RECEIVED
NOV 02 1998
OSTI

*GASFLOW: A Computational
Fluid Dynamics Code for Gases,
Aerosols, and Combustion*

VOLUME 2
User's Manual

Los Alamos
NATIONAL LABORATORY

*Los Alamos National Laboratory is operated by the University of California
for the United States Department of Energy under contract W-7405-ENG-36.*

An Affirmative Action/Equal Opportunity Employer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither The Regents of the University of California, the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by The Regents of the University of California, the United States Government, or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of The Regents of the University of California, the United States Government, or any agency thereof. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

DISCLAIMER

**Portions of this document may be illegible
electronic image products. Images are
produced from the best available original
document.**

*GASFLOW: A Computational
Fluid Dynamics Code for Gases,
Aerosols, and Combustion*

VOLUME 2

User's Manual

J. W. Spore

*J. R. Travis**

*P. Roysl**

K. L. Lam

T. L. Wilson

C. Müller

*G. A. Necker**

B. D. Nichols

**Forschungszentrum Karlsruhe, Box 3640, 976021 Karlsruhe, Germany.*

CONTENTS

	Page
ACKNOWLEDGMENTS.....	xi
EXECUTIVE SUMMARY	xiii
1 INTRODUCTION	1
1.1 Code Capabilities	1
1.2 Computational Method	2
1.3 General Code Features	3
2 INPUT DATA FORMAT	6
3 GEOMETRY DEFINITION	12
3.1 Cell Labeling Convention	12
3.2 Mesh Generation	12
3.2.1 Direct Input of Grid Locations	13
3.2.2 Automatic Mesh Generation	14
3.3 Definition of Walls and Obstacles	18
3.3.1 Walls	18
3.3.2 Obstacles	19
3.3.3 Holes	20
3.3.4 Fractional Area	25
3.4 Definition of the 1D Duct Network	27
3.4.1 Defining Node Points	29
3.4.2 Defining Ducts	30
3.4.3 Defining Junctions	34
3.4.4 Network Connection to a 3D Block	35
3.4.5 Defining Flow Pressure Losses	35
3.4.6 Defining Wall Friction	37
3.4.7 Defining Filters	37
3.4.8 Defining Dampers	38
3.4.9 Defining Blowers	39
3.5 Checking Geometric Model	41
4 SPECIFICATION OF GAS SPECIES AND PROPERTIES	46
4.1 Definition of Gas Species	46
4.2 Definition of Transport Properties	49
5 INITIAL AND BOUNDARY CONDITIONS	52
5.1 Specification of Initial Conditions	52
5.1.1 Fluid Composition and State	52
5.1.2 Fluid Velocities	59

CONTENTS (cont)

	Page
5.2 Specification of Boundary Conditions	59
5.2.1 Global Definition.....	60
5.2.2 Local Definition.....	62
5.2.3 Diffusion Cutoff and Mass Balance with Source Reservoirs	70
5.2.4 Boundary Conditions from SORTAM File.....	73
 6 DEFINITION OF SOLID HEAT STRUCTURES	 77
6.1 Wall/Slab Heat Structures.....	77
6.2 Configuration Control for Slabs and Walls.....	79
6.3 Heat Conduction in Wall Heat Structures	83
6.4 Heat Conduction in Sink Heat Structures.....	92
6.5 Other Heat Structure Input.....	96
6.6 Heat Conduction in Slab Heat Structures (Boundary Cells)	97
6.7 Background for Defining Steady-State Temperature Profiles	102
6.7.1 Fluid Conditions from GASFLOW Define Both Sides (Applies Only to Walls).....	103
6.7.2 Steady-State Fluid Conditions Input for the Negative or Positive Side of the Structure	104
6.7.3 Direct Input of Steady-State Heat Flux on the Negative or Positive Side of the Structure	105
6.8 Heat Fluxes into Slabs, Walls, and Sinks.....	106
6.9 Balancing of Heat Structure Surfaces.....	107
 7 PHYSICAL MODEL OPTIONS.....	 108
7.1 Body Forces.....	108
7.2 Diffusion of Mass, Energy, and Momentum.....	108
7.3 Turbulence.....	109
7.3.1 Algebraic Model.....	110
7.3.2 The κ - ϵ Model.....	111
7.4 Combustion.....	113
7.5 Heat Transfer	114
7.6 Aerosol Model	114
7.6.1 Description of Particle Initialization Input Parameters.....	115
7.6.2 Description of Particle Transport Input Parameters.....	119
7.6.3 Description of Particle Deposition Model Input.....	121
7.6.4 Description of Entrainment Input Parameters.....	124
7.6.5 Description of Particle Cloud Model Input Parameters.....	126
7.6.6 Particle Model Restart	127
7.7 Special Containment Models.....	127
7.7.1 Ignitor Model.....	127

CONTENTS (cont)

	Page
7.7.2 Sump Model.....	129
7.7.3 Recombiner Model.....	132
8 OPTIONS ON NUMERICAL SOLUTION PROCEDURE.....	138
8.1 Pressure Iteration.....	138
8.2 Time-Step Control.....	140
8.3 Advection Scheme.....	141
9 OUTPUT AND RESTART.....	143
9.1 Graphical Outputs.....	143
9.1.1 Basic Plots.....	143
9.1.2 Geometry Plots.....	144
9.1.3 Time-History Plots.....	144
9.1.4 Profile Plots.....	148
9.1.5 Contour Plots.....	150
9.1.6 Velocity Vector Plots.....	151
9.1.7 Graphic and Tabular Particle Data Output.....	153
9.1.8 Graphic Display of Detonation Criterion.....	156
9.2 Printed Output.....	160
9.3 Output To Terminal.....	161
9.4 Restart.....	161
10 GENERAL USER GUIDANCE.....	163
11 SAMPLE PROBLEMS.....	165
11.1 Von Kármán Vortex Street.....	165
11.2 Hydrogen Burn.....	165
11.3 Particle Model Sample Problem.....	173
12 REFERENCES.....	190

APPENDICES

	Page
A Summary of Variables in NAMELIST Group xput.....	A-1
B Summary of Variables in NAMELIST Group meshgn.....	B-1
C Summary of Variables in NAMELIST Group rheat.....	C-1
D Summary of Variables in NAMELIST Group grafic.....	D-1
E Summary of Variables in NAMELIST Group innet.....	E-1
F Summary of Variables in NAMELIST Group parts.....	F-1
G Sample Input Deck with Minimum Data Required.....	G-1

FIGURES

2-1a	Sample GASFLOW input listing, showing typical comments that can be included in an <code>ingf</code> input file	7
2-1b	Sample GASFLOW input listing, showing <code>innet</code> NAMELIST group	7
2-1c	Sample GASFLOW input listing, showing <code>xput</code> NAMELIST group	8
2-1d	Sample GASFLOW input listing, showing two input NAMELIST groups, <code>meshgn</code> and <code>grafic</code>	9
3-1	GASFLOW numbering convention for cells (or cell centers) and cell faces using the <i>i</i> -direction with <code>imax = 7</code> for illustration	13
3-2	Two-dimensional <i>x-z</i> view of a mesh generated by the GASFLOW automatic mesh generator	16
3-3	Two-dimensional, <i>r-θ</i> view of a mesh generated by the GASFLOW automatic mesh generator	17
3-4	Two-dimensional <i>x-z</i> view of a mesh containing two wall surfaces and two obstacle regions generated by walls and moobs input definitions	21
3-5	A geometric model with one obstacle and two walls to demonstrate the holes option	23
3-6	A geometric model with one obstacle (moobs) and two walls (walls) with three penetrations cut through the <i>j</i> =6 plane (<i>y</i> -plane cut) to demonstrate the holes option	24
3-7	Network geometry	28
3-8	Example network	29
3-9	Four cell duct	30
3-10	Mesh fragment of a 3D block connected to a 1D network	36
5-1	Example of applying an internal mass, momentum, and energy source	73
6-1	Illustration of an obstacle surface belonging to a wall heat structure or to a slab heat structure, depending on the value of slabthk	79
6-2	Definition of heat-conducting structures for obstacles and boundary cells	81
6-3	Example of the definition of slabs and walls on a dice obstacle and on a thin obstacle with thickness <i>d</i> in the <i>x</i> -direction	82
6-4	Noding scheme for 10 heat-conducting elements with small surface nodes and internal mesh expansion in sinks, slabs, and walls	87
6-5	Steady-state solution for Example 1, the 10 cm thick composite structure	89
6-6	Calculations of the wall surface temperature on the positive side and an analytical solution for 1D heat conduction through a 10 cm wall of steel or concrete	91

FIGURES (cont)

