delta/L = 1$.

	GLIMPS (W)	Manifest (W)	Ratio
Wall-to-Gas Heat Transfer	28.1	25.1	0.9
Viscous Dissipation	10.1E-1	7.8-1	0.8

Table 2.3: Manifest and GLIMPS comparisons at $R_e = 100$, $V_a = 5$, $\delta/L = 1$.

Manifest predicts about 1.4 times more heat transfer than GLIMPS for the second case. This is as expected, although, somewhat less than that predicted from the steady-flow thermal entry length correction factor mentioned above. Evidently, Vanlensi number is high enough to have some affect. GLIMPS has no Valensi-number-dependent film heat transfer correction factors built in because of the lack of an exact analytic model.

The effect of oscillating flow on the growth of the thermal boundary layer can be seen in figures 2.31 and 2.32. Shown there are temperature contour plots at the instant of mean-flow reversal (from left to right) and 60 and 120 degrees beyond. (Degrees in the phasor sense: one full period is 360 degrees.) The second half-cycle is just the mirror image of the first half. Figure 2.31 is closest to steady flow, but even so, the effects of flow oscillation are apparent. The temperature boundary layer at the flow reversal point shows significant remnants from the previous half cycle. As flow advances in the rightward direction, a more normal boundary layer develops, but it continues to change over the course of the cycle. Figure 2.32 is qualitatively different. The lumpy boundary layer

	GLIMPS (W)	Manifest (W)	Ratio
Wall-to-Gas Heat Transfer	93.1	132.9	1.4
Viscous Dissipation	9.9E-1	8.4E-1	0.8

Table 2.4: Manifest and GLIMPS comparisons at $R_e = 1000$, $V_a = 50$, $\delta/L = 1$.

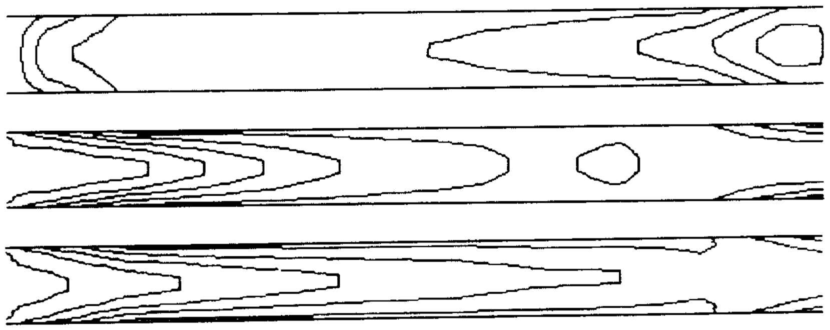


Figure 2.31: Thermal boundary layer development for the first heat transfer problem. $R_e = 100$, $V_a = 5$ and $\delta/L = 1$. Top to bottom: 0, 60 and 120 degrees after left-to-right flow reversal. The contour interval is 2.25K

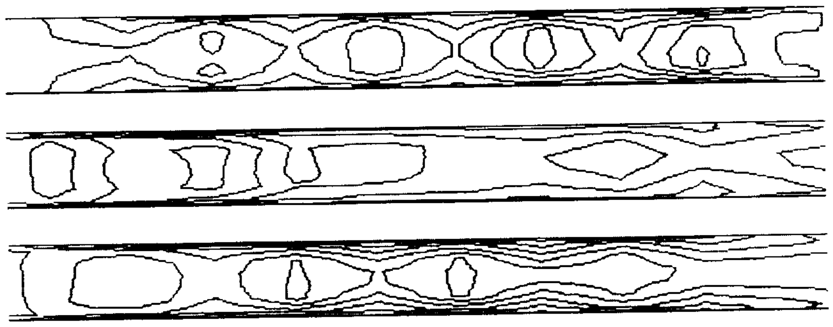


Figure 2.32: Thermal boundary layer development for the second heat transfer problem. $R_e = 1000$, $V_a = 50$ and $\delta/L = 1$. Top to bottom: 0, 60 and 120 degrees after left-to-right flow reversal. The contour interval is 2.25K

is, presumably, due to the beginnings of flow instability at the high Reynolds number. Averaging out that effect, the boundary layer is much thinner than before. This is due to the blunter velocity profile. As before, the thermal boundary layer continues to develop over the entire course of the cycle.

2.5 Epilogue

So these are the sample problems. Although it was a struggle to produce them with my tenuous, slow and unreliable communications link to the NASA Lewis Cray computer, I am pleased with the results. I think they have borne out Hamming's motto (quote at chapter beginning). I, at any rate, have seen new light.

While Manifest is already a powerful investigative tool, it could be better. Should there be a next version of Manifest, the first-priority new feature should be, in my mind, a turbulence model. High Reynolds number modeling in heat exchangers and piston-cylinders could lead to important insights into areas which dramatically affect stirling performance. Another priority, would be some sort of adaptive control over the computational grid. It would be nice to have a means to concentrate the grid around points of interest and avoid problems near concave corners of the computational domain. A less pressing need, but fair thought nonetheless, would be the ability to generate curved boundaries for the computational domain. After all, not all boundaries in stirling machines are straight. And the icing on the cake would be a fully three-dimensional Manifest. However, the current generation of computers — even supercomputers — is too slow for this. Assuming the required hardware improvements are forthcoming, it would be fairly straight-forward to convert the basic Manifest computational algorithm to 3-D, but the graphical input and output routines would require major re-thinking. So then, if the present version of Manifest is a milestone for stirling flow modeling, the end of the road is still a ways off.

This ends the general-interest section of this report. Many readers will want to stop here. Those who want to actually use Manifest should read the following user's guide. Those who want to delve even deeper can go so far as to read the subsequent chapters on theory.

Part II

User's Guide

This part of the report assumes that you have access to the Manifest software and want to learn to use it. Even without the software in hand, the overview of chapter 3 might be of general interest. To make best use of the tutorial in chapter 4 and the reference guide in chapter 5, you will need the PC version of Manifest. If you have access to the mainframe version of Manifest, then you will want to read chapter 6.

Chapter 3

Overview

Manifest comprises three program modules — MFSETUP, MF and MFOUT — which perform the input, main simulation and output respectively. The three modules communicate by reading and writing disk files. Using three separate modules in sequence gives flexibility to Manifest: For example, you may choose to run the computationally-intensive main simulation on a Cray rather than a PC. Or you may review the output at any time without the need to re-run the main simulation.

MFSETUP contains graphics-based input routines that allow you to interactively specify the shape of the computational region and its associated numerical parameters. The source-level data files read and written by MFSETUP are named *username*.DAT where *username* is up to you. In addition to specifying input data, MFSETUP is also responsible for calculating the curvilinear computational grid on which the solution will be obtained. When finished, the source-level and derived variables produced by MFSETUP are written to two disk files named _MF1.TXT and _MF2.TXT in preparation for the simulation stage. The reason the source file is duplicated in _MF1.TXT is to avoid synchronization problems in case the source file is modified before the output file is created.

The main simulation, MF, reads the files _MF1.TXT and _MF2.TXT and time-steps its finite-difference solution algorithm for a prescribed number of iterations, or until convergence. MF comes in two versions: a PC version and a mainframe version. The PC version displays the solved velocity field after each time step to indicate that progress is being made in what otherwise might appear to be a dead machine. Execution time may be several minutes (or even hours!) depending on the size of the problem. The mainframe version runs in batch mode and produces no visible output until finished. The PC version writes the solved gasdynamic variables in files _MFSTEP.nnn — one for each output time step. The file extension nnn is the time-step number. The mainframe version writes the solved variables in a single file _MFSTEP.ALL which is essentially the union of the _MFSTEP.nnn files. Each version also writes a small status file _MFSTATU.TXT containing information about the last step simulated and the state of convergence. And sometimes, each writes a file _MFSTEP.DV containing solved-variable increments for the most recent integration time step. Manifest's solution algorithm needs this data in case you want to re-start the solution from where it left off. The _MFSTEP.DV file is only written in the event there is more than one integration time step per output time step.

The output program, MFOUT, reads the _MF1.TXT, _MF2.TXT, _MFSTEP.nnn and _MFSTATU.TXT files produced previously, and then enters an interactive output mode which allows you to examine the results of the simulation in your own way. Output file _MFSTEP.ALL, produced by the mainframe MF module, must be converted to _MFSTEP.nnn format before running MFOUT. There are two ways to view the output graphically: in terms of a mass-flux vector field or in terms of contour plots for pressure or temperature. You can scale the graphics view-port to home-in on regions of interest and also change vector lengths and contour intervals. Using arrow keys, you can play back the simulation step-by-step in a sort of moving animation of the solution. In addition there are several categories of tabular information that you can examine: energy integrals in regions, energy fluxes through boundaries and Fourier Coefficients. In each of these categories you are presented with a menu of possible sub-choices and then asked to select the region, boundary or point of interest by point-and-clicking the mouse. While this is going on, MFOUT is constantly updating a diskfile named OUTPUT with a log of any tabular output displayed in interactive mode. OUTPUT also contains a human-readable listing of the input variables, but since it is an ASCII file, no graphical information.

3.1 System Requirements

The recommended system is:

- IBM-XT or -AT equivalent or Intel 80386-based computer.
- MS-DOS operating system; version 2.0 or higher.
- Floating-point math coprocessor.
- 512–640k RAM
- Graphics display capability
- Memory-resident graphics display printing utility
- Hard disk
- Mouse pointing device
- Link to a Cray Supercomputer

A math coprocessor is all but essential for running floating-point bound programs such as Manifest. It speeds up execution by a factor of about 15. Without one, the program will limp along badly.

Because Manifest displays everything in bit-mapped graphics mode, some sort of graphics display capability is required — the higher the resolution the better. Manifest is fairly flexible on this point since it is able to automatically adapt itself to a wide range of common display devices.

Distributed with Manifest is a memory resident driver PRTSCRN.EXE which allows you to dump graphics-screen images to an IBM-compatible dot matrix printer. You must first load PRTSCRN (by typing it at the DOS command line), after which you activate it by pressing the Shft-PrtSc key combination. For wider printer support, including Hewlett Packard compatible laser printers, third-party software is available for the job. One such package is the GRAF-PLUS utility marketed by Jewell Technologies[1].

A mouse is needed to support graphics editing functions. You can, in theory, get by without one: All mouse operations can be done via the keyboard. However, not having a mouse will slow you down a lot and may drive you crazy. The Microsoft and Mouse Systems mice are both supported.

A hard disk is recommended since the Manifest programs read and write some large data files. Also the programs themselves take up a lot of space. You should have one or two megabytes of storage available to keep Manifest related files. If you really had to, you could probably get by without a hard disk but you might wind up doing a lot of floppy disk swapping.

Having a link to a Cray supercomputer means that you will be able to run the computationally-intensive MF module in a supercomputing environment (provided it is installed there) thereby speeding things up and increasing your ability to resolve small-scale phenomena in the solution. This capability is completely optional: Manifest can run just fine within a PC environment. Moreover, with the rapidly developing status of microcomputers, the need for a supercomputer is becoming less and less important. However, for the moment, the lack of a supercomputer means that you will bump into speed and memory limitations somewhat sooner than otherwise.

3.2 Files

The following files are distributed with the PC version of Manifest

[1]Jewell Technologies, 4740 44th Ave. Southwest, Suite 203, Seattle WA 98116, tel: (206) 937-1081.

MFSETUP.EXE	Input program module
MF.EXE	Main simulation module
MFOUT.EXE	Output program module
PRTSCRN.EXE	Graphics screen dump utility
DSPLYATR.TXT	Display attribute specifications
*.FNT	Font files

If you plan to use the program a lot, you should copy the above files to a hard-disk directory specifically set up for Manifest: \MANIFEST, for example.

The display attribute file DSPLYATR.TXT is an ASCII file which tells Manifest how to set its display colors. It contains three lines, each with the name of a variable followed by an integer value. The variables are:

PenColNum	Color of text and other objects drawn on the screen
BackgroundColNum	Background color
BorderColNum	Color of the border outside the normal display region

You may change the numeric values using any text editor. Just be sure not to delete any of the spaces between a variable name and its value. The first twenty columns are reserved for the variable name. The color palette is based on the IBM EGA coding scheme. Table 3.1 tells you what to expect for various numeric values.

Except for the font files, all files that Manifest reads or writes are in standard ASCII format. This means you can examine (or even modify) them with any text editor. More importantly, it means that you can pass files back and forth to another computer system without difficulty. This comes in handy when running the MF module on a supercomputer. However, there is a price: File input and output are relatively slow. It takes longer to interpret an ASCII file than, say, a binary file containing an exact memory image of the data. As disk access times get faster and faster, this should become less of a problem.

Value	Color	Value	Color
0	black	8	dark gray
1	blue	9	light blue
2	green	10	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	brown	14	yellow
7	white	15	intensified white

Table 3.1: Decimal color codes for DSPLYATR.TXT file.

Chapter 4

Tutorial

4.1 MFSETUP Tutorial

The preeminent feature of MFSETUP is its interactive graphics-oriented input editor. Using operations similar to those in PC drawing or CAD programs, MFSETUP facilitates the job of building up a complex two-dimensional computational domain from a stack of quadrilateral-shaped sub-regions. MFSETUP gives you considerable freedom to choose the order of events you will use to define a computational region and to correct mistakes as they occur. In current computer-science jargon, MFSETUP is pretty much a *modeless* editor, which means that you control the program rather than the other way around.

The purpose of this tutorial is to introduce you to the input editor without emphasizing the physics of two-dimensional flow problems. Accordingly, a simple but pointless example is worked out that illustrates all the major features of the input editor but is of little interest beyond that. However, once you have worked through the tutorial, you should begin to see how MFSETUP can be used to set up problems of practical value.

You will find important additional information in the input variable glossary of section 5.1.5. By understanding the variables defined there (all accessible through the input editor) you will begin to develop a feel for exactly what is possible with Manifest.

4.1.1 Starting

Make sure the default drive or directory is the one containing the Manifest program files. Type **MFSETUP** then press return. The first thing MFSETUP does is to automatically determine the display adapter and mouse driver your computer is using. If all goes well, MFSETUP will correctly identify your system, pause briefly, then continue. If something goes wrong — and MFSETUP gets confused about your system — there is currently no way to set things straight. Sorry.

If things have gone well so far, you will see the display shift over to graphics mode and the following prompt line appear:

```
Enter data file name (.DAT assumed)
```

Type SAMPLE then press return. MFSETUP will search for SAMPLE.DAT on the default drive. Not finding it, the prompt line changes to

```
New file:  enter name of file to start from, or <ret>
```

Just press the return key and MFSETUP will start from its default input values. If instead you had entered the name of an existing input file (from a previous session) MFSETUP would have read that file to obtain starting point data values.

4.1.2 Main Menu

Now displayed on the top line of the display screen is the main menu for MFSETUP.

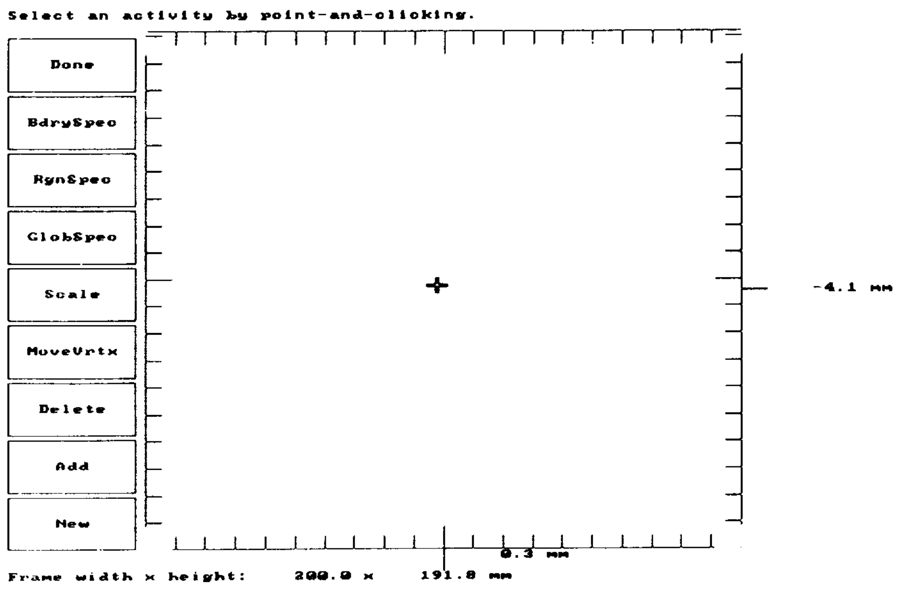
```
sample.DAT: F1 edit, F2 save then compile, F3 just save, Esc done
```

The name of your source-level data file appears first. When you are finished editing data and choose to save it, this is the disk file to which your data will be written. The four options presented in this menu are selected by pressing one of the function keys F1, F2 or F3, or the Escape key. Pressing the Escape key allows you to exit MFSETUP without updating your input file. You might want to choose this option if you have screwed things up terribly and want to try again. Pressing F3 key allows you to update your input file and return to the main menu in the event you get nervous about a power glitch in the middle of a long editing session; or if you simply want to protect existing data changes in case you are about to screw things up terribly in the near future. Pressing the F1 or F2 keys will set into motion the two major sub-procedures of MFSETUP.

4.1.3 Graphics Editor

At this point, press F1 to enter the graphics-based input editor. Your display screen should now look like figure 4.1. Variations in appearance are possible depending on your monitor and graphics display adapter.

The blank rectangle with the "+" (cursor) in the middle is the drawing area in which you will soon create a computational domain. The computational domain is the area in which the gasdynamic variables will eventually be solved; think of it now as just a picture inside the drawing area. The drawing area is just a window through which you look at the computational domain. The status line at the bottom of the screen tells you the dimensions of the drawing


Select an activity by point-and-clicking.

Frame width x height: 200.0 x 191.8 mm

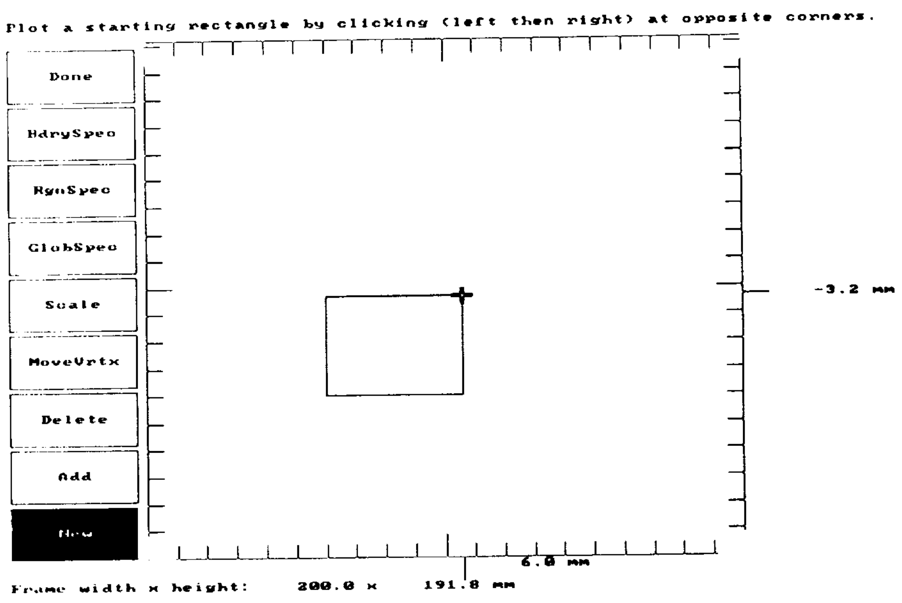
Figure 4.1: Starting graphics-editor display screen.

area. You will see later how the drawing area can be scaled and shifted about to accommodated the actual physical dimensions of your computational domain.

Along the bottom and right edges of the drawing area are displayed the coordinates of the cursor. Try moving the mouse around a bit. The cursor moves with the mouse and its coordinates are updated as rapidly as possible. The tic marks along the edges of the drawing area are located at decade subdivisions of the basic meter unit. These tic marks automatically adapt to the scale and position of the drawing area.

Using the mouse to point-and-click is the way things are intended to be done in the graphics-editor. But do not despair if you have no mouse: the keyboard can be used to do everything the mouse can. As you might expect, the arrow keys move the cursor around — one step at a time. Moreover, pressing the shift key at the same time as an arrow key (or activating Caps Lock) increases the distance moved. To click mouse buttons without a mouse, simply press the F9 or F10 keys instead. These F9 key is equivalent to the left mouse button and F10 is equivalent to the right button. Even for mouse users, the arrow keys can come in handy when the cursor must be moved exactly vertically or horizontally or in small increments. Keep this in mind when you are trying to exactly position vertices of the computational domain later on.

The prompt line at the top of the screen is continually updated to give you some clue as to what you might want to do next. This particular prompt suggests that you select an activity by point-and-clicking. For example, move


Plot a starting rectangle by clicking (left then right) at opposite corners.

Frame width x height: 200.0 x 191.8 mm

Figure 4.2: Creating the first region.

the cursor now to the box labeled New and click the right or left mouse button. The New box lights up and you are ready to define the first region of the computational domain.

4.1.4 First Region

Referring to figure 4.2 as a guide, position the cursor at any vertex of the small rectangle shown in the drawing area (not visible yet) and click the left mouse button. Then move the cursor to the opposite vertex and click the right mouse button. Note that after clicking the left button and before clicking the right button, a rectangle is dragged about on the screen with the cursor. This allows you to see what you're about to create before fixing it. To start over just click the left mouse button again and you will erase the old rectangle and begin dragging a new one.

The rectangle you have just drawn serves as the starting region for your computational domain. Eventually It might represent a regenerator matrix or a manifold. Later, you will be able to change its shape by moving its vertices around or change its properties by modifying numerical data parameters associated with it. In effect you will be able to fill it with a porous material; or leave it empty. You will be able to change it boundaries into active flow inlets or keep them as solid walls. And you will be able to stack together several regions together to form complex shapes.

Before continuing, be sure your starting rectangle has plenty of room between its right edge and the limit of the drawing area, as shown in figure 4.2.

4.1.5 More Regions

When you have finished drawing the starting region move the cursor to the box labeled **Add** and click either mouse button. You are now ready to add regions to the initial region. Regions are added to the starting region by tagging squares onto its left or right boundary. Additions to the top or bottom boundaries are not allowed.

Move the cursor to a position just right of the right boundary of the starting region and click a button. A square appears, attached to the starting rectangle's right boundary. Had you positioned the cursor on the other side of the starting rectangle and clicked a button, a square would have appeared there. Nothing would have appeared had you clicked a button while the cursor was above or below the starting rectangle, or while it was too far to the left or right.

Each time a region is added, its left or right boundary in turn becomes a boundary where yet another region can be added. In principle, there is no limit to the number of regions that can be built up in this way. However, each region carries with it a fairly large memory overhead so there are practical limits.

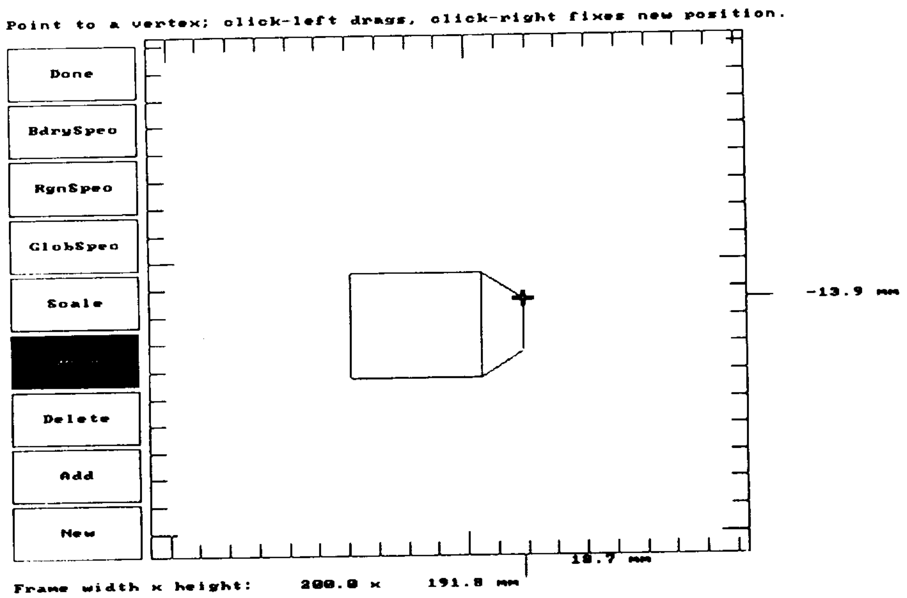
The designations "left" and "right" used above are somewhat artificial since the orientation of the computational domain may be altered in the course of editing. However, once you understand the basic idea — that the computational domain is built up of a linear sequence of elementary regions — you should have no trouble adding regions.

4.1.6 Changing Shapes

At this point you should have a rectangle adjoined by a square displayed on your screen. Now move the cursor to the box labeled **MoveVrtx** and click a button. You are ready to move vertices around.

Move the cursor to the upper right vertex of the right-most region — the square — and click the left mouse button. Any further cursor movement will drag a line segment between the selected vertex and the cursor. If you do not see this drag-line, re-position the cursor around the upper right vertex — this time more accurately — and click the left mouse button again. Once you are dragging a line from the selected vertex, move the cursor to the new vertex position shown in figure 4.3 and click the right mouse button. The selected vertex snaps to the new position. If you want to adjust the vertex position, just move the cursor and click the right button again. In general, clicking the left mouse button selects a vertex for relocation while clicking the right button re-positions the selected vertex.

Now, using the same basic sequence, move the lower right vertex of the right-most region to the position shown in figure 4.3. You do not have to get things



Point to a vertex; click-left drags, click-right fixes new position.

Frame width x height: 200.0 x 191.8 mm

Figure 4.3: Moving Vertices.

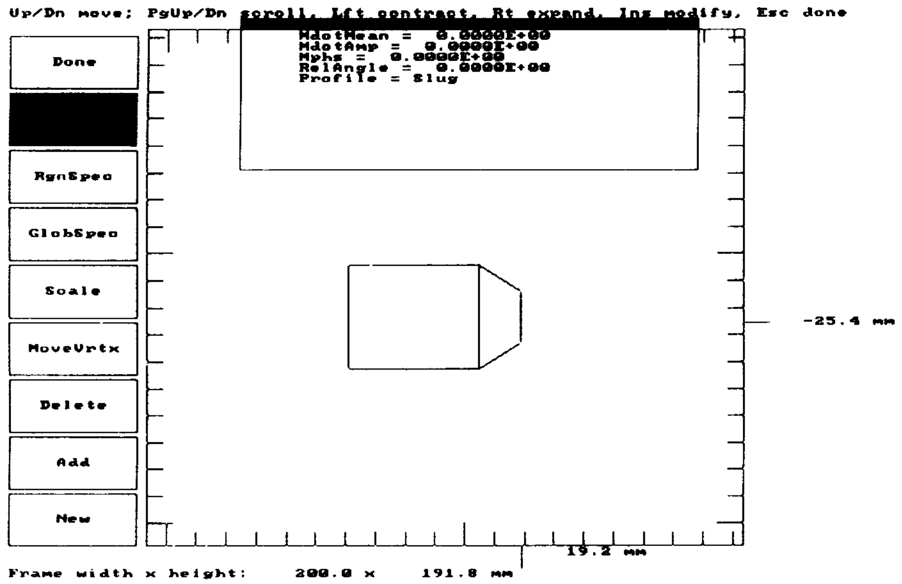
exact, just try to wind up with something that looks similar to the figure so that the rest of this tutorial will make sense.

4.1.7 Numerical Data

The selection boxes labeled BdrySpec, RgnSpec and GlobSpec are used for entering numerical data for the exterior boundaries of individual regions, the interiors of individual regions and the computational domain as a whole. In the cases of the BdrySpec and RgnSpec options, there are several possible data sets. To select one you move the cursor to the appropriate region boundary or interior then click either mouse button. The format for entering data is the same in each case. The basic ideas are these: A window, containing a list of variables to be modified, is opened in the display. The variables, defined in section 5.1.5, are initially displayed with default values which you are free to change. Selection and modification of variables is performed with keyboard — not mouse — operations.

By working through a few examples you will quickly get the hang of entering numerical data. First, Move the cursor to the box labeled BdrySpec and click a mouse button. Following the prompt message that appears at the top of the screen, move the cursor to the right-most boundary of the computational domain and click the mouse again to select that boundary for numerical data modification. A window now opens up and the display looks like figure 4.4.

ORIGINAL PAGE IS
OF POOR QUALITY



```
Up/Dn move; PgUp/Dn scroll, Lft contract, Rt expand, Ins modify, Esc done
        MdotMean =   0.0000E+00
        MdotAmp =  0.0000E+00
        Mphs =  0.0000E+00
        RelAngle =   0.0000E+00
        Profile = Slug
Frame width x height:    200.0 x    191.8 mm
```

Figure 4.4: A typical numerical-data entry screen.

Numerical input data is always displayed in outline form with indenting used to separate logical hierarchies. Internally, numerical data lists are stored in tree form. Branches in numerical data trees can have either a down branch or a right branch. A down branch is displayed in the same column and signifies a branch of similar hierarchy. A right branch is indented to the right and signifies a lower hierarchical level. A plus in front of a displayed name indicates the presence of a right branch. Note that the root branch boundary is preceded by a plus. A plus is needed since right-branch subtrees may be initially hidden from view and you should at least be reminded that they are there.

Colons after names indicate label branches that have no associated input variable but rather serve as signposts and storage locations for user comments. For example, a comment can be entered for the branch boundary. In general, all branch modification occurs at the cursor position — currently at boundary. Press the Ins key (usually below the numeric keypad) to enter input mode. The prompt line immediately changes to

```
Enter comment
```

Type **tube inlet** then press return. The original null comment after +boundary: on the display screen is now replaced with the new comment. Note that the cursor automatically moves down one line after a value is modified; this feature simplifies sequential data entry.

The cursor can be moved up and down using the arrow keys of the numeric keypad. Try it. The PgDn and PgUp keys give added control over the display;

they cause the display page to scroll down or up by one line leaving the cursor fixed. Cntrl-PgDn and Cntrl-PgUp scroll the display down or up an entire page. Of course with the small display now showing, the scroll commands are of little use. But in some cases, the display can overflow the available screen at which point scrolling control becomes valuable.

Move the cursor to MdotAmp (Amplitude of mass flow rate across boundary) by pressing the appropriate arrow key then press Ins to modify the current value. The prompt line now changes to

```
Enter real value
```

If you simply press Esc now you will return to the main environment without having changed anything. Try it. Now go back up to MdotAmp by pressing the up arrow key then press Ins again to actually modify MdotAmp. Enter 1.0E–2 and press return. If you enter a number outside the range suggested in the prompt you will have to re-enter the number until it falls in the correct range.

Pressing the Ins key to enter numerical data quickly becomes awkward. So, there is a shortcut that works for numerical data input only. Just start entering the number and you will automatically enter input mode for the variable at the cursor position. You'll find that this feature allows you to enter variable lists very efficiently.

This is enough BdrySpec data entry for now. Changing MdotAmp from 0.0 to 1.0E–2 has the effect of converting the formerly solid boundary into an active inlet. Press the Escape key now to leave data entry mode. Any changes you made are stored in memory but the disk file is not yet updated.

The display screen should now be restored to its former state except that arrows appear on the right boundary of the computational domain; see figure 4.5. In general, arrows indicate the direction and amplitude of the mass flux per unit area for an active inlet. The length of the arrow is not an absolute measure of mass flux amplitude. Rather, it corresponds to a relative scale that depends on the largest mass flux amplitude for the entire exterior boundary of the computational domain.

The numerical input editor still has a few more features. To see these, move the cursor to the selection box labeled Rgnspec and click. Then move the cursor to the interior of the left region of the computational domain and click again. When the numerical-input window comes up, move the cursor down to the variable Matrix using the down arrow key. Matrix is known as a decision branch; it determines whether the region contains a matrix (TRUE) or is empty (FALSE). In general, a decision branch gives rise to two or more possible sub-branches of the input tree; the branch selected depends on the actual value stored in the decision branch. Another example of a decision branch variable is Profile in BdrySpec.