	Page
6-7 Schematic representation of the distributed heat sink used in GASFLOW.	93
7-1 Example of a sump.....	132
7-2 Examples of recombiner types and setup for a vertical cut through the computational mesh at $j = 6$	137
9-1 Example problem to demonstrate the utility of the iroomdef input option	157
9-2 Another example problem to demonstrate the utility of the iroomdef input option.....	159
11-1 Input listing for Von Kármán Vortex Street test problem.....	166
11-2 The GASFLOW mesh for calculating flow field behind a rectangular block.....	169
11-3 Velocity vectors for flow around flat plate.....	170
11-4 Time history velocity in the Y-direction at $i=18, j=3, k=2$ for flow around flat plate.....	170
11-5 Input listing for the hydrogen burn sample problem.....	171
11-6 Temperature contours at 0.5 s.....	173
11-7 Velocity vectors at 0.5 s showing upward motion near the ignition location.....	174
11-8 Temperature contours at 1.0 s. The hydrogen flame is split between the lower and middle compartment.....	175
11-9 Velocity vectors at 1.0 s, showing fluid motion spreading to other locations from the ignition point	176
11-10 Temperature contours at 2.0 s. The flame has spread to most areas, except in regions below the bottom and top compartments	177
11-11 Aerosol test input deck.....	178
11-12 All particles at Time = 1.1 seconds	184
11-13 Class 1, size 1 particles at 1.2 seconds.....	185
11-14 Class 2 particles at 1.2 seconds.....	186
11-15 All particles at 1.5 seconds.....	187
11-16 Time history plots for particle concentrations and deposition.....	188
11-17 Time history plots for particles	189
G-1 Input deck showing minimum set of data required to run GASFLOW	G-2

TABLES

	Page
1-1 Description of input and output files in GASFLOW	5
3-1 Specification of 2D grid plots at cell face and/or cell center locations via input variable maxcyc in NAMELIST group xput.....	45
4-1 Properties of gas species available in GASFLOW	48
9-1 Solution variables available for plotting.....	145

ACKNOWLEDGMENTS

The authors would like to thank Pratap Sadasivan for his development of the general slab heat-transfer boundary condition model. We would like to thank Maurizio Bottoni, Allen Notafrancesco, Asimios Malliakos, and George Niederauer for reviewing the document. We thank Carolyn Robinson for her careful preparation and technical editing of this document.

GASFLOW: A Computational Fluid Dynamics Code for Gases, Aerosols, and Combustion

by

**J. W. Spore, J. R. Travis, P. Royle, K. L. Lam, T. L. Wilson,
C. Müller, G. A. Necker, and B. D. Nichols**

EXECUTIVE SUMMARY

Los Alamos National Laboratory (LANL) and Forschungszentrum Karlsruhe (FzK) are developing GASFLOW, a three-dimensional (3D) fluid dynamics field code as a best-estimate tool to characterize local phenomena within a flow field. Examples of 3D phenomena include circulation patterns; flow stratification; hydrogen distribution mixing and stratification; combustion and flame propagation; effects of noncondensable gas distribution on local condensation and evaporation; and aerosol entrainment, transport, and deposition.

An analysis with GASFLOW will result in a prediction of (1) the gas composition and discrete particle distribution in space and time throughout the facility and (2) the resulting pressure and temperature loadings on the walls and internal structures with or without combustion. A major application of GASFLOW is for predicting the transport, mixing, and combustion of hydrogen and other gases in nuclear reactor containments and other facilities. It has been applied to situations involving transporting and distributing combustible gas mixtures. It has been used to study gas dynamic behavior (1) in low-speed, buoyancy-driven flows, as well as sonic flows or diffusion dominated flows; and (2) during chemically reacting flows, including deflagrations. The effects of controlling such mixtures by safety systems can be analyzed.

The code version described in this manual is designated GASFLOW 2.1, which combines previous versions of the United States Nuclear Regulatory Commission code HMS (for Hydrogen Mixing Studies) and the Department of Energy and FzK versions of GASFLOW. The code was written in standard Fortran 90. This manual comprises three volumes. Volume I describes the governing physical equations and computational model. Volume II describes how to use the code to set up a model geometry, specify gas species and material properties, define initial and boundary conditions, and specify different outputs, especially graphical displays. Sample problems are included. Volume III contains some of the assessments performed by LANL and FzK. GASFLOW is under continual development, assessment, and application by LANL and FzK. This manual is considered a living document and will be updated as warranted.

GASFLOW is a finite-volume code based on robust computational fluid dynamics numerical techniques that solve the compressible Navier-Stokes equations for 3D

volumes in Cartesian or cylindrical coordinates. The code can model geometrically complex facilities with multiple compartments and internal structures in a computational domain of multiple 3D blocks of cells connected by one-dimensional flow paths. GASFLOW has transport equations for multiple gas species, liquid water droplets, and total fluid internal energy. A built-in library contains 23 gas species and 1 liquid water species.

GASFLOW can simulate the effects of two-phase dynamics with the homogeneous equilibrium model, two-phase heat transfer (steam condensation and water evaporation) to walls and internal structures, chemical kinetics from catalytic hydrogen recombination and combustion processes, and fluid turbulence. The code can model two-phase heat transfer to and from walls and internal structures by convection and mass diffusion.

Wall shear stress models are provided for bulk laminar and turbulent flow. Two turbulence models available: algebraic and κ - ϵ , which provide zero- and two-transport-equation models, respectively, that determine turbulent velocity and length scales needed to compute the turbulent viscosity. Terms for the turbulent diffusion of different species are included in the mass and internal energy equations.

Chemical energy of combustion involving hydrogen provides a source of energy within the gaseous regions. A one-step global chemical kinetics model based on a modified Arrhenius law accounts for local hydrogen and oxygen concentrations. A two-step chemical kinetics model divides the chemical reaction into two parts: (1) an induction phase, which builds radicals and releases little energy; and (2) an energy release phase, where the radicals recombine. Hydrogen is ignited using a generalized ignitor model that represents both spark- and glow-plug-type designs. A catalytic hydrogen combination with oxygen is modeled using data from both the Nuclear Ingenieur Service and Siemens recombiner box designs.

The aerosol model comprises the following models: Lagrangian discrete particle transport, stochastic turbulent particle diffusion, particle deposition, particle entrainment, and particle cloud. These models incorporate the physics of particle behavior to model discrete particle phenomena and allow the code user to track the transport, deposition, and entrainment of discrete particles, as well as clouds of particles.

GASFLOW 2.1 models have been extended beyond GASFLOW 1.0 with the following developments:

- independent multiblock computational domains;

- independent multiblocks connected on external boundaries by a ventilation system;
- a fractional area treatment to model flow areas smaller than a cell face area;
- accurate internal energy as a function of temperature to fourth-degree polynomials;
- gas properties library of thermochemical and transport extended to 30 species;
- homogeneous equilibrium model for fluid mixture;
- droplet depletion or "rainout";
- two-phase heat and mass transfer to structural components;
- both spark- and glow-plug ignitor models;
- empirical hydrogen combustion limits;
- hydrogen recombiner models; and
- transport, deposition, and entrainment of discrete particles.

Each version of GASFLOW is tested with a Standard Test Matrix of 126 problems in four categories: (1) feature tests for the computer science aspects of the code; (2) functional tests for code algorithms, equations, logic paths, and decision points; (3) comparisons with analytical solutions; and (4) comparisons with data. During the development of GASFLOW 2.1, many experiments were modeled and analyzed. All 19 analytical solutions and the following 6 experiments are documented in Volume III: (1) the Bureau of Mines Spherical Test Chamber; (2) the Sandia FLAME Experiment; (3) Battelle Model Containment (BMC) Test GX6; (4) Battelle Model Containment Test HYJET JX7; (5) Heißdampfreaktor (HDR) Test T31.5; and (6) Phebus Tests 4A, 4B, 6A, and 10A. All of the problems in the Standard Test Matrix and in the initial set of assessments were executed successfully by GASFLOW 2.1 without modification, and the results are in agreement with the analytical solution or the test data. For example, other assessment analyses of the HDR, BMC, Russian RUT, and LANL TA-55 facilities are documented in the literature.

The Bureau of Mines Spherical Test Chamber (having a volume of 120 L) has been used to investigate the flammability of hydrogen, ammonia, nitrous oxide, and air mixtures under quiescent and turbulent conditions. GASFLOW calculated combustion of mixtures of hydrogen concentrations for 5 to 40 vol % diluted in air, 1:1 ratios of hydrogen and nitrous oxide in air, and very low hydrogen and nitrous oxide concentrations in air with 1 and 2% ammonia added. Beginning with ambient temperature and pressure, GASFLOW accurately calculated that the behaviors of both

fuel-lean and fuel-rich mixtures for hydrogen are (1) little pressure rise for very low hydrogen concentrations, (2) downward flame propagation and nearly complete combustion for approximately 8% hydrogen, and (3) and maximum pressure for a stoichiometric (29.6% hydrogen) mixture. Good results were obtained for the mixtures where nitrous oxide was added, although the pressure rise rate was conservatively high by up to 20%. In conclusion, the one-step finite-rate chemical kinetics model yields good results but is limited. For more complex combustion problems, a reduced set of chemical kinetics mechanisms or a two-step method based on an induction parameter model should be used. Both improved models are derived from detailed chemical kinetics mechanisms.