Decision-branch values are modified using the right and left arrow keys. Press the Ins key, the right arrow key and then return to change the value of Matrix from FALSE to TRUE. There now appears on the display a new list of

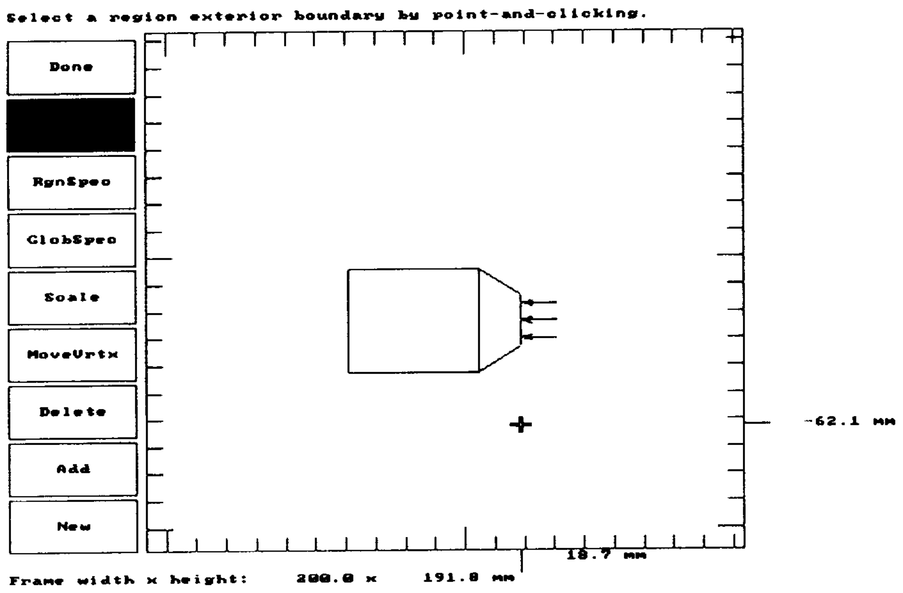

Select a region exterior boundary by point-and-clicking.
Done
RgnSpec
GlobSpec
Scale
MoveVrtx
Delete
Add
New
-62.1 mm
18.7 mm
Frame width x height: 200.0 x 191.8 mm

Figure 4.5: An active inlet noted by mass flux arrows.

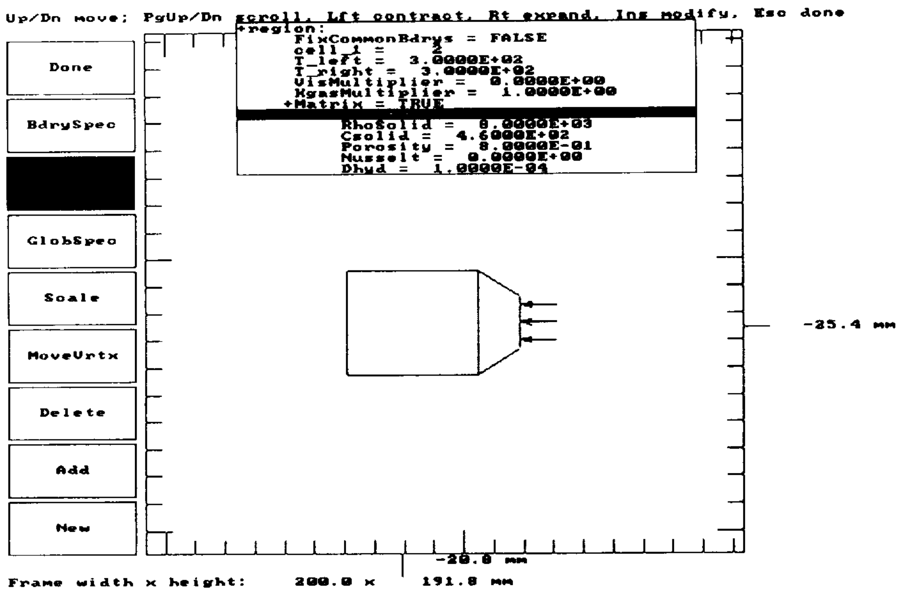
variables — previously missing — displayed after **Matrix** and indented to the right. See figure 4.6. These are variables that only make sense when a matrix is present in the region.

While you are looking at the current display, you might as well modify some variables to make things more interesting later on in the output part of this tutorial. Move the cursor to the following variables and change them as indicated:

- **VisMultiplier** = 0.0
- **Porosity** = 0.80
- **Dhyd** = 1.0E–4
- **Cergun[1,1]** = 150
- **Cergun[2,2]** = 150

You need do this for the left-most region only.

It is possible to hide (or re-display) right-branch subtrees from view. For example, move the cursor back up to the **Matrix** variable and press the left arrow key. The right-branch subtree previously displayed under **Matrix** disappears from view. It is still in memory — you just can't see it. To re-display the hidden right branch just press the right arrow key or the return key. Keep this in mind whenever you see a variable preceded by a "plus" with nothing



```
Up/Dn move; PgUp/Dn scroll, Lft contract, Rt expand, Ins modify, Esc done
+region:
     FixCommonBdrys = FALSE
     cell_1 =    2
     T_left =  3.0000E+02
     T_right =  3.0000E+02
     VisMultiplier =  0.0000E+00
     KgasMultiplier =  1.0000E+00
    +Matrix = TRUE
         RhoSolid =  8.0000E+03
         Csolid =  4.6000E+02
         Porosity =  8.0000E-01
         Nusselt =  0.0000E+00
         Dhyd =  1.0000E-04
Frame width x height:   200.0 x   191.0 mm
```

Figure 4.6: Hidden variables under the Matrix decision branch.

displayed indented to the right. This situation will arise if you press escape now to enter graphics-edit mode then re-enter numerical-input mode by clicking the mouse with the cursor in the same region as before. The display comes up with Matrix = TRUE preceded by a "plus" but without the right subtree displayed.

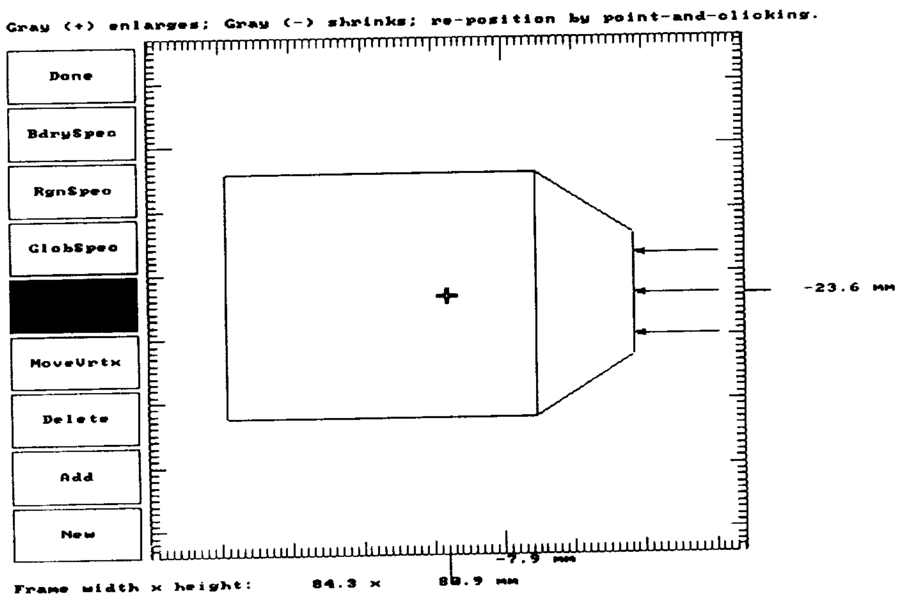
There is one more data entry required to avoid difficulty in the solution stage later on. Enter GlobSpec and change the frequency variable to 500. This rather fast frequency has the effect of reducing the default time step to the stable range for this particular problem. More on this time-step limitation in section 5.2.1.

4.1.8 Scaling

At some point you will want to magnify or reduce your picture, or shift it's position in the drawing area. This is done with the Scale function.

Move the cursor to the selection box labeled Scale and click. Pressing the gray "+" key increases the size of the picture while pressing the gray "–" key reduces the size. Try it. The physical coordinates of the computational domain are not affected by all this, the limits of the drawing area are all that change. (There is no way to scale the physical coordinates of the computational domain other than with the MoveVrtx function.) Note that the status line at the bottom of the display changes to reflect the changing dimensions of the drawing area.

Shifting the position of the computational domain is a two-step process. First move the cursor to the point you want to be in the center of the display rectangle. This point can be anywhere on the boundary or interior of the computational



Gray (+) enlarges; Gray (-) shrinks; re-position by point-and-clicking.

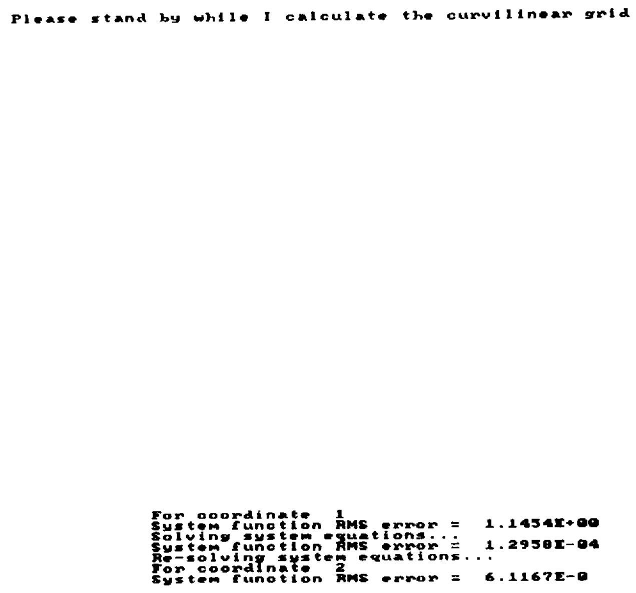
Frame width x height: 84.3 x 80.9 mm

Figure 4.7: Scaling the display rectangle.

domain. (It can even by on the exterior although this ordinarily will not make much sense.) Then click either mouse button. The computational region now shifts so the selected point lies at the center of the display rectangle. It is not possible to rotate the computational domain. In preparation for the next stage, press the gray "+" key a few times and shift the display around until it looks about like figure 4.7.

4.1.9 Removing Regions

The right-most or left-most regions of the computational domain can be removed by selecting the box labeled **Delete**. This feature allows you to correct mistakes or edit existing computational domains into new ones. When the **Delete** box is highlighted, clicking the mouse in either the right- or left-most region will remove it from the computational domain — permanently! There is no *undo* function so be careful. Any numerical data entries you made for a deleted region are lost forever. Your only recourse to a mistaken delete is to reconstruct the missing region from scratch. For now, if you want, you can first select **Add** to add an extra region, then delete it. But before continuing, be sure your display continues to look something like figure 4.7.



```
Please stand by while I calculate the curvilinear grid
```

```
For coordinate  1
System function RMS error =  1.1454E+00
Solving system equations...
System function RMS error =  1.2958E-04
Re-solving system equations...
For coordinate  2
System function RMS error =  6.1167E-0
```

Figure 4.8: Calculating the curvilinear grid.

4.1.10 Return To Main Menu

At this point you have seen everything that the input editor can do and are ready to move on. Move the cursor to the box labeled Done and click. The prompt line changes to the main menu.

```
sample.DAT: F1 edit, F2 save then compile, F3 just save, Esc done
```

Press F2 and MFSETUP will first save your input data to disk, then calculate the curvilinear coordinate grid that satisfies the boundary conditions specified by your computational domain.

4.1.11 Curvilinear Coordinate Generation

If all goes well your display will look like figure 4.8 for a few seconds while it calculates the curvilinear grid. Each of the two coordinates is solved in a separate iterative process. The "System RMS error" is an overall measure of the degree to which convergence has been obtained — the smaller the better. When the value gets below the tol input parameter convergence is deemed satisfactory and iteration stops. The message "Solving System equations..." refers to the solution of a large set of linear equations involving a matrix triangularization step followed by a back-substitution step. The message "Re-solving system equations" refers to the case where only the right-hand side of the equation system has changed, eliminating the need for the matrix triangularization step.

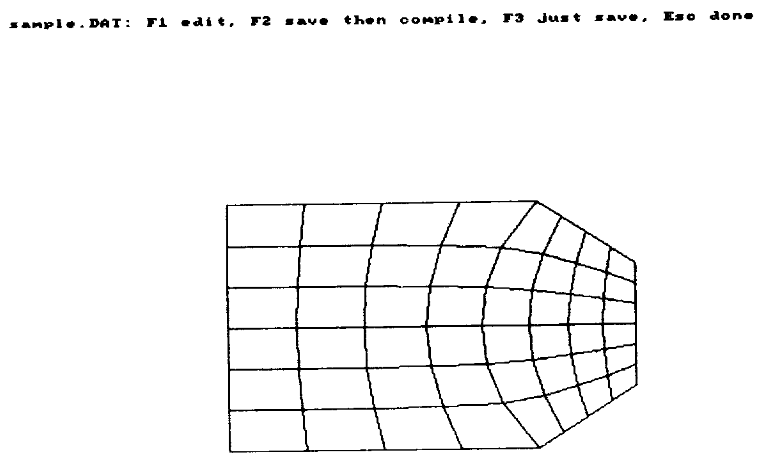

sample.DAT: F1 edit, F2 save then compile, F3 Just save, Esc done

Figure 4.9: The solved curvilinear grid.

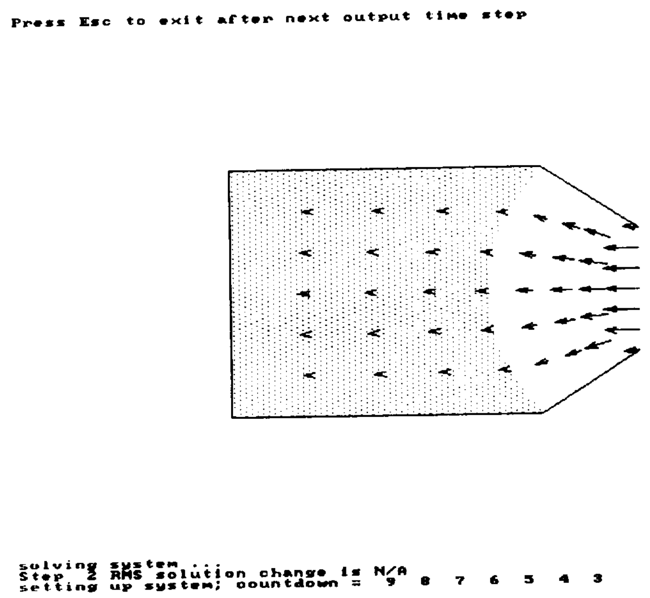
When the program is finished calculating, the display changes to look like figure 4.9, the output files _MF1.TXT and _MF2.TXT are written to disk and control returns to the main menu. The computational grid is presented as a net of coordinate curves, the intersections of which are the *nodes* of the grid where the gasdynamic variables are solved. The displayed size of the grid corresponds to the size of the computational region displayed in the graphics editor. You can affect the size here by using the Scale function there.

At this point you should see the main menu displayed at the top of the screen. You can now re-enter the graphics editor to play around some more or simply press the escape key to return to the operating system. If you have followed this tutorial more-or-less faithfully, you should now have a viable input file on the disk for the main simulation to follow. Pressing Escape will allow you to run that simulation next.

4.2 MF Tutorial

This section describes only the PC version of MF. The mainframe version is conceptually similar except that it runs in batch mode and produces no visual display during execution — see section 6.

At this point you should have on disk, the two MF input files: _MF1.TXT and _MF2.TXT, which you produced by running the input program MFSETUP as described in section 4.1. MF always gets its input from the most recent run



```
Press Esc to exit after next output time step
```

```
solving system ...
Step  2 RMS solution change is N/A
setting up system; countdown =  9  8  7  6  5  4  3
```

Figure 4.10: A typical display during MF execution.

of MFSETUP.

Type MF at the DOS prompt to start execution. The program begins by reading its input files and initializing data structures. It then starts time-stepping through the solution until you halt execution or it decides on its own that it has attained convergence to a steady state cycle. This may take quite a while.

Figure 4.10 shows a typical display screen during execution. The main display window shows the mass flux vector field for the most recent time step. The size of the computational domain in the window is the same as it was in edit mode of MFSETUP. Another window at the bottom of the screen gives a blow-by-blow account of the solution.

The left region of the computational domain is shaded because it contains a porous matrix (Matrix = TRUE). The matrix right boundary is curved because it follows the coordinate lines of the computational grid.

The line that reads: "Setting up system; countdown = 9 8 7 ..." is there mainly to assure you that your computer is doing something productive while you wait. A number is printed for each row or column of the computational grid while MF is evaluating partial derivatives for the system Jacobian matrix.

The line "solving system ..." tells you that the system of linear equations, whose coefficients are the Jacobian matrix, is being solved. For more information about the solution algorithm see section 9.

The line that reads: "Step 2 RMS solution change is N/A " tells you that time step 2 has just been solved but insufficient steps have so far been taken for MF to compare the present solution with that of one cycle previous. Had the simulation been running for more than one cycle, the N/A would be replaced by a number giving the root-mean-square difference of the present dimensionless solution variables with those of the same time for the previous cycle. The time-step numbers are displayed modulo node_T — where node_T is the number of time steps per cycle — so that they repeat at intervals of exactly one cycle period. Should the RMS solution change ever fall below the input variable GasdynamicTol, then MF will assume convergence has been achieved and stop time-stepping.

As the top line of the display tells you, you can temporarily stop execution at the next time step by pressing the escape key. No matter when you press it, execution will continue until the current step has been solved. Once you've stopped execution, you can resume again at the place you left off by simply re-starting MF. This feature is handy if you want to examine the output in more detail with program MFOUT, or, if you just want access to your computer for any reason. This restart feature is available until the next time you run MFSETUP to produce new input files. If you are really pressed for time and want to halt execution immediately, just re-boot your computer. Since, MF updates the solution variable files _MFSTEP.nnn and status file _MFSTATU.TXT after each time step, you will only lose information from the pending time-step. And, you will still be able to restart MF from that point.

In a nutshell, that's it. There is very little interaction required with MF. You need only monitor the display from time to time to check the progress of the solution. If something does go awry, then you will need to go back to MFSETUP and revise your source data. For more information on what might go wrong with MF see section 5.2.

For purposes of this tutorial you should let the sample problem set up in section 4.1 run for at least one cycle before halting execution. Then you will be able to make best use of the following MFOUT tutorial. However, if you cannot bear to wait that long, you need only complete only a few time steps before stopping. In this case, you will be able to examine most of — but not all of — the features of MFOUT.

4.3 MFOUT Tutorial

The output module MFOUT is an interactive program designed to let you view Manifest's solution in your own way. At this point you should have run MF for at least a few time steps, as discussed in section 4.2, in order to have produce some _MFSTEP.nnn files containing the raw solution data.

Now type MFOUT at the DOS prompt to start the program. MFOUT spends a little time at the beginning reading the source-level data file _MF1.TXT, the compiled-level data file _MF2.TXT, the _MFSTEP.nnn files and the status file _MFSTATU.TXT; with the bulk of its initialization time spent reading and processing the _MFSTEP.nnn files. The status line at the bottom of the display tells you which time step is currently being processed.

MFOUT produces an output disk file named OUTPUT which is overwritten each time you run the program. OUTPUT contains a log — in ASCII text format — of your output session. The first part of the output file is a copy of your source-level input data in human readable form; after that it depends on your interaction with MFOUT. All textual information displayed on the screen is copied to the output file but no graphical information. When finished with MFOUT, you can examine the OUTPUT listing with any text editor or word processor, or print it on any printer. More information on the format for OUTPUT is provided in section 5.3

When you see the screen shown in figure 4.11 you are ready to begin interactive output mode. Just as in MFSETUP, the idea here is to point-and-click with the mouse to make activity selections from the boxes at the left of the display. The line in the display that reads,

```
RmsChange > GasdynamicTol; Solution has not yet converged.
```

is a reminder that although a full cycle has been simulated, convergence has not yet been attained. Your display may or may not show this line. Another possibility is,

```
This output is for an incomplete cycle.
```

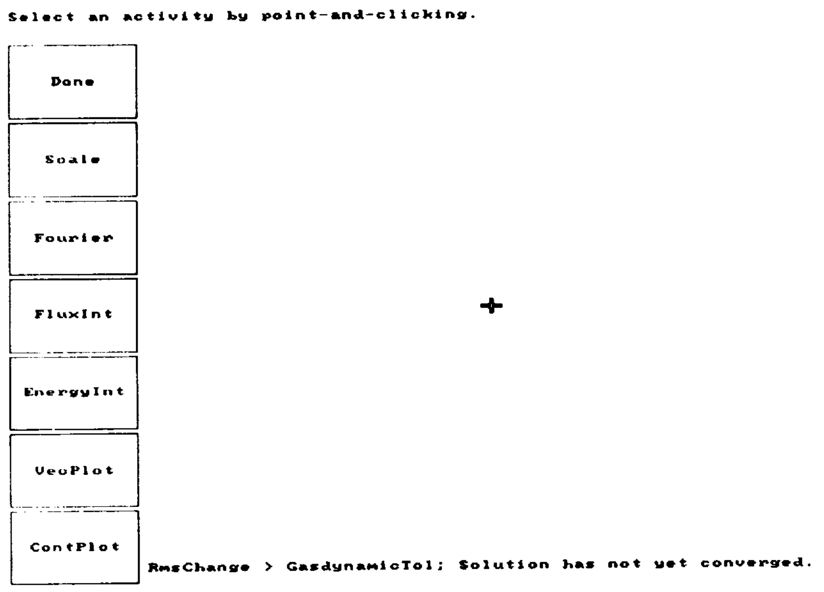
which is a reminder when there are insufficient _MFSTEP.nnn files produced to make up a complete cycle.

4.3.1 Scaling

Several of the options in MFOUT start out by drawing the basic solution domain outline. In each case, the relative size and position of the domain within its display window are the same. The **Scale** function allows you to control the part of the solution domain you wish to view and the magnification at which you view it.

Scale works exactly as in MFSETUP. The gray "+" key increases the apparent size of the picture while the gray "–" key reduces it. Pointing-and-clicking the mouse at a point within the display window shifts that point to the center.

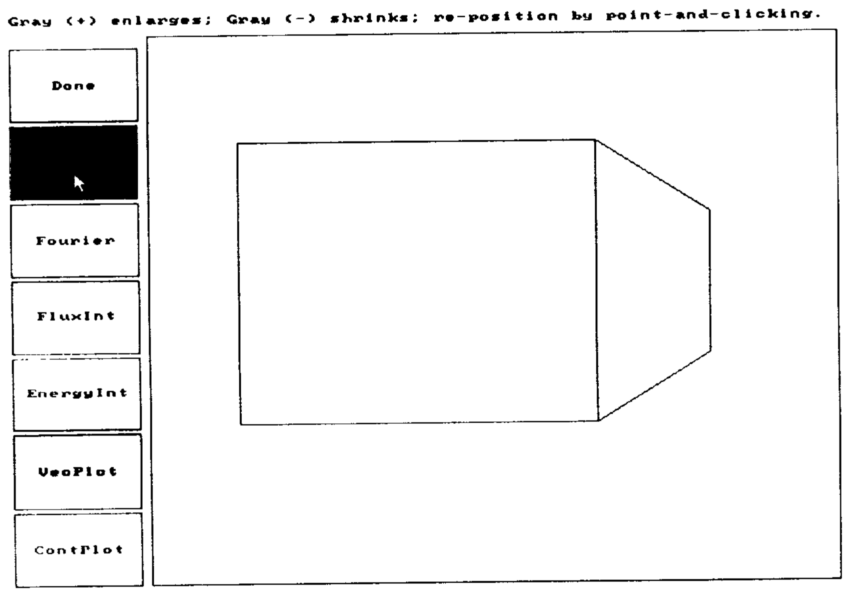
Point-and-click the SCALE box at this time to enter scale mode. You should now see the outline of the computational domain in the same proportions (relative to the display window) as you left it in MFSETUP. You are welcome now to play around until the picture looks good to you. Figure 4.12 shows a typical screen appearance in scale mode.



Select an activity by point-and-clicking.

RmsChange > GasdynamicTol; Solution has not yet converged.

Figure 4.11: The starting screen for interactive output in MFOUT.



Gray (+) enlarges; Gray (-) shrinks; re-position by point-and-clicking.

Figure 4.12: The display appearance during Scale.

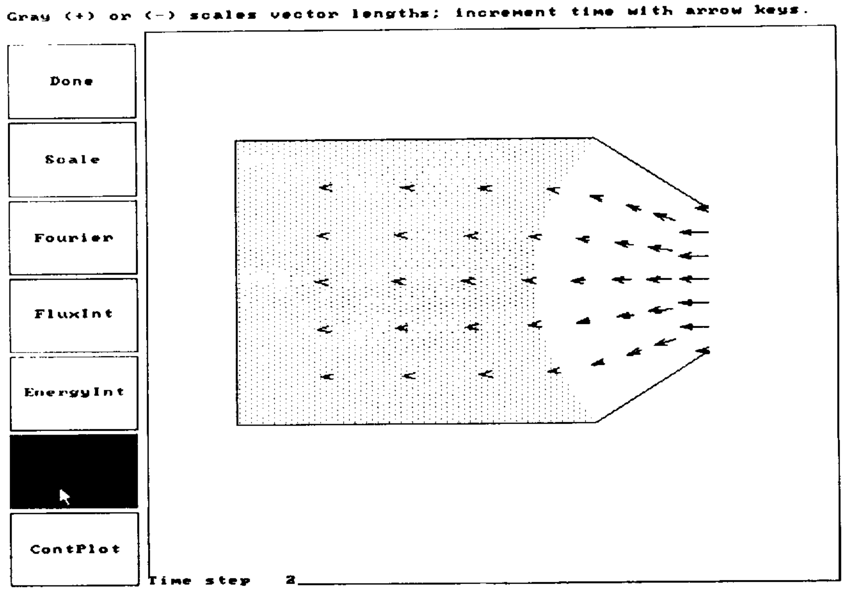

Gray (+) or (-) scales vector lengths; increment time with arrow keys.
Done
Scale
Fourier
FluxInt
EnergyInt
ContPlot
Time step 2

Figure 4.13: A typical vector plot display

4.3.2 Vector Plots

Move the cursor to the box labeled VecPlot and click the mouse. You are now in vector-plot mode which produces plots of mass flux per unit total area for each time step. Pressing the right or left arrow keys increments or decrements the time-step number by one. Try it. The current number appears at the bottom left of the display window. Pressing the gray "+" or "−" keys scales the lengths of the arrows. Play around with this until you get the picture most pleasing to you. Figure 4.13 shows a vector plot for our sample problem. If for some reason you want to home-in on a particular part of the computational domain, just use the Scale function to re-position the display window, then return to VecPlot.

4.3.3 Contour Plots

Move the cursor to the box labeled ContPlot and click the mouse. You are now in Contour-plot mode which can produce pressure or temperature contours for each time step. Since there is more than one option you are presented with a menu list which you can scroll through with the up and down arrow keys. Try it. Now scroll up to the top selection Pressure and press return.

You are now ready to view pressure contour plots. The right and left arrow keys change the time-step number as before. Press the right arrow key several times to get to a time step where the velocities are significant — about time step 4. The gray "+" and "−" keys control the spacing of the contours. Press

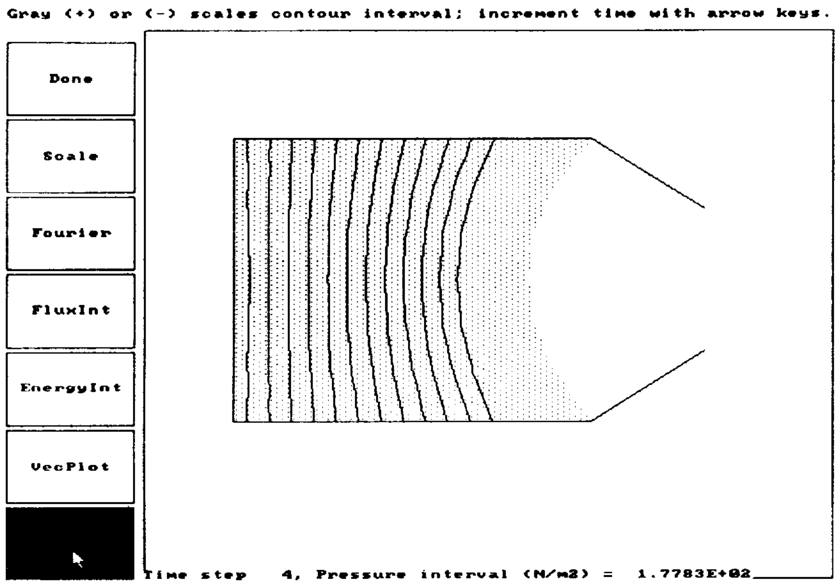

Gray (+) or (-) scales contour interval; increment time with arrow keys.
Done
Scale
Fourier
FluxInt
EnergyInt
VecPlot
Time step 4, Pressure interval (N/m2) = 1.7783E+02

Figure 4.14: A typical pressure contour plot display

the gray "–" key a few times until you see a pleasing number of contour lines. Figure 4.14 shows a contour plot for our sample problem.

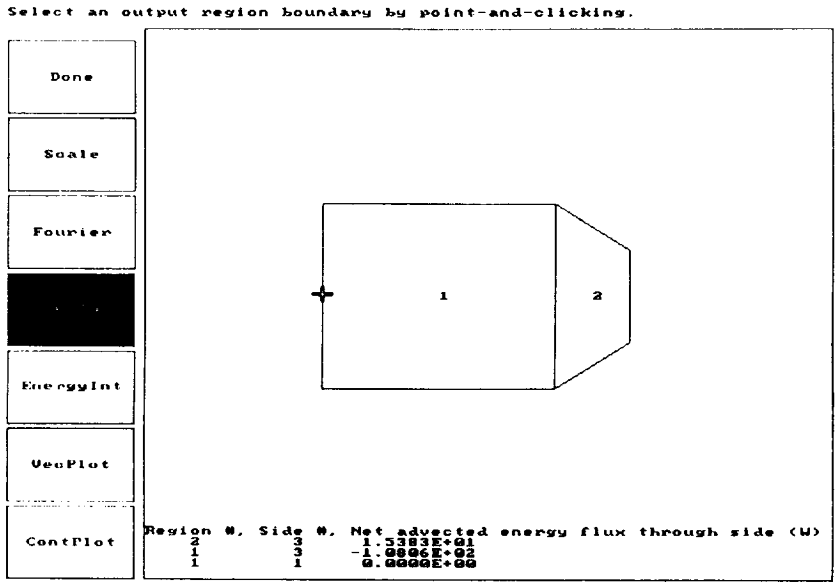
4.3.4 Energy Flux Integrals

This option tabulates cyclic time integrals of energy flux passing through the various region boundaries within the computational domain. Although you are allowed to select this option without having simulated a complete cycle, Manifest assumes that flux integrands are zero for any time step not represented by a _MFSTEP.nnn file.

Move the cursor to the box labeled FluxInt and click the mouse. You are presented with a menu of several possible selections. Press return to select the Advected Energy Flux option. The display screen should now look about like figure 4.15. Point-and-click the cursor on the right most boundary of the region labeled 2. Immediately the net cyclic advected energy (enthalpy + kinetic energy + viscous stress work) through the selected boundary is displayed. You can point-and-click on any region boundary. The results are displayed and also recorded in the OUTPUT file.

4.3.5 Energy Volume Integrals

This option tabulates cyclic time integrals of net energy change within the various regions of the computational domain. Again, for an incomplete cycle, Man-


Select an output region boundary by point-and-clicking.

```
Region #, Side #, Net advected energy flux through side (W)
    2          3     1.5383E+01
    1          3    -1.0806E+02
    1          1     0.0000E+00
```

Figure 4.15: A typical energy flux integral display

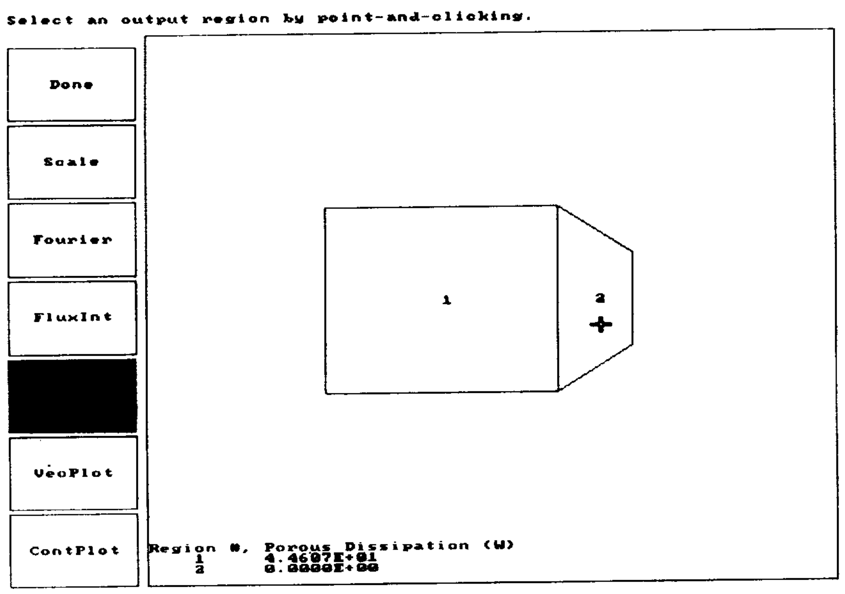
ifest assumes that energy integrands are zero for any time step not represented by a _MFSTEP.nnn file.