The FLAME facility (1.8 m wide, 2.4 m high, 30 m long = 136 m³) at Sandia National Laboratories was used to study flame acceleration and deflagration-to-detonation transition of hydrogen-air mixtures. In experiment F-21, 13% hydrogen was added and mixed with a fan. After 30 min, gas samples indicated stratification ranging from 15% at an upper location to 10% at a lower location. The test was chosen to assess transient mixing and stratification calculations by GASFLOW, beginning with stagnant air and followed by hydrogen injection, mechanical mixing, and natural stratification. The algebraic turbulence model was used, and results were not very sensitive to the chosen turbulent length scale because mixing was largely achieved by molecular diffusion. GASFLOW showed that the hydrogen jet does not penetrate very far before it becomes a buoyant plume that gradually expands. At 100 s after injection ends, stratification is maximum (approximately 17% at the top and approximately 4% at the bottom) in the early part of the mixing phase, which lasted for 1800 s, or 30 min. When the gas samples were taken, GASFLOW calculated upper and lower volume fractions of 15.4% and 9.8%, respectively, within the reported uncertainty of 0.5%.

The Battelle Model Containment (BMC) Test GX6 was used to validate the catalytic recombiner model in GASFLOW. This test was conducted using the central cylindrical room R1/R3 and the annular segment compartments R5 through R8; these rooms were sealed off from the remainder of the BMC. Each compartment has a gas volume of 49 m³; the total for test GX6 was 209 m³. A Siemens catalytic recombiner was placed next to the inner wall of R5, not far from the opening to R6. Steam was injected into rooms R5, R6, and R7. Hydrogen was injected into R8. Ten hours of the test was calculated with GASFLOW. The calculated pressure was slightly higher than measured and stayed more level, indicating a combination of too much in-leakage in the calculation and slightly different boundary condition pressures. The drop in pressure at approximately 6 h into the transient, which was caused by terminating the inflow of steam, was calculated to be slightly earlier and larger. This drop indicated that the condensation rate in GASFLOW is slightly larger than is inferred from the test data. The overall agreement is good and captures the dominant trends in the data. Temperature comparisons at several different locations are excellent and capture dominant trends in the data. In particular, the temperature rise through the recombiner was calculated quite accurately.

The BMC Test JX7, also known as HYJET, was used to validate the capability to model small- and large-scale effects of a turbulent jet and convection with containment within one calculation. This test used all of the rooms in the BMC, except the annular segment compartment R7, which has a total gas volume of 600 m^3 . Near the bottom of R6, a helium mass of 9.25 kg was injected vertically under the openings to R5 and to the dome over a period of 200 s. The nozzle had a diameter of 95 mm and an average injection velocity of 42 m/s. The jet from the nozzle extended all the way from the source location into the dome region, where the helium then stratifies as a cloud. Using a model with 50,000 computational cells, GASFLOW calculated the helium concentrations at four sensor locations: in the dome, near the bottom of the central room, and in the upper and lower parts of the ring room. In general, GASFLOW predicts the dominate trends well and predicts the helium stratification consistently with the data. Good agreement was obtained in the central room and in the lower part of the ring room. GASFLOW overpredicted the helium concentration in the dome region; this is attributed to insufficient air entrainment along the jet surface.

HDR Test T31.5, also known as an International Standard Problem, ISP-23, was used to validate the capability to model hydrogen and steam transport and distributions in a large-scale containment during a simulated large-break loss-of-coolant accident. The containment has a steel shell with a height of 60 m, an internal diameter of 20 m, and a free volume of $11,300 \text{ m}^3$. The containment has 72 compartments connected with about 200 passageways below the approximately 5000-m^3 volume of the open hemispherical-dome region. In this test, a blowdown from the pressure vessel injected 30,000 kg of a steam-water mixture over the first 50 s at the 22-m level. From 21 to 36 min, superheated steam was injected at a rate of 2.2 kg/s. From 36 to 48 min, a light gas (14% hydrogen and 85% helium) was injected. The GASFLOW model for T31.5 used 12,300 computational cells with obstacles and barriers to model all major floors and walls and all compartments. Heat structures were modeled extensively through the containment. An isentropic expansion model was used for the blowdown. GASFLOW predicts the dominate trends well for pressure, temperature, and concentration for this complex test.

Pressure peaks and rapid pressure equilibrium calculated by GASFLOW throughout the containment agreed quite well with the data. After 5 min, GASFLOW starts predicting a slower pressure decay, reaching a difference of 0.08 bars after about 15 min. One explanation is the lack of a film relocation model. The film vaporization model provides a continuous steam source that slows down the pressure decay; however, that should be cut off as dryout occurs. Calculated temperature histories reflect the somewhat higher asymptotic pressures. Calculated values are high by about 10°C in the middle and upper containment and by 20° to 25°C in the bottom containment. Three effects could cause high bottom temperatures; the most likely is that a significant amount of water collects in the bottom, creating an additional heat sink, especially after passing over cold surfaces on the way down. GASFLOW calculations of light-gas concentrations in the spiral staircase and in the dome are in excellent agreement with data. In two places, GASFLOW concentrations vary significantly from the data. One is near the source, where concentration gradients vary strongly and the computational

mesh is not fine enough to resolve them well. Another is at the bottom, where the concentration is very low (only about 0.5%).

Four steady-state tests (4A, 4B, 6A, and 10A) in the Phebus facility were used to validate the condensation model in GASFLOW. The facility (approximately 5.7 m high and 1.8 m in diameter = 10 m^3) has a insulated double skinned wall through which coolant flows to maintain a homogeneous inner wall temperature of approximately 110°C . Inside the top part of the facility is (1) a wet condenser, which consists of three steel rods internally cooled to a specific temperature, and (2) a dry condenser, which is heated to prevent condensation. Initial conditions were air diluted with 39 vol % steam at 110°C and 1.9 bars. Steam sources for this set of tests varied from 1 g/s at 114°C and 1.65 bar to 4 g/s at 125°C and 2.3 bar, directly vertically upward from the bottom. A GASFLOW model of 715 computational cells modeled a 60-degree sector. Condensation parameters were set to standard values. The calculated final pressures during the condensation process were found to agree with the data within 1% to 5% for the various cases.

1 INTRODUCTION

1.1 Code Capabilities

GASFLOW is a best-estimate computer code developed at Los Alamos National Laboratory (LANL) and Forschungszentrum Karlsruhe (FZK) for use by the US Nuclear Regulatory Commission (NRC) and FZK for predicting the transport, mixing, and combustion of hydrogen and other gases, liquid water droplets, and aerosols in nuclear reactor containments and other nonnuclear buildings. The code can model geometrically complex facilities having internal structures and multiple compartments and is useful for facilities that are required to address various nuclear and nonnuclear safety issues. The gas/liquid mixture modeled may consist of components included in a built-in library of 25 species. The aerosols modeled can be of different material densities and sizes. The fluid flow modeled may be laminar or turbulent, subsonic or supersonic, single phase or two phase, and with or without aerosols. Momentum, heat, and mass transfer within the fluid is determined by physical mechanisms such as diffusion (molecular and/or turbulent) and convection. Heat conduction in solid structures is calculated and is coupled to the fluid dynamics through the wall temperatures and heat fluxes at the fluid-solid interfaces. If steam is present, the code predicts its rate of condensation based on the local wall temperature and bulk fluid conditions. The (simplified) chemical kinetics of the burn of a hydrogen-air-steam mixture can be solved simultaneously with the fluid dynamics to predict flame propagation and acceleration.

A typical application of GASFLOW may be in predicting stratification of hydrogen distribution in a nuclear reactor containment building during the course of a severe accident in which large amounts of the flammable gas are produced. In analyzing containment designs [such as the European Power Reactor (EPR)] that lack active mixing mechanisms such as fans and internal sprays but rather rely on natural circulation for cooling and mixing the containment atmosphere, the three-dimensional (3D), multispecies, variable-density capabilities of an analysis tool such as GASFLOW are useful. The calculation will identify *local* regions of high hydrogen concentration within the multicompartment containment geometry where steam condensation also is occurring. GASFLOW modeling capability allows for the positioning of catalytic recombiners at various locations throughout the containment. The GASFLOW model for recombiners simulates a slow recombination of hydrogen and oxygen to form water even though the mixture is not flammable. The maximum hydrogen concentration can then be compared against flammability and detonation limits established experimentally to assess the risk of a hydrogen burn. The calculation can be carried one step further by assuming that the hydrogen gas mixture is ignited to determine the resulting pressure and temperature loads on the containment structures and safety-related equipment. Combustion modes that can be calculated include diffusion flames and propagating deflagrations. GASFLOW is especially useful in predicting local

pressure spikes in narrow passages between subcompartments where the interaction between fluid turbulence, temperature, and fuel concentration in the accelerated jet can be expected to give rise to instantaneous ignition.