Move the cursor to the box labeled EnergyInt and click the mouse. You are presented with a menu of several possible selections. Press return to select the Porous Dissipation option. The display screen should now look about like figure 4.15. Point-and-click the cursor in the interior of the region labeled 1. Immediately the net dissipation due to porous flow resistance in that region is displayed. You can point-and-click in either region interior. The results are displayed and also recorded in the OUTPUT file.

4.3.6 Fourier Coefficients

This option tabulates Fourier coefficients of several fundamental variables at the various nodes of the computational grid. You are not allowed to select this option unless you have simulated a complete cycle. Manifest will beep at you if you try.

Move the cursor to the box labeled Fourier and click the mouse. You are presented with a menu of several possible selections. Press the down arrow key three times then press return to select the rho V_1 option (horizontal component of mass flux per unit total area). The display screen should now look about like figure 4.17. Point-and-click at any computational node of your choice (the intersections of the coordinate curves). Immediately the mean value of rho


Select an output region by point-and-clicking.

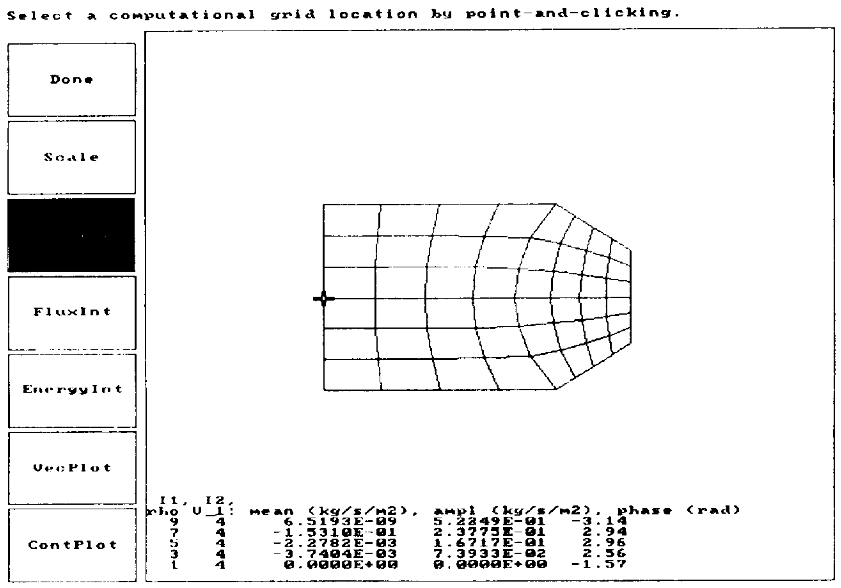
```
Region #, Porous Dissipation (W)
    1       4.4607E+01
    2       0.0000E+00
```

Figure 4.16: A typical volumetric energy integral display

V_1 for that node together with the first-harmonic amplitude and phase are displayed. You can point-and-click to any node in the computational domain. The results are displayed and also recorded in the OUTPUT file.

4.3.7 When Finished

This concludes the MFOUT tutorial. You are welcome now to play around with the output program. When you are finished, move the cursor to the box labeled Done and click the mouse. This will return you to DOS. You can now print the OUTPUT file or examine it with a text editor. Remember, if you re-start MFOUT you will start over with a completely new OUTPUT file.



```
Select a computational grid location by point-and-clicking.
```

```
 I1,  I2,
rho  U_1:  mean (kg/s/m2),  ampl (kg/s/m2),  phase (rad)
  9    4      6.5193E-09     5.2249E-01   -3.14
  7    4     -1.5310E-01     2.3775E-01    2.94
  5    4     -2.2782E-03     1.6717E-01    2.96
  3    4     -3.7404E-03     7.3933E-02    2.56
  1    4      0.0000E+00     0.0000E+00   -1.57
```

Figure 4.17: A typical Fourier coefficient display

Chapter 5

Reference Guide

5.1 MFSETUP Reference Guide

5.1.1 File Selection

After starting MFSETUP, one or more of the following initial prompts will appear:

```
Enter data file name (.DAT assumed)
```

Type the name of an existing data file (omitting the DAT file extension) or a new name if you are starting from scratch. Your data entries will be stored in this file. If the name of a new file is entered, the prompt line changes to

```
New file:  enter name of file to start from, or <ret>
```

Type the name of an existing file you wish to use for initial variable values in your new file. If you simply press the return key, default initial values will be used. If the starting-value data file you typed in cannot be found on the default directory, the prompt line changes to

```
File not found:  enter name of file to start from, or <ret>
```

This is a second-chance version of the previous prompt. If you need to examine the disk directory at this point, press return, then exit to DOS by pressing Escape.

5.1.2 Main Menu Commands

The main menu is in force whenever you see the following prompt displayed at the top of the screen.

```
source.DAT: F1 edit, F2 save then compile, F3 just save, Esc done
```

The name of your source-level data file (here source.DAT) appears first on the prompt line. When you see the main menu, the following commands are available:

F1 Enters the graphics editor.

F2 Sets the following sequence into motion:

- Saves your source-level data file to disk.
- Calculates the curvilinear computational grid.
- Writes the output file _MF2.TXT to disk.
- Returns to the main menu.

F3 Saves the source-level data file to disk then returns to the main menu.

Esc Returns to DOS. Saves nothing to disk.

5.1.3 Graphics Editor Command Summary

Pressing the F1 key from the main menu invokes the graphics editor. The editor is designed so that point-and-click operations with the mouse perform most of the operations. A menu of activities or functions is presented along the left edge of the screen. Each activity is represented by a word in a rectangular box. Moving the cursor to the interior of an activity box and clicking the right or left mouse button, selects that activity for execution.

Moving the Cursor

Depending on its location, the cursor is represented on the display by either a "+" or an arrow. Normally the cursor *tracks* the motion of the mouse as you move it. If you don't have a mouse, or if you need finer control than the mouse provides, the keyboard arrow keys can also be used to move the cursor. The unadorned arrow keys move the cursor about one pixel in the indicated direction. Holding a shift key down while pressing an arrow key increases the size of the step.

Mouse Button Substitutes

The right and left mouse buttons are needed frequently in the graphics editor. If you have no mouse, use the F9 key as a substitute for the left button and the F10 key as a substitute for the right button.

New

Use this function to create the first region of the computational domain. You may create exactly one rectangle, positioned anywhere in the drawing area. When the cursor is located in the drawing area the mouse buttons work as follows:

- **Left Click**: Fixes the first vertex of the rectangle.
- **Right Click**: Fixes the opposite vertex.

After a left-click, but before a right-click, a rectangle is dragged along with the cursor. You may erase an existing rectangle by clicking the left mouse button again.

Add

Use this function to add regions to the computational domain. You may add squares only to the right- or left-most boundaries of the existing computational domain. To select which one, position the cursor just to the right or left of the computational domain (in the interior of the square to be added) and click either mouse button.

The designations "right" and "left" refer to the orientation of the initial rectangular region. As the computational domain increases in complexity, the boundaries to which regions may be added may be re-oriented. If you think of the computational domain as a topologically distorted stack of rectangles, the boundaries where additions can occur are obvious.

Delete

Use this function to delete regions from the computational domain. You may delete only the right-most or left-most region at a time. To delete a region position the cursor in its interior and click either mouse button. The affect is permanent and any numerical data entered for a deleted region is lost. The only possible way to recoup from a mistaken **delete** is to abort the current edit session and restart from a recently saved data file. To do this, select the **Done** function then press the Escape key from the main menu. This will return you to DOS without updating your source-level data file. Then restart MFSETUP as usual, reading your saved data file for input.

MoveVrtx

Use this function to re-position the vertices of the computational domain. When the cursor is located in the drawing area the mouse buttons work as follows:

- **Left Click**: Selects a target vertex.
- **Right Click**: Fixes its new location.

When a target vertex is selected, moving the cursor drags a line from the target vertex. If you select a vertex but do not see this line, try again: you probably did not have the cursor cross-hairs close enough to the target vertex when you clicked the left button. Once you have successfully selected a target vertex you can experiment with its new location by moving the cursor and clicking the right button repeatedly. Each time you, do the computational region is redrawn with the target vertex at the current cursor position. Once you have fixed the new position of the target vertex, you can select a new target by clicking the left button.

Scale

Use this function to re-position or scale the size of the drawing area. The physical coordinates of computational domain are not affected by scaling. Two keyboard commands are available:

- **Gray +**: Increases the apparent size of the computational domain.
- **Gray −**: Decreases the apparent size of the computational domain.

In addition, clicking either mouse button when the cursor is in the drawing area, re-positions the drawing area so it is centered at the cursor location.

GlobSpec

Use this function to change or examine the values of the numerical data whose scope is the computational domain as a whole. Once in numerical data entry mode, the commands of section 5.1.4 are available. See section 5.1.5 for the meanings of the variables.

RgnSpec

Use this function to change or examine the values of the numerical data applying to the interior of an individual region of the computational domain. Before entering numerical input mode you must first select the region of interest by moving the cursor to its interior then clicking either mouse button. Once in numerical input mode, the commands of section 5.1.4 are available. See section 5.1.5 for the meanings of the variables.

BdrySpec

Use this function to change or examine the values of the numerical data applying to an exterior boundary of an individual region of the computational domain. Once in numerical input mode, the commands of section 5.1.4 are available. See section 5.1.5 for the meanings of the variables.

Done

Use this function to exit the graphics editor and return to the main menu.

5.1.4 Numerical Input Command Summary

Numerical input mode is invoked automatically through the BdrySpec, RgnSpec or BdrySpec functions of the graphics editor.

Numerical data is stored internally in branching linked lists referred to as *trees*. At any given time, some of the branches of the tree might be hidden from view on the display. You may use the commands below to view hidden branches or hide them again. Even if it's currently hidden, the (+) prefix displayed before a branch denotes the presence of a right subtree. You must make sure that all the variable values in hidden subtrees are correct.

Branch names followed by the (:) suffix contain comments which you may enter using the Ins command. Comments are displayed after the (:) and are optional. Branches followed by the (=) suffix are variable names containing values which you *must* enter correctly using the Ins command or usually by simply entering the appropriate number. Variable values are displayed after the (=).

While in numerical input mode, the following commands are available:

Up Arrow Moves cursor up.

Down Arrow Moves cursor down.

Left Arrow Hides display of right subtree originating from branch at cursor.

Right Arrow Displays next hierarchical level of right subtree originating from branch at cursor.

return Same as Right Arrow.

PgUp Scrolls display window up one line.

PgDn Scrolls display window down one line.

Ctrl-PgUp Scrolls display window up one page.

Ctrl-PgDn Scrolls display window down one page.

Ins Enters modify mode for variable (comment) at cursor position

- Follow prompts; normally enter a number, then press return.
- Examples: 10, 2.5, .001, 1.0E−3
- Select enumerated variables with right or left arrow keys, then press return.
- Pressing return immediately will exit modify mode leaving the previous value unchanged if it is within the allowable range.
- Pressing Esc at any time will exit modify mode leaving the previous value unchanged if it is within the allowable range.

0...9 + − . Any one of these keys also enters modify mode as a shortcut for numerical or character string input.

Esc Leaves numerical input mode and returns to graphics editor. Data changes are saved in memory but not yet updated to disk.

Numerical entries made with the numerical keypad must be done in *NumLock* mode. The ordinary shift key will not work because Manifest interprets shft-2, shft-4, shft-6 and shft-8 as shifted arrow keys.

To remove an existing comment, you cannot simply press Ins, then return, because this leaves the old comment unchanged. Instead press Ins, space then return. This will replace the comment with an invisible space.

5.1.5 Glossaries of Numerical Input Variables

MFSETUP leaves it to you to make sure that all the numerical data variables have the correct values for the simulation you want to perform. You must check the variables accessible with the ...Spec functions of the graphics editor as follows:

BdrySpec Check the variables for each active inlet. These are the exterior boundaries of regions where the variable **Mamp** is strictly greater than zero. Active inlets are displayed with arrows. Do not worry about variables for region boundaries that are not active inlets.

RgnSpec Check the variables for each region of the computational domain.

GlobSpec Check these variables for the computational domain as a whole.

BdrySpec Glossary

These variables apply to exterior boundaries of the individual regions of the computational domain. Each boundary has its own variable list.

MdotMean : (kg/s)
$1/\cos(\textbf{RelAngle})\times$ the mean component of mass flow rate across the total region boundary. See equation below.

MdotAmp : (kg/s)
$1/\cos(\textbf{RelAngle})\times$ the amplitude of mass flow rate across the total region boundary. See equation below.

Mphs : (radians)
Phase angle of mass flow rate. See equation below.

NullBeg : (dimensionless)
An auxiliary variable for the **StepWise** velocity profile. The profile is uniform except for a null band beginning at x = **NullBeg** and ending at x

= NullEnd. x is the dimensionless position along the boundary; $x = -1$ at the counter-clockwise-most endpoint and $x = 1$ at the other endpoint. Both NullEnd and NullEnd are restricted to the interval $[-1, 1]$. If NullBeg $<$ NullEnd then the null band is in the middle of the boundary. If NullBeg $>$ NullEnd then the null band is split into two parts at each end with the active flow in the middle.

NullEnd : (dimensionless)
See NullBeg.

Profile : (K)
Specifies the inlet velocity profile. An enumerated selection variable with the following choices:

Slug Uniform flow across inlet.

Parabolic Parabolic profile as in developed steady laminar flow in channels.

seventhPower As in developed steady turbulent flow in channels.

Ramp A linear profile where the slope is provided in the auxiliary variable RampSlope.

StepWise Similar to Slug except a null band may be defined in the middle or at the endpoints of the boundary according to the values of the auxiliary variables NullEnd and NullBeg.

All profiles are normalized so that the net mass flux through the boundary is unaffected.

RampSlope : (dimensionless)
An auxiliary variable for the Ramp velocity profile. The profile form is

$$f = 1 + mx$$

where m is RampSlope and x is the dimensionless position along the boundary. $x = -1$ at the counter-clockwise-most endpoint and $x = 1$ at the other endpoint. RampSlope $= \pm 1$ gives zero at one or the other of the endpoints. |RampSlope| > 1 produces a bi-directional profile.

RelAngle : (radians)
Angle (measured counterclockwise) of the positive flow direction with respect to inward-pointing boundary normal.

The total mass flow rate $\dot{m}$ through the region boundary is given by

$$\dot{m} = [\text{MdotMean} + \text{MdotAmp} \sin(\omega t + \text{Mphs})] \cos(\text{RelAngle})$$

where ω is the circular frequency and t is time. The positive sense of $\dot{m}$ is always into the region. The influx temperature is determined by the region temperature as specified by the **RgnSpec** variables **T_left** and **T_right**. If both **MdotMean** and **MdotAmp** are zero then Manifest treats the boundary as a solid wall.

The current version of Manifest allows several different velocity profiles across any given region boundary, but you might wonder why there is no way to specify flow boundary conditions at individual nodes? There is a good reason for this. In the current way of doing things, Manifest can easily reconstruct the node boundary conditions after vertices are moved or the computational cell count along the boundary is changed. If you were allowed to enter mass flow conditions for individual nodes, then it would be much harder for Manifest to figure out how to update node boundary conditions after any changes.

RgnSpec Glossary

These variables apply to the individual regions comprising the computational domain. The interior boundaries dividing regions are not always the straight line segments displayed by the input editor. The true boundaries will follow the coordinate curves of the computational grid. If they wind up curved along an interior boundary, then so will be the boundary.

cell_1 : (integer)
The number of computational cells along the *horizontal* dimension of the region — the dimension not in common with other regions.

Cergun : (dimensionless)
Ergun coefficients in tensor form. For principle porosity axes aligned with the coordinate axes of the display:

- Cergun[1,1] = first Ergun coefficient in the X-direction (horizontal).
- Cergun[2,2] = first Ergun coefficient in the Y-direction (vertical).
- Cergun[1,2] = 0.
- Cergun[2,1] = 0.

The first Ergun coefficient is c_1 in the Darcy friction factor f of the form

$$f = \frac{c_1}{R_e} + c_2$$

where R_e is the Reynolds number based on hydraulic diameter. For fibrous materials, $c_1 \approx 150\beta^{-0.6}$ and $c_2 \approx 1.5\beta^{-0.6}$ where β is porosity — see reference [11]. For non-aligned principal porosity axes you will have to use the appropriate coordinate transformation rules for tensors — see chapter 7. This should rarely be necessary.

Csolid : $(J/kg\,K)$
Solid matrix material specific heat. Csolid = 0.0 causes the solid temperature to be held constant.

Dhyd : (m)

Hydraulic diameter for the porous matrix. Defined as

$$\text{Dhyd} = 4V/S$$

where V is void volume and S is wetted surface area. Used for matrix heat transfer and porous flow resistance calculations. Dhyd must be strictly positive to avoid division by zero. For wire matrices with fiber diameter d_w and porosity β

$$\text{Dhyd} = \frac{\beta}{1-\beta} d_w$$

For matrices composed of spherical particles having diameter d_p

$$\text{Dhyd} = \frac{2}{3}\frac{\beta}{1-\beta} d_p$$

ErgunRatio : (dimensionless)

The ratio of the second/first Ergun coefficients which is assumed to hold for each component of the Cergun tensor. Used for a high Reynolds number perturbation for the porous flow resistance. See variable Cergun.

FixCommonBdrys : (True or False)

Choosing *True* forces the coordinate curves of the curvilinear grid to follow the straight-line boundaries of the region — including those held in common with other regions within the interior of the computational domain. Choosing *False* relaxes the constraint on common boundaries. Because it gives the smoothest grid, the *False* choice is best for contiguous regions; either empty or filled with the same material. However, the *False* choice might allow a regenerator matrix to sag into or away-from an empty space. To prevent this, choose the *True* option for a filled region next to an empty neighbor. The status of a common boundary is actually determined by the value of FixCommonBdrys for both adjoining regions. If either one is *True* then the grid will be constrained there.

KgasMultiplier : (dimensionless)

Gas-only conductivity enhancement multiplier, defined as KgasMultiplier $= k_e/k$ where k is the molecular gas conductivity and k_e is the effective gas conductivity. Normally KgasMultiplier $= 1$. Values greater than one can be used to account for conduction-like sub-grid-scale terms in the energy equation — if you know of a suitable correlation for them. See section 8.2.6.

Ksolid : (W/mK)

Matrix material solid conductivity.

Matrix : (True or False)

True if a matrix is present in the region or *False* if it's empty.

Nusselt : (dimensionless)

Nusselt number for the region. Defined as

$$\text{Nusselt} = \frac{\bar{h}d}{k}$$

where $\bar{h}$ is the mean film coefficient, d is the hydraulic diameter and k is the molecular gas conductivity. Reference [8] provides a useful correlation for Nusselt number in screens:

$$\text{Nusselt} = pR_e^m$$

where R_e is Reynold's number based on hydraulic diameter, and p and m are defined in terms of porosity β by

$$\begin{array}{lcll} m & = & 0.85 - 0.43\beta & \\ p & = & 0.537\beta; & \text{if } \beta < 0.39 \\ p & = & 1.54 - 6.36\beta + 7.56\beta^2; & \text{if } \beta \geq 0.39 \end{array}$$

In terms of Stanton number S_t, Reynolds number R_e and Prandtl number P_r, Nusselt number is also given by

$$\text{Nusselt} = S_t P_r R_e$$

Porosity : (dimensionless)

Void/total volume in the region. Porosity = 1.0 causes the solid temperature to be held constant.

RhoSolid : (Kg/m^3)

Solid matrix material density. RhoSolid = 0.0 causes the solid temperature to be held constant.

T_left : (K)

Initial temperature at the left end of the region. The designation "left" refers to the initial orientation of the display.

T_right : (K)

Initial temperature at the right end of the region. The designation "right" refers to the initial orientation of the display.

VisMultiplier : (dimensionless)

Viscosity enhancement multiplier for the region. Defined as VisMultiplier = μ_e/μ where μ is molecular gas viscosity and μ_e is effective gas viscosity. μ_e replaces μ in the viscous stress tensor. Normally VisMultiplier = 1. Values greater than one can be used to account for viscosity-like sub-grid-scale terms in the momentum equation — if you know of a suitable correlation for them. See section 8.2.6.

VisMultiplier = 0.0 causes Manifest to ignore the viscous stress tensor in both momentum and energy equations (but still model porous flow resistance), thereby speeding up the computation considerably.

GlobSpec Glossary

These variables apply to the computational domain as a whole.

c1_Suth : $(kg/(msK^{0.5}))$
A Sutherland coefficient. Molecular viscosity μ is calculated as

$$\mu = \frac{\text{c1\_Suth}\, T^{1.5}}{T + \text{s1\_Suth}}$$

where T is gas temperature.

cell_2 : (integer)
The number of cells along the vertical dimension of the computational domain — the dimension common to all regions.

Cp : $(J/kg\,K)$
Gas specific heat at constant pressure.

Cv : $(J/kg\,K)$
Gas specific heat at constant volume.

frequency : (Hz)
Frequency of underlying flow cycle.

GasdynamicTol : (dimensionless)
RMS error tolerance for convergence of main solution. After each time step, Manifest calculates the root mean square change of the solution variables over exactly one cycle period. If less than GasdynamicTol then Manifest stops. (The solution variables from the preceding cycle are in the _MFSTEP.nnn file about to be overwritten.) The default value of GasdynamicTol = 1.0E–4 is a reasonable place to start. You may have to play around with this value depending on the details of your particular problem. The dimensionless solution variables M, E and T_s (see section 8.5.2) are normalized to the order of one. Velocity, on the other hand, is normalized by the sonic velocity which means that variables G^1 and G^2 are on the order of the local Mach number.

GridTol : (dimensionless)
RMS error tolerance for convergence of dimensionless curvilinear coordinate solution. GridTol = 1.0E−4 is the default value.

MaxTimeSteps : (dimensionless)
Maximum number of output time steps to be simulated. Manifest halts execution if the solution has not converged before this.

node_T : (integer)
Number of output time steps (_MFSTEP.nnn files written) per period of the oscillating flow cycle.

Prandtl : (dimensionless)

Prandtl number. Used for calculating molecular gas conductivity, $k = C_p\mu/\text{Prandtl}$.

pressure : (N/m^2)

Time-mean gas pressure.

s1_Suth : (K)

A Sutherland viscosity coefficient. See c1_Suth.

thk_domain : (m)

Thickness of the computational domain. The dimension normal to the display screen. This dimension is not discretized.

TstepSubdivide : (integer)

Number of integration time steps per output time step. Integration time step Δt is given by

$$\frac{1}{\Delta t} = \text{TstepSubdivide} \times \text{node\_T} \times \text{frequency}$$

TstepSubdivide should normally be set to 1. Values greater than 1 are used to reduce the integration time step without increasing the number of _MFSTEP.nnn files written.

5.1.6 More on Curvilinear Coordinates

The curvilinear grid is normally produced as the last step of MFSETUP. This part of the program is not interactive in any way; you just sit back and wait for the results. Even so, there are a few things you should be aware of.

The number of nodes in the curvilinear grid is determined by the variables Cell_2 (accessible with edit function GlobSpec) and Cell_1 (accessible with edit function RgnSpec). Cell_2 has global scope in the computational domain while there is a Cell_1 defined for each region. Actually, these variables determine the number of computational *cells* — the basic solution elements — in the domain. Each cell spans two coordinate curves of the display. That is, the nodes of the grid fall at the corners, mid-points of sides and in the centers of the computational cells.

If Cell_2 or the Cell_1's become too large, the time required to solve the curvilinear grid will grow rapidly and, at some point, MFSETUP may even run out of memory and crash ignobly. If this happens your only recourse is to reboot your machine (warm boot) and try again. One might suppose that speed and memory requirements would depend mainly on the total number of cells in the computational domain. This is not quite true: Long and thin domains fare better than square domains having the same number of cells. In the previous sentence, the terms *long* and *thin* refer to the number of cells across the domain, not the physical dimensions.

In some cases, part of the curvilinear grid will spill over to the exterior of the computational domain. This can happen when the domain contains sharp V-shaped concave corners. Manifest is currently too dumb to prevent this, or even know when it has happened. It is up to you to spot the problem and take corrective action. Setting the variable FixCommonBdrys to TRUE for at least one of the regions on either side of the offending V, will constrain the grid on the common boundary, and fix the problem.

5.1.7 Modeling Steady Flows

Occasionally, you may want to set up Manifest to model steady flow instead of oscillating flow. Many flow maldistribution problems are just as evident in a steady-flow situation, and Manifest usually converges to the solution faster. The BdrySpec variable MdotMean allows you to specify a steady-flow boundary condition.

You might wonder what to do about the GlobSpec variables frequency and node_T which seem to be somewhat arbitrary in steady flow. Since node_T determines the number of _MFSTEP.nnn files written to the disk, it is a good idea to keep it small: say about node_T = 4. Then, once you have set node_T, choose frequency to determine the time step Δt from the following equation

$$\frac{1}{\Delta t} = \text{TstepSubdivide} \times \text{node\_T} \times \text{frequency}$$

TstepSubdivide can be anything, in principle, but it is probably a good idea to set it equal to 1 so that a _MFSTEP.nnn file is written for each integration time step.

5.1.8 Special Effects

Some of the MFSETUP source-level variables have special values which trigger Manifest to modify its normal simulation model.

Fixing Solid Temperatures

Setting Porosity = 1.0, RhoSolid = 0.0 or Csolid = 0.0 for a region cause the solid temperature in that region to be held constant. This feature is useful in regions where the solid temperature T_s is supposed to represent an isothermal heat exchanger wall, or when the region is essentially adiabatic (in which case you will also have to set Nusselt = 0). Fixed T_s is selected automatically whenever the Matrix input variable is set FALSE.

Fixing Gas Temperature at Walls

A constant wall-temperature boundary condition can come in handy when modeling conductive heat flux through the walls of regions. You might be interested in this for gas spring loss analysis or deriving film heat transfer coefficients for oscillating channel flow from first principles. The key idea here is that for solid walls (not active flow inlets) Manifest sets the gas temperature at the wall equal to T_s extrapolated to the wall. By properly initializing T_s and holding it constant as explained above, you can model the normal temperature gradient at the wall (and therefore the conductive heat flux) with any wall boundary condition temperature you choose.

Neglecting the Viscous Stress Tensor

Setting VisMultiplier = 0.0 for a region causes the viscous stress tensor $\boldsymbol{\tau}_e$ to be neglected (not evaluated at all) in that region for the momentum and energy equations. You should take advantage of this feature whenever you are modeling a porous material where the sub-grid-scale flow resistance is large compared to the grid-scale viscous forces. The sub-grid-scale flow resistance is modeled in the form $\phi \cdot \beta \mathbf{V}$ in the porous momentum equation (8.59), and is completely separate from the visous stress term.

You will probably do well to set VisMultiplier = 0.0 for almost all porous materials you are likely to model. You will notice a significant speed-up of the solution process with this feature in effect.

Of course there are some cases where VisMultiplier should not be set to 0. These are for flow in empty manifolds or when evaluating small scale phenomenon near the boundaries of a diffuse matrix — such as jet penetration into a matrix. In general, whenever grid-scale spatial velocity gradients are likely to produce significant viscous forces, set VisMultiplier to a realistic value > 0.

5.2 MF Reference Guide

When MF is running smoothly, all you have to do is sit back and watch. When problems arise you might find help in this section.

5.2.1 Instability

When the arrows in the mass flux vector field run amok (grow large and randomly oriented), Manifest has gone unstable. Unfortunately this is possible even though the program uses an implicit solution method.

Instablity seems to correlate fairly well with Courant number N_c defined as

$$N_c = c\frac{\Delta t}{\Delta x}$$

where c is the gas sonic velocity, Δt is the integration time step and Δx is the grid spacing. Experience indicates that Manifest is stable up to a limiting Courant number of about 100. This value is not exact: It can be more or less depending on the size of viscous and porous damping terms as well as the *curviness* of the curvilinear coordinate grid for a particular problem.

For a grid spacing $\Delta x = 1.0\text{E}{-2}\ m$ and a sonic velocity $c = 1000\ m/s$ (helium at 300 K), the likely not-to-exceed time step works out to

$$\Delta t_{max} = \frac{N_{cmax}\Delta x}{c} \approx 1.0\ \text{E}{-3}\ s$$

For a 60 Hz cycle this would imply a minimum of 16 integration time steps per cycle (node_T × TstepSubdivide), which is reasonable.

There is a problem with making node_T too big: The number of _MFSTEP.nnn files that must be read by the output program increases. If you have the need to model a low frequency cycle or a very small physical region, you may want to model your problem as a number of separate steady-flow cases as discussed in section 5.1.7. This is reasonable since in this sort of problem non-steady effects are likely to be small anyway. Another alternative is to keep node_T small but make TstepSubdivide large. This will limit the number of _MFSTEP.nnn files produced while still giving you freedom to reduce the integration time step as small as needed.

5.2.2 Reynolds Number Limits

Just like in real fluids, Manifest flows are characterized by Reynolds number. If Reynolds number becomes too large, flow instabilities may develop. These are different from the numerical instabilities mentioned above. Reynolds number instabilities begin with gradual wave-like phenomena and gradually grow to complete chaos. But since real fluids behave this way too, there is no need to fix the *problem*. Just reduce Reynolds number. The critical Reynolds number beyond which chaos ensues is on the order of 100 to 1000, depending on the flow details.

Unlike real fluids though, Manifest has limits to its resolving power. Small-scale eddies or thin wall-boundary layers are unknowable to Manifest once they become smaller than the mesh of the computational grid. The user must be the judge of whether or not the computational grid is fine enough to resolve the important phenomena in the problem. Manifest is not smart enough to do this on its own.

5.2.3 Running Out of Memory

Abnormal termination of MF accompanied by the system error message *No room in heap* is the definitive symptom of running out of memory. The maximum addressable memory of 640K should be available; 512K is the minimum recommended. Be sure that there is no superfluous memory resident software competing for memory space. The actual memory requirement for any given simulation depends mainly on the fineness of the computational mesh. Reducing cell_2 or cell_1 for one or more regions will reduce memory needs. Setting VisMultiplier = 0.0 for regions containing porous materials — as described in section 5.1.8 — will also help.

5.2.4 Restarting

You can halt execution of MF at any time and resume later on where you left off. Either press the Esc key to stop after the current time step is solved or, more drastically, re-boot your computer. There is usually a special key sequence to perform a *warm* boot that is faster and less damaging than momentarily interrupting the line power. To resume execution, just load MF as usual. The program remembers where it left off by reading the file _MFSTATU.TXT.

The only time you will be *unable* to restart MF is if you run MFSETUP again. Actually, this is not quite true: If you use MFSETUP to merely inspect a source file without going so far as to compile or save it (menu options F2 or F3) then you will be OK. Only when you choose menu options F2 or F3 does MFSETUP erase the previous _MF... files.

5.2.5 Convergence

MF will keep running until the convergence criterion established by the GlobSpec variable GasdynamicTol is met. Basically, after each output time step, Manifest calculates the root mean square change of the solution variables over exactly one cycle period. If less than GasdynamicTol then Manifest stops. (See section 5.1.5.)

The only exceptions to this are if the time-out number of output time steps (set by GlobSpec variable MaxTimeSteps) have been exceeded, or if you manually abort the program. In either of these cases you will be warned in MFOUT that convergence has not been attained.

Even if Manifest thinks convergence has been attained, you should still be on guard. Some problems may require a finer GasdynamicTol than others. You should be suspicious if an energy balance does not hold in any region of your computational domain and reduce GasdynamicTol accordingly. Matrix solid temperatures are apt to be especially slow to converge.

5.3 MFOUT Reference Guide

5.3.1 Indexing Conventions

MFOUT has an indexing system which makes it possible to refer to the various regions, region boundaries and grid points of the computational domain. When reading this section think of your computational domain as a chain of rectangles strung out from left to right on the display. Even if your actual domain doesn't actually look like this, it is still topologically equivalent; you simply have to mentally stretch it into compliance. Having done this, everything else is simple.

Regions These are numbered sequentially starting at 1 for the left-most region on the display.

Boundaries These are indexed by two numbers: first by the region containing the boundary then clockwise by side number starting with 1 for the left-most side. Thinking of a region as a rectangle, the left side = 1, top = 2, right side = 3 and bottom = 4. Since regions share common boundaries, side 3 of some region is the same as side 1 of the following one.

About these common boundaries. During some selection processes you will see them displayed on the screen as straight line segments, even though, in some cases, the true boundary, defined by the computational grid, is curved. Rest assured that all integrals performed within regions and on region boundaries reflect the true grid- defined boundaries.

Grid Points These are indexed by ordered pairs following the usual convention of listing the horizontal coordinate first. Grid point (1,1) is at the lower left corner.