1.2 Computational Method

In this section, we briefly summarize the computational method adopted in GASFLOW. This is included so that the code user can quickly review the numerical approach and models. Further details on the theoretical aspects of GASFLOW are given in the Theory and Computational Model manual.

GASFLOW is a finite-volume code that solves the time-dependent, 3D, compressible Navier-Stokes equations. Transport equations for the internal energy and for multiple gas species, a liquid droplet species, and multiaerosol sizes are also solved. The computational domain is discretized by a mesh of regular orthogonal cells in either Cartesian or cylindrical geometry. The computational domain can be a single 3D block, multiple 3D blocks, a single 1D network of ducts and ventilation system components, or a combination of 3D blocks and a 1D network of ducts and ventilation system components. Primary hydrodynamic variables such as density, internal energy, and pressure are defined at cell centers whereas the components of vector quantities such as velocity and mass flux are defined at the appropriate cell faces. A linearized Arbitrary-Lagrangian-Eulerian method is used for approximating the solution to the coupled mass, momentum, and energy conservation equations. The implicit, iterative pressure computation in this method, which uses an efficient preconditioned conjugate gradient matrix solver, allows simulation of both high- and low-speed (low-Mach-number) flows without the time-step restrictions that are caused by the fluid sound speed. The computational time-step size, however, is controlled automatically in the code so that the material Courant limit and numerical stability criteria resulting from various diffusion processes are not violated.

To model fluid turbulence, GASFLOW provides an option for two turbulent models. These are the algebraic and $k-\epsilon$ models, which are the respective zero- and two-transport equation models that compute the turbulent velocity and length scales required to determine the turbulent diffusivity. Turbulent diffusivity, together with its molecular counterpart, is used to determine gradient diffusion fluxes in the momentum, the internal energy, and the species mass transport equations.

Heat conduction within walls and structures is one-dimensional. The solid heat conduction equations are approximated with an implicit, finite-difference formulation that results in the solution of tridiagonal matrices. The GASFLOW mesh expansion capability allows for small nodes at the surface and expanding larger nodes within the heat structure. This will result in accurate calculations of the surface temperatures. Rates of heat transfer and condensation to walls and structures are calculated from the

Reynolds analogy between momentum, heat, and mass transfer. A model is available to account for the enhanced mass- and heat-transfer rates in the presence of high mass fluxes toward the wall (e.g., during steam condensation). A term that accounts for the cooling effect caused by gas expanding into the volume space vacated by steam condensation is included in the energy equation.

Chemical energy of combustion involving hydrogen or other fuels provides a source of energy within the gaseous region, in addition to changing the composition of the gas mixture. GASFLOW uses a one-step, global, chemical-kinetics model to simplify the actual chemical processes. (In the case of a hydrogen-nitrogen-oxygen-steam system, detailed chemical kinetics may involve up to about 50 intermediate reactions.) The model is based on a modified Arrhenius rate law that calculates the local fuel and oxidizer concentrations. The finite-rate chemical equation is solved implicitly for the fuel concentration when the fuel-oxidizer mixture is fuel lean and for the oxidizer or reactant concentration when the fuel-oxidizer mixture is fuel rich. The procedure ensures that combustion gas components will never be driven negative, regardless of the time-step size.

The GASFLOW code aerosol model comprises a Lagrangian discrete particle transport model, a stochastic turbulent particle diffusion model, a particle deposition model, a particle entrainment model, and a particle cloud model. These models incorporate the physics of particle behavior to model discrete particle phenomena and allow the code user to track the transport, deposition, and entrainment of discrete particles, as well as clouds of particles, in nuclear systems.

1.3 General Code Features

GASFLOW is a Fortran 90 computer code originally developed to run on the Cray supercomputers at LANL. It has been optimized to take advantage of the vector processing units in such computer architecture to gain performance speed-up of almost an order of magnitude (compared to running the code in scalar mode) for some problems. So that it can be readily installed, the code has recently been converted to run on Sun and IBM workstations and on the VPP supercomputer.

A code calculation may be started from prescribed initial conditions or from the solution of a previous run. The restart capability is very useful when performing large-scale computations in which a complete run may require many hours of CPU time on a supercomputer. Definition of initial conditions is rather flexible—fluid temperature, pressure, and composition at arbitrary regions of the mesh as well as temperature and material of solid thermal structures may be specified. Different types of boundary conditions, which may be time dependent, can be specified on various portions of the computational domain boundaries and on internal wall and obstacle surfaces, and many of the boundary conditions can be changed during restart runs.

In GASFLOW , the centimeter-gram-second (cgs) system is used for the units of dimensional quantities. Therefore, the user should use the following units when preparing input data for the code:

Length	Mass	Time	Pressure	Temperature	Energy
cm	g	sec	$\frac{\text{dynes}}{\text{cm}^2}$ (0.1 Pa)	K	ergs (10^{-7} J)

If the cylindrical coordinate system is used to set up the mesh for the computation, then input values for azimuthal coordinates, if required, must be in degrees (rather than in radians).

GASFLOW has a built-in graphics package for displaying different views of the mesh and the computational results. In this regard, it is different from many codes that require a separate postprocessor to display the results. Actually, the code can be conveniently used as a graphics postprocessor to plot additional results after an analysis run has been completed. Available options include 1D profile, 2D contour and vector, 3D vector, and time history plots for all hydrodynamic variables and for temperatures in all heat-conducting solid structures. Although the graphics have been developed as the primary tool for analyzing the computed data, several printed output files are written by the code to provide the user additional information about the run. Table 1-1 lists all the files used or written by GASFLOW.

Table 1-1. Description of input and output files in GASFLOW

File Name	Description
ingf	Input data text file.
pgf	Graphics metafile.
gfout	Output listing text file.
meshmap	Text file containing cell status and neighbor list information.
cyclinfo	Text file containing time-step and iteration information for each cycle of calculation.
tape16	Text file echoing all NAMELIST input variables.
gfd1, gfd2, ...	Binary dump files used for restart calculations. The number of dump files produced can be controlled through user input.
pth1, pth2, ...	Binary file containing time history data. The number of files generated depends on the number of time history plots being asked for.

2 INPUT DATA FORMAT

To run GASFLOW, the user must prepare an input file that contains data required for the problem calculation and for specifying any desirable output options. The input file is called `ingf`. The user must limit the input file to 80 columns wide except for optional comments. Figures 2-1a and 2-1b show the listing of the input for a sample problem. The first three lines contain alphanumeric data for problem identification purposes. These input data follow:

Line	Data Format	Description
1	A80	Title of problem to appear on all pages of graphical output and printed output.
2	A10	Label to appear on printed output.
3	A64	Special plot file label.

1D/3D Problem ESDEF test problem

JWS

GASFLOW V2.0

NOTES:

- * 2 3D block, 1 duct, air
- * Uniform energy source term of 8325.2 ergs/cm³
- * BLOCK #1 - 300 x 300 x 100 cm
 - 3 x 3 x 1 = 9 real cells m222 = 32
 - 5 x 5 x 3 = 75 total cells mrrr = 44
- * DUCT #1 - 400 x (100 x 100) cm - 4 cells
 - W end: connected to 3D block
 - E end: connected to 3D block
- * BLOCK #2 - 300 x 300 x 100 cm
 - 3 x 3 x 1 = 9 real cells m222 = 32
 - 5 x 5 x 3 = 75 total cells mrrr = 44

J=4	+-----+-----+-----+									
	42	43	44							
J=3	+-----+-----+-----+									
	37	38	39							
J=2	+-----+-----+-----+-----+-----+									
	32	33	151	152	153	154	155			
			1	2	3	4	5			
J=1	+-----+-----+-----+-----+-----+									
	1	2	3	4	2	3	4	5		
				1						

NODE 1

NODE 2

3D BLOCK 1

NETWORK DUCT 1

Connected to Duct 1	+-----+-----+-----+ J=4			
		117	118	119
	+-----+-----+-----+ J=3			
		112	113	114
	+-----+-----+-----+ J=2			
	155	107	108	109
	5	6		
	+-----+-----+-----+ J=1			
	1	2	3	4
NODE 2 3D BLOCK 2				

Figure 2-1a. Sample GASFLOW input listing, showing typical comments that can be included in an ingf input file

```

-----
      N E T W O R K      I N P U T
-----
$innet
  cpnt      = 300.0,  50.0,  50.0,
              700.0,  50.0,  50.0,
ductdef     = 100.0, 100.0, 100.0, 100.0, 100.0, 100.0,
              1,      2,      4,      1.0,  0.0,  0.0,
  nwcx      = 4 , 4 , 1 , 2 , 1, 2 , 1 , 1 ,
              1 , 1 , 1 , 2 , 1, 2 , 2 , 2 ,
iwshear     = 0,
$end