5.3.2 Output Listing

Each time you run MFOUT it writes a new OUTPUT file to the disk. Any earlier version that may have been present is erased. The things which appear in OUTPUT are

- Date and time stamps. This is for the time when MFOUT is executing, not when the simulation ran or when the source data file was created.
- Source data file name.
- Global input variables from GlobSpec of MFSETUP.
- Region specific input variables from RgnSpec of MFSETUP. Listed for each region separately and indexed by the region number.
- Boundary-condition variables from BdrySpec of MFSETUP. Listed only for active inlets (boundaries where either MdotAmp or MdotMean are not zero). Indexed by region and side numbers.
- Run status variables StepNumber and RmsChange. These give the last time step simulated and the RMS solution change from the previous cycle (see GasdynamicTol in section 5.1.5).
- A log of your interactive output session. This is text format only. No graphics are included.

The GlobSpec, RgnSpec and BdrySpec listings are in the same format as they appear in the MFSETUP display: in a hierarchical list with variables of similar scope indented the same amount.

OUTPUT is a standard ASCII file and you can examine it with any text editor or most word processors. You can also print it using the DOS PRINT or TYPE>PRN commands. If you want, you can even edit it before printing.

5.3.3 Interactive Options

After reading and processing its input files, MFOUT enters interactive output mode. Then, you select interactive output options by point-and-clicking the appropriate activity box along the left edge of the screen. In this respect, MFOUT is similar to MFSETUP.

Moving the Cursor

Depending on its location, the cursor is represented on the display by either a "+" or an arrow. Normally the cursor *tracks* the motion of the mouse as you move it. If you don't have a mouse, the keyboard arrow keys can also be used to move the cursor. The arrow keys move the cursor about one pixel in the indicated direction. Holding a shift key down while pressing an arrow key increases the size of the step.

Mouse Button Substitutes

The right and left mouse buttons are needed frequently. If you have no mouse, use the F9 key as a substitute for the left button and the F10 key as a substitute for the right button.

ContPlot

Use this function to display contour plots of

- gas pressure
- gas temperature
- matrix solid temperature

After you select ContPlot you are presented with a sub-menu of the above options. The up and down arrow keys scroll through the menu and pressing return selects the highlighted option.

The initial contour plot is drawn for time step 0 with the default contour spacing. The right and left arrow keys increment or decrement the time-step number by one. The gray "+" or "–" keys increase or decrease the contour spacing.

VecPlot

Use this function to display vector plots of mass flux per unit total area. There are no other options at this time.

The initial vector plot is drawn for time step 0 with the default arrow length. The right and left arrow keys increment or decrement the time-step number by one. The gray "+" or "−" keys increase or decrease the arrow lengths.

EnergyInt

Use this function to display integrals within regions of

- Porous Dissipation
- Viscous Dissipation
- Gas-to-Solid Heat Transfer

Technically speaking each of these quantities has units of work (Joules), but in accordance with engineering conventions they are all multiplied by frequency to give mean effective powers expressed in Watts.

After you select EnergyInt you are presented with a sub-menu of the above options. The up and down arrow keys scroll through the menu and pressing return selects the highlighted option.

The initial display shows an outline of the computational domain with the individual quadrilateral regions comprising it numbered sequentially starting from 1. The bottom of the display window gives an initial heading — depending on the sub-menu option selected — of something like:

```
Region #, Porous Dissipation (W)
```

Point-and-clicking the mouse inside a region of the computational domain produces a line of numerical output at the bottom of the display window. This output is simultaneously copied to the OUTPUT file.

Porous dissipation is based on the integral of

$$W_f = -\beta \nabla P_f \cdot \mathbf{V}$$

where β is porosity, ∇P_f is the force per unit void volume due to sub-grid-scale porous-material friction and $\mathbf{V}$ is void-average velocity. W_f is the dimensionless pumping power per unit total volume. ∇P_f is given by

$$\nabla P_f = -\boldsymbol{\phi} \cdot \mathbf{V}$$

where $\boldsymbol{\phi}$ is the flow resistance tensor described in section 8.2.3.

Viscous dissipation is based on the integral of the dissipative part of the stress work term

$$W_d = \nabla \cdot (\boldsymbol{\tau}_e \cdot \beta \mathbf{V})$$

occurring in the porous-material thermal energy equation (8.60). $\boldsymbol{\tau}_e$ is the apparent viscous stress tensor (see section 8.2.6). In tensor notation, the dissipative part of W_d is

$$\beta(\boldsymbol{\tau}_e)_{mk}\frac{\partial U_k}{\partial x_m}$$

this is often called the dissipation function which represents the mechanical energy per unit total volume dissipated into thermal energy.

Gas-to-solid heat transfer is based on the integral of

$$Q_h = \beta Q$$

where β is porosity and Q is the heat flux per unit void volume transferred between the gas and the solid matrix material.

FluxInt

Use this function to display integrals across boundaries of

- Advected Energy Flux
- Conductive Heat Flux

Technically speaking each of these quantities has units of work (Joules), but in accordance with engineering conventions they are all multiplied by frequency to give mean effective powers expressed in Watts.

After you select FluxInt you are presented with a sub-menu of the above options. The up and down arrow keys scroll through the menu and pressing return selects the highlighted option.

The initial display shows an outline of the computational domain with the individual quadrilateral regions comprising it numbered sequentially starting from 1. The bottom of the display window gives an initial heading — depending on the sub-menu option selected — of something like:

```
Region #, Side #, Net advected energy flux through side (W)
```

Point-and-clicking the mouse on a region side produces a line of numerical output at the bottom of the display window. This output is simultaneously copied to the OUTPUT file. For interior sides, the flux integral is performed along the actual region boundary defined by the curvilinear coordinate grid.

Advected energy flux is based on the integral of the advected terms appearing in the porous-material thermal energy equation (8.60)

$$\mathbf{Q}_e = \beta\rho e\mathbf{V} + P\beta\mathbf{V} - \boldsymbol{\tau}_e \cdot \beta\mathbf{V}$$

where β is porosity, ρ is dimensionless gas density, e is dimensionless mass specific total gas energy, $\mathbf{V}$ is void-average velocity, P is pressure, and $\boldsymbol{\tau}_e$ is the apparent viscous stress tensor (see section 8.2.6). $\mathbf{Q}_e$ is comprised of enthalpy flux, kinetic energy flux and viscous stress-work flux. Integrated over a cycle, the kinetic energy term will presumably cancel (for sinusoidal $\mathbf{V}$) leaving only enthalpy and viscous stress-work flux.

Conductive heat flux is based on the integral of

$$\mathbf{Q}_k = \beta\mathbf{q}_e$$

where β is porosity and $\mathbf{q}_e$ is the apparent void-average heat flux vector which appears in the porous-material thermal energy equation (8.60)

Fourier

Use this function to display Fourier coefficients at computational grid points of

- Pressure
- Gas Temperature
- Matrix Solid Temperature
- rho V_1
- rho V_2

The quantities (rho V_1) and (rho V_2) are the rectangular components (horizontal and vertical) of the mass flux per unit total area vector **G** defined as

$$\mathbf{G} = \beta\rho\mathbf{V}$$

where β is porosity, ρ is dimensionless gas density and $\mathbf{V}$ is void-average velocity.

After you select Fourier you are presented with a sub-menu of the above options. The up and down arrow keys scroll through the menu and pressing return selects the highlighted option.

The initial display shows the computational grid with individual nodes lying at the intersections of the coordinate curves. The bottom of the display window gives an initial heading — depending on the sub-menu option selected — of something like:

```
I1, I2, rho V_1:  mean (kg/s m2), ampl (kg/s m2), phase (rad)
```

Point-and-clicking the mouse at a node produces a line of numerical output at the bottom of the display window. This output is simultaneously copied to the OUTPUT file. I1 and I2 are the coordinates of the node, **mean** is the cycle-average value and **ampl** and **phase** are the amplitude and phase of the first harmonic.

Scale

Use this function to scale the computational domain relative to its display window. The dimensions set here carry over to all of the other interactive functions of MFOUT. This function works identically to the scaling function in MFSETUP.

Two keyboard commands are available:

- **Gray +**: Increases the apparent size of the computational domain.
- **Gray –**: Decreases the apparent size of the computational domain.

In addition, clicking either mouse button when the cursor is in
the display window, re-positions the display window so it is centered at the cursor location.

Done

Use this function to exit MFOUT and return to DOS.

5.3.4 Graphics Hard Copy

MFOUT has no internal commands for copying graphics images to a printer. Instead, you will have to load some form of memory resident screen printing utility prior to running MFOUT and activate it when needed — generally by pressing a *hot key* combination such as Shft-PrtSc. Execution of MFOUT is temporarily interrupted while the screen dump is in process. This may take several minutes, depending on the resolution of your display screen and the speed of data transfer.

The chief problem will be finding a screen-dump utility that works with your particular display adapter and printer. See chapter 3 for some guidance here. The PRTSCRN.EXE program bundled with the Manifest software may work with IBM-compatible dot-matrix printers. More sophisticated systems will require more powerful software.

Keep in mind that the display colors will affect the printer image. You may have to set the display background color to black by making BackgroundColNum = 0 in the attribute file DSPLYATR.TXT. See chapter 3 for more details on how to do this.

Chapter 6

Mainframe Manifest

PC's are limited in speed and memory, and this is especially noticeable in Manifest's number crunching MF module. So it is natural to want to run at least this part of Manifest on a bigger computer. Mainframe manifest makes this possible. The input and output modules MFSETUP and MFOUT run on a PC exactly as before, except now one has the option of running the MF module on a supercomputer.

The mainframe version of Manifest's MF module is written in a standard dialect of Pascal and can probably be installed on virtually any computer system by re-compiling. This chapter documents the version installed at NASA Lewis on the VM/Cray system residing there. (VM stands for the virtual-machine IBM mainframe system that serves as front-end to the Cray X-MP.)

As in the MS-DOS environment, MFSETUP, MF and MFOUT communicate with each other by disk files. This requires that certain files be sent to VM before MF is run and be sent back again when finished. The details of this process depend upon how one's PC is connected to the host mainframe. Users within NASA Lewis can use the Lewis local network to transfer files. More remote users, connected via modem and phone lines, can use the CMS KERMIT file transfer program as long as their PC communications software supports it.

In addition to the files required for the PC version of Manifest, mentioned in chapter 3, the following files are vital for mainframe Manifest:

MANIFEST PAS	Source code for mainframe MF module
MFBATCH DAT	Job submittal data file
MFRECV EXEC	Job receiving command file
UPCRAY.EXE	PC input file conversion program
DOWNCRAY.EXE	PC output file conversion program

The first three file names are in VM format since they normally reside there. The last two file names are in MS-DOS format since they stay on your PC. The

Pascal source code is stored under the name MANIFEST PAS to avoid confusion with the PC version MF.PAS.

6.1 Running a Job

Regardless of the PC-to-Mainframe link, the process of running MF on the Cray can be blocked out as follows:

1. Run PC program MFSETUP as usual to produce input files.
2. Run PC program UPCRAY to initialize Cray-specific files.
3. Logon to VM and upload the files: _MF1.TXT, _MF2.TXT, _MFSTATU.TXT and _MFSTEP.ALL.
4. Submit the batch-mode job to the Cray by entering the VM command: CRSUBMIT MFBATCH DAT.
5. Wait until output files appear in the reader signifying the job is done.
6. Type: MFRECV to receive all the reader files to the VM disk.
7. Download to your PC the files: _MFSTATU TXT, _MFSTEP ALL, _MF1 TXT and _MF2 TXT.
8. Run PC program DOWNCRAY to break up _MFSTEP.ALL output file into individual files for each time step.
9. Run PC program MFOUT as usual to examine output.

To run a completely new job (new input files compiled by MFSETUP) you must repeat the above procedure in its entirety. To restart a job from the point where it last left off, just jump into the procedure at step 4.

What follows in this section is further elaboration on some of the steps in the submittal procedure.

Step 2. UPCRAY initializes the files _MFSTATU.TXT and _MFSTEP.ALL. These files are mainly output files but are also read as input to remind MF where it left off in case it is restarting. The initial values placed in these files by UPCRAY tell MF to start from scratch.

A better way to tell MF to start from scratch would be to just not write the files in the first place. Then MF could sense that the files were not present and default to its start-up mode. But Cray Pascal does support checking to see if a file exists before reading it.

Step 3. With luck, uploading files can be done using the wildcard specification _MF*.*, depending on your file-transfer method.

Step 4. MFBATCH DAT is a JCL file that causes the following to happen when executed in the Cray batch environment:

- Sets the job time limit (number after -lT command, seconds).
- Fetches the _MF... input files from VM.
- Links to the appropriate Cray directory containing manifest.o, the compiled MF object code.
- Loads and runs the program.
- Disposes the _MF... output files to the VM reader.

See section 6.2 if MANIFEST PAS has not yet been compiled on the Cray.

Step 5. Since MF executes in batch mode, you do not have to be logged onto VM for it to execute. At the same time, you must anticipate the possibility that it will run out of time before achieving convergence. If this happens, you will not get any output back. To avoid this you can set the input variable MaxTimeSteps to an appropriately small number or set the time limit in MFBATCH DAT to an appropriately big number. When MF completes MaxTimeSteps output time steps it will stop. MF is designed to restart where it left off the next time it is submitted. It will then run until another block of MaxTimeSteps is completed, or until convergence is achieved.

Step 6. MFRECV EXEC is a VM command file which receives all the files in the reader using the RECEIVE command with the (REPLACE option. If the reader was not previously empty, MFRECV will read those files as well.

Assuming the reader was previously empty, the first reader file, MANIFEST CONSOLE, contains human-interest output that would have been displayed on the console had you run the program interactively. Normally, you can ignore this file. If you have trouble, though, you might want to look here for a clue as to what went wrong.

The next two files, _MF1.TXT and _MF2.TXT, are input-only files that are returned from the Cray system unchanged. That is, they are the same files as those fetched from VM in step 4. They are sent back to the reader for the sake of completeness in case the corresponding VM versions have since been modified. Program MFOUT reads these two files as well as the remaining output files so, needless to say, it will get confused unless they all correspond to the same problem.

The next three files, _MFSTATU.TXT, _MFSTEP.ALL and _MFSTEP.DV (optional), are files containing information about the most recently completed time step, the solved variables themselves and the solution-variable increments for the most recent time step. MF both reads from and writes to these files. Output program MFOUT reads _MFSTATU.TXT and _MFSTEP.ALL. The file _MFSTEP.DV is only needed for restarting MF and is not present in all cases.

The Cray system, of its own accord, returns to the reader a file named something like name____ OUTPUT which contains messages from UNICOS. The name____ part is the name of the virtual machine associated with the job (padded with underscores to eight characters). Here you can find Cray job accounting information — like CPU time, etc. — and perhaps other information that may be of interest, especially if your job bombed.

Step 7. You do not actually have to download the files _MF1.TXT or _MF2.TXT unless you have changed them on your PC while the Cray job was running. MF only reads from these files.

Step 8. To accommodate the limitations of Cray Pascal and to simplify the JCL, the Cray version of MF reads and writes its solution values to one big file _MFSTEP.ALL containing values for all points of the computational grid, for all time steps computed so-far. The MS-DOS version, on the other hand, breaks its output down into separate files for each time step _MFSTEP.nnn, where nnn is the time step number. Program DOWNCRAY converts the _MFSTEP.ALL file into separate _MFSTEP.nnn files.

6.2 The Source Code

The version of MANIFEST PAS currently residing on the VM system has appropriate JCL statements included at the beginning and end of the file so that it may be submitted to the Cray for compilation by entering the VM command: CRSUBMIT MANIFEST PAS. The compiled object code is automatically stored in a Cray subdirectory as file manifest.o.

Needless to say, the Pascal source code for the MS-DOS version of the MF module and the Cray version differ substantially. There are two reasons for this:

1. Cray Pascal does not support nearly as many extensions to the ISO Level 1 Pascal standard as Microsoft Pascal does, and
2. The VM/Cray operating environment does not support graphics.

The remainder of this section documents some of the major surgery done on the PC MF module in order to produce a Cray version.

6.2.1 Units

A unit is a separately compiled block of Pascal source code that allows big programs — like Manifest — to be separated into logically distinct chunks. Units

C-2

export types, variables and procedures, to be used by other units or the main program, in a structured way. The Microsoft Pascal version of Manifest relies heavily on units. But, alas, Cray Pascal does not support units. Separately compiled modules are as close as it comes, but modules are essentially unstructured and are not allowed to import or export variables or type declarations from other modules. The result of all this is that the Cray version MF is one long program.

6.2.2 Graphics

During execution, the PC version of MF displays a graphics image of the most-recently-completed solution vector field. This feature is absent in the Cray version.

6.2.3 Dynamic Arrays

One of the very nice things about Microsoft Pascal is that it allows you to establish upper bounds and allocate memory for arrays at run-time. You do not have to allocate memory in advance to hold the largest possible array. Unfortunately, Cray Pascal does not support this feature, with the result that the maximum dimensions of the computational array and the maximum number of output time steps per cycle are specified as constants at compile time (constants: MaxNode_1, MaxNode_2, and MaxNode_T). If you change these constants you have to re-compile the program. You may bump into these limits, in the form of an error message followed by an abrupt halt, if you become too bold in your problem design.

6.2.4 String Handling

Microsoft Pascal supports string types of fixed and variable length and a large number of special functions for dealing with them. Cray Pascal supports only fixed-length strings. This fact required several minor but annoying changes in procedures dealing with reading variable names during disk I/O or passing file names as arguments.

6.2.5 File I/O

Microsoft Pascal has a way for you to check to see if a file is present before trying to read it. This makes it possible for the PC version of MF to determine whether it is starting from scratch or restarting, merely by checking for the existence of certain input files. Cray Pascal does not support this sort of thing, which explains some of the added complication of the submittal procedure (programs UPCRAY and DOWNCRAY) in the previous section.

6.2.6 Restarting

The PC version of MF is somewhat interactive; you can stop it at any time by pressing the escape key, or by re-booting the system. The next time MF runs it will resume the simulation where it left off (unless you create new input files with MFSETUP). This is possible because MF is designed to write a permanent disk file of solution variables _MFSTEP.nnn for each time step immediately after it is solved. The idea is that these files are cyclically overwritten so that the collective set of them contains the solution for the most recently simulated cycle. The files _MFSTATU.TXT and _MFSTEP.DV are also overwritten after each time step and contain information about the status of the solution, and the solution-variable increments dV for the current time step. This information is needed to restart the simulation in case of MF is stopped for some reason. (Actually, _MFSTEP.DV only written if input variable TstepSubdivide > 1 because, otherwise, dV is available by differencing the two most recent _MFSTEP.nnn files.)

The Cray version attempts to do all this as much as possible. It actually maintains the equivalent of the _MFSTEP.nnn files — but in RAM, not on the disk. Only at the very beginning or end does it read/write them from/to the permanent disk file _MFSTEP.ALL. I decided to do it this way because I didn't want to deal with the JCL required to reference files of unknown number having incremented extension identifiers. Files _MFSTATU.TXT and _MFSTEP.DV are overwritten each time step as before. Restarting is still possible, but the ways in which the program can halt gracefully are more limited. Pressing the escape key or re-booting are out. The only alternatives are to achieve convergence or to exceed the maximum-time-step limit specified by input variable MaxTimeSteps. If the system times you out, you lose your output files.

Part III

Theory

Chapter 7

Coordinate Transformation Theory

Manifest uses two-dimensional boundary-fitted coordinates in order to solve general stirling flow problems in non-rectangular regions. This chapter develops the foundation for this sort of thing from pretty-nearly first principles. The reader familiar with coordinate transformations can skip over this chapter, returning to it later on for reference if needed.

Since this subject requires a lot of formulas involving coordinates, **the summation convention from tensor analysis is used**. Whenever two like indices occur in a formula they should be summed from 1 to 3 (or 1 to 2 in two-dimensional formulations). For example the expression $\frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial y_j}\frac{\partial y_j}{\partial x_i}$ should be read $\frac{\partial f}{\partial x_i} = \sum_{j=1}^{3} \frac{\partial f}{\partial y_j}\frac{\partial y_j}{\partial x_i}$.

Also, please note that although most results are derived in three-dimensions, application to two dimensions is straight forward — simply sum indices from 1 to 2 instead of 1 to 3 or omit quantities corresponding to the third dimension.

7.1 Coordinate Systems

First, a word about coordinate systems in Euclidian three-dimensional space, also known as $\mathbf{E}^3$. Since we all live in $\mathbf{E}^3$ (or a close approximation) we can relate to it pretty well. Its only when we think about coordinate systems for $\mathbf{E}^3$ that the trouble begins. In the familiar rectangular system (x_1, x_2, x_3) are the position coordinates and t is the time coordinate. In the unfamiliar curvilinear coordinates I will speak of the same position and time in terms of coordinates (y_1, y_2, y_3) and τ where.

$$y_i = y_i(x_1, x_2, x_3, t) \tag{7.1}$$

for $i = 1 \ldots 3$, and

$$\tau = t \tag{7.2}$$

Distinguishing between t and τ will help us keep things straight later on when it is not otherwise clear whether I'm talking about a function of rectangular or curvilinear coordinates.

I make no assumptions about the form of equations (7.1) other than that they are smooth enough so one can evaluate partial derivatives and that at any fixed time the inverse transformation exists

$$x_i = x_i(y_1, y_2, y_3, \tau) \tag{7.3}$$

at least in the sub-region of $\mathbf{E}^3$ we are considering.

It will be convenient to define the transformation matrices

$$\mathbf{Y_X} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \frac{\partial y_3}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \frac{\partial y_3}{\partial x_2} \\ \frac{\partial y_1}{\partial x_3} & \frac{\partial y_2}{\partial x_3} & \frac{\partial y_3}{\partial x_3} \end{pmatrix} \tag{7.4}$$

and

$$\mathbf{X_Y} = \begin{pmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_2}{\partial y_1} & \frac{\partial x_3}{\partial y_1} \\ \frac{\partial x_1}{\partial y_2} & \frac{\partial x_2}{\partial y_2} & \frac{\partial x_3}{\partial y_2} \\ \frac{\partial x_1}{\partial y_3} & \frac{\partial x_2}{\partial y_3} & \frac{\partial x_3}{\partial y_3} \end{pmatrix} \tag{7.5}$$

and the so-called Jacobian of the coordinate transformation

$$J = \det(\mathbf{Y_X}) \tag{7.6}$$

where det denotes the determinant function.

Now it turns out that $\mathbf{X_Y}$ and $\mathbf{Y_X}$ are inverses but the reason why is based on an argument involving functions, coordinate representations and the chain rule. Functions are so important to understanding everything that follows that I will take a slight detour here to talk about them in their own section. If you feel comfortable with functions skip over the next section and read it later on if you start to get confused.

7.2 Function Representations and the Chain Rule

A function f defined on $\mathbf{E}^3$ is an abstract concept independent of coordinate systems. To any point $\mathbf{p}$ (also coordinate independent) in $\mathbf{E}^3$, f assigns a value $f(\mathbf{p})$. The value $f(\mathbf{p})$ may be a real number or another point in $\mathbf{E}^3$ — it doesn't matter. So that we can do useful work with them, we talk about the representations of functions in a particular coordinate system. For example, in terms of rectangular coordinates (x_1, x_2, x_3) we might represent f as $f(x_1, x_2, x_3) = x_1 + x_2 + x_3$ which tells us everything we need to know about f.

The problem comes when we have two coordinate systems around. In rectangular coordinates we should represent f by a notation like $f_x(x_1, x_2, x_3)$ and in curvilinear coordinates we should represent f by $f_y(y_1, y_2, y_3)$ because the algebraic form of f_x and f_y are usually completely different. Most authors (including this one) ignore this distinction most of the time because it would only add to the confusion of the notation. However in this section I will maintain the distinction in order to talk about the chain rule.

Let (x_1, x_1, x_3) be rectangular coordinates of a variable point in $\mathbf{E}^3$ and (y_1, y_2, y_3) be the coordinates of the *same* point in curvilinear coordinates. Then by definition

$$f_x(x_1, x_1, x_3) = f_y(y_1(x_1, x_2, x_3, t), y_2(x_1, x_2, x_3, t), y_3(x_1, x_2, x_3, t)) \tag{7.7}$$

and the chain rule for partial derivatives tells us that (summation convention)

$$\frac{\partial f_x}{\partial x_i} = \frac{\partial f_y}{\partial y_j}\frac{\partial y_j}{\partial xi} \tag{7.8}$$

The chain rule will be used a lot in subsequent sections.

From now on I will drop the x and y subscripts on f and we can keep things straight according to whether, in the context of a formula, f is represented in rectangular or curvilinear coordinates. This is almost always obvious except when we begin to think of functions of time as well as position. In this case, if t were used to denote time in both coordinate systems, time partial derivatives would get confusing without the x and y subscripts on f. This is the reason that time is referred to by t and τ in rectangular and curvilinear coordinates respectively.

7.3 Important Properties of the Transformation Matrices

The fact that $\mathbf{X_Y}$ and $\mathbf{Y_X}$ are inverses follows immediately from the chain rule. Note that for any function f on $\mathbf{E}^3$, the chain rule given above in equation (7.8) can be rewritten as

$$\begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{pmatrix} = \mathbf{Y_X} \begin{pmatrix} \frac{\partial f}{\partial y_1} \\ \frac{\partial f}{\partial y_2} \\ \frac{\partial f}{\partial y_3} \end{pmatrix} \tag{7.9}$$

using the customary notation for matrix multiplication. Applying the chain rule in reverse gives

$$\begin{pmatrix} \frac{\partial f}{\partial y_1} \\ \frac{\partial f}{\partial y_2} \\ \frac{\partial f}{\partial y_3} \end{pmatrix} = \mathbf{X_Y} \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{pmatrix} \tag{7.10}$$

Applying the preceding results in succession shows that

$$\mathbf{X_y} \cdot \mathbf{Y_x} \begin{pmatrix} \frac{\partial f}{\partial y_1} \\ \frac{\partial f}{\partial y_2} \\ \frac{\partial f}{\partial y_3} \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial y_1} \\ \frac{\partial f}{\partial y_2} \\ \frac{\partial f}{\partial y_3} \end{pmatrix} \tag{7.11}$$

and since f is arbitrary

$$\mathbf{X_y} \cdot \mathbf{Y_x} = \mathbf{I} \tag{7.12}$$

where $\mathbf{I}$ is the identity matrix. Since $\mathbf{X_y}$ and $\mathbf{Y_x}$ are inverses, it immediately follows from the usual rules on determinants that

$$\det(\mathbf{X_y}) = 1/J \tag{7.13}$$

Later on, I will need to express the $\frac{\partial y}{\partial x}$'s in terms of the $\frac{\partial x}{\partial y}$'s. Writing equation (7.12) columnwise gives

$$\mathbf{X_y} \begin{pmatrix} \frac{\partial y_j}{\partial x_1} \\ \frac{\partial y_j}{\partial x_2} \\ \frac{\partial y_j}{\partial x_3} \end{pmatrix} = \delta_j \tag{7.14}$$

where δ_j is the column vector with 1 in the j^{th} index and 0 in all other indices. Equation (7.14) is a linear system whose solution is given by Cramer's rule as

$$\frac{\partial y_j}{\partial x_i} = J \det(\mathbf{X_y}|_i^j) \tag{7.15}$$

where $\mathbf{X_y}|_i^j$ is the matrix obtained by replacing the i^{th} column of $\mathbf{X_y}$ with δ_j. For example

$$\mathbf{X_y}|_2^1 = \begin{pmatrix} \frac{\partial x_1}{\partial y_1} & 1 & \frac{\partial x_3}{\partial y_1} \\ \frac{\partial x_1}{\partial y_2} & 0 & \frac{\partial x_3}{\partial y_2} \\ \frac{\partial x_1}{\partial y_3} & 0 & \frac{\partial x_3}{\partial y_3} \end{pmatrix} \tag{7.16}$$

and so forth. Although not especially simple, equation (7.15) does give $\frac{\partial y_j}{\partial x_i}$ in terms of the $\frac{\partial x}{\partial y}$'s.

7.4 Basis Vectors

By definition, the position vector $\mathbf{r}$ in $\mathbf{E}^3$ is represented as a function of rectangular coordinates as

$$\mathbf{r}(x_1, x_2, x_3) = x_i \mathbf{e}_i \tag{7.17}$$

where $\mathbf{e}_i$ are the mutually-perpendicular unit basis vectors for rectangular coordinates with which we are all familiar. Note that

$$\mathbf{e}_i = \frac{\partial \mathbf{r}}{\partial x_i} \tag{7.18}$$

According to coordinate transformation (7.3) one can also represent $\mathbf{r}$ as a function of curvilinear coordinates $\mathbf{r}(y_1, y_2, y_3, \tau)$ and define a new set of basis vectors by

$$\mathbf{g}_j = \frac{\partial \mathbf{r}}{\partial y_j} \tag{7.19}$$

This time the $\mathbf{g}_j$ basis vectors depend on position and time and are not necessarily mutually perpendicular or of unit length. It is customary to think of the $\mathbf{g}_j$ as attached to the point of study $\mathbf{r}$ rather than the origin of the coordinate system. Then the vector $\mathbf{g}_j$ is tangent to the y_j coordinate curve passing through $\mathbf{r}$.

There is an important relationship between the $\mathbf{g}_j$ basis vectors and the normals to coordinate surfaces y_i = *constant*. The vectors $\mathbf{g}_2$ and $\mathbf{g}_3$, for example, are tangent to the surface y_1 = *constant* since both the y_2 and y_3 coordinate curves lie in this surface. Therefore $\mathbf{g}_2$ and $\mathbf{g}_3$ are perpendicular to the surface normal to y_1 = *constant* — that is, to the gradient of y_1. Here one thinks of y_1 as a scalar function on $\mathbf{E}^3$. This result holds in general as can be seen from a little algebra. In rectangular coordinates the gradient of y_i has the simple representation

$$\nabla y_i = \frac{\partial y_i}{\partial x_k} \mathbf{e}_k \tag{7.20}$$

But from the chain rule, the vector $\mathbf{g}_j = \frac{\partial \mathbf{r}}{\partial y_j} = \frac{\partial \mathbf{r}}{\partial x_l} \frac{\partial x_l}{\partial y_j}$ or

$$\mathbf{g}_j = \frac{\partial x_l}{\partial y_j} \mathbf{e}_l \tag{7.21}$$

From (7.20) and (7.21), the dot product $\nabla y_i \cdot \mathbf{g}_j = \frac{\partial y_i}{\partial x_k} \frac{\partial x_l}{\partial y_j} \mathbf{e}_k \cdot \mathbf{e}_l = \frac{\partial y_i}{\partial x_k} \frac{\partial x_l}{\partial y_j} \delta_{kl} = \frac{\partial y_i}{\partial x_k} \frac{\partial x_k}{\partial y_j}$, which is just a component in the product matrix $\mathbf{X_Y} \cdot \mathbf{Y_X}$. Using the fact that $\mathbf{Y_X}$ and $\mathbf{X_Y}$ are inverses

$$\nabla y_i \cdot \mathbf{g}_j = \delta_{ij} \tag{7.22}$$

where δ_{ij} is the Kronecker delta ($\delta_{ij} = 1$ if $i = j$ or 0 if $i \neq j$). Note that whereas (7.22) implies that ∇y_i and $\mathbf{g}_j$ are perpendicular for $i \neq j$ it does not imply that they are parallel for $i = j$ (neither ∇y_i nor $\mathbf{g}_j$ are necessarily unit vectors). The sets of vectors ∇y_i and $\mathbf{g}_j$ are sometimes called reciprocal bases because of relation (7.22).

7.5 Vector Fields

A vector field is a vector valued function on $\mathbf{E}^3$. For example, the velocity of a fluid stream is a vector field. In this section $\mathbf{r}$ will denote the position vector in $\mathbf{E}^3$and $\mathbf{A} = \mathbf{A}(\mathbf{r})$ will denote a vector field. The mathematics of vector fields in curvilinear coordinates is covered in any textbook on vector analysis — see for example Newell [12].

In rectangular coordinates, when we represent a vector field $\mathbf{A}$ by (A_1, A_2, A_3) we mean by definition

$$\mathbf{A} = A_i \mathbf{e}_i \tag{7.23}$$

where A_i are scalar functions on $\mathbf{E}^3$. In curvilinear coordinates I will represent $\mathbf{A}$ in terms of components (A^1, A^2, A^3) which will mean by definition

$$\mathbf{A} = A^j \mathbf{g}_j \tag{7.24}$$

where, again, the A^j are scalar functions. The A^j components are called *contravariant* components by some authors, but other authors use the term contravariant to mean vectors in terms of a reciprocal basis. To avoid confusion I shall avoid the term contravariant. I use the superscript index notation for curvilinear components to avoid having to define new symbols every time I wish to talk about the curvilinear components of a vector.