```

Figure 2-1b. Sample GASFLOW input listing, showing innet NAMELIST group

B A S I C I N P U T

```

$хput
autot      = 1.0,   cyl = 0.0,   omg =   34.0 ,
delt0      = 00.10000 ,
deltmin    = 1.000e-08,
deltmax    = 1.000e-00,
epsi0      = 1.000e-08,
epsimax    = 1.000e-08,
epsimin    = 1.000e-08,
gz         = -000.0,
itdowndt   = 25,
itupdt     = 100,
itmax      = 100,
lpr        = 1,
maxcyc     = 02000,
ittyfreq   = 5,
nu         = 0.15 ,
pltdt      = 100.00,
prtdt      = 010.000,
twfin      = 020.00,
tddt       = 100.0000,
velmx      = 1.5,
ibw        = 1,
ibe        = 1,
ibs        = 1,
ibn        = 1,
ibb        = 1,
ibt        = 1,
mat        = 'air',

; 3D BLOCK # 1 I.C. -----
gasdef(1,1) = 1 , 'im1', 1 , 'jm1' , 1 , 'km1', 1,
              1.000000000e6, 300.0, 1, 0., 0.,
              'air', 1.00000,

; 3D BLOCK # 2 I.C. -----
gasdef(1,2) = 1 , 'im1', 1 , 'jm1' , 1 , 'km1', 2,
              1.000000000e6, 300.0, 1, 0., 0.,
              'air', 1.00000,

; 1D NETWORK -----
gasdef(1,3) = 0 , 6, 1 , 0 , 0 , 0 , 0 , ; D#1 I.C.
              1.000000000e6, 300.0, 1, 0., 0.,
              'air', 1.00000,
esdef(1,1) = 1, 5, 1, 0, 0, 0, 0, 8325.2, 0.0, 99999.0,
esdef(1,2) = 1, 4, 1, 4, 1, 2, 1, 8325.2, 0.0, 99999.0,
esdef(1,3) = 1, 4, 1, 4, 1, 2, 2, 8325.2, 0.0, 99999.0,
$end

```

Figure 2-1c. Sample GASFLOW input listing, showing xput NAMELIST group

```

-----
M E S H
-----
$meshgn
  iblock = 1,
  xgrid = 0.0, 100.0, 200.0, 300.0,
  ygrid = 0.0, 100.0, 200.0, 300.0,
  zgrid = 0.0, 100.0,
$end
$meshgn
  iblock = 2,
  xgrid = 0.0, 100.0, 200.0, 300.0,
  ygrid = 0.0, 100.0, 200.0, 300.0,
  zgrid = 0.0, 100.0,
$end
$meshgn
$end
-----
G R A P H I C S
-----
$grafic
  thdt = 1.0,
  thp(1,1) = 4, 2, 2, 1, 'tk', 0,
              4, 3, 2, 1, 'tk', 0,
              4, 2, 2, 2, 'tk', 0,
              4, 3, 2, 2, 'tk', 0,
              2, 1, 0, 0, 'tk', 0, ; duct #1
              5, 1, 0, 0, 'tk', 0, ; duct #1
              2, 1, 0, 0, 'pn', 0, ; duct #1
              5, 1, 0, 0, 'pn', 0, ; duct #1
$end

```

Figure 2-1d. Sample GASFLOW input listing, showing two input NAMELIST groups, meshgn and grafic

The rest of the input data are read into the code via eight groups of NAMELIST variables.

The main purpose of each NAMELIST group is listed below:

NAMELIST Group	Purpose
xput	Definition of physical properties, initial and boundary conditions, code control, and numerical solution option data.
innet	Ventilation system description.
meshgn	Specification of computational mesh.
rheat	Specification of heat-transfer data.
grafic	Definition of graphical output options.
parts	Specification of variables related to particle transport.
special	Definition of miscellaneous 3D plotting variables.
specialp	Definition of additional 3D plotting variables.

Note that the NAMELIST groups `parts`, `rheat`, `special`, and `specialp` are included but left blank in the sample input deck shown in Figures 2-1a through 2-1d. Input variables in these four groups are not used in this sample problem, but the input file must contain the group names (though no data), so the input data will be processed successfully. All variables for the NAMELIST groups are described in Appendices A-F.

The NAMELIST feature offers an easy way of specifying input data. Within each NAMELIST group, both scalar and array variables can be defined conveniently with their desired values. The order of appearance of the variables is unimportant. As can be seen in Figures 2-1a and 2-1b, all input data values are clearly associated with the corresponding variable names, which makes it very simple for a user to modify the input deck to run other problems. However, the input for the `meshgn` NAMELIST group must be followed by a `meshgn` NAMELIST input with no data. This provides a signal to the GASFLOW computer code that there is no additional 3D multiblock input. In addition, the 3D multiblocks must be input in consecutive order starting with **iblock** = 1, as shown in the example in Figures 2-1a through 2-1d.

An input NAMELIST group record can consist of one or more lines (physical records). Column 1 and column 81 and beyond are ignored. In the first line, `$name` (the dollar sign delimiter followed immediately by the name of the NAMELIST group) must appear beginning in column 2 and then be followed by one or more blanks. The remaining portion of the input record may contain as many variables as needed, with their assigned values, and in any order. Commas are used to separate items and to separate input values for elements of the same array. Input items take the following forms:

```

variable           = value,
array              = value[,value,] . . . ,
array(subscripts) = value[,value,] . . . ,

```

where subscripts are integer constants identifying particular elements of the array. (Brackets indicate optional entries.) Multidimensional array values are assigned in storage order. Any value can be repeated by $n \times \text{value}$, where n is the repetition count. A delimiter (\$end) terminates the NAMELIST group record.

Blanks can be used to improve legibility but must not be embedded in names, values, or between an array name and the open (left) parenthesis that encloses the array indices. For example,

```
gasdef (1,1) = . . . ,    is correct, whereas
```

```
gasdef (1,1) = . . . ,    will lead to input processing errors.
```

Optional comments can appear between input NAMELIST group records. They can also be placed within a NAMELIST group. A comment within the record must be preceded by a semicolon. No input data can be specified after a comment on the same line; i. e., entries after a semicolon on the same line will be ignored. (See sample input decks in Figures 2-1a through 2-1d.) An input NAMELIST group record may contain only comments or may be entirely blank.

3 GEOMETRY DEFINITION

3.1 Cell Labeling Convention

In GASFLOW two coordinate systems are available. In the Cartesian or rectangular system, the coordinate axes are x , y , and z , and their corresponding logical indices are i , j , and k . If the cylindrical system is used, then the logical coordinate indices i , j , and k correspond, respectively, to the radial (r), azimuthal (θ), and axial (z) directions. To define regions in the computational domain where initial and boundary conditions are to be applied, the user must understand the cell numbering scheme. The same scheme is used as the basis for specifying regions (lines, surfaces, or volumes) where graphical displays of the calculated results are desired.

The finite-difference mesh used for discretizing the geometry consists of computational cells that are ordered logically in three dimensions with indices i , j , and k . The maximum number of cells in each direction is designated $i_{\max}$, $j_{\max}$, or $k_{\max}$, depending on the direction. In GASFLOW, a layer of fictitious cells is used just beyond each boundary of the computational domain to accommodate general boundary condition treatments. Therefore, in the z -direction, for example, $k = 1$ and $k_{\max}$ are the fictitious boundary cells while only cells with k indices from 2 to $k_{\max}-1$ are active or real. So the total number of real cells in the entire mesh is the product $(i_{\max}-2)(j_{\max}-2)(k_{\max}-2)$.

Besides labeling cells, it is useful sometimes to refer to the cell faces between them, which form the grid lines. The GASFLOW convention is that the i^{th} grid line refers to the cell face between a cell with index i and the next cell with index $i+1$. This computational cell labeling scheme is shown in Figure 3-1.

3.2 Mesh Generation

Before generating a mesh, the user must specify which coordinate system is to be used for the computation. The input variable for this is **cy1** in the NAMELIST group **xput**. Set **cy1** = 0 (default) to use Cartesian coordinates or **cy1** = 1 to use cylindrical coordinates. Then the computational mesh is defined by one of two methods available. Input variables for both methods are in the NAMELIST group **meshgn**. Note that the user only defines geometry for the *real*, physical domain. Fictitious boundary cells are assigned automatically by the code.

Figure 3-1. GASFLOW numbering convention for cells (or cell centers) and cell faces using the i -direction with $imax = 7$ for illustration. The fictitious boundary cells are shaded, i. e., cell numbers 1 and 7. The real fluid cells are numbered from 2 to 6. The physical computational volume ranges from cell face number 1 to cell face number 6.

3.2.1 Direct Input of Grid Locations

The first method of defining the mesh is simply direct entering of the coordinate value of each grid point in each direction. The input array variables **xgrid**, **ygrid**, and **zgrid** are used to specify grid point locations in the x -, y -, and z -directions in Cartesian coordinates. The length unit must be in centimeters. For example,

```
xgrid = 0., 1., 2., 3., 4., 5., 6., 7., 8., 10.
```

specifies that the mesh in the x -direction goes from 0 to 10 cm and has nine cells. The first eight cells have a cell-width of 1 cm, and the last one is 2 cm wide. Note that **xgrid**, **ygrid**, and **zgrid** values define the coordinates of cell faces.

If cylindrical coordinates are used, then **xgrid** refers to grid point locations in the radial (r) direction, and **ygrid** and **zgrid** refer respectively to the azimuthal (θ) and axial (z) directions. The measure of θ must be in degrees. For example,

```
ygrid = 0., 15., 30., 45., 60., 75., 90.
```

specifies a mesh that is a quadrant of a cylinder and has six layers of cells in the azimuthal direction, all evenly spaced 15° apart.

3.2.2 Automatic Mesh Generation

The above method of directly entering grid coordinates is useful when such information is available, for example, from a separate mesh-generation program. In many cases, it is more convenient to use the second method offered by the code. This method uses an automatic mesh generator which allows easy generation of a mesh composed of cells with either fixed or variable sizes. The basic idea is to build a mesh by stacking together a series of submeshes in each coordinate direction. For example, consider the x -direction. The x -dimension of the problem to be solved is subdivided into a set of nx intervals. The k^{th} interval extends from its left (lower) end, $x1(k)$, to the left end of the next interval, $x1(k+1)$. Within each interval there is a location, $xc(k)$, where the mesh cells will be smallest. In other words, the grid lines in the k^{th} submesh converge to location $xc(k)$. The number of cells between $x1(k)$ and $xc(k)$ is specified as $nx1(k)$, and the number from $xc(k)$ to $x1(k+1)$ is specified as $nxr(k)$. The minimum cell size, which is located at $xc(k)$, is specified as $dxmn(k)$.

Using the above information, the mesh generator expands cell sizes from a value of $dxmn(k)$ at $xc(k)$ in a quadratic manner such that the required number of cells will lie on each side of $xc(k)$ and fill the subinterval. If $dxmn(k)$ is larger than the cell size corresponding to uniform zoning, then the generator will produce a uniformly spaced mesh in the x -direction.

Any number of cells can be defined on either side of $xc(k)$, including zero. A choice of zero is useful when the minimum cell size is desired at the beginning or end of a subinterval. This is often done in problems in which a fine mesh resolution is required, in the vicinity of a surface where steep gradients in the temperature or velocity profile are expected.

GASFLOW supports definition of up to 49 mesh subintervals in each of the three dimensions. Thus, it is possible to generate complicated meshes with locally fine resolution around any number of points. Furthermore, because the minimum cell sizes are specified as part of the input data, there should be no unexpected cell-size-related numerical stability difficulties.

In summary, the input parameters for mesh subdivision k in the x -direction are as follows:

- nx defines the total number of subintervals in the x -direction.
- $x1(k)$ sets the location of the left boundary of subdivision k .
- $xc(k)$ sets the "convergence point" where the minimum cell spacing occurs in subdivision k .
- $nx1(k)$ specifies the number of cells to the left of $xc(k)$, i. e., between locations $x1(k)$ and $xc(k)$ in subdivision k .

- n_{xr}(k)** specifies the number of cells to the right of **x_c(k)**, i. e., between locations **x_c(k)** and **x_l(k+1)** in subdivision k.
- d_{xmn}(k)** specifies the minimum cell size in the x-direction in subdivision k.

The maximum number of mesh subdivisions allowed is 50. A similar treatment is used in the y- and z-directions. In addition, the input variables used for the r- and θ -directions, when cylindrical coordinates are chosen, are the same as those for the x- and y-directions, respectively. A list of the variables for all the directions is given below:

x- or r-direction:	n_{kx} , x_l(k) , x_c(k) , n_{xl}(k) , n_{xr}(k) , d_{xmn}(k)
y- or θ -direction:	n_{ky} , y_l(k) , y_c(k) , n_{yl}(k) , n_{yr}(k) , d_{ymn}(k)
z-direction:	n_{kz} , z_l(k) , z_c(k) , n_{zl}(k) , n_{zr}(k) , d_{zmn}(k)

Consider the following two examples that illustrate the use of the automatic mesh generator.

Cartesian Mesh. The first example involves Cartesian geometry (**cy1** = 0.0). Here we show how to generate a uniform mesh in the z-direction extending from 0 to 12 cm containing 10 cells. In the x-direction, the mesh also extends from 0 to 12 cm and consists of 10 cells, but has a minimum cell size of 0.2 cm on both sides of the line $x = 5$ cm. The following input specifications in NAMELIST group meshgn will generate such a mesh for 3D block number 1, as depicted in Figure 3-2.

```
iblock = 1,
nkx = 1, nkz = 1,
zl(1)=0.,zc(1)=0.,nzl(1)=0,nzr(1)=10,dzmn(1)=1.e9,zl(2)=12.,
xl(1)=0.,xc(1)=5.,nxl(1)=5,nxr(1)=5,dxmn(1)=0.2,xl(2)=12.
```

Cylindrical Mesh. In the second example, the coordinate system chosen is cylindrical (**cy1** = 1.0 specified in NAMELIST group xput). Figure 3-3 shows the mesh in two dimensions generated by the following input in NAMELIST group meshgn:

```
iblock = 1,
nkx = 1, nky = 1,
xl(1)=0.,xc(1)=15.,nxl(1)=10,nxr(1)=0,dxmn(1)=0.5,xl(2)=15.,
yl(1)=0.,yc(1)=0.,nyl(1)=0,nyr(1)=24,dymn(1)=1.e9,yl(2)=360.
```

In the azimuthal (θ) direction, there are 24 cells, which are evenly spaced because the minimum cell size, **d_{ymn}**, is greater than the average cell width obtained by uniform zoning (i.e., $10^9 > 360/24$). In the radial (r) direction, there are 10 cells that discretize the total radius of 15 cm. The minimum cell size, **d_{xmn}**, is specified as 0.5 cm, which is smaller than the "uniform" cell width given by 15 cm/10. Therefore, the cell size gradually expands from this minimum value at $r = 10$ cm to a maximum value at $r = 0$.

Figure 3-2. Two-dimensional x - z view of a mesh generated by the GASFLOW automatic mesh generator. Note that the vertical mesh lines converge on the $x = 5$ line.

Figure 3-3. Two-dimensional, r - θ view of a mesh generated by the GASFLOW automatic mesh generator. The cell spacing becomes finer as r increases, but is uniform in the θ -direction. The numbers 1 to 11 below x are the r -mesh numbers.

In many problems, it is useful to know the largest, as well as the smallest, cell sizes to have a feel for the computational length scales as compared to the physical scales. Note that the maximum cell size, $\delta_{\max}$ in any submesh generated by this automatic mesh generator can be easily determined from the relation

$$(\delta_{\min} + \delta_{\max})/2 = \delta_{\text{avg}} ,$$

or
$$\delta_{\max} = 2 \delta_{\text{avg}} - \delta_{\min} ,$$

where δ_{avg} is the average cell size corresponding to uniform zoning. Therefore, $\delta_{\max}$ in the x -direction in the mesh generated in the above example is

$$2 [(5 - 0)/5] - 0.2 = 1.8 \text{ cm}$$

on the left side of $x = 5$ cm. On the right side, the maximum cell size is

$$2 [(12 - 5)/5] - 0.2 = 2.6 \text{ cm} .$$

3.3 Definition of Walls and Obstacles

The previous section describes how to generate a computational mesh that represents a discretized model of the region over which the conservation equations for the fluid are solved. In most practical problems, the fluid flow region is more complex than an empty rectangular box. There may be flow obstacles or interconnected subcompartments. In GASFLOW, walls and obstacles can be defined within the mesh to model complex flow paths. The nomenclature used by the code is that a *wall* is a surface dividing two adjacent layers of fluid cells that forbids flow across it. An obstacle is a volume consisting of an arbitrary number of cells, namely *obstacle cells*, through which no fluid flow is allowed. In other words, obstacle cells are blocked out from the fluid-dynamics calculations. (However, in problems involving heat transfer, conduction inside the obstacle cells is calculated.) Once *walls* and *obstacles* have been placed within the computational mesh, the user has the option of putting *holes* through both *walls* and *obstacles*. This allows a more general construction of the complexities involved in developing complicated geometries.