7.5.1 Transformation Rules

Transforming (A_1, A_2, A_3) components to (A^1, A^2, A^3) components and vice-versa is relatively easy. Start with the representation $A_i \mathbf{e}_i = A_i \frac{\partial \mathbf{r}}{\partial x_i}$ in rectangular coordinates. Using the chain rule, $A_i \frac{\partial \mathbf{r}}{\partial x_i} = A_i \frac{\partial \mathbf{r}}{\partial y_j} \frac{\partial y_j}{\partial x_i}$. But by definition, $\mathbf{A} = A^j \frac{\partial \mathbf{r}}{\partial y_j}$. Therefore, for any vector field $\mathbf{A}$

$$A^j = A_i \frac{\partial y_j}{\partial x_i} \tag{7.25}$$

And likewise, it is easy to show that

$$A_i = A^j \frac{\partial x_i}{\partial y_j} \tag{7.26}$$

7.5.2 The Velocity Field

A vector field of central importance is the flow velocity field $\mathbf{V}$. I will use u_i for the components of $\mathbf{V}$ in rectangular coordinates and u^j for the components in curvilinear coordinates. Then the u_i and the u^j are related by (7.25) and (7.26).

7.6 Tensor Fields

A tensor field is a tensor valued function on $\mathbf{E}^3$. Like vectors, tensors are understood to exist independently of coordinate systems although they are ultimately used via their coordinate representations in terms of matrix notation. In fact, I will use the matrix representation of a tensor in rectangular coordinates as its basic definition. There are lots of ways to define and use tensors — see for example Borisenko and Tarapov [5].

Let **A** and **B** be two vector fields having rectangular coordinate representations (A_1, A_2, A_3) and (B_1, B_2, B_3). Then the vector product **AB** is a tensor defined by the matrix representation

$$\mathbf{AB} = \begin{pmatrix} A_1B_1 & A_1B_2 & A_1B_3 \\ A_2B_1 & A_2B_2 & A_2B_3 \\ A_3B_1 & A_3B_2 & A_3B_3 \end{pmatrix} \tag{7.27}$$

This type of vector product was historically known as a ***dyad***. Note that $\mathbf{AB} \neq \mathbf{BA}$ in general. I define a general tensor field **T** in rectangular coordinates by the sum

$$\mathbf{T} = T_{ij}\mathbf{e}_i\mathbf{e}_j \tag{7.28}$$

for arbitrary components T_{ij}, or equivalently

$$\mathbf{T} = \begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix} \tag{7.29}$$

In terms of this representation, tensor operations are formally equivalent to matrix operations — although only in rectangular coordinate systems. It is through operations on tensors — yielding vectors or scalars — that we begin to understand what tensors are.

For example, the product of a tensor field **T** and a vector field **A** yields a vector field $\mathbf{T} \cdot \mathbf{A}$ defined by the following representation in rectangular coordinates

$$\mathbf{T} \cdot \mathbf{A} = T_{ij}A_j\mathbf{e}_i \tag{7.30}$$

$\mathbf{T} \cdot \mathbf{A}$ is just the matrix product of **T** with the column vector **A**.

The ∇ operator $(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_3})$ can also operate on tensors. For example, the divergence of a tensor field **T** yields a vector field $\nabla \cdot \mathbf{T}$ defined by the following representation in rectangular coordinates

$$\nabla \cdot \mathbf{T} = (\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_3}) \cdot \begin{pmatrix} T_{1j} \\ T_{2j} \\ T_{3j} \end{pmatrix} \mathbf{e}_j = \frac{\partial T_{ij}}{\partial x_i}\mathbf{e}_j \tag{7.31}$$

7.6.1 Transformation Rules

A tensor field $\mathbf{T} = T_{ij}\mathbf{e}_i\mathbf{e}_j$ in rectangular coordinates can also be represented in terms of curvilinear coordinates by components T^{kl} defined by

$$\mathbf{T} = T^{kl}\mathbf{g}_k\mathbf{g}_l \tag{7.32}$$

Transforming Tij components to T^{kl} components and vice-versa is done analogously to transformations of vector components. Start with the representation $\mathbf{T} = T_{ij}\mathbf{e}_i\mathbf{e}_j = T_{ij}\frac{\partial \mathbf{r}}{\partial x_i}\frac{\partial \mathbf{r}}{\partial x_j}$ in rectangular coordinates. Using the chain rule, $T_{ij}\frac{\partial \mathbf{r}}{\partial x_i}\frac{\partial \mathbf{r}}{\partial x_j} = T_{ij}\frac{\partial y_k}{\partial x_i}\frac{\partial y_l}{\partial x_j}\frac{\partial \mathbf{r}}{\partial y_k}\frac{\partial \mathbf{r}}{\partial y_l} = T_{ij}\frac{\partial y_k}{\partial x_i}\frac{\partial y_l}{\partial x_j}\mathbf{g}_k\mathbf{g}_l$. But by definition, $\mathbf{T} = T^{kl}\mathbf{g}_k\mathbf{g}_l$. Therefore, for any tensor field $\mathbf{T}$

$$T^{kl} = T_{ij}\frac{\partial y_k}{\partial x_i}\frac{\partial y_l}{\partial x_j} \tag{7.33}$$

And likewise, it is easy to show that

$$T_{ij} = T^{kl}\frac{\partial x_i}{\partial y_k}\frac{\partial x_j}{\partial y_l} \tag{7.34}$$

In general, matrix algebra does not work for tensor operations in curvilinear coordinates. For example, consider the tensor-vector product $\mathbf{T} \cdot \mathbf{A}$ defined in rectangular coordinates by (7.30). Using transformation equations (7.26) and (7.34), $T_{ij}A_j\mathbf{e}_i = T^{kl}\frac{\partial x_i}{\partial y_k}\frac{\partial x_i}{\partial y_l}A^n\frac{\partial x_i}{\partial y_n}\mathbf{e}_i = T^{kl}(\frac{\partial x_i}{\partial y_n}\frac{\partial x_i}{\partial y_l})A^n(\frac{\partial x_i}{\partial y_k}\mathbf{e}_i) =$ $T^{kl}\mathbf{g}_l{\cdot}\mathbf{g}_nA^n\mathbf{g}_k$. Only if $\mathbf{g}_l{\cdot}\mathbf{g}_n = \delta_{ln}$ will $\mathbf{T} \cdot \mathbf{A}$ reduce to $T^{kl}A^l\mathbf{g}_k$. That is, matrix algebra will work only if the curvilinear basis vectors form an orthonormal set. Subsequent sections will derive transformation rules for some important vector and tensor operations in curvilinear coordinates.

7.7 Transforming the Divergence of a Vector Field

Let $\mathbf{A}$ be a vector field with coordinates (A_1, A_2, A_3) in rectangular coordinates and (A^1, A^2, A^3) in curvilinear coordinates. Using the definition of the divergence in rectangular coordinates, $\nabla \cdot \mathbf{A} = \frac{\partial A_i}{\partial x_i}$. Writing A_i in terms of curvilinear coordinates using transformation rule (7.26) gives

$$\nabla \cdot \mathbf{A} = \frac{\partial}{\partial x_i}(A^j\frac{\partial x_i}{\partial y_j}) \tag{7.35}$$

and using the chain rule

$$\nabla \cdot \mathbf{A} = \frac{\partial y_k}{\partial x_i}\frac{\partial}{\partial y_k}(A^j\frac{\partial x_i}{\partial y_j}) \tag{7.36}$$

Now replace $\frac{\partial y_k}{\partial x_i}$ with its equivalent in terms of curvilinear coordinates using (7.15) to obtain

$$\nabla \cdot \mathbf{A} = J\det(\mathbf{Xy}|_i^k)\frac{\partial}{\partial y_k}(A^j\frac{\partial x_i}{\partial y_j}) \tag{7.37}$$

Using the product rule in the form $u\,dv = d(uv) - v\,du$ this can be expanded to

$$\nabla \cdot \mathbf{A} = J\left(\frac{\partial}{\partial y_k}(\det(\mathbf{Xy}|_i^k)\frac{\partial x_i}{\partial y_j}A^j) - A^j\frac{\partial x_i}{\partial y_j}\frac{\partial}{\partial y_k}(\det(\mathbf{Xy}|_i^k)\right) \tag{7.38}$$

Now the second term in the large brackets drops out, since with a good deal of algebra $\frac{\partial}{\partial y_k}(\det(\mathbf{Xy}|_i^k))$ can be shown to equal zero. The first term simplifies by noting that $\det(\mathbf{Xy}|_i^k)\frac{\partial x_i}{\partial y_j}A^j = \frac{1}{J}\frac{\partial y_k}{\partial x_i}\frac{\partial x_i}{\partial y_j}A^j = \frac{1}{J}\delta_{jk}A^j = \frac{1}{J}A^k$. The final result is

$$\nabla \cdot \mathbf{A} = J\frac{\partial}{\partial y_k}(\frac{1}{J}A^k) \tag{7.39}$$

7.8 Transforming the Divergence of a Tensor Field

Let $\mathbf{T}$ be a tensor field with components T_{ij} in rectangular coordinates and T^{kl} in curvilinear coordinates. In terms of its rectangular components, the divergence of $\mathbf{T}$ was defined earlier by equation (7.31). Note that the factor $\frac{\partial T_{ij}}{\partial x_i}$ appearing on the right side of (7.31) can be written in rectangular coordinates as the vector divergence $\nabla \cdot (T_{1j}, T_{2j}, T_{3j})$ which, according to (7.39) and (7.25) is equal to $J\frac{\partial}{\partial y_k}(\frac{1}{J}T_{ij}\frac{\partial y_k}{\partial x_i})$. It follows then that

$$\nabla \cdot \mathbf{T} = J\frac{\partial}{\partial y_k}(\frac{1}{J}T_{ij}\frac{\partial y_k}{\partial x_i})\mathbf{e}_j \tag{7.40}$$

Now since $\mathbf{e}_j$ is constant it can be brought into the parenthesis. But $\mathbf{e}_j = \frac{\partial \mathbf{r}}{\partial x_j} = \frac{\partial y_l}{\partial x_j}\frac{\partial \mathbf{r}}{\partial y_l} = \frac{\partial y_l}{\partial x_j}\mathbf{g}_l$, so that

$$\nabla \cdot \mathbf{T} = J\frac{\partial}{\partial y_k}(\frac{1}{J}T_{ij}\frac{\partial y_k}{\partial x_i}\frac{\partial y_l}{\partial x_j}\mathbf{g}_l) \tag{7.41}$$

And using the transformation rule (7.33), the final result is

$$\nabla \cdot \mathbf{T} = J\frac{\partial}{\partial y_k}(\frac{1}{J}T^{kl}\mathbf{g}_l) \tag{7.42}$$

7.9 Transforming Terms of the Form $\frac{\partial f}{\partial t}$

This is the first section in which the time-dependence of the basic transformation equations (7.1) is explicitly important. All the previous results are valid for time-dependent transformations; the time terms just dropped out so they weren't emphasized.

Let f represent a scalar- or vector-valued function on $\mathbf{E}^3$ with representations $f(x_1, x_2, x_3, t)$ in rectangular coordinates and $f(y_1, y_2, y_3, \tau)$ in curvilinear coordinates. Then, using the chain rule, $\frac{\partial f}{\partial t}$ can be transformed to

$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial \tau} + \frac{\partial f}{\partial y_j}\frac{\partial y_j}{\partial t} \tag{7.43}$$

Note that if the coordinate transformation (7.1) is time-independent then $\frac{\partial y_i}{\partial t} = 0$ and $\frac{\partial f}{\partial t} = \frac{\partial f}{\partial \tau}$.

For time-dependent transformations one can put (7.43) into strong conservative form as follows. The first task is to transform $\frac{\partial y_i}{\partial t}$ to curvilinear coordinates. Using the chain rule in the form $\frac{\partial y_i}{\partial \tau} = \frac{\partial y_i}{\partial t} + \frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial \tau}$, noting that $\frac{\partial y_i}{\partial \tau} = 0$ (y_j is not a function of τ) and solving one obtains

$$\frac{\partial y_j}{\partial t} = -\frac{\partial y_j}{\partial x_i}\frac{\partial x_i}{\partial \tau} \tag{7.44}$$

Here one looks at y_j alternately as a function of curvilinear and rectangular coordinates. If you're confused at this point try re-reading section 7.2. Using (7.15) one can substitute for $\frac{\partial y_i}{\partial x_i}$ giving

$$\frac{\partial y_j}{\partial t} = -J \det(\mathbf{X_y}|^j_i)\frac{\partial x_i}{\partial \tau} \tag{7.45}$$

Substituting this into (7.43) gives

$$\frac{\partial f}{\partial t} = J[\frac{1}{J}\frac{\partial f}{\partial \tau} - \det(\mathbf{X_y}|^j_i)\frac{\partial x_i}{\partial \tau}\frac{\partial f}{\partial y_j}] \tag{7.46}$$

To put this into strong conservative form, note that $\frac{1}{J}\frac{\partial f}{\partial \tau} = \frac{\partial}{\partial \tau}(f/J) - f\frac{\partial}{\partial \tau}(1/J)$ which is obvious, and

$$\begin{aligned}\det(\mathbf{X_y}|^j_i)\frac{\partial x_i}{\partial \tau}\frac{\partial f}{\partial y_j} &= \frac{\partial}{\partial y_j}\left(\det(\mathbf{X_y}|^j_i)\frac{\partial x_i}{\partial \tau}f\right) - \\ &\quad f\left(\frac{\partial x_i}{\partial \tau}\frac{\partial}{\partial y_j}(\det(\mathbf{X_y}|^j_i)) + \det(\mathbf{X_y}|^j_i)\frac{\partial^2 x_i}{\partial y_j \partial \tau}\right) \\ &= \frac{\partial}{\partial y_j}\left(\det(\mathbf{X_y}|^j_i)\frac{\partial x_i}{\partial \tau}f\right) - f\frac{\partial}{\partial \tau}(1/J)\end{aligned}$$

which isn't obvious at all and requires a great deal of algebra to verify. Putting all this together gives

$$\frac{1}{J}\frac{\partial f}{\partial t} = \frac{\partial}{\partial \tau}(f/J) - \frac{\partial}{\partial y_j}\left(\det(\mathbf{X_y}|^j_i)\frac{\partial x_i}{\partial \tau}f\right) \tag{7.47}$$

And if we are willing to mix coordinates, (7.45) can be re-introduced into (7.47) giving

$$\frac{1}{J}\frac{\partial f}{\partial t} = \frac{\partial}{\partial \tau}(f/J) + \frac{\partial}{\partial y_j}\left(\frac{1}{J}\frac{\partial y_j}{\partial t}f\right) \tag{7.48}$$

7.10 Integrals in Curvilinear Coordinates

After solving a gasdynamic problem in curvilinear coordinates one is generally interested in quantities such as total flux of energy across a particular boundary or net change of mass in some region. These quantities require numerical surface or volume integrals respectively. Since the boundary of the solution region is, by design, generally comprised of curvilinear coordinate surfaces (y_k = *constant*) it will be most natural to perform these integrals in curvilinear coordinates. Here are some results that will help out.

7.10.1 Volume Integrals

Volume integrals in curvilinear coordinates are covered in any advanced calculus textbook — see for example Taylor [16]. Since the following result is so well documented elsewhere, I'll just quote it for future reference. Let $\mathcal{R}$ denote some physical region in $\mathbf{E}^3$ and f a function defined on $\mathcal{R}$. Then

$$\int_{\mathcal{R}} f\, dv_x = \int_{\mathcal{R}} \frac{1}{|J|} f\, dv_y \tag{7.49}$$

where $dv_x = dx_1 dx_2 dx_3$ is the volume element in rectangular coordinates, $dv_y = dy_1 dy_2 dy_3$ is the volume element in curvilinear coordinates and J is the transformation Jacobian defined by (7.6).

7.10.2 Integrals of Flux Across Boundaries

Surface integrals in curvilinear coordinates are a lot messier than volume integrals except in the special case where the boundary $\mathcal{B}$ of region $\mathcal{R}$ is comprised of curvilinear coordinate surfaces (y_k = *constant*). But, after all, this is the point of boundary-fitted coordinates in the first place, so there is no problem with this restriction.

In coordinate-free form, the flux-integral of a vector field A across boundary $\mathcal{B}$ is defined as

$$\int_{\mathcal{B}} (\mathbf{A} \cdot \mathbf{n}) ds$$

where $\mathbf{n}$ is the outward pointing unit normal to the surface and ds denotes the surface-area element. Using the divergence theorem and the above result for the transformation of volume integrals $\int_{\mathcal{B}} (\mathbf{A} \cdot \mathbf{n}) ds = \int_{\mathcal{R}} (\nabla \cdot \mathbf{A})\, dv_x =$ $\int_{\mathcal{R}} \frac{1}{|J|} (\nabla \cdot \mathbf{A})\, dv_y$. Then equation (7.39) for the divergence in curvilinear coordinates gives the intermediate result

$$\int_{\mathcal{B}} (\mathbf{A} \cdot \mathbf{n}) ds = \int_{\mathcal{R}} \frac{\partial}{\partial y_k} \left(\frac{1}{|J|} \mathbf{A}^k \right) dv_y \tag{7.50}$$

Now assume the boundary $\mathcal{B}$ is made up of curvilinear coordinates surfaces. Define a positive y_k surface as a part of the boundary defined by $y_k = b$ such that if $a < b$ then the surface $y_k = a$ passes through the interior of region $\mathcal{R}$, at least for a sufficiently close to b. Define a negative y_k surface in a similar manner except in the opposite sense. Then $\int_{\mathcal{R}} \frac{\partial}{\partial y_k}(\frac{1}{|J|}\mathbf{A}^k)\, dv_y$ reduces to the sum of surface integrals on the y_k surfaces so that (7.50) can be written in the final form

$$\int_{\mathcal{B}} (\mathbf{A} \cdot \mathbf{n}) ds = \int_{\mathcal{B}} \left(\frac{1}{|J|} \mathbf{A}^k \right) \eta_k \, ds_y \tag{7.51}$$

where

$$\eta_k = \begin{cases} 1 & \text{on positive } y_k \text{ surface} \\ -1 & \text{on negative } y_k \text{ surface} \\ 0 & \text{on } y_l \text{ surface, } l \neq k \end{cases} \tag{7.52}$$

and

$$ds_y = \begin{cases} dy_2 dy_3 & \text{on } y_1 \text{ surface} \\ dy_1 dy_3 & \text{on } y_2 \text{ surface} \\ dy_1 dy_2 & \text{on } y_3 \text{ surface} \end{cases} \tag{7.53}$$

Chapter 8

The Porous Flow Equations

This chapter develops a set of modified Navier-Stokes equations with special terms pertaining to flow through porous materials. The results of chapter 7 are then used to represent these equations in general curvilinear coordinates.

The general Navier-Stokes equations require modification in order to address the computational reality that the particle size in a porous bed is generally much smaller than a practical computational mesh size. In philosophy at least, porous flow modeling has much in common with turbulence modeling — both attempt to deal with sub-grid-scale phenomena using various approximations.

8.1 The General Navier-Stokes equations

The general Navier-Stokes equations in conservative form and in coordinate-independent notation are as follows. For a very thorough development of the Navier-Stokes equations see Schlichting [15]. For a development more oriented toward computation than basic physics see Peyret and Taylor [13] or Anderson, Tannehill and Pletcher [1]. The equations presented here assume there are no external forces.

Continuity

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0 \tag{8.1}$$

Momentum

$$\frac{\partial}{\partial t}(\rho \mathbf{V}) + \nabla \cdot (\rho \mathbf{V}\mathbf{V} - \boldsymbol{\sigma}) = 0 \tag{8.2}$$

Energy

$$\frac{\partial}{\partial t}(\rho e) + \nabla \cdot (\rho e \mathbf{V} - \boldsymbol{\sigma} \cdot \mathbf{V} + \mathbf{q}) = 0 \tag{8.3}$$

where

ρ	=	density scalar field
t	=	time
$\mathbf{V}$	=	velocity vector field
$\boldsymbol{\sigma}$	=	stress tensor field
e	=	$\varepsilon + \frac{1}{2}\mathbf{V}\cdot\mathbf{V}$ mass specific total energy
ε	=	mass specific internal energy
		$\varepsilon = c_v T$ for an ideal gas
T	=	temperature scalar field
$\mathbf{q}$	=	$-\kappa\nabla T$ heat-flux vector field
κ	=	conductivity

Here, the $\nabla\cdot$ operator is used to indicate the divergence of a vector or tensor field, both of which are concepts independent of coordinates even though $\nabla\cdot$ is most easily understood in terms of its rectangular coordinate representations. Likewise, tensors (dyads) such as $\mathbf{VV}$ and tensor-vector products such as $\boldsymbol{\sigma}\cdot\mathbf{V}$ are coordinate independent concepts even though the notation strongly suggests a representation in terms of matrix multiplication in rectangular coordinates.

8.2 Porous-Flow Formulation

The starting point for deriving the porous-flow equations will be the integral form of the preceding Navier-Stokes equations presented in reference [13], section 1.1. However, first I will review the mathematical framework for decomposing quantities into local averages and fluctuating parts.

8.2.1 Averaged Quantities

Following somewhat the conventions established in the porous-flow literature (for example, reference [4]) I define a representative elemental volume $\mathcal{C}$ to be a cube aligned with the coordinate axes whose side dimension, $2dx$, is much larger than the pore size but much smaller than the characteristic length of the overall matrix. $\mathcal{C}$ is the finite-difference equivalent of the infinitesimal volume element commonly used to derive so-called *continuum* differential equations. The factor of 2 in the side dimension will make sense later in terms of centered finite-difference formulae and the staggered-grid solution method. The fact that $\mathcal{C}$ is a cube, or that it is aligned with the coordinate axes, are not strictly necessary assumptions but convenient simplifications. The volume element may be moved arbitrarily about the matrix; sometimes I use the notation $\mathcal{C}(\mathbf{r})$ to denotes that $\mathcal{C}$ is centered at the point $\mathbf{r}$. The faces of $\mathcal{C}$ are comprised of elemental squares oriented in three basic directions. These squares are denoted $\mathcal{S}_i$, $i = 1\ldots3$, where the i subscript refers to the coordinate direction normal to the surface; the notation $\mathcal{S}_i(\mathbf{r})$ denotes that $\mathcal{S}_i$ is centered at the point $\mathbf{r}$. See figure 8.1.

The void volume within $\mathcal{C}$ is denoted $\mathcal{V}$. The surface $\mathcal{S}_v$ of $\mathcal{V}$ is broken into two disjoint pieces: $\mathcal{S}_{v_1}$, the part intersecting the faces of $\mathcal{C}$, and $\mathcal{S}_{v_2}$, the interior

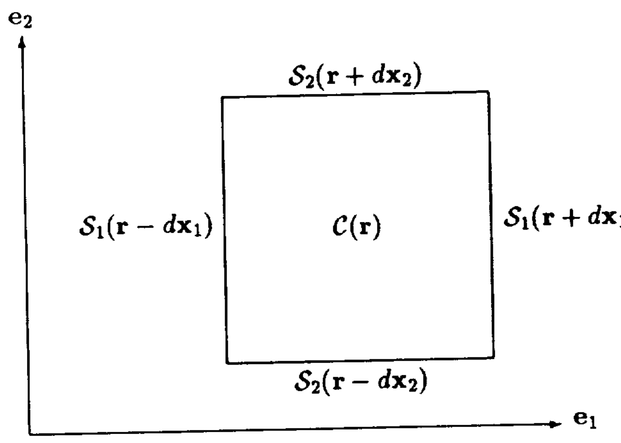

e_2
$\mathcal{S}_2(\mathbf{r}+d\mathbf{x}_2)$
$\mathcal{S}_1(\mathbf{r}-d\mathbf{x}_1)$
$\mathcal{C}(\mathbf{r})$
$\mathcal{S}_1(\mathbf{r}+d\mathbf{x}_1)$
$\mathcal{S}_2(\mathbf{r}-d\mathbf{x}_2)$
e_1

Figure 8.1: Cross section of a representative elemental volume centered at point $\mathbf{r}$, and its faces. $\mathbf{e}_i$ are the rectangular coordinate basis vectors and $d\mathbf{x}_i = dx\,\mathbf{e}_i$.

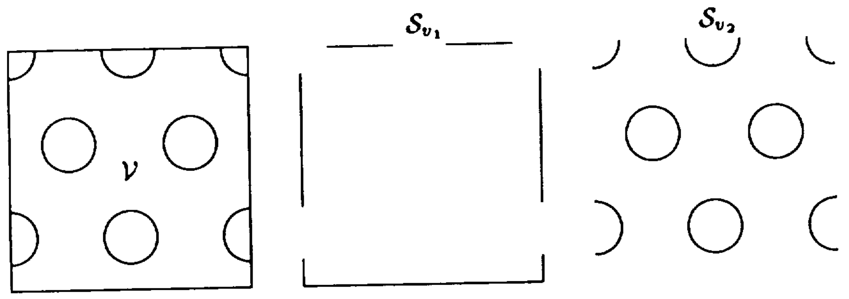


Figure 8.2: The void part $\mathcal{V}$ of a typical representative elemental volume showing its surface components $\mathcal{S}_v = \mathcal{S}_{v_1} \cup \mathcal{S}_{v_2}$.

part. This is illustrated in figure 8.2.

The idea behind the representative volume element is that the average of any gasdynamic quantity over $\mathcal{C}$ or $\mathcal{V}$ removes sub-grid-scale inhomogeneity without eliminating macroscopic inhomogeneity. The problem now is how to reformulate the Navier-Stokes equations in terms of averaged variables. But first some preliminaries.

Basic Definitions

Let $G(\mathbf{r})$ represent any gasdynamic quantity such as density, velocity, etc. Since gasdynamic quantities are only defined in the void part of the matrix, I assume G is zero in the solid portion. The notation $\overline{G}^c(\mathbf{r})$ will denote the average of G

over the volume element $\mathcal{C}(\mathbf{r})$. That is

$$\overline{G}^c(\mathbf{r}) = (1/\mathcal{C}) \int_{\mathcal{C}(\mathbf{r})} G dv \tag{8.4}$$

where $\mathcal{C}$ denotes the volume of $\mathcal{C}(\mathbf{r})$. A similar average, this time taken over only the void part of $\mathcal{C}$ is denoted $\overline{G}$.

$$\overline{G}(\mathbf{r}) = (1/\mathcal{V}) \int_{\mathcal{V}(\mathbf{r})} G dv \tag{8.5}$$

Since the integrals are equal in the two expressions, it is clear that

$$\overline{G}^c = \beta \overline{G} \tag{8.6}$$

where

$$\beta(\mathbf{r}) = \mathcal{V}/\mathcal{C} \tag{8.7}$$

is the matrix porosity. Note that I am allowing β to be a function of $\mathbf{r}$, not necessarily a constant.

Volume vs Surface Averages

A question that will come up is: How are averages over the elemental surfaces $\mathcal{S}_i$ related to volume averages? I assume that surface averages are carried out in the ensemble sense — on all possible realizations of the porous matrix. This randomization effectively smooths out any variability in the surface void fraction for any particular matrix realization. Also under this convention, the orientation of the elemental surface is unimportant. Then, the average of a gasdynamic quantity G over an elemental surface centered on a point $\mathbf{r}$ is the same as $\overline{G}^c(\mathbf{r})$. This follows because volume integrals are nothing more than surface integrals swept through space. Also, the ensemble-average void fraction of $\mathcal{S}_i$ is the same as the volumetric void fraction of the matrix. From this it follows that the average of G over the void part of a surface $\mathcal{S}_i$ centered on a point $\mathbf{r}$ is the same as $\overline{G}(\mathbf{r})$.

Microscopic Deviations

For a fixed point $\mathbf{r}_0$, the microscopic deviation of G from $\overline{G}(\mathbf{r}_0)$ will be denoted by $\acute{G}_{r_0}$. That is

$$\acute{G}_{r_0}(\mathbf{r}) = G(\mathbf{r}) - \overline{G}(\mathbf{r}_0) \tag{8.8}$$

Note that $\acute{G}$ is a family of functions — one for each point of application $\mathbf{r}_0$. As $\mathbf{r}$ deviates more and more from $\mathbf{r}_0$, $\acute{G}_{r_0}$ deviates more and more from G. But this will not be important since $\acute{G}_{r_0}$ will only be used within a *small* representative volume or surface element. An important property of $\acute{G}$ is that

$$\overline{\acute{G}_r}(\mathbf{r}) = 0 \tag{8.9}$$

which follows immediately from the definition since

$$\begin{aligned}\overline{\acute{G}_r}(\mathbf{r}) &= \int_{\mathcal{V}(\mathbf{r})} (G(\cdot) - \overline{G}(\mathbf{r}))\, dv \\ &= \overline{G}(\mathbf{r}) - \overline{G}(\mathbf{r})\end{aligned}$$

In the above integral, the notation $(\cdot)$ denotes the integration variable.

At this point it may not be clear why $\acute{G}$ is based on the void-average instead of the total volume average. The reason is that it makes more physical sense this way. Consider density ρ for example. In an incompressible fluid, $\acute{\rho} = 0$ as defined above — as expected. Were the definition based on the total volume, $\acute{\rho}$ would be non-zero because $\overline{\rho}^c$ is smaller than ρ owing to the zero-weight fluid density within the solid part of the matrix.

Averaging Sums and Products

What follows are two useful properties for sums and products of void-averaged quantities, that will come in handy when deriving the porous Navier-Stokes equations. Although the results are derived for volume averages, by the remarks earlier, they apply also to surface averages. Let F and G be two gasdynamic quantities. First, it follows immediately from the additive property of integrals that

$$\overline{F+G} = \overline{F} + \overline{G} \tag{8.10}$$

A slightly more complicated result applies to products. Note that for any point $\mathbf{r}$

$$\begin{aligned}\overline{FG}(\mathbf{r}) &= 1/\mathcal{V} \int_{\mathcal{V}(\mathbf{r})} FG\, dv \\ &= 1/\mathcal{V} \int_{\mathcal{V}(\mathbf{r})} \left(\overline{F}(\mathbf{r}) + \acute{F}_r(\cdot)\right)\left(\overline{G}(\mathbf{r}) + \acute{G}_r(\cdot)\right) dv \\ &= 1/\mathcal{V} \int_{\mathcal{V}(\mathbf{r})} \left(\overline{F}(\mathbf{r})\overline{G}(\mathbf{r}) + \overline{F}(\mathbf{r})\acute{G}_r(\cdot) + \acute{F}_r(\cdot)\overline{G}(\mathbf{r}) + \acute{F}_r(\cdot)\acute{G}_r(\cdot)\right) dv\end{aligned}$$

Factoring the constants $\overline{F}(\mathbf{r})$ and $\overline{G}(\mathbf{r})$ out of the integrals and applying equation (8.9) to the middle two terms gives the final result

$$\overline{FG} = \overline{F}\,\overline{G} + \overline{\acute{F}\acute{G}} \tag{8.11}$$

8.2.2 Continuity Equation

The integral form of the continuum continuity equation applied to volume $\mathcal{V}$ centered on a point $\mathbf{r}$ is

$$\frac{\partial}{\partial t}\int_{\mathcal{V}} \rho\, dv + \int_{S_{v_1}} \rho\mathbf{V}\cdot\mathbf{n}\, ds + \int_{S_{v_2}} \rho\mathbf{V}\cdot\mathbf{n}\, ds = 0 \tag{8.12}$$

where $\mathbf{n}$ is the outward-pointing unit surface normal. The integral in the first term is by definition $\mathcal{V}\overline{\rho} = \beta\mathcal{C}\overline{\rho}$. The second integral may be simplified by breaking $\mathcal{S}_{v_1}$ into pieces lying on the six faces $\mathcal{S}_i$ of $\mathcal{C}$ and using the $\overline{}$ notation to represent each one. Note that on $\mathcal{S}_i$, $\mathbf{n} = \pm\mathbf{e}_i$, the i^{th} rectangular basis vector, so that $\mathbf{V}\cdot\mathbf{n} = \pm u_i$, the i^{th} velocity component. The third integral vanishes since $\mathbf{V}\cdot\mathbf{n} = 0$ on $\mathcal{S}_{v_2}$ because of the solid impermeability. Putting all this together gives

$$\mathcal{C}\frac{\partial}{\partial t}(\beta\overline{\rho}(\mathbf{r})) + \sum_i \mathcal{S}\,[\beta\overline{\rho u_i}]_{\mathbf{r}-d\mathbf{x}_i}^{\mathbf{r}+d\mathbf{x}_i} = 0 \tag{8.13}$$

where $\mathcal{S}$ denotes the area of surface element $\mathcal{S}_i$, $d\mathbf{x}_i = dx\,\mathbf{e}_i$ and the expression in square brackets symbolizes, for example, $[F]_a^b = F(b) - F(a)$. Dividing through by $\mathcal{C}$ gives

$$\frac{\partial}{\partial t}(\beta\overline{\rho}(\mathbf{r})) + \sum_i \frac{1}{2dx}\,[\beta\overline{\rho u_i}]_{\mathbf{r}-d\mathbf{x}_i}^{\mathbf{r}+d\mathbf{x}_i} = 0 \tag{8.14}$$

The sum term is nothing more than the centrally-differenced finite-difference divergence of the quantity in square brackets; using the $\nabla\cdot$ operator to replace the summation notation simplifies the notation greatly

$$\frac{\partial}{\partial t}(\beta\overline{\rho}) + \nabla\cdot(\beta\overline{\rho\mathbf{V}}) = 0 \tag{8.15}$$

The fact that the $\nabla\cdot$ operator now refers to a macroscopic volume instead of the customary infinitesimal one will cause no confusion. After all, the customary infinitesimal $\nabla\cdot$ used in the *continuum* Navier-Stokes equations is really no better since it actually refers to a volume element much larger than the molecular scale — certainly not *infinitesimal.* I am simply enlarging the scale a bit further.