3.3.1 Walls

To define a wall means specifying a surface normal to any of the three orthogonal dimensions with logical indices i , j , and k . This is done via the input array variable **walls** in the NAMELIST group `xput`. The array **walls** is 2D with the second index identifying the wall definition and the first index specifying eight numbers that are required to define the wall surface:

- walls(1,*)** Beginning i mesh index (cell face number).
- walls(2,*)** Ending i mesh index (cell face number).
- walls(3,*)** Beginning j mesh index (cell face number).
- walls(4,*)** Ending j mesh index (cell face number).
- walls(5,*)** Beginning k mesh index (cell face number).
- walls(6,*)** Ending k mesh index (cell face number).
- walls(7,*)** Block number.
- walls(8,*)** Integer to identify the type of wall (thickness and material). Used only for heat transfer; ignored if heat transfer is not invoked.

The asterisk (*) should be replaced by an integer (≤ 500) that identifies the particular wall definition.

The last element **walls(8,*)** is reserved for specifying an input that is only required if heat transfer is invoked (by setting **ihflag** = 1 in NAMELIST group **rheat**) but is otherwise ignored. This is explained in Section 7.5. The rest of the input, **walls(1,*)** to **walls(6,*)**, specifies the location and extent of the wall. Because a surface has only two dimensions, one of the three pairs of beginning and ending mesh indices must be the same.

Consider the following input which defines two walls:

```
walls = 2, 2, 1, 2, 2, 10, 1, 0,
        2, 9, 1, 2, 2, 2, 1, 0
```

Because the input data are read in consecutively in the order of memory storage, no indices have to be explicitly written for the two-dimensional array **walls**. (In other words, the first input line in the example above defines values for **walls(1,1)**, **walls(2,1)**, etc., up to **walls(8,1)**.) The 16 numbers will be used correctly by the code to define two walls. An equivalent way to write the above input is

```
walls(1,1) = 2, 2, 1, 2, 2, 10, 1, 0
walls(1,2) = 2, 9, 1, 2, 2, 2, 1, 0
```

In this example, the first line defines a wall at the *i* cell face index 2, and extends from *j*-index 1 to *j*-index 2 and from *k*-index 2 to *k*-index 10. The second line describes a wall that is normal in the *k*-direction, i.e., perpendicular to the first wall. If the input is applied to the mesh shown in Figure 3-3, then the two walls will appear on the *x-z* plane of the mesh, as shown in Figure 3-4.

3.3.2 Obstacles

Figure 3-4 also shows obstacle cells that further restrict fluid flow in the computational domain. These "mesh obstacles" are specified by the input array variable **mobs** in NAMELIST group **xput**:

mobs(1,*)	Beginning <i>i</i> mesh index (cell face number).
mobs(2,*)	Ending <i>i</i> mesh index (cell face number).
mobs(3,*)	Beginning <i>j</i> mesh index (cell face number).
mobs(4,*)	Ending <i>j</i> mesh index (cell face number).
mobs(5,*)	Beginning <i>k</i> mesh index (cell face number).
mobs(6,*)	Ending <i>k</i> mesh index (cell face number).
mobs(7,*)	Block number.
mobs(8,*)	Integer to identify the material that the solid obstacle

is made of. Used only for heat transfer; ignored if heat transfer is not invoked. In future versions this will direct (connect ?) to a definition statement

similar to **walldf**.

The asterisk (*) should be replaced by an integer that identifies the particular **mobs** definition (≤ 3000).

The elements in the array **mobs** have the same meaning as those in **walls**, except for the last element, which is explained in Section 6, where solid heat conduction is discussed. However, mesh obstacles refer to a *volume* region where no flow is allowed to penetrate. Therefore, the beginning and ending i, j, and k mesh indices should define any two vertices of a three-dimensional volume that are orthogonal to each other. The following two **mobs** definitions specify obstacle regions that are shown in Figure 3-4:

```
mobs = 3, 7, 1, 2, 9, 10, 1, 0,  
        6, 9, 1, 2, 7, 9, 1, 0
```

GASFLOW supports 500 definitions for **walls** and 3000 definitions for **mobs**.

3.3.3 Holes

In order to model complicated geometries, GASFLOW has generalized input variables to represent *obstacles* (solid structures which eliminate fluid cells in the computational mesh) and *walls* (which provide a zero flux condition on any computational fluid face). These surfaces described by *obstacles* and *walls* may, at the user's discretion, provide two-phase heat-transfer regions within the computational mesh. It is convenient to add another geometry modeling capability, and we call this input *holes*. This allows regions that have been removed as fluid cells by using *obstacles* to be set back to fluid cells. It must be noted that all *obstacles* (**mobs**) and *walls* (**walls**) are processed first, and then the *holes* (**holes**) are positioned. In the current version of GASFLOW, the number of *holes* is limited to the PARAMETER variable MXA (MXA=300). Each *hole* is defined by thirteen entries in the *holes* input array in the NAMELIST input block **xput**.

Figure 3-4. Two-dimensional x - z view of a mesh containing two wall surfaces and two obstacle regions generated by **walls** and **mobs** input definitions, respectively

The *holes* input variable array, [**holes**(i , *), $i = 1, 13$], is defined for the n^{th} hole where (*) is replaced by an integer that identifies the particular n^{th} hole definition:

holes (1, *)	Beginning i mesh index (cell face number).
holes (2, *)	Ending i mesh index (cell face number).
holes (3, *)	Beginning j mesh index (cell face number).
holes (4, *)	Ending j mesh index (cell face number).
holes (5, *)	Beginning k mesh index (cell face number).
holes (6, *)	Ending k mesh index (cell face number).
holes (7, *)	Block number.
holes (8, *)	Flag for fluxing on the beginning i mesh face [holes (1,*)]: = 0 implies no fluxing allowed; = 1 implies fluxing allowed.
holes (9, *)	Flag for fluxing on the ending i mesh face [holes (2,*)]:

= 0 implies no fluxing allowed;
 = 1 implies fluxing allowed.
holes(10,*) Flag for fluxing on the beginning j mesh face [**holes(3,*)**]:
 = 0 implies no fluxing allowed;
 = 1 implies fluxing allowed.
holes(11,*) Flag for fluxing on the ending j mesh face [**holes(4,*)**]:
 = 0 implies no fluxing allowed;
 = 1 implies fluxing allowed.
holes(12,*) Flag for fluxing on the beginning k mesh face [**holes(5,*)**]:
 = 0 implies no fluxing allowed;
 = 1 implies fluxing allowed.
holes(13,*) Flag for fluxing on the ending k mesh face [**holes(6,*)**]:
 = 0 implies no fluxing allowed;
 = 1 implies fluxing allowed.

The asterisk (*) should be replaced by an integer that identifies the particular holes definition (≤ 300).

An example of the input for **holes** follows:

Consider a computational mesh, i.e., *block 1*, that has 10 fluid cells in each of the three coordinate directions x , y , and z . This means that including the boundary cells, there are 12 cells in each coordinate direction. There is an obstacle block that penetrates the entire computing domain that is 3 cells high, 7 cells wide, and 10 cells deep made of material 1; there is a horizontal wall (ceiling) 8 cells wide, 2 cells above the bottom boundary, and 10 cells deep made of material 2; and there is a vertical wall 4 cells high, 1 cell from the west or left side boundary, and 10 cells deep made of material 2. The input is as follows:

```

$xput
.
.
.
  mobs =  3,10,  1,11,  5, 8,  1,  1,  ; solid obstacle

  walls =  2,10,  1,11,  3, 3,  1,  2,  ; horizontal wall
          2, 2,  1,11,  6,10,  1,  2,  ; vertical wall
.
.
.
$end
  
```

This configurations is plotted as shown in Figure 3-5.

Figure 3-5. A geometric model with one obstacle and two walls to demonstrate the **holes** option

We wish to position three holes in the existing geometric model:

1. a hole two vertical cells high penetrating the obstacle from the top,
2. a vertical hole penetrating through the entire obstacle, and
3. a hole penetrating the horizontal wall.

This new configuration is shown in Figure 3-6.