8.2.3 Momentum Equation

The integral form of the continuum momentum equation applied to volume $\mathcal{V}$ centered on a point $\mathbf{r}$ is

$$\frac{\partial}{\partial t}\int_{\mathcal{V}} \rho\mathbf{V}\,dv + \int_{\mathcal{S}_{v_1}} ((\mathbf{n}\cdot\mathbf{V})\rho\mathbf{V})\,ds - \int_{\mathcal{S}_v} \mathbf{n}\cdot\boldsymbol{\sigma}\,ds = 0 \tag{8.16}$$

This equation is based on Newton's second law: *force = mass × acceleration.* The first term is the local contribution to *mass × acceleration* while the surface integral of $((\mathbf{n}\cdot\mathbf{V})\rho\mathbf{V})ds$ over $\mathcal{S}_{v_1}$ is the advected contribution. The term $(\mathbf{n}\cdot\mathbf{V})\rho\mathbf{V}$ is zero on the solid-boundary surface $\mathcal{S}_{v_2}$ because of the solid impermeability. The vector $\mathbf{n}\cdot\boldsymbol{\sigma}$ is the matrix product of the unit surface-normal row-vector $\mathbf{n}$ with the stress tensor $\boldsymbol{\sigma}$. The surface integral of $\mathbf{n}\cdot\boldsymbol{\sigma}\,ds$ over the total void surface $\mathcal{S}_v$ contributes the *force* part in Newton's law. A net force results from thermodynamic pressure and viscous stresses on $\mathcal{S}_v$.

To make possible some porous-flow approximations, I chose to regroup equation (8.16) as

$$\frac{\partial}{\partial t}\int_{\mathcal{V}} \rho\mathbf{V}\,dv + \int_{\mathcal{S}_{v_1}} ((\mathbf{n}\cdot\mathbf{V})\rho\mathbf{V} - \mathbf{n}\cdot\boldsymbol{\tau})\,ds + \int_{\mathcal{S}_v} \mathbf{n}\cdot P\mathbf{I}\,ds - \int_{\mathcal{S}_{v_2}} \mathbf{n}\cdot\boldsymbol{\tau}\,ds = 0 \quad (8.17)$$

Here, the total stress tensor has been decomposed into

$$\boldsymbol{\sigma} = -P\mathbf{I} + \boldsymbol{\tau} \quad (8.18)$$

where P is the thermodynamic pressure, $\mathbf{I}$ is the identity tensor and $\boldsymbol{\tau}$ is the viscous stress tensor.

The First Term

The integral in the first term of equation (8.17) is by definition $\mathcal{V}\overline{\rho\mathbf{V}} = \beta\mathcal{C}\,\overline{\rho\mathbf{V}}$ so that

$$\frac{\partial}{\partial t}\int_{\mathcal{V}} \rho\mathbf{V}\,dv = \mathcal{C}\frac{\partial}{\partial t}(\beta\overline{\rho\mathbf{V}}) \quad (8.19)$$

The Second Term

The second integral of equation (8.17) may be simplified by breaking the integral over $\mathcal{S}_{v_1}$ into pieces lying on the six faces $\mathcal{S}_i$ of $\mathcal{C}$ then applying the sum rule (8.10) and product rule (8.11). Remember that on $\mathcal{S}_i$, $\mathbf{n} = \pm\mathbf{e}_i$ and $\mathbf{n}\cdot\mathbf{V} = \pm u_i$, the i^{th} rectangular basis vector and velocity vector components. This gives

$$\int_{\mathcal{S}_{v_1}} ((\mathbf{n}\cdot\mathbf{V})\rho\mathbf{V} - \mathbf{n}\cdot\boldsymbol{\tau})\,ds = \sum_i \mathcal{S}\left[\beta\left(\overline{u_i}\,\overline{\rho\mathbf{V}} + \overline{\acute{u}_i(\acute{\rho\mathbf{V}})} - \overline{\mathbf{e}_i\cdot\boldsymbol{\tau}}\right)\right]_{\mathbf{r}-d\mathbf{x}_i}^{\mathbf{r}+d\mathbf{x}_i} \quad (8.20)$$

Rewriting in terms of the finite-difference divergence operator gives

$$\int_{\mathcal{S}_{v_1}} ((\mathbf{n}\cdot\mathbf{V})\rho\mathbf{V} - \mathbf{n}\cdot\boldsymbol{\tau})\,ds = \mathcal{C}\,\nabla\cdot\beta\left(\overline{\rho\mathbf{V}}\,\overline{\mathbf{V}} + \overline{(\acute{\rho\mathbf{V}})\acute{\mathbf{V}}} - \overline{\boldsymbol{\tau}}\right) \quad (8.21)$$

where the vector product notation (for example: $\overline{\rho\mathbf{V}}\,\overline{\mathbf{V}}$) represents matrix multiplication of a column vector times a row vector — yielding a tensor. Like Reynold's apparent stress term of turbulent flow, the term $(\acute{\rho\mathbf{V}})\acute{\mathbf{V}}$ cannot be ignored since $(\acute{\rho\mathbf{V}})$ and $\acute{\mathbf{V}}$ are both significant and correlated. More on the meaning of this term later, but for the moment, I just leave it as is.

The Third Term

The third integral in equation (8.17) is the net thermodynamic pressure force acting on the total surface of $\mathcal{V}$. One can, perhaps, best understand this term by reflecting on a simple problem in hydrostatics. In a motionless fluid ($\mathbf{V} = 0$)

subject to a gravitational field, it is well known that pressure increases with depth according to

$$\nabla P = \rho \mathbf{a} \tag{8.22}$$

where $\mathbf{a}$ is the acceleration of gravity. Certainly this equation remains valid in the presence of a porous matrix. On the other hand, a simple force balance on volume $\mathcal{V}$ under these conditions gives

$$\int_{S_v} \mathbf{n} \cdot P\mathbf{I}\, ds = \int_{\mathcal{V}} \rho \mathbf{a}\, dv = \beta \mathcal{C} \mathbf{a} \overline{\rho} \tag{8.23}$$

That is: The net pressure force on its surface is balanced by the total external gravitational force on $\mathcal{V}$. Combining this with (8.22) gives

$$\int_{S_v} \mathbf{n} \cdot P\mathbf{I}\, ds = \beta \mathcal{C} \nabla \overline{P} \tag{8.24}$$

The fact that $\nabla \overline{P}$ is caused by a gravitational field is irrelevant to (8.24). The same result can be derived by purely mathematical arguments based on a careful accounting of the integral on the left.

The Fourth Term

Finally, the last integral in equation (8.17) is the total drag force in an elemental volume $\mathcal{C}$ arising from the viscous stresses between the gas and the solid matrix — an experimentally measurable quantity. I assume

$$\int_{S_{v_2}} \mathbf{n} \cdot \boldsymbol{\tau}\, ds = -\beta \mathcal{C} (\boldsymbol{\phi} \cdot \overline{\mathbf{V}}) \tag{8.25}$$

where $\boldsymbol{\phi}$ is an empirical permeability-related tensor.

To justify approximation (8.25), consider the momentum equation (8.17) in the case of uni-directional, steady incompressible flow in a matrix of uniform porosity. Under these conditions, the first term, obviously, drops out. The second term also drops out as can be seen from equation (8.21) where $\overline{\rho \mathbf{V}}\, \overline{\mathbf{V}}$ is constant, since $\overline{\mathbf{V}}$ is constant, and both $\overline{(\rho \mathbf{V})' \mathbf{V}'}$ and $\overline{\boldsymbol{\tau}}$ are constant by symmetry arguments in the ensemble-average sense. Approximating the third term using (8.24), the momentum equation simplifies to

$$\int_{S_{v_2}} \mathbf{n} \cdot \boldsymbol{\tau}\, ds = \beta \mathcal{C}\, \nabla \overline{P} \tag{8.26}$$

Now, there are various ways to approximate the pressure gradient $\nabla \overline{P}$ in uni-directional, steady incompressible flow. A widely used correlation is the Ergun equation [11] which assumes a friction factor f of the form

$$f = \frac{c_1}{Re} + c_2 \tag{8.27}$$

where Re is the flow Reynolds number, and c_1 and c_2 are two constants depending on the porous material structure but where always $c_1 \gg c_2$. Therefore, for low Reynolds numbers — which occur in many porous flow cases of interest — the frictional pressure gradient works out to

$$\begin{aligned} \nabla\overline{P} &= -(f/d_h)\rho V\overline{\mathbf{V}}/2 \\ &\approx -\frac{\mu c_1}{2d_h^2}\overline{\mathbf{V}} \end{aligned} \tag{8.28}$$

at least for isotropic materials. Here d_h is the hydraulic diameter, $V = |\overline{\mathbf{V}}|$ and μ is the fluid viscosity. Approximation (8.25) follows by combining equations (8.26) and (8.28) then replacing $\frac{\mu c_1}{2d_h^2}\overline{\mathbf{V}}$ with the more general form $\boldsymbol{\phi}\cdot\overline{\mathbf{V}}$.

The reason $\boldsymbol{\phi}$ is a tensor is because the tensor product $\boldsymbol{\phi}\cdot\mathbf{V}$ allows non-isotropic flow resistance to be modeled. In the three-dimensional case, three different resistance coefficients can be defined in each of three principal directions. Usually the porous matrix will be situated so the principal directions coincide with the directions of the rectangular basis vectors $\mathbf{e}_i$. The detailed representation of the porosity tensor $\boldsymbol{\phi}$ in rectangular coordinates will be given in section 8.3.

For higher Reynolds number flow, where c_2 is no longer negligible compared to c_1/Re, one can use the complete Ergun friction factor to correlate pressure gradient as

$$\nabla\overline{P} = -\frac{\mu c_1}{2d_h^2}(1+\epsilon)\overline{\mathbf{V}} \tag{8.29}$$

where ϵ is a perturbation given by

$$\epsilon = \frac{c_2}{c_1}Re \tag{8.30}$$

Again, it is reasonable to represent pressure gradient in the general form $\nabla\overline{P} = -\boldsymbol{\phi}\cdot\overline{\mathbf{V}}$, only this time, the tensor $\boldsymbol{\phi}$ is a function of V instead of a constant.

Almost Final Form

Combining all the previous results and dividing through by C gives the following porous-material momentum equation

$$\frac{\partial}{\partial t}(\beta\overline{\rho\mathbf{V}}) + \nabla\cdot\beta\left(\overline{\rho\mathbf{V}}\,\overline{\mathbf{V}} + \overline{(\rho\dot{\mathbf{V}})\dot{\mathbf{V}}} - \overline{\tau}\right) + \beta\nabla\overline{P} + \boldsymbol{\phi}\cdot\beta\overline{\mathbf{V}} = 0 \tag{8.31}$$

8.2.4 Thermal Energy Equation

The integral form of the continuum energy equation applied to volume $\mathcal{V}$ centered on a point $\mathbf{r}$ is

$$\frac{\partial}{\partial t}\int_{\mathcal{V}}\rho e\,dv + \int_{S_{v_1}}\mathbf{n}\cdot(\rho e\mathbf{V} - \boldsymbol{\sigma}\cdot\mathbf{V} + \mathbf{q})\,ds + \int_{S_{v_2}}\mathbf{n}\cdot\mathbf{q}\,ds = 0 \tag{8.32}$$

This equation is a macroscopic statement of the energy conservation principle in void volume $\mathcal{V}$. The first term is the local rate of change of internal energy while the surface integral of $\mathbf{n} \cdot (\rho e \mathbf{V})$ over $\mathcal{S}_{v_1}$ is the rate of internal energy advected out of $\mathcal{V}$. The integral of $\mathbf{n} \cdot (\boldsymbol{\sigma} \cdot \mathbf{V})$ over $\mathcal{S}_{v_1}$ is the rate of mechanical stress-work done on $\mathcal{V}$, where the vector $\boldsymbol{\sigma} \cdot \mathbf{V}$ is understood in terms of the matrix product of a tensor and a column-vector. The term $\mathbf{n} \cdot (\rho e \mathbf{V} - \boldsymbol{\sigma} \cdot \mathbf{V})$ does not appear in the third integral because of the no-slip velocity boundary condition on the solid-boundary surface $\mathcal{S}_{v_2}$. The surface integrals of $\mathbf{n} \cdot \mathbf{q}$ represent the rate of heat leaving $\mathcal{V}$ through conduction.

The First Term

The integral in the first term of equation (8.32) is by definition $\mathcal{V}\,\overline{\rho e} = \beta \mathcal{C}\,\overline{\rho e}$ so that

$$\frac{\partial}{\partial t} \int_{\mathcal{V}} \rho e \, dv = \mathcal{C} \frac{\partial}{\partial t} (\beta \overline{\rho e}) \tag{8.33}$$

The Second Term

The second integral of equation (8.32) may be simplified by breaking the integral over $\mathcal{S}_{v_1}$ into pieces lying on the six faces $\mathcal{S}_i$ of $\mathcal{C}$ then applying the sum rule (8.10) and product rule (8.11) for averages. Note that on $\mathcal{S}_i$, $\mathbf{n} = \pm \mathbf{e}_i$, $\mathbf{n} \cdot \mathbf{V} = \pm u_i$ and $\mathbf{n} \cdot \mathbf{q} = \pm q_i$, the i^{th} rectangular basis vector, velocity and heat-flux components. This gives

$$\int_{\mathcal{S}_{v_1}} \mathbf{n} \cdot (\rho e \mathbf{V} - \boldsymbol{\sigma} \cdot \mathbf{V} + \mathbf{q}) \, ds = \sum_i \mathcal{S} \left[\beta \left(\overline{e}\, \overline{\rho u_i} + \overline{\acute{e}(\rho \acute{u}_i)} - \mathbf{e}_i \cdot (\overline{\boldsymbol{\sigma}} \cdot \overline{\mathbf{V}}) - \mathbf{e}_i \cdot \left(\overline{\acute{\boldsymbol{\sigma}} \cdot \acute{\mathbf{V}}} \right) + \overline{q_i} \right) \right]_{\mathbf{r}-d\mathbf{x}_i}^{\mathbf{r}+d\mathbf{x}_i} \tag{8.34}$$

Rewriting in terms of the finite-difference divergence operator gives

$$\int_{\mathcal{S}_{v_1}} \mathbf{n} \cdot (\rho e \mathbf{V} - \boldsymbol{\sigma} \cdot \mathbf{V} + \mathbf{q}) \, ds = \mathcal{C} \nabla \cdot \beta \left(\overline{e} \overline{\rho \mathbf{V}} + \overline{\acute{e}(\rho \acute{\mathbf{V}})} - \overline{\boldsymbol{\sigma}} \cdot \overline{\mathbf{V}} - \overline{\acute{\boldsymbol{\sigma}} \cdot \acute{\mathbf{V}}} + \overline{\mathbf{q}} \right) \tag{8.35}$$

The Third Term

I assume the sub-grid-scale heat flux per unit void-volume between the gas and the porous solid is given by a scalar $Q = hs(\overline{T}^s - \overline{T})$, where h is a film heat-transfer coefficient, s is the matrix surface area per unit void volume and $\overline{T}^s$ is the local average solid temperature

$$\overline{T}^s(\mathbf{r}) = \frac{1}{\mathcal{C} - \mathcal{V}} \int_{\mathcal{C}(\mathbf{r}) - \mathcal{V}(\mathbf{r})} T dv \tag{8.36}$$

Then the third integral in equation (8.32) becomes

$$\int_{S_{v_2}} \mathbf{n} \cdot \mathbf{q}\, ds = -\beta C\, Q \tag{8.37}$$

Almost Final Form

Combining all the previous results and dividing through by C gives the following thermal energy equation for the fluid within a porous material

$$\frac{\partial}{\partial t}\left(\beta\overline{\rho e}\right) + \nabla \cdot \beta \left(\overline{e}\,\overline{\rho \mathbf{V}} + \overline{\acute{e}(\rho \acute{\mathbf{V}})} - \overline{\boldsymbol{\sigma}} \cdot \overline{\mathbf{V}} - \overline{\acute{\boldsymbol{\sigma}} \cdot \acute{\mathbf{V}}} + \overline{\mathbf{q}}\right) - \beta Q = 0 \tag{8.38}$$

At this point I go ahead and assume that the term $\overline{\mathbf{q}}$ has lumped with it, the heat flux leaving C through the solid part of its boundary. This avoids having to deal with heat flux separately in the solid energy equation — a big help computationally.

8.2.5 Solid Energy Equation

In addition to the gasdynamic equations, an energy equation is required for the porous solid in order to complete the set of equations. I ignore solid-mode conductive heat flux arguing that for all cases of practical interest the porous solid is in good thermal contact with the gas so that, as noted before, the heat flux term $\nabla \cdot \overline{\mathbf{q}}$ already present in the gas energy equation suffices for both, with perhaps an empirical adjustment in gas conductivity. I consider only the local heat transfer between the gas and the solid. The resulting solid energy equation is then very simple

$$\frac{\partial E_s}{\partial t} + \beta Q = 0 \tag{8.39}$$

where

E_s	$=$	$(1-\beta)\rho_s c_s T_s$ solid volume-specific energy
ρ_s	$=$	solid density
c_s	$=$	solid specific heat
T_s	$=$	$\overline{T}^s$, the local-average solid temperature

8.2.6 Closure

There are a few loose ends in the forgoing development which must be tied up before the porous-flow equations are computationally useful. Quantities of the form $\overline{\acute{F}\acute{G}}$ that have been accumulating must be approximated in terms of computable variables. Keep in mind that I'm looking for equations that can accommodate the whole spectrum of porous materials — from densely packed, all the way to no packing at all. For this reason, if it seems I retain some terms that might reasonably be neglected for densely packed materials, it is because they are important in diffuse materials or empty manifolds.

Independent Variables

Although there is some freedom in choosing them, one must have exactly as many independent variables as there are equations to be solved. I choose the following quantities to be independent variables:

$$\overline{\rho} \quad \overline{\rho\mathbf{V}} \quad \overline{\rho e} \quad T_s$$

This choice reflects typical practice in computational fluid dynamics. All the remaining variables used in the porous-flow equations must be expressed in terms of these.

Void-Average Velocity

Unfortunately the local void-average velocity $\overline{\mathbf{V}}$ is not an independent variable, nor is it directly calculable from the preceding choice of independent variables. The most natural approximation for $\overline{\mathbf{V}}$ is

$$\overline{\mathbf{V}} = \overline{\rho\mathbf{V}}/\overline{\rho} \tag{8.40}$$

which is equivalent to neglecting $\overline{\acute{\rho}\acute{\mathbf{V}}}$ in the expansion $\overline{\rho\mathbf{V}} = \overline{\rho}\,\overline{\mathbf{V}} + \overline{\acute{\rho}\acute{\mathbf{V}}}$. Neglecting $\overline{\acute{\rho}\acute{\mathbf{V}}}$ might not always be a good idea. There are situations where $\acute{\rho}$ and $\acute{\mathbf{V}}$ might well be correlated — in the presence of a density gradient (temperature gradient), for example. However, lacking a suitable way to approximate $\overline{\acute{\rho}\acute{\mathbf{V}}}$ in terms of independent computational variables, I assume equation (8.40) is correct.

Void-Average Energy

Similarly, the local void-average energy density $\overline{e}$ is not an independent variable. The most natural approximation for $\overline{e}$ is

$$\overline{e} = \overline{\rho e}/\overline{\rho} \tag{8.41}$$

which is equivalent to neglecting $\overline{\acute{\rho}\acute{e}}$ in the expansion $\overline{\rho e} = \overline{\rho}\,\overline{e} + \overline{\acute{\rho}\acute{e}}$. This is justified since both $\acute{\rho}$ and $\acute{e}$ are relatively small, leading to second-order *smallness* in the product.

Void-Average Temperature

An ideal gas assumption relates the mass-specific total energy e and temperature T in the gas-filled part of the matrix $\mathcal{V}$.

$$e = c_v T + \frac{1}{2}\mathbf{V}\cdot\mathbf{V} \tag{8.42}$$

I assume that $\overline{T}$ may be solved from

$$\bar{e} = c_v\overline{T} + \frac{1}{2}\overline{\mathbf{V}}\cdot\overline{\mathbf{V}} \tag{8.43}$$

which is equivalent to neglecting $\overline{\acute{\mathbf{V}}\cdot\acute{\mathbf{V}}}$ in the expansion $\overline{\mathbf{V}\cdot\mathbf{V}} = \overline{\mathbf{V}}\cdot\overline{\mathbf{V}} + \overline{\acute{\mathbf{V}}\cdot\acute{\mathbf{V}}}$. Since the kinetic-energy fraction of the total energy is generally small in porous flow, this approximation is reasonably close.

Void-Average Pressure

Pressure is determined via an ideal gas equation of state which holds in the gas-filled part of the matrix $\mathcal{V}$

$$P = R\rho T \tag{8.44}$$

where P is pressure and R is a gas constant. I assume that

$$\overline{P} = R\bar{\rho}\overline{T} \tag{8.45}$$

which is equivalent to neglecting $\overline{\acute{\rho}\acute{T}}$ in the expansion $\overline{\rho T} = \bar{\rho}\overline{T} + \overline{\acute{\rho}\acute{T}}$. This is justified since both $\acute{\rho}$ and $\acute{T}$ are relatively small leading to second-order *smallness* in the product.

Enhanced Viscosity

At this point I lump together the stress tensor $\overline{\boldsymbol{\tau}}$, occurring in the momentum equation (8.31), with the term $\overline{(\rho\acute{\mathbf{V}})\acute{\mathbf{V}}}$ to produce an effective stress tensor $\boldsymbol{\tau}_e$.

First, recall how the molecular stress tensor $\boldsymbol{\tau}$ is defined in terms of molecular viscosity μ. According to reference [13] (section 1.1) it is possible to write the stress tensor $\boldsymbol{\tau}$, somewhat awkwardly, as

$$\boldsymbol{\tau} = \mu\left(2\,\mathrm{def}\,\mathbf{V} - \frac{2}{3}(\nabla\cdot\mathbf{V})\mathbf{I}\right) \tag{8.46}$$

where $\mathbf{I}$ is the identity tensor and $\mathrm{def}\,\mathbf{V} = \frac{1}{2}(\nabla\mathbf{V} + (\nabla\mathbf{V})^t)$, the t superscript denoting transpose.

Arguing that the term $\overline{(\rho\acute{\mathbf{V}})\acute{\mathbf{V}}}$ is akin to Reynold's apparent stress of turbulent flow theory, I now lump $\overline{\boldsymbol{\tau}}$ and $\overline{(\rho\acute{\mathbf{V}})\acute{\mathbf{V}}}$ together into an overall effective stress tensor $\boldsymbol{\tau}_e$ which I assume can be written in the form

$$\boldsymbol{\tau}_e = \mu_e\left(2\,\mathrm{def}\,\overline{\mathbf{V}} - \frac{2}{3}\left(\nabla\cdot\overline{\mathbf{V}}\right)\mathbf{I}\right) \tag{8.47}$$

where μ_e is an effective viscosity coefficient. The actual value of μ_e cannot be obtained from the sort of analysis presented here — it must be determined experimentally.

Enhanced Conductivity and Stress Work

The term $\overline{\acute{e}(\rho\acute{\mathbf{V}})}$, which occurs in the energy equation (8.38), gives rise to two separate energy transport mechanisms. One can see this by recalling that for an ideal gas, $e = c_v T + \frac{1}{2}\mathbf{V}\cdot\mathbf{V}$, so that

$$\overline{\acute{e}(\rho\acute{\mathbf{V}})} = c_v\overline{\acute{T}(\rho\acute{\mathbf{V}})} + \frac{1}{2}\overline{(\mathbf{V}\overset{\prime}{\cdot}\mathbf{V})(\rho\acute{\mathbf{V}})} \qquad (8.48)$$

The first term on the right of (8.48) represents the flux of thermal energy per unit area carried by the fluctuating velocity components. In the presence of a temperature gradient, $\acute{T}$ and $(\rho\acute{\mathbf{V}})$ are likely to be correlated. Based partly on experimentally observed enhancement of thermal diffusivity in flow through porous materials, and partly on theoretical arguments similar to those used in turbulent flow, it is reasonable to assume that $\overline{\acute{T}(\rho\acute{\mathbf{V}})}$ is proportional to $-\nabla\overline{T}$. Therefore, I lump together $\overline{\mathbf{q}}$ and $c_v\overline{\acute{T}(\rho\acute{\mathbf{V}})}$ into an effective overall heat-flux vector $\mathbf{q}_e$ defined by

$$\mathbf{q}_e = \overline{\mathbf{q}} + c_v\overline{\acute{T}(\rho\acute{\mathbf{V}})} = -\kappa_e\nabla\overline{T} \qquad (8.49)$$

where κ_e is an overall effective conductivity. Again, the value of κ_e would be determined experimentally; where $\mathbf{q}_e$ would be calculated from the total heat-flux across a matrix section divided by the mean void area. $\mathbf{q}_e$ lumps together the molecular gas conduction, the eddy conduction and the solid-mode conduction.

The second term on the right of (8.48) represents the flux of fluctuating kinetic energy per unit area transported by the fluctuating velocity components. This phenomenon bears the same relationship to the computational equations as does molecular kinetic energy transport to the continuum Navier-Stokes equations. In the Navier-Stokes equations, the effects of molecular momentum flux are embodied in the fluid stress tensor $\boldsymbol{\tau}$ and the flux of molecular kinetic energy works out to $-\boldsymbol{\tau}\cdot\mathbf{V}$. At the scale of the macroscopic (yet still sub-grid scale) velocity fluctuations, the role of $\boldsymbol{\tau}$ is superseded by the apparent excess-over-molecular stress tensor $(\boldsymbol{\tau}_e - \boldsymbol{\tau})$, so by analogy, it is reasonable to account for the fluctuating kinetic energy flux as

$$\frac{1}{2}\overline{(\mathbf{V}\overset{\prime}{\cdot}\mathbf{V})(\rho\acute{\mathbf{V}})} = \overline{\boldsymbol{\tau}}\cdot\overline{\mathbf{V}} - \overline{\boldsymbol{\tau}_e}\cdot\overline{\mathbf{V}} \qquad (8.50)$$

Putting this all together, and decomposing $\boldsymbol{\sigma}$ into $\boldsymbol{\tau} - P\mathbf{I}$, the three terms $\overline{\acute{e}(\rho\acute{\mathbf{V}})}$, $\overline{\mathbf{q}}$ and $\overline{\boldsymbol{\sigma}}\cdot\overline{\mathbf{V}}$ occuring in energy equation (8.38) may be re-written as

$$\overline{\acute{e}(\rho\acute{\mathbf{V}})} - \overline{\boldsymbol{\sigma}}\cdot\overline{\mathbf{V}} + \overline{\mathbf{q}} = \overline{P}\,\overline{\mathbf{V}} - \overline{\boldsymbol{\tau}_e}\cdot\overline{\mathbf{V}} + \mathbf{q}_e \qquad (8.51)$$

The Term $\overline{\acute{\sigma} \cdot \acute{\mathbf{V}}}$

I choose to neglect the term $\overline{\acute{\sigma} \cdot \acute{\mathbf{V}}}$ in the energy equation (8.38). From the definition of $\boldsymbol{\sigma}$, this term may be expanded as

$$\overline{\acute{\sigma} \cdot \acute{\mathbf{V}}} = \overline{\acute{\tau} \cdot \acute{\mathbf{V}}} - \overline{\acute{P}\acute{\mathbf{V}}} \tag{8.52}$$

It is reasonable to neglect the second term on the right on the grounds that $\acute{P}$ is not correlated with $\acute{\mathbf{V}}$. Neglecting first term, however, requires a bit more justification since $\acute{\tau}$ and $\acute{\mathbf{V}}$ are definitely correlated.

I will argue component-wise that $\overline{\acute{\tau} \cdot \acute{\mathbf{V}}}$ is negligible — at least to the extent that the gas behaves as an incompressible fluid. The j^{th} component of $\overline{\acute{\tau} \cdot \acute{\mathbf{V}}}$ in rectangular coordinates is given by

$$\overline{\left(\acute{\tau} \cdot \acute{\mathbf{V}}\right)_j} = \overline{\acute{\tau}_{jk}\acute{u}_k} \tag{8.53}$$

where the summation convention applies to the repeated k index. Now for an incompressible fluid, the stress tensor components τ_{jk} are

$$\tau_{jk} = \mu\left(\frac{\partial u_j}{\partial x_k} + \frac{\partial u_k}{\partial x_j}\right) \tag{8.54}$$

so that

$$\acute{\tau}_{jk}\acute{u}_k = \mu\left(\frac{\partial \acute{u}_j}{\partial x_k}\acute{u}_k + \frac{\partial \acute{u}_k}{\partial x_j}\acute{u}_k\right) \tag{8.55}$$

Making use of the product rule for differentiation gives

$$\acute{\tau}_{jk}\acute{u}_k = \mu\left(\frac{\partial}{\partial x_k}\left(\acute{u}_j\acute{u}_k\right) - \acute{u}_j\frac{\partial \acute{u}_k}{\partial x_k} + \frac{\partial}{\partial x_j}\left(\frac{\acute{u}_k^2}{2}\right)\right) \tag{8.56}$$

But the middle term on the right vanishes since $\frac{\partial \acute{u}_k}{\partial x_k} = 0$ for an incompressible fluid. (To see this, use definition (8.8) to write $\acute{u}_k = u_k - \overline{u}_k$, then note that $\frac{\partial u_k}{\partial x_k} = 0$ by the usual contintity equation and $\frac{\partial \overline{u}_k}{\partial x_k} = 0$ by the porous continuity equation (8.15) — at least assuming porosity β is constant.) So $\overline{\acute{\tau}_{jk}\acute{u}_k}$ may now be written

$$\overline{\acute{\tau}_{jk}\acute{u}_k} = \mu\left(\overline{\frac{\partial}{\partial x_k}\left(\acute{u}_j\acute{u}_k\right)} + \overline{\frac{\partial}{\partial x_j}\left(\frac{\acute{u}_k^2}{2}\right)}\right) \tag{8.57}$$

The right side of (8.57) can be broken down into a sum of volume integrals each of which has an integrand that is a perfect differential in one of the integration directions. By appropriately choosing the integration order, these integrals reduce to planar integrals of $[\acute{u}_j\acute{u}_k]$ and $[\acute{u}_k^2]$ evaluated at the endpoints (including those in the interior) of the broken line segments formed by the intersection of the coordinate lines (in the direction of the perfect differential) with $\mathcal{V}$. Now, on the boundaries of $\mathcal{V}$, $\mathbf{V} = 0$ so that $\acute{\mathbf{V}} = -\overline{\mathbf{V}}$ and the above evaluated quantities tend to cancel in pairs ($\overline{\mathbf{V}}$ does not vary much at the scale of the porosity void size). I conclude from all this that, in the case of an incompressible fluid at least, $\overline{\acute{\tau} \cdot \acute{\mathbf{V}}}$ is small. If it is not exactly zero, then at least it does not scale in proportion to the intensity of the fluctuating velocity components as, for example, $\overline{\acute{e}(\rho\acute{\mathbf{V}})}$ does.

The Term $\bar{e}\,\overline{\rho\mathbf{V}}$

The term $\bar{e}\,\overline{\rho\mathbf{V}}$, which appears in the energy equation (8.38), is conventionally evaluated as $\overline{\rho e}\,\overline{\mathbf{V}}$ instead. There is no problem in doing this here as well. By assumptions previously made, $\overline{\rho\mathbf{V}}$ can be decomposed into $\bar{\rho}\overline{\mathbf{V}}$, then $\bar{\rho}\,\bar{e}$ can be re-combined into $\overline{\rho e}$. Again, the decomposing part assumes $\overline{\acute{\rho}\acute{\mathbf{V}}}$ is negligible and the re-combining part assumes $\overline{\acute{\rho}\acute{e}}$ is negligible.

Summary

At this point it is safe to drop the $\overline{}$ notation in order to simplify things. Hereafter, the gasdynamic variables ρ, $\mathbf{V}$, e, T and P will always refer to their local void-fraction averages. In terms of computational variables this makes perfect sense and will cause no confusion.

The porous-flow gasdynamic equations may now be written in their final form as follows:

Porous Continuity

$$\frac{\partial}{\partial t}(\beta\rho) + \nabla \cdot (\beta\rho\mathbf{V}) = 0 \tag{8.58}$$

Porous Momentum

$$\frac{\partial}{\partial t}(\beta\rho\mathbf{V}) + \nabla \cdot (\beta\rho\mathbf{V}\mathbf{V} - \beta\boldsymbol{\tau}_e) + \beta\nabla P + \boldsymbol{\phi} \cdot \beta\mathbf{V} = 0 \tag{8.59}$$

Porous Thermal Energy

$$\frac{\partial}{\partial t}(\beta\rho e) + \nabla \cdot (\beta\rho e\mathbf{V} + P\beta\mathbf{V} - \boldsymbol{\tau}_e \cdot \beta\mathbf{V} + \beta\mathbf{q}_e) - \beta Q = 0 \tag{8.60}$$

where $\boldsymbol{\tau}_e$ is the effective stress tensor defined by equation (8.47), $\boldsymbol{\phi}$ is the porosity tensor discussed in section 8.2.3 and $\mathbf{q}_e$ is the effective heat flux vector defined by equation (8.49). Note that for problems where porosity is constant, β may be factored out of all the equations.

The above equation set is especially suited to numerical solution in domains containing porosity discontinuities. If one uses a staggered-grid finite-difference solution scheme, it is possible to arrange the momentum equation to be solved at the discontinuity and the continuity and energy equations at staggered locations.

This way, the only terms that need be evaluated at the discontinuity are: $\beta\rho\mathbf{V}$, $\beta\nabla P$, $\phi\cdot\beta\mathbf{V}$, $\beta\rho e\mathbf{V}$, $P\beta\mathbf{V}$, $\tau_e\cdot\beta\mathbf{V}$ and $\beta\mathbf{q}_e$. The term $\beta\rho\mathbf{V}$ is time-differenced in the momentum equation and spatially differenced in the continuity equation, the terms $\beta\nabla P$ and $\phi\cdot\beta\mathbf{V}$ are just evaluated in the momentum equation, while the remaining terms are spatially differenced in the energy equation. With the exception of $\beta\nabla P$ and $\phi\cdot\beta\mathbf{V}$, all of these terms are continuous at porosity discontinuities since they represent net *fluxes* per unit total area: $\beta\rho\mathbf{V}$ is momentum flux, $\beta\rho e\mathbf{V}$ is internal gas energy flux, $P\beta\mathbf{V}$ is normal-pressure work flux, $\tau_e\cdot\beta\mathbf{V}$ is viscous-stress work flux and $\beta\mathbf{q}_e$ is conductive heat flux. The terms $\beta\nabla P$ and $\phi\cdot\beta\mathbf{V}$ may be evaluated by averaging β and ϕ on either side of the discontinuity.

8.3 Rectangular Coordinate Representations

Some of the key terms that are used in the porous-flow equations are defined here in terms of rectangular coordinate representations.

8.3.1 Effective Viscous Stress Tensor

The components of the effective viscous stress tensor τ_e in rectangular coordinates are (summation convention)

$$\tau_{eij} = \mu_e\left(\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)-\frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\right) \tag{8.61}$$

where

δ_{ij}	=	Kronecker delta function
μ_e	=	effective viscosity
P	=	pressure
u_i	=	velocity components in rectangular coordinates

In two dimensions the components of τ_e are explicitly

$$\tau_{e11} = \frac{2}{3}\mu_e\left(2\frac{\partial u_1}{\partial x_1}-\frac{\partial u_2}{\partial x_2}\right) \tag{8.62}$$

$$\tau_{e12} = \mu_e\left(\frac{\partial u_1}{\partial x_2}+\frac{\partial u_2}{\partial x_1}\right) \tag{8.63}$$

$$\tau_{e21} = \mu_e\left(\frac{\partial u_2}{\partial x_1}+\frac{\partial u_1}{\partial x_2}\right) \tag{8.64}$$

$$\tau_{e22} = \frac{2}{3}\mu_e\left(2\frac{\partial u_2}{\partial x_2}-\frac{\partial u_1}{\partial x_1}\right) \tag{8.65}$$

8.3.2 Permeability Tensor

Motivated by the discussion in section 8.2.3, I define the following representation for permeability tensor ϕ in terms of rectangular coordinates

$$\phi_{ij} = \frac{\mu}{2d_h^2}(1+\epsilon)C_{ij} \tag{8.66}$$

where ϵ is a scalar perturbation which grows with Reynolds number according to equation (8.30) and the C_{ij} are dimensionless coefficients that depend on the nature of the porous material. For isotropic flow

$$C_{ij} = c_1\delta_{ij} \tag{8.67}$$

where c_1 is the first coefficient in the friction factor of the Ergun equation (8.27). For non-isotropic flow where the principal directions are the rectangular basis vectors $\mathbf{e}_i$

$$\mathbf{C} = \begin{pmatrix} C_1 & 0 \\ 0 & C_2 \end{pmatrix} \tag{8.68}$$

where C_i is the first Ergun coefficient in the $\mathbf{e}_i$ direction. For non-isotropic flow where the principal directions have been rotated off-axis, one would have to resort to tensor transformation rules such as (7.33) and (7.34) to obtain the C_{ij} from the previous case.

8.3.3 Two-Dimensional Equations in Rectangular Coordinates

For reference purposes, the porous-flow equations are written here in terms of rectangular coordinates and in two-dimensions. The solid energy equation (8.39) is not included since it is already in simplest form.

Continuity

$$\frac{\partial}{\partial t}(\beta\rho) + \frac{\partial}{\partial x_1}(\beta\rho u_1) + \frac{\partial}{\partial x_2}(\beta\rho u_2) = 0 \tag{8.69}$$

Momentum

$$\begin{aligned}\frac{\partial}{\partial t}(\beta\rho u_1) &+ \frac{\partial}{\partial x_1}\left(\beta(\rho u_1^2 - \tau_{e11})\right) + \frac{\partial}{\partial x_2}\left(\beta(\rho u_1 u_2 - \tau_{e21})\right) \\ &+ \frac{\partial P}{\partial x_1} + \phi_{11}\beta u_1 + \phi_{12}\beta u_2 = 0 \end{aligned} \tag{8.70}$$

$$\begin{aligned}\frac{\partial}{\partial t}(\beta\rho u_2) &+ \frac{\partial}{\partial x_1}\left(\beta(\rho u_2 u_1 - \tau_{e12})\right) + \frac{\partial}{\partial x_2}\left(\beta(\rho u_2^2 - \tau_{e22})\right) \\ &+ \frac{\partial P}{\partial x_2} + \phi_{21}\beta u_1 + \phi_{22}\beta u_2 = 0 \end{aligned} \tag{8.71}$$

Energy

$$\begin{aligned}\frac{\partial}{\partial t}(\beta\rho e) &+ \frac{\partial}{\partial x_1}(\beta(\rho e u_1 + P u_1 - u_1\tau_{e11} - u_2\tau_{e12} + q_{e1})) \qquad (8.72)\\ &+ \frac{\partial}{\partial x_2}(\beta(\rho e u_2 + P u_2 - u_1\tau_{e21} - u_2\tau_{e22} + q_{e2})) - \beta Q = 0\end{aligned}$$

8.4 Dimensionless Form

One can choose normalization quantities so that, in dimensionless form, the porous flow equations will appear exactly as before. This is standard practice except for a few twists due to special terms in the porous-flow equations and the oscillating boundary conditions present in regenerator flow problems.

First choose characteristic values ρ_0, u_0, μ_0 and κ_0 for density, velocity, viscosity and conductivity. The choice is arbitrary but Manifest uses $u_0 = \sqrt{\gamma R T_0}$, the speed of sound at characteristic temperature T_0, because then velocities are represented as Mach numbers. The angular frequency ω of the oscillating inlet boundary conditions is another characteristic value which is used in place of a characteristic time.

Now define dimensionless time and position coordinates by

$$t^* = \omega t \qquad (8.73)$$

$$x^* = \frac{\omega}{u_0} x \qquad (8.74)$$

Before tackling the complicated porous flow equations, it is useful to see how the simple generic equation

$$\frac{\partial A}{\partial t} + \frac{\partial B}{\partial x} + C = 0 \qquad (8.75)$$

is made dimensionless. Using the chain rule one can transform the left side of (8.75) to $\frac{\partial A}{\partial t^*}\frac{\partial t^*}{\partial t} + \frac{\partial B}{\partial x^*}\frac{\partial x^*}{\partial x} + C = \frac{\partial A}{\partial t^*}\omega + \frac{\partial B}{\partial x^*}\omega/u_0 + C$. Factoring out the ω gives

$$\frac{\partial A}{\partial t^*} + \frac{1}{u_0}\frac{\partial B}{\partial x^*} + \frac{C}{\omega} = 0 \qquad (8.76)$$

The conclusion is: If a normalization quantity N makes A/N dimensionless, then $B/(u_0 N)$ and $C/(\omega N)$ are also dimensionless. This observation saves a lot of work since one has only to worry about how to normalize the first term in an equation.

In light of the preceding observation, the porous-flow equations are made dimensionless by the substitution of the following starred quantities:

$$\begin{aligned} \rho^* &= \rho/\rho_0 & \tau_e^* &= \tau_e/(\rho_0 u_0^2) & q_e^* &= q_e/(\rho_0 u_0^3)\\ u_i^* &= u_i/u_0 & \phi^* &= \phi/(\rho_0\omega) & Q^* &= Q/(\omega\rho_0 u_0^2)\\ P^* &= P/(\rho_0 u_0^2) & e^* &= e/u_0^2 & E_s^* &= E_s/(\rho_0 u_0^2)\end{aligned}$$

The form of the equations remains exactly the same.

The constitutive relations for dimensionless τ_e^*, ϕ^*, q_e^*, Q^* and E_s^* are derived next. They are almost — but not quite — in their previous form.

First, define some dimensionless quantities that will be required along the way.

$$
\begin{aligned}
T^* &= T(c_p/u_0^2)\text{; dimensionless temperature} \\
R_{e0} &= \rho_0 u_0^2/(\mu_0 \omega)\text{; characteristic Reynolds number} \\
W_{m0} &= \tfrac{d_h}{2}\sqrt{\rho_0 \omega/\mu_0}\text{; characteristic Womersley number} \\
P_{r0} &= \mu_0 c_p/\kappa_0\text{; characteristic Prandtl number} \\
N_{tu} &= hs/(\omega \rho_0 c_p)\text{; number of transfer units} \\
\lambda &= (1-\beta)\rho_s c_s/(\beta \rho_0 c_p)\text{; solid-to-void heat capacity ratio}
\end{aligned}
$$

Here, c_p is the gas specific heat at constant pressure which is assumed to be a constant. Also recall that s used in N_{tu} is the matrix surface area per unit void volume, not the total surface area. And note that N_{tu} may vary if h is considered a function of flow velocity.

From the preceding results one can derive the following dimensionless relations in rectangular coordinates which replace the earlier dimensional formulations. **Hereafter I drop the stars from the notation** because all subsequent expressions will involve only dimensionless quantities.

$$\tau_{eij} = \frac{1}{R_{e0}}\mu_e \left((\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k} \right) \tag{8.77}$$

$$\phi_{ij} = \frac{1+\epsilon}{8W_{m0}^2}\mu c_{ij} \tag{8.78}$$

$$q_{ei} = \frac{-\kappa_e}{R_{e0}P_{r0}}\frac{\partial T}{\partial x_i} \tag{8.79}$$

$$Q = N_{tu}(T_s - T) \tag{8.80}$$

$$E_s = \beta \lambda T_s \tag{8.81}$$

Also, the following dimensionless relations hold for an ideal gas

$$T = \gamma(e - \frac{1}{2}V^2) \tag{8.82}$$

$$P = (\gamma - 1)\rho(e - \frac{1}{2}V^2) \tag{8.83}$$

$$P = \frac{\gamma - 1}{\gamma}\rho T \tag{8.84}$$

where $\gamma = c_p/c_v$ is the ratio of gas specific heats. The third relation is the dimensionless version of the ideal-gas equation of state.

8.5 Porous Flow Equations in Curvilinear Coordinates

This section uses the coordinate transformation theory developed in chapter 7 to write the porous-flow equations in arbitrary curvilinear coordinates. The formulations are, for the most part, a mélange of the work of several authors who have come this way before. Viviand [20] is usually credited with first deriving strong conservative forms for the gasdynamic equations under general time-dependent coordinate transformations. Less practical though more mathematically elegant are the results of Vinokur [19] who presents his results entirely in terms of curvilinear components. Peyret and Taylor [13] and Anderson, Tannehill and Pletcher [1] review the literature on the subject in textbook form. From the point of view of the non-expert, all of the above sources tend to *quote* rather than derive results, which is one of the reason that this presentation attempts to be reasonably detailed.

Remember that the reason for bothering with curvilinear coordinates in the first place is so that a rectangular array can always be used for the computational grid even though the actual physical solution domain is non-rectangular. The idea is to generate a coordinate transformation so that the curvilinear coordinate surfaces (y_k = *constant*) fall on the boundaries of the physical solution domain. Then if the system differential equations are written so that terms of the form $\frac{\partial}{\partial x_i}$ are replaced with terms of the form $\frac{\partial}{\partial y_k}$, one can set up the finite-difference solution scheme in a rectangular grid. Note that nothing has been said about actually generating curvilinear coordinates; this will be discussed in chapter. 10.

There are several different ways to proceed depending on whether the equations are cast in conservative or non-conservative form, and whether curvilinear coordinates are used to represent all quantities or are mixed with rectangular coordinate representations. Since all that is really required is to replace $\frac{\partial}{\partial x_i}$ terms with $\frac{\partial}{\partial y_k}$ terms, one has several choices on how to represent everything else. In particular, some authors choose to retain the rectangular coordinate representation for velocity in terms of components (u_1, u_2, u_3) while others choose to represent the transformed equations in terms of curvilinear velocity components (u^1, u^2, u^3). I choose a mixed approach out of the need for computational speed and because of the staggered-grid solution scheme used by Manifest.

8.5.1 Staggered Grids

In a staggered-grid solution in rectangular coordinates the flow-related variables ρu_i are solved at different grid locations from the property-related variables ρ and ρe. In addition to making good conceptual sense, staggered grid formulations also prevent the *checkerboard* instability problem common to non-staggered solutions — see Issa [9]. The most natural way to stagger the solution is the

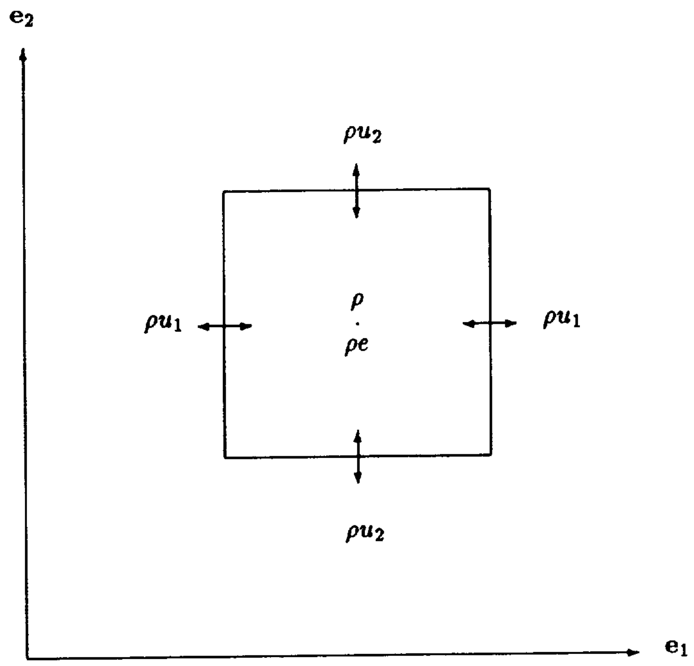


Figure 8.3: The MAC staggered-grid scheme for rectangular coordinates in two-dimensions

MAC (marker and cell) scheme illustrated for two dimensions and rectangular coordinates in figure 8.3. The labeled quantities are solved at the indicated locations of a typical control volume. The control volume itself is made up of coordinate line segments (x_i = *constant*) of length equal to twice the spacing of nodes in the computational grid. In other words, a computational node lies at each corner, the midpoint of each side and the center of the control volume. The MAC scheme is intuitively satisfying because, in an array of such control volumes, the solved pressures ($\approx \rho e$) are located in the natural position for central differencing of the $\frac{\partial P}{\partial x_i}$ terms found in the momentum equations for the ρu_i variables.

Figure 8.4 shows the logical extension of the MAC scheme to curvilinear coordinates. Now the typical control volume is made up of curvilinear coordinate line segments (y_k = *constant*) and the components of momentum flux solved at the midpoints of each side are the curvilinear components (ρu^i). Again, the solved pressures are located in the natural position for central differencing of the $\frac{\partial P}{\partial y^i}$ terms found in the curvilinear momentum equations for the ρu^i variables — as will be seen — although the dominance of the $\frac{\partial P}{\partial y^i}$ term diminishes as the curvilinear coordinates become more and more skewed. One of the key requirements for the transformed equation set is that the curvilinear momentum flux components ρu^i should wind up as independent variables in the momentum equations.

The curvilinear equations presented below are consistent with the MAC

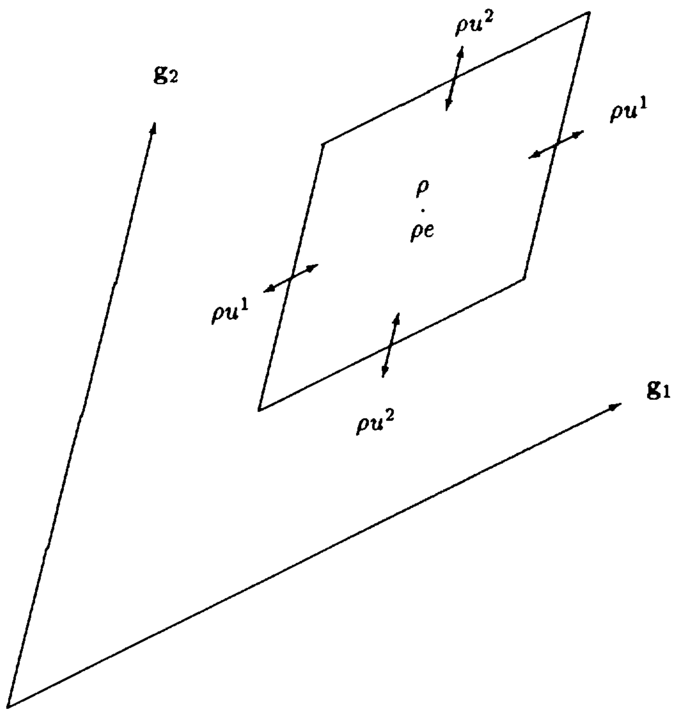


Figure 8.4: The MAC staggered-grid scheme for curvilinear coordinates in two-dimensions

staggered-grid scheme and also strongly conservative. All equations are in dimensionless form consistent with the conventions presented earlier in section 8.4.

8.5.2 Computational Variables

At this point it is convenient to define three new independent variables in order to simplify the following material. A good all-around choice is

$$\begin{aligned} M &= \rho \\ \mathbf{G} &= \beta\rho\mathbf{V} \\ E &= \rho e \end{aligned}$$

where ρ is dimensionless void-average gas density, β is matrix porosity, $\mathbf{V}$ is dimensionless void-average gas velocity and e is dimensionless void-average mass-specific gas energy density. The components of $\mathbf{G}$ are denoted G_j or G^k in rectangular or curvilinear coordinates respectively. A computationally important property shared by all three variables is that they are continuous across porosity discontinuities. These variables supersede the previous independent variables ρ, $\rho\mathbf{V}$ and ρe defined earlier. But, since β is assumed known everywhere, the two sets of variables are entirely equivalent. The solid temperature T_s remains the fourth independent variable, the same as before.

Actually, the most concise equations would result by choosing $\beta\rho/J$, $\beta\rho\mathbf{V}/J$ and $\beta\rho e/J$ as independent variables. But for computational reasons, it turns out that this is a poor choice. The problem is that in a staggered-grid solution, the independent variables must be interpolated or extrapolated to non-solved grid points. The factor of $1/J$ in these variables causes difficulty in regions where the coordinate transformation is highly nonlinear. Likewise, in the variables $\beta\rho/J$ and $\beta\rho e/J$, the factor of β causes difficulty near porosity discontinuities. The result is that, in some situations, unacceptably large truncation errors can arise — especially when using course computational grids.

8.5.3 Continuity Equation

Starting with the continuity equation (8.58), use (7.48) to replace the $\frac{\partial}{\partial t}(\beta\rho)$ term and (7.39) to replace the $\nabla \cdot (\beta\rho\mathbf{V})$ term, obtaining

$$J\frac{\partial}{\partial\tau}\left(\frac{\beta M}{J}\right) + J\frac{\partial}{\partial y_k}\left(\frac{G^k}{J} + \frac{\partial y_k}{\partial t}\frac{\beta M}{J}\right) = 0 \tag{8.85}$$

For the case of a time-independent coordinate transformation this becomes simply

$$\frac{\partial M}{\partial\tau} + \frac{J}{\beta}\frac{\partial}{\partial y_k}\left(\frac{G^k}{J}\right) = 0 \tag{8.86}$$

8.5.4 Momentum Equation

Start with the porous-flow momentum equation (8.59). First rewrite ∇P as $\nabla \cdot P\mathbf{I}$, then use (7.48) to replace the $\frac{\partial}{\partial t}(\beta\rho\mathbf{V})$ term and (7.42) to replace the $\nabla\cdot(\beta\rho\mathbf{V}\mathbf{V} - \beta\boldsymbol{\tau}_e)$ and $\nabla \cdot P\mathbf{I}$ terms, obtaining finally the following general form of the porous flow momentum equation in curvilinear coordinates.

$$\begin{aligned} J\frac{\partial}{\partial\tau}\left(\frac{G^l}{J}\mathbf{g}_l\right) \quad &+ \quad J\frac{\partial}{\partial y_k}\left(\frac{G^k}{J}u^l\mathbf{g}_l - \frac{\beta}{J}\tau_e^{kl}\mathbf{g}_l + \frac{\partial y_k}{\partial t}\frac{G^l}{J}\mathbf{g}_l\right) \\ &+ \quad \beta J\frac{\partial}{\partial y_k}\left(\frac{P}{J}\mathbf{I}^{kl}\mathbf{g}_l\right) + \boldsymbol{\phi}\cdot\beta\mathbf{V} = 0 \end{aligned} \tag{8.87}$$

Unfortunately the curvilinear components of the stress tensor (τ_e^{kl}) are expensive to compute. It is more efficient to back off from (8.87) by using (7.40) instead of (7.42) to replace the $\nabla\cdot(\beta\rho\mathbf{V}\mathbf{V} - \beta\boldsymbol{\tau}_e)$ and $\nabla \cdot P\mathbf{I}$ terms in the original momentum equation. After noting that $u_i\frac{\partial y_k}{\partial x_i} = u^k$ and $\boldsymbol{\phi}\cdot\mathbf{V} = \phi_{jk}u_k\mathbf{e}_j$ from (7.30), the resulting momentum equation involving mixed rectangular and curvilinear components is

$$\begin{aligned} J\frac{\partial}{\partial\tau}\left(\frac{G^l}{J}\mathbf{g}_l\right) \quad &+ \quad J\frac{\partial}{\partial y_k}\left(\frac{G^k}{J}u_j\mathbf{e}_j - \frac{\beta}{J}\tau_{eij}\frac{\partial y_k}{\partial x_i}\mathbf{e}_j + \frac{\partial y_k}{\partial t}\frac{G_j}{J}\mathbf{e}_j\right) \\ &+ \quad \beta J\frac{\partial}{\partial y_k}\left(\frac{P}{J}\frac{\partial y_k}{\partial x_j}\mathbf{e}_j\right) + \beta\phi_{jk}u_k\mathbf{e}_j = 0 \end{aligned} \tag{8.88}$$

Mixed components are no problem since both rectangular and curvilinear velocity components are readily available and the τ_{eij} can be easily calculated as shown below.

For the case of a time-independent coordinate transformation (8.88) becomes

$$\begin{aligned}\frac{\partial}{\partial \tau}(G^l \mathbf{g}_l) &+ J\frac{\partial}{\partial y_k}\left(\frac{G^k}{J}u_j\mathbf{e}_j - \frac{\beta}{J}\tau_{eij}\frac{\partial y_k}{\partial x_i}\mathbf{e}_j\right) \\ &+ \beta J\frac{\partial}{\partial y_k}\left(\frac{P}{J}\frac{\partial y_k}{\partial x_j}\mathbf{e}_j\right) + \beta\phi_{jk}u_k\mathbf{e}_j = 0 \qquad (8.89)\end{aligned}$$

Now, the constant basis vectors $\mathbf{e}_j$ can be factored out of the second term of (8.89) and since the $\mathbf{g}_l$ are assumed time-independent, they can be factored out of the first term. The equation can then be broken down into three scalar equations by forming the dot product with the alternate basis vectors ∇y_1, ∇y_2 and ∇y_3 in succession. Using the relations $\mathbf{g}_l \cdot \nabla y_n = \delta_{ln}$ (equation 7.22) and $\mathbf{e}_j \cdot \nabla y_n = \frac{\partial y_n}{\partial x_j}$, the momentum equation becomes

$$\begin{aligned}\frac{\partial G^n}{\partial \tau} &+ \left(J\frac{\partial}{\partial y_k}\left(\frac{G^k}{J}u_j - \frac{\beta}{J}\tau_{eij}\frac{\partial y_k}{\partial x_i}\right) + \beta J\frac{\partial}{\partial y_k}\left(\frac{P}{J}\frac{\partial y_k}{\partial x_j}\right) + \beta\phi_{jk}u_k\right)\frac{\partial y_n}{\partial x_j} \\ &= 0 \qquad (8.90)\end{aligned}$$

for $n = 1\ldots 3$.

The rectangular components of the dimensionless stress tensor $\boldsymbol{\tau}_e$ are obtained from (8.77), but first, the equation must be transformed to use $\frac{\partial}{\partial y_l}$ partial derivatives. Note that the factor $\frac{\partial u_k}{\partial x_k} = \nabla \cdot \mathbf{V}$ can be written using (7.39) as $J\frac{\partial}{\partial y_l}(u^l/J)$. Then use the chain rule on the remaining terms to obtain

$$\tau_{eij} = \frac{\mu_e}{R_{e0}}\left(\frac{\partial u_i}{\partial y_l}\frac{\partial y_l}{\partial x_j} + \frac{\partial u_j}{\partial y_l}\frac{\partial y_l}{\partial x_i} - \frac{2}{3}\delta_{ij}J\frac{\partial}{\partial y_l}(u^l/J)\right) \qquad (8.91)$$

Here rectangular and curvilinear velocity components are intentionally mixed to simplify the formula. For actual computation in two dimensions, the explicit forms for τ_e given in (8.62) through (8.65) can be transformed according to the chain rule to give the less general, although more efficient, form

$$\tau_{e11} = \frac{2}{3}\mu_e(2\frac{\partial u_1}{\partial y_l}\frac{\partial y_l}{\partial x_1} - \frac{\partial u_2}{\partial y_l}\frac{\partial y_l}{\partial x_2}) \qquad (8.92)$$

$$\tau_{e12} = \mu_e(\frac{\partial u_1}{\partial y_l}\frac{\partial y_l}{\partial x_2} + \frac{\partial u_2}{\partial y_l}\frac{\partial y_l}{\partial x_1}) \qquad (8.93)$$

$$\tau_{e21} = \mu_e(\frac{\partial u_2}{\partial y_l}\frac{\partial y_l}{\partial x_1} + \frac{\partial u_1}{\partial y_l}\frac{\partial y_l}{\partial x_2}) \qquad (8.94)$$

$$\tau_{e22} = \frac{2}{3}\mu_e(2\frac{\partial u_2}{\partial y_l}\frac{\partial y_l}{\partial x_2} - \frac{\partial u_1}{\partial y_l}\frac{\partial y_l}{\partial x_1}) \qquad (8.95)$$

8.5.5 Thermal Energy Equation

Starting with the energy equation (8.60), use (7.48) to replace the $\frac{\partial}{\partial t}(\beta \rho e)$ term and (7.39) to replace the $\nabla \cdot (\beta \rho e \mathbf{V} + P\beta\mathbf{V} - \boldsymbol{\tau}_e \cdot \beta\mathbf{V} + \beta q)$ term, obtaining

$$J\frac{\partial}{\partial \tau}\left(\frac{\beta E}{J}\right) + J\frac{\partial}{\partial y_k}\left(\frac{E}{J}\beta u^k + \frac{1}{J}(P\beta\mathbf{V})^k - \frac{1}{J}(\boldsymbol{\tau}_e \cdot \beta\mathbf{V})^k + \frac{\beta}{J}q_e^k + \frac{\partial y_k}{\partial t}\frac{\beta E}{J}\right) - \beta Q = 0 \qquad (8.96)$$

For the case of a time-independent coordinate transformation this becomes

$$\frac{\partial E}{\partial \tau} + \frac{J}{\beta}\frac{\partial}{\partial y_k}\left(\frac{E}{J}\beta u^k + \frac{1}{J}(P\beta\mathbf{V})^k - \frac{1}{J}(\boldsymbol{\tau}_e \cdot \beta\mathbf{V})^k + \frac{\beta}{J}q_e^k\right) - Q = 0 \qquad (8.97)$$

Using the transformation rule (7.25) followed by definition (7.30), the curvilinear components of $\boldsymbol{\tau}_e \cdot \beta\mathbf{V}$ may be obtained from their rectangular components as

$$(\boldsymbol{\tau}_e \cdot \beta\mathbf{V})^k = \beta\tau_{e_{ij}} u_j \frac{\partial y_k}{\partial x_i} \qquad (8.98)$$

where the $\tau_{e_{ij}}$ are calculated as in the previous section.

Similarly the curvilinear components of $\mathbf{q}_e$ may be written $q_e^k = q_{e_i}\frac{\partial y_k}{\partial x_i}$, where the q_{e_i} are calculated from the dimensionless form (8.79). Using the chain rule to rewrite $\frac{\partial T}{\partial x_i} = \frac{\partial T}{\partial y_l}\frac{\partial y_l}{\partial x_i}$ gives the dimensionless form

$$q_e^k = \frac{-\kappa_e}{R_{e0}P_{r0}}\frac{\partial y_l}{\partial x_i}\frac{\partial y_k}{\partial x_i}\frac{\partial T}{\partial y_l} \qquad (8.99)$$

8.5.6 Solid Energy Equation

Start with the solid energy equation (8.39), except replace E_s with $\beta\lambda T_s$ according to equation (8.81). Then use (7.48) to replace the $\frac{\partial}{\partial t}(\beta\lambda T_s)$ term, obtaining

$$J\frac{\partial}{\partial \tau}\left(\frac{\beta\lambda}{J}T_s\right) + J\frac{\partial}{\partial y_j}\left(\frac{\partial y_j}{\partial t}\frac{\beta\lambda}{J}T_s\right) + \beta Q = 0 \qquad (8.100)$$

For the case of a time-independent transformation this becomes

$$\frac{\partial}{\partial \tau}(\lambda T_s) + Q = 0 \qquad (8.101)$$

Why is T_s taken as an independent variable rather than E_s? The reason is purely a computational one. In some cases it makes sense to treat the matrix as isothermal and ignore the solid energy equation. Manifest selects this option whenever the input parameter λ is set equal to zero. But, it is still necessary to have T_s defined in order to compute the heat transfer term Q. This would be tough to do if E_s were the independent variable, since $T_s = E_s/(\beta\lambda)$ would then be undefined. The problem disappears if T_s is the independent variable: In the isothermal case T_s can simply be held fixed at its initial value.

Chapter 9

The Computational Algorithm

This chapter discusses Manifest's numerical algorithms without actually getting involved in programming details. Readers interested in actual software structure will find plenty of comments nested within the source code to guide them. But this chapter is a prerequisite.

9.1 The Grid

Manifest's computational grid is a two-dimensional array of discrete curvilinear coordinates $(y_1, y_2)[i,j] = (i\Delta y, j\Delta y)$ where indices $[i,j]$ range between limits $i = 1 \ldots N_1$ and $j = 1 \ldots N_2$ and Δy is some fixed but arbitrary spacing. Array upper bounds N_1 and N_2 vary from problem to problem depending on the resolution desired in the solution. This computational grid ultimately corresponds, by way of a coordinate transformation, to an actual physical grid in Euclidean space where the points are not necessarily uniformly spaced nor arranged in rectangular fashion. But the computational algorithm only knows this by the transformation Jacobian and partial derivative matrices $\mathbf{X_y}$ and $\mathbf{Y_x}$ stored at every point of the grid. The matter of solving the coordinate transformation is discussed separately in chapter 10.

9.2 The Gasdynamic Variables

At each location of the computational grid is defined a set of gasdynamic variables which will eventually hold the numerical problem solution at successive time steps. These variables are M, $\mathbf{G}$, E and T_s defined in section 8.5.2. However, not all these variables are actually solved independently. Some are interpolated from nearby solved variables in accordance with the staggered-grid MAC (marker and cell) solution scheme discussed in section 8.5.1, and some are determined by boundary conditions. I will get back to the matters of interpolation and boundary conditions later on, but for the moment, all that matters is that the gas dynamic variables can be partitioned into two sets: those that are independently solved and those that are not. By ordering the components of these two sets in some way (details not important) I arrive at two arrays V and D. The components V_k of V are the independently solved variables. The components D_k of D are dependent on V and time τ in the form

$$D_k = F_k(V, \tau) \tag{9.1}$$

where the various functions F_k take care of the interpolations or extrapolations required by the staggered-grid scheme as well as boundary conditions.

It is in terms of the array V that the solution algorithm proceeds. Again, the ordering of components in V is not important. Each component is either M, G_1, G_2 or T_s for some grid index $[i, j]$. The computer keeps track of the ordering. We do not have to. To understand the following solution algorithm just think of V as an ordinary array or vector.

9.3 Time Stepping

Manifest is built around a solution scheme originally proposed by Beam and Warming [2,3]. However, I depart from their presentation in order to handle, in a general way, exceptions to the rule produced by staggered-grids and arbitrary boundary conditions. Also, I have eliminated a matrix factoring step that was central to their method. For small or medium sized gasdynamic problems, the resulting full-system scheme is both faster and more accurate, although its asymptotic behavior for large problems remains an open question.

It will help to first review the essentials of the Beam and Warming solution method. The basic problem is formulated as a coupled nonlinear system of differential equations in the form

$$\frac{\partial V}{\partial \tau} + R(V, \tau) = 0 \tag{9.2}$$

where V is the array of solved gasdynamic variables discussed above, and R contains approximations to everything besides the time-partial terms in the underlying partial differential equations — finite-difference formulations for spatial partial derivatives and so forth. At this point, I still consider time to be a continuum variable. A solution to the system (9.2) is an approximate solution to the original system of continuum differential equations.

The temporal identity upon which the solution algorithm is based is

$$\Delta V^n = \frac{c_1 \Delta\tau}{1+c_2}\frac{\partial \Delta V^n}{\partial \tau} + \frac{\Delta\tau}{1+c_2}\frac{\partial V^n}{\partial \tau} + \frac{c_2}{1+c_2}\Delta V^{n-1}$$

$$+\mathcal{O}\left((c_1 - c_2 - 1/2)\Delta\tau^2 + \Delta\tau^3\right) \tag{9.3}$$

where $V^n = V(n\Delta\tau)$, $\Delta V^n = V^{n+1} - V^n$, and c_1 and c_2 are constants of the method. For $(c_1, c_2) = (1/2, 0)$, equation (9.3) represents the implicit trapezoid method

$$\frac{V^{n+1} - V^n}{\Delta\tau} = \frac{1}{2}\left(\frac{\partial V^{n+1}}{\partial\tau} + \frac{\partial V^n}{\partial\tau}\right) \tag{9.4}$$

and for $(c_1, c_2) = (1, 1/2)$ it represents the three-point backward implicit method.

$$\frac{3V^{n+1} - 4V^n + V^{n-1}}{2\Delta\tau} = \frac{\partial V^{n+1}}{\partial\tau} \tag{9.5}$$

Assuming the solution V^n at time-level n is known, (9.3) allows us to step the solution forward in time. In principle, all we need do is substitute $-R$ for $\frac{\partial V}{\partial\tau}$ on the right side of (9.3). The problem is: How does one evaluate R^{n+1}? The fact that R is a nonlinear function of V doesn't help matters but in any case we can make use of the Taylor series expansion

$$R^{n+1} = R^n + \frac{\partial R^n}{\partial\tau} + \mathbf{J}^n \Delta V^n + \mathcal{O}(\Delta\tau^2) \tag{9.6}$$

where $\mathbf{J}^n$ is the Jacobian matrix $\frac{\partial R}{\partial V}$ at time-level n. Making this substitution and grouping all the ΔV^n terms on the left, the implicit scheme for advancing the solution to level $n+1$ becomes

$$\left(I + \frac{c_1\Delta\tau}{1+c_2}\mathbf{J}^n\right)\Delta V^n = -\frac{c_1\Delta\tau^2}{1+c_2}\frac{\partial R^n}{\partial\tau} - \frac{\Delta\tau}{1+c_2}R^n + \frac{c_2}{1+c_2}\Delta V^{n-1} \tag{9.7}$$

Note that the $\mathcal{O}(\Delta\tau^3)$ temporal accuracy of (9.3) is not affected by the $\mathcal{O}(\Delta\tau^2)$ R^{n+1} approximation because the R terms already contain a factor $\Delta\tau$. Manifest takes its initial time step with $(c_1, c_2) = (1/2, 0)$ because then the ΔV^{n-1} term on the right (which is unknown) is moot. Subsequent steps use $(c_1, c_2) = (1, 1/2)$.

A key step in the Beam and Warming algorithm is to decompose the Jacobian into three parts $\mathbf{J} = \mathbf{J}_1 + \mathbf{J}_2 + \mathbf{J}_x$ (dropping the n superscript for clarity), bringing the $\mathbf{J}_x$ term over to the right-hand side to be explicitly (and approximately) evaluated as $-\frac{c_1\Delta\tau}{1+c_2}\mathbf{J}_x\Delta V^{n-1}$ and approximating the remaining terms on the left using the factorization

$$I + \frac{c_1\Delta\tau}{1+c_2}(\mathbf{J}_1 + \mathbf{J}_2) \approx \left(I + \frac{c_1\Delta\tau}{1+c_2}\mathbf{J}_1\right)\left(I + \frac{c_1\Delta\tau}{1+c_2}\mathbf{J}_2\right) \tag{9.8}$$

Roughly speaking, the terms of $\mathbf{J}_1$, $\mathbf{J}_2$ and $\mathbf{J}_x$ result from spatial differences of R in the y_1-direction, y_2-direction and cross coordinate directions respectively. The factorization of (9.8) makes the solution more efficient because $\mathbf{J}_1$ and $\mathbf{J}_2$ are narrow-bandwidth block-diagonal matrices.

Unfortunately, factorization reduces accuracy. Equation (9.8) leaves out the term $(\frac{c_1\Delta\tau}{1+c_2})^2\mathbf{J}_1\mathbf{J}_2$ which can be significant if $\Delta\tau$ is not too small or $\mathbf{J}_1$ and $\mathbf{J}_2$ contain large terms. Both of these problems seem likely in normal use of Manifest. Also, evaluating the $\mathbf{J}_x$ term explicitly introduces another error of uncertain magnitude, and there is no good way to proceed for the first time step, where ΔV^{n-1} is not known. John Lavery at NASA Lewis (with whom I had the good fortune to discuss my algorithm) pointed out that even though there is no simple way to do an error analysis for the factored method, one thing is certain: it throws away information in the Jacobian. If that information turns out to be vital, then accuracy will suffer.

To avoid these problems, Manifest just solves the full system (9.7) directly. The only disadvantages to doing this are — and they are potentially big ones — that both execution time and memory requirements grow more quickly with increasing system dimension N than for a factored scheme. The key questions are:

1. How do the times for the linear system solutions vary with N for the full-system vs factored method?

2. How does accuracy vary with N?

The answer to question (1) is relatively easy: The factored scheme has asymptotic time behavior of $\mathcal{O}(N)$ while the full-system scheme goes as $\mathcal{O}(N^2)$ — at least for a square computational domain and the Gaussian elimination algorithm in Manifest. The reason for the factored-scheme behavior is that the block size and bandwidth of the block-diagonal system are independent of N. Thus, each of the N steps of the Gaussian elimination process requires a fixed number of calculations. In the full-system matrix, on the other hand, the bandwidth (which depends on how the terms are ordered) is proportional to the number of cells on the shortest side of the computational domain — to $\mathcal{O}(\sqrt{N})$ for a square domain. In this case, each of the N steps of the Gaussian elimination process requires a number of calculations itself proportional to N — resulting in $\mathcal{O}(N^2)$ total operations. At first thought it might seem that, since most of the terms in the full-system matrix are zero, the time for each step of the elimination process would be somewhat less than $\mathcal{O}(N)$. Ah... were it only true. Because of the so-called *filling* problem of sparse matrices, zero terms become filled-in with non-zero terms as the elimination process is carried out.

The answer to question (2) is not so simple. The accuracy of the full-system scheme of equation (9.7) certainly improves with increasing system dimension because the accuracy of the spatial finite-difference terms comprising R increases with decreasing node spacing. However, in the case of the factored scheme, this effect is countered by the behavior of the factorization error term $(\frac{c_1\Delta\tau}{1+c_2})^2\mathbf{J}_1\mathbf{J}_2$. The actual magnitude of the coefficients in $\mathbf{J}_1$ and $\mathbf{J}_2$ (at least those derived from spatial differences) are between $\mathcal{O}(\Delta y^{-1})$ and $\mathcal{O}(\Delta y^{-2})$, where Δy is the node spacing. This is because of the Δy and Δy^2 factors that occur in the denominators of finite-difference expressions for first- and second-partial derivatives. Therefore, the factorization error term may grow quite fast with decreasing Δy. A detailed theoretical analysis of this problem would be quite complicated.

Testing on PC-sized problems indicated that the full-system scheme was somewhat faster than the factored-scheme for a given level of accuracy. The obvious question is: What is the break-even point for the two methods? That is, for what size problem does the increased solution speed of the factored scheme make up for its reduced accuracy? I have no hard answers to this question. In light of the growing error terms of the factored method, it is entirely possible that the break-even point never occurs — the full-system solution might always be better.

Before turning to other matters, one computational point is worth discussing: How does one actually calculate the Jacobian $\mathbf{J}$ required in equation (9.7)? The problem is that the staggered-grid structure and exceptions at boundaries make the ordering of the components of V and calculation of R terms quite complicated. Rather than attempting to algebraically compute $\mathbf{J}$, with the high likelihood of human error, I chose to let Manifest do the work. By virtue of structured programming techniques and numerical differencing, Manifest is able to calculate reliable estimates of $\mathbf{J}$ components in the form

$$J_{kl} = \frac{R_k(V + h\delta_l) - R_k(V)}{h} \tag{9.9}$$

where δ_l is the Kronecker delta vector (zero except one for l-th component). The program includes time-saving logic to avoid differencing in the case where R_k has no dependence on V_l and to store intermediate values that may be needed for subsequent components of $\mathbf{J}$. Also, when calculating a R_k term, Manifest automatically replaces any reference to a dependent variable D_k by the appropriate function $F_k(V, \tau)$ according to (9.1). In this way R_k is always, ultimately, a function of V and τ. This feature makes it especially straightforward to code the finite-difference formulae in the R_k terms.

One disadvantage of the Manifest code is that, while efficient on a scalar processor, it seems to be beyond the means of the current generation of compilers to vectorize. The result is that Manifest cannot, at present, take full advantage of vector processors such as the Cray-series of computers.

9.4 Dependent Variables

Most of the dependent variables D_k are present because of the MAC staggered-grid scheme used by Manifest. The ideas behind and advantages of the MAC scheme are discussed in section 8.5.1. See especially figures 8.3 and 8.4. This section expounds on some of the computational details.

Essentially, the computational grid is subdivided into unit cells each of which is two nodes on a side. Variables M, E and T_s (density, pressure and solid

Grid Indices i, j	M, E and T_s	G_1	G_2
i even, j even	solved	y_1-interpolation	y_2-interpolation
i even, j odd	y_2-interpolation	cross-interpolation	solved
i odd, j even	y_1-interpolation	solved	cross-interpolation
i odd, j odd	cross-interpolation	y_2-interpolation	y_1-interpolation

Table 9.1: Interpolation scheme at various positions of the computational grid.

temperature) are solved in the centers of the unit cells and the curvilinear components of mass flow rate per unit area G^1 and G^2 are solved at the midpoints of the appropriate sides. The MAC scheme has much in common with a control volume formulation where the unit cell plays the role of the control volume.

Non-solved values of E, M, and $\mathbf{G}$ are interpolated or extrapolated. In principle, the interpolation formulae are arbitrary, however, the choice affects the accuracy of the solution and the bandwidth of the Jacobian matrix $\mathbf{J}$. Manifest uses first-order central and one-sided interpolation formulae. Three general-purpose interpolating routines cover all possible cases: y_1-direction interpolation, y_2-direction interpolation and cross-interpolation. The first case, y_1-interpolation is defined as follows for a variable $S[i,j]$ representing E, M, G^1 or G^2 at grid index $[i,j]$

$$S[i,j] = \begin{cases} \frac{3S[i+1,j]-S[i+3,j]}{2} & \text{if } i = 1 \\ \frac{S[i+1,j]+S[i-1,j]}{2} & \text{if } 1 < i < N_1 \\ \frac{3S[i-1,j]-S[i-3,j]}{2} & \text{if } i = N_1 \end{cases} \tag{9.10}$$

The case of y_2-interpolation is exactly analogous, except in the other direction. Cross-interpolation is defined as the function-composition of the two — y_1-interpolation of y_2-interpolated values or vice-versa. In most cases, cross-interpolation amounts to calculating the value at the center of a box as the average of the four corner values.

Table 9.1 presents the essentials of the Manifest interpolation scheme. An entry of ***solved*** in the table indicated that the corresponding variable is an independent solution variable — a component of array V. Node indices i, j are both even at the centers of unit cells, both odd at the corners and mixed odd/even at side mid-points.

9.5 Boundary Conditions

Exceptions to the staggered-grid conventions of table 9.1 occur on the boundaries of the computational grid because of the boundary conditions imposed there. Generally, one arrives at these boundary conditions by reflecting upon the physics of the problem instead of by precise mathematical deduction. In the case of Manifest:

- Mass flux G is specified on the boundary; either as zero, in the case of a no-slip impermeable wall, or as some prescribed (steady or sinusoidal) value, in the case of an active inlet.

- Gas temperature T is specified on the boundary; either as solid temperature T_s in the case of an impermeable wall, some prescribed constant value for incoming flow through an inlet or extrapolated from interior values for outgoing flow through an inlet.

The G boundary condition and the initial gas mass content completely determine system gas mass for all time. The temperature boundary condition is designed so that, (1) the outgoing enthalpy flux at the inlets has freedom to differ from the incoming value and (2), heat flux can be modeled at impermeable walls. However, the temperature boundary condition is not directly implemented in Manifest since T is not a primary gasdynamic variable. Instead, the boundary condition is used to evaluate density M on the boundary from the low Mach-number approximation to the dimensionless equation of state $M = \gamma E/T$ (ignoring kinetic energy terms), where E is evaluated on the boundary according to the scheme of table 9.1.

There is a minor ambiguity in the above boundary condition definitions for points at the end of one logically distinct segment of the boundary and the beginning of another. Such points are found at the corners of the computational domain and along the sides where an active inlet changes to an impermeable wall, and so forth. Manifest resolves these ambiguities by the simple technique of calculating boundary conditions for all points lying at the corners of unit cells as the average of the values for the neighboring two boundary points on either side, which always lie at unit-cell side midpoints. Viva the MAC scheme!

9.6 Initial Values

Specifying initial values for the solved system variables is easy in some cases but difficult in others. Mass flux $\mathbf{G}$ is easy. Manifest just initializes $\mathbf{G} = 0$ everywhere in the interior of the computational domain. Solid temperature T_s is also easy since it is, essentially, specified in the input data set. The only hard ones are M and E. They are initialized so that the initial gas temperature T equals T_s and so that the time-mean pressure, over one period in an oscillating flow solution, will equal the prescribed input value.

Here's how. First integrate the dimensionless equation of state (see section 8.4) over the physical computational domain.

$$\int \frac{P}{T} dv = \frac{\gamma - 1}{\gamma} \int M dv \tag{9.11}$$

Now, $\int M dv$ is the initial total dimensionless mass of the system and can be written

$$\int M dv = \int \overline{M} dv + \int \acute{M} dv \tag{9.12}$$

where $\overline{M}$ is the time-average local value of density and $\acute{M}$ is the initial surplus or deficit. Assuming P has negligible spatial variation it can be factored out of the left side integral of (9.11) leaving, after a little algebra

$$P = \overline{P} + \frac{\gamma - 1}{\gamma} \frac{\int \acute{M} dv}{\int T dv} \tag{9.13}$$

where $\overline{P}$ is the time-average pressure. Now $\overline{P}$ is specified as input, and one can calculate $\int \acute{M} dv$ from the sinusoidal inlet mass flux boundary conditions and $\int T dv$ from the assumed temperature profile. So, P can be solved. All that remains is to estimate the initial across-the-board value of E from the approximation $E = P/(\gamma - 1)$, and the local value for M from $M = \gamma E/T$.

Chapter 10

Grid Generation

Grid Generation is a separate field unto itself. Although the basic ideas are straight-forward, in some software packages the details can get rather involved. Manifest tries to keep things simple by generating a grid which is the simple solution to Laplace's equation. No frills.

Generally speaking, one starts with an irregularly shaped physical domain in Euclidian space (x-coordinates) and seeks to transform this into a *rectangular* computational domain in curvilinear y-coordinates. The term *rectangular* simply refers to the fact that the Euclidian domain maps onto a set of points $\{(y_1, y_2, y_3) | a_k \leq y_k \leq b_k,\ k = 1 \ldots 3\}$ where the interval endpoints a_k and b_k are fixed but otherwise arbitrary. The boundaries of the x-coordinate irregular domain map onto the curvilinear-coordinate surfaces ($y_k = a_k$ or b_k), hence the origin of the term boundary-fitted coordinates.

The inverse transformation $x_k = x_k(y_1, y_2, y_3)$ from y to x coordinates is the one that really interests us. This is because it is the y-coordinates that are suitable for use as independent variables in a computational grid. Remember that, two-dimensional Manifest specifies its computational grid as a set of discrete coordinates discrete coordinates $(y_1, y_2)[i,j] = (i\Delta y, j\Delta y)$ where indices $[i,j]$ range between limits $i = 1 \ldots N_1$ and $j = 1 \ldots N_2$ and Δy is some fixed but arbitrary spacing. If the points (x_1, x_2), corresponding to this computational grid, are plotted in Euclidean space and connected with line segments, one sees the stretched grid superimposed on the physical domain one normally associates with curvilinear coordinates. Figure 2.3 of chapter 2 shows a good Example.

The process of actually solving for the coordinate transformation $x_k = x_k(y_1, y_2, y_3)$ is the subject of the remainder of this section. In general, one needs three things:

- A differential equation that the transformation satisfies in the interior of the computational domain.

- Transformation values on the boundary.

- A solution algorithm.

10.1 The Governing Differential Equation

Arguably the simplest differential equation that gives pleasing results for grid generation is Laplace's equation. Applied to the problem at hand, it is written

$$\frac{\partial^2 x_k}{\partial y_1^2} + \frac{\partial^2 x_k}{\partial y_2^2} + \frac{\partial^2 x_k}{\partial y_3^2} = 0 \tag{10.1}$$

for $k = 1 \ldots 3$. For those of us who are not comfortable with Laplace's equation as an abstraction, an analogy from the realm of physical experience may help at this point. Consider thermal conduction in solids, which science tells us is also governed by Laplace's equation — at least after time-varying terms have settled out. Applying our intuitive notions of thermal equilibrium, look at x_k as the temperature along the boundary of the y-coordinate computational domain. Then, the equilibrium interior temperature is equivalent to x_k.

Manifest uses (10.1) as the basis for its grid generation. Considerable embellishments are possible, but for physical domains that are not too *irregular* Manifest seems to get along just fine as is. One obvious modification might be to include a source term on the right side of (10.1). This might be called an *attractor* in the jargon of those practiced in the art, and serve to concentrate the grid about regions of special interest. The only problem is: How does one specify the region of interest? (Not a trivial problem.) One might even abandon Laplace's equation entirely and replace it with a more complicated equation designed to do one thing or another. Interested readers may turn to reference [17] for a discussion of more state-of-the-art grid generation techniques.

One problem with Laplace's equation that should not go unmentioned is its difficulty near concave corners of the physical domain. In some cases the grid fails to properly stretch around the corner and winds up partly outside the domain. Presently, this can be avoided by tinkering with the corner shape or grid spacing, but would, perhaps, best be handled by adjustments at the level of the governing differential equation.

10.2 Boundary Conditions

Boundary conditions for grid generation are created by simply specifying the location of the boundary points of the discrete computational grid on the actual boundary of the physical domain. In principle this is easy to do, but the details have a great deal to do with how the final grid looks. For example, it is possible to concentrate points along the boundary to improve the local resolution of the grid, and so forth.

Manifest, however, does nothing complicated. It generates its grid boundary conditions in a completely natural way as a result of the way it breaks the physical domain into a chain of quadrilateral sub-regions. Each quadrilateral carries with it the physical x-coordinates of its corners and the fineness of the desired discretization in each direction. Manifest calculates the physical x-coordinates of the required number of equally spaced points along the exterior quadrilateral boundaries and assigns these as computational grid boundary conditions.

One useful feature of Manifest is that it can group the quadrilaterals comprising the total physical domain into distinct sub-domains for grid-generation purposes. By solving each sub-domain separately, Manifest insures that any common boundaries are undistorted in the computational grid. That is, if $\mathcal{B}$ is a physical boundary common to two adjacent sub-domains, then $\mathcal{B}$ falls on a $y_1 = constant$ coordinate line in the computational grid. This feature is handy if one wants to include a quadrilateral filled with porous material as part of a larger domain yet preserve the physical locations of its boundaries.

10.3 Solution Algorithm

An observation, which may or may not be profound, is that grid generation has much in common with the actual solution of the computational-fluid-dynamics problem. Both start with a multi-dimensional array of variables to be solved. In grid generation, the array elements hold physical coordinate values for the curvilinear grid. In a fluid-dynamics problem, the array elements hold the various gasdynamic variables of the solution. It is no surprise then that many of the algorithms of Manifest work equally well for both purposes. Since, historically, the fluid-dynamics part of Manifest was in place before the grid-generation part, it was a relatively simple task to extend Manifest to grid generation.

Here's how the grid generation algorithm works. Think of all the discrete non-boundary points in the computational grid as ordered in some way, say from 1 to M, and let $X(m)$ denote the value of physical coordinate x_k at the m-th location for $m = 1 \ldots M$. The details of the ordering are not important, nor does it matter whether the k in x_k is 1, 2 or 3 — each is considered as a separate problem in turn. The components $X(m)$ form an array; call it X. Now, at each grid location m, we can write down a function $F_m(X)$, defined as the finite-difference equivalent of the left-hand side of governing differential equation (10.1). $F_m(X)$ is defined in terms of $X(m)$, nearby X values and, perhaps, boundary conditions, if grid location m is near the boundary. Collectively, all the components $F_m(X)$ may be designated $F(X)$. From this point of view, our problem is to solve for X that makes $F = 0$, for then, the finite difference version of (10.1) will be satisfied everywhere.

Manifest uses the multi-dimensional version of Newton's iterative method to solve $F = 0$. Starting from an initial guess X^0 Manifest computes successive

X^n by solving for ΔX^n in the linear system of equations

$$J^n \Delta X^n = -F(X^n) \tag{10.2}$$

where $\Delta X^n = X^{n+1} - X^n$ and J^n is the Jacobian matrix $\frac{\partial F}{\partial X}$ evaluated at X^n. Iteration stops when $||F||$ gets smaller than some convergence tolerance.

Already armed with the heavy artillery necessary to execute the fluid dynamics computational algorithm, it was easy to implement this relatively simple algorithm for solving Laplace's equation. Convergence is generally obtained in only a few iterations.

Bibliography

[1] D. A. Anderson, J. C. Tannehill, R. H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*, Hemisphere (1984).

[2] R.M. Beam and R.F. Warming, *An Implicit Finite-difference Algorithm for Hyperbolic Systems in Conservation-Law Form*, Journal of Computational Physics, 22, 87–110 (1976).

[3] R.M. Beam and R.F. Warming, *An Implicit Factored Scheme for the Compressible Navier-Stokes Equations*, AIAA Journal, Vol. 16, No. 4, 393–402 (1978).

[4] J. Bear, *Transport Phenomena in Porous Media — Basic Equations*, In: *Fundamentals of Transport Phenomena in Porous Media*, Editors: J. Bear and M.Y. Corapcioglu, Martinius Nijhoff, 5–61 (1984).

[5] A. I. Borisenko, I. E. Tarapov, Translated by R. A. Silverman, *Vector and Tensor Analysis With Applications*, Prentice-Hall (1968).

[6] D. Gedeon, *Computational Techniques for the Two-Dimensional Gasdynamic Equations in Stirling Engine Regenerators and Associated Manifolds*, 20th IECEC, Society of Automitive Engineers, 3.354–3.359 (1985).

[7] D. Gedeon, *A Globally-Implicity Stirling cycle Simulation*, 21st IECEC, Vol. 1, American Chemical Society, 550–556 (1986).

[8] Heames T.J., Uherka D.J., Zabel J.C., Daley J.G., *Stirling Engine Thermodynamic Analysis: A Users Guide to SEAM1*, Argonne National Laboratory, ANL-82-59 (1982).

[9] R.I. Issa, *Numerical Methods for Two- and Three-Dimensional Recirculating Flows*, in: *Computational Methods for Turbulent, Transonic and Viscous Flows*, Editor: J.A. Essers, Hemisphere, 183–211 (1983).

[10] W.M. Kays and A.L. London, *Compact Heat Exchangers, 3rd Edition*, McGraw-Hill (1984).

[11] I.F. Macdonald, M.S. El-Sayed, K. Mow and F.A.L. Dullien, *Flow through Porous Media - the Ergun Equation Revisited*, Ind. Eng. Chem. Fundam., Vol. 18, No. 3, 199–208 (1979).

[12] H. E. Newell, *Vector Analysis*, McGraw-Hill (1955).

[13] R. Peyret, T. D. Taylor, *Computational Methods for Fluid Flow*, Springer-Verlag (1983).

[14] J.R. Seume and T.W. Simon, *Oscillating Flow in Stirling Engine heat Exchangers*, 21st IECEC, Vol. 1, American Chemical Society, 533–538 (1986).

[15] H. Schlichting, *Boundary-Layer Theory, Seventh Edition*, McGraw-Hill (1979).

[16] A.E. Taylor, *Advanced Calculus*, Ginn and Company (1955).

[17] J.F. Thompson, F.C. Thames, and C.W. Mastin, *TOMCATM — A Code for Numerical Generation of Boundary-Fitted Curvilinear Coordinate Systems on Fields Containing Any Number of Arbitrary Two-Dimensional Bodies*, Journal of Computational Physics, Vol. 24, 274–302 (1977).

[18] M. Van Dyke, *An Album of Fluid Motion*, The Parabolic Press (1982).

[19] M. Vinokur, *Conservation Equations of Gasdynamics in Curvilinear Coordinate Systems*, Journal of Computational Physics, Vol. 14, 105–125 (1974).

[20] H. Viviand, *Conservative Forms of Gas Dynamic Equations*, La Recherche Aerospatiale, No. 1, 65–68 (1974).

Nomenclature

Vectors and tensors are set in bold face type. Their components in rectangular or curvilinear coordinates are set in normal face with subscripts or superscripts respectively. The summation convention for repeated indices applies.

Variables may be either dimensional or dimensionless depending on their context. Generally, in the theory part of this report they are dimensional prior to section 8.4 and dimensionless thereafter.

Roman Letter Symbols

c_1, c_2	Coefficients of Ergun friction factor $f = c_1/R_e + c2$
c_v, c_p	Gas specific heats
c_s	Solid specific heat
d_h	Hydraulic diameter
D	Set of dependent computational variables
e	$\varepsilon + 1/2\mathbf{V}\cdot\mathbf{V}$, void-average gas mass specific total energy
$\mathbf{e}_i$	Rectangular coordinate basis vectors $i = 1\ldots3$
E	ρe, void-average gas energy density
E_s	$(1-\beta)\rho_s c_s T_s$, solid volume-specific energy
$\mathbf{E}^3$	Euclidean three-dimensional space
f	Darcy friction factor
$\mathbf{g}_j$	Curvilinear coordinate basis vectors $j = 1\ldots3$
$\mathbf{G}$	$\beta\rho\mathbf{V}$, mass flux per unit total area; continuous at porosity discontinuities
$\mathbf{I}$	Identity matrix
J	Jacobian of the coordinate transformation matrix $\det(\mathbf{Y_X})$
$\mathbf{J}$	Finite-difference system matrix $\frac{\partial R}{\partial V}$
M	Void-average gas density
$\mathbf{n}$	Outward-pointing unit surface normal
P	Void-average pressure

$\mathbf{q}$	$-\kappa\nabla T$, heat flux vector field
$\mathbf{q}_e$	$-\kappa_e\nabla T$ effective heat flux vector field
Q	Gas-to-solid heat flux per unit void volume
R	Array of finite-difference approximations (without time partials) to the gasdynamic equations at each point in V
R_e	Reynolds number
t	Time
T	Void-average gas temperature
T_s	Solid-average temperature
u	Void-average gas velocity components
$\mathbf{V}$	Void-average gas velocity vector field
V	Set of independent computational variables
x	Rectangular coordinates
$\mathbf{X_y}$	Matrix of coordinate transformation partials $\frac{\partial x}{\partial y}$
y	Curvilinear coordinates
$\mathbf{Y_x}$	Matrix of coordinate transformation partials $\frac{\partial y}{\partial x}$

Greek Letter Symbols

β	Porosity
ε	Void-average gas mass-specific internal energy (C_vT for an ideal gas)
γ	Ratio of specific heats c_p/c_v
κ	Thermal conductivity
κ_e	Effective thermal conductivity for lumped gas and solid
λ	$(1-\beta)\rho_s c_s/(\beta\rho_0 c_p)$, solid-to-void heat capacity ratio
μ	Molecular viscosity
μ_e	Effective viscosity in porous materials
τ	Time in curvilinear coordinates
$\boldsymbol{\tau}$	Viscous stress tensor
$\boldsymbol{\tau}_e$	Effective viscous stress tensor in porous materials (based on μ_e)
ρ	Void-average gas density
ρ_s	Solid-average density
$\boldsymbol{\sigma}$	$\boldsymbol{\tau} - P\mathbf{I}$, total stress tensor
$\boldsymbol{\phi}$	Empirical permeability tensor for porous materials
ω	Angular frequency

Script Letter Symbols

$\mathcal{B}$	Physical domain boundary
$\mathcal{C}$	Elemental volume for porous flow equation derivation
$\mathcal{S}_i$	Surfaces of $\mathcal{C}$ oriented in coordinate directions $i = 1 \ldots 3$
$\mathcal{S}_v$	Surface of $\mathcal{V}$
$\mathcal{S}_{v_1}$	Part of $\mathcal{S}_v$ on the faces of $\mathcal{C}$
$\mathcal{S}_{v_2}$	Part of $\mathcal{S}_v$ in the interior of $\mathcal{C}$
$\mathcal{V}$	Void volume within $\mathcal{C}$

☆U.S. GOVERNMENT PRINTING OFFICE: 1989 - 648-162/81030

National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-182290	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Manifest: A Computer Program for 2-D Flow Modeling in Stirling Machines		5. Report Date May 1989
		6. Performing Organization Code
7. Author(s) David Gedeon		8. Performing Organization Report No. None
		10. Work Unit No. 586-01-11
9. Performing Organization Name and Address Gedeon Associates 16922 South Canaan Road Athens, Ohio 45701		11. Contract or Grant No. NAS3-25195
		13. Type of Report and Period Covered Contractor Report Final
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Lewis Research Center Cleveland, Ohio 44135-3191		14. Sponsoring Agency Code

15. Supplementary Notes

Project Manager, Roy C. Tew, Jr., Power Technology Division, NASA Lewis Research Center.

16. Abstract

This report is about a computer program named Manifest, after *manifold* and *estimate* which more or less describe its intended purpose. That is, Manifest is a program one might want to use to model the fluid dynamics in the manifolds commonly found between the heat exchangers and regenerators of stirling machines. But not just in the manifolds—in the regenerators as well. And in all sorts of other places too, such as: in heaters or coolers, or perhaps even in cylinder spaces. There are probably nonstirling uses for Manifest also. In broad strokes, Manifest will:

- Model oscillating internal compressible laminar fluid flow in a wide range of two-dimensional regions—either filled with porous materials or empty.
- Present a graphics-based user-friendly interface, allowing easy selection and modification of region shape and boundary condition specification.
- Run on a personal computer, or optionally (in the case of its number-crunching module) on a supercomputer.
- Allow interactive examination of the solution output so the user can view vector plots of flow velocity, contour plots of pressure and temperature at various locations and tabulate energy-related integrals of interest.

17. Key Words (Suggested by Author(s))

Stirling engine
Computer program
Two-dimensional model

Date for general release May 1991.

Subject Category 34

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No of pages 157	22. Price* A08