Figure 3-6. A geometric model with one obstacle (**mobs**) and two walls (**walls**) with three penetrations cut through the $j=6$ plane (y -plane cut) to demonstrate the **holes** option

The entire input stream follows:

```
$xput
.
mobs = 3,10, 1,11, 5, 8, 1, 1, ; solid obstacle
walls = 2,10, 1,11, 3, 3, 1, 2, ; horizontal wall
          2, 2, 1,11, 6,10, 1, 2, ; vertical wall
holes = 5, 7, 4, 7, 6, 8, 1, 0,0, 0,0, 0,1, ; top hole
          8, 9, 5, 6, 5, 8, 1, 0,0, 0,0, 1,1, ; thru hole
          8, 9, 5, 6, 2, 4, 1, 1,1, 1,1, 1,1, ; wall hole
.
$end
```

3.3.4 Fractional Area

Often in modeling complex geometries the actual flow area is different from the structure that the computational mesh allows. For example, in Figure 3-6, the small room that we created with a **holes** option in the large obstacle has an inlet flow area that is actually 10% of the flow area shown in the mesh. GASFLOW has the capability to reduce the flow area and volumes of selected cells. The user can input a fractional area at the cell edge between two adjacent cells and GASFLOW will automatically include a sharp edge orifice flow loss at that location and calculate velocities based on the actual flow area. Fractional area reductions are input with the **areardef** array in the xput NAMELIST group.

areardef (1,*)	Beginning i mesh index (cell face number).
areardef (2,*)	Ending i mesh index (cell face number).
areardef (3,*)	Beginning j mesh index (cell face number).
areardef (4,*)	Ending j mesh index (cell face number).
areardef (5,*)	Beginning k mesh index (cell face number).
areardef (6,*)	Ending k mesh index (cell face number).
areardef (7,*)	Block number.
areardef (8,*)	Fraction of geometric flow area available for fluid flow. If areardef (8,*) is less than zero and areardef (9,*) equals zero, then no flow loss coefficient is included at this location. If areardef (8,*) is less than zero and areardef (10,*) equals zero, then no drag loss is included at this location.
areardef (9,*)	User input flow loss coefficient. If areardef (9,*) is zero, then correlations for flow through an orifice are used to calculate the flow loss coefficient. If areardef (8,*) is less than zero and areardef (9,*) equals zero, then no flow loss coefficient is included at this location. Default is zero.
areardef (10,*)	Hydraulic diameter to be used for the laminar drag loss. If areardef (10,*) equals zero, then no laminar drag loss is included at this location. Default is zero.
areardef (11,*)	Coefficient in the laminar drag loss formula. Default is 64.
areardef (12,*)	Fraction of the cell-center distance between neighboring cells to be used in the laminar drag loss formula. Default is one.

The asterisk (*) should be replaced by an integer (≤ 300) that identifies the particular **areardef** definition. GASFLOW supports 300 descriptions for **areardef**.

If a beginning and ending mesh index is the same in for the i, j, or k input, then the flow area fraction is applied only at the cell edge identified by the repeated mesh index. For example, the flow area into the room we created by the **holes** option in Figure 3-6 can be reduced to 10% of the geometric flow area by the following statement:

```
areardef(1,1) = 5,7,4,7,8,8,1,0.1, 4*0.0,
```

If the user requires that the volume and flow area be reduced, then the beginning and ending mesh indices for i, j, and k must be different. For example, the following input reduces geometric volumes and flow areas by 50% for the hole that we drilled through the entire obstacle in Figure 3-6:

```
areardef(1,2) = 7,8,5,6,5,8,1,0.5, 4*0.0,
```

The following input reduces geometric volumes and flow areas by 75% for flow through the horizontal wall or ceiling in Figure 3-6:

```
areardef(1,3) = 7,8,5,6,3,3,1,0.75, 4*0.0,
```

We can create the entire geometric model in Figure 3-6 including the fractional areas and volumes with the following input:

\$xput

```
.
.
.
mobs   = 3,10, 1,11, 5, 8, 1, 1, ; solid obstacle

walls   = 2,10, 1,11, 3,  3, 1, 2, ; horizontal wall
           2, 2, 1,11, 6, 10, 1, 2, ; vertical wall

holes    = 5, 7, 4, 7, 6, 8, 1, 0,0, 0,0, 0,1, ; top hole
           8, 9, 5, 6, 5, 8, 1, 0,0, 0,0, 1,1, ; thru hole
           8, 9, 5, 6, 2, 4, 1, 1,1, 1,1, 1,1, ; wall hole

areardef= 5, 7, 4, 7, 8, 8, 1, 0.1, 4*0.0,  ; fract area
           ; top hole
           8, 9,  5, 6,  5, 8,  1,  0.5, 4*0.0,  ; fract area
           ; thru hole
```

```

      8, 9, 5, 6, 3, 3, 1, 0.75, 4*0.0, ; fract area
                                           ; wall
      .
      .
      .
$end

```

GASFLOW supports 300 definitions for **areardef**.

3.4 Definition of the 1D Duct Network

GASFLOW has the capability for computing the flow of multiple gas species in 1D duct networks. Physical elements containing gas are *ducts*, where the flow is locally one dimensional, and *junctions*, which accommodate the connection of two to six ducts. Junctions and the ends of ducts are located at *nodes* (predefined locations in space). At the end of each duct, there is either a junction, a user-defined boundary condition, or a 3D block. The assembled collection of ducts, junctions, 3D blocks, and boundary conditions is called the network geometry. Several general capabilities are illustrated in the network fragment shown in Figure 3-7. The network fragment consists of 3 ducts, 1 junction, and three boundary conditions. *Junction 3* is connected to three ducts (numbers 2, 5, and 12). *Duct 2* has a constant cross section and 3 cells of uniform length, *duct 5* has a constant cross section and 5 cells of nonuniform length, and *duct 12* has a nonconstant cross section and 4 cells of uniform length. Note that the cross-sectional area of the end of a duct need not be the same as the area of the junction face to which it connects (see ducts 2 and 5).

Flow within the network is modified by various ventilation system components. Filters, dampers, curved ducts, wall friction, and other flow pressure losses retard the flow; that is, they induce pressure drops in the direction of flow, according to their physical characteristics and local fluid conditions. Blowers, on the other hand, add momentum and energy to the flow. Each of these mechanisms is modeled explicitly and is referred to as a ventilation component or simply a component.

The network geometry plus the ventilation components is called the 1D network or simply the network. Instructions for defining the network geometry are given below, and the ventilation components are described in Sections 3.4.5, 3.4.6, 3.4.7, 3.4.8, and 3.4.9.

Defining the basic network geometry—ducts, junctions, and their logical connectivity—is done by specifying **cpnt**, **ductdef**, and **defjct** in NAMELIST group **innet**. An example input using the **innet** NAMELIST group is given below:

Figure 3-7. Network geometry

\$innet

; node point coordinates

```

;      x      y      z
cpnt(1,1) = 0.0, 0.0, 0.0,
cpnt(1,2) = 400.0, 0.0, 0.0,
cpnt(1,3) = 400.0, -800.0, 0.0,
cpnt(1,4) = 1000.0, -800.0, 0.0,

```

; duct descriptions

```

;      dlw      dle      d2w      d2e      d3w      d3e
ductdef(1,1) = 0.0, 0.0, 100.0, 100.0, 100.0, 100.0,
;      npw      npe      ncd      betax      df1      df2
1, 2, 4, 1.0, 0, 0,
ductdef(1,2) = 100.0, 125.0, 100.0, 125.0, 0.0, 0.0,
2, 3, 6, 0.0, 0, 0,
ductdef(1,3) = 135.0, 95.0, 100.0, 125.0, 80.0, 120.0,
3, 4, 5, 0.0, 0, 0,

```

; junction descriptions

```

;      dxj      dyj      dzj      nodej

```

```
defjct(1,1) = 100.0, 100.0, 100.0, 2,
defjct(1,2) = 125.0, 125.0, 130.0, 3,
$end
```

These instructions will be illustrated for the 3 duct/2 junction/2 boundary condition network shown in Figure 3-8.

3.4.1 Defining Node Points

Nodes are defined to make it easier to define the location and connectivity among ducts, junctions, and boundary conditions. Define the Cartesian coordinates (x, y, z) of each node point in centimeters with **cpnt** (3 numbers required per definition). Note **cpnt** is part of NAMELIST group **innet**. For the example network given in Figure 3-8, the four nodes are defined by

Figure 3-8. Example network

```

cpnt(1,1) = 0.0,    0.0,    0.0,
cpnt(1,2) = 400.0,    0.0,    0.0,
cpnt(1,3) = 400.0, -800.0,    0.0,
cpnt(1,4) = 1000.0, -800.0,    0.0,

```

Currently, GASFLOW allows for up to 400 nodes in a single problem.

3.4.2 Defining Ducts

Ducts occupy spatial volumes between node points. The geometry is defined by the transverse dimensions (d2w, d2e, d3w, d3e) describing the cross section at each end. The location in space is defined by the node points at the ends (npw, npe). The duct is discretized by a uniform or stretched grid of ncd computational cells that vary geometrically in size from d1w to d1e according to the stretching factor betax. Figure 3-9 shows these definitions for a duct meshed by four expanding cells (betax = 1.5). The cell numbering convention for ducts starts with one at the west end of a duct and increases toward the east end of a duct. A boundary cell is included at both the east end and the west end of each duct. Therefore, the duct given in Figure 3-9 would have a total of six cells with the real cells numbering 2, 3, 4, and 5 and the boundary cells numbering 1 and 6.

This is defined with the variable **ductdef(12, ndmax)** in a manner similar to **gasdef**. Note **ductdef** is input as part of **innet** NAMELIST input.

Ten numbers are required for each duct definition:

```

ductdef(1,*) = d1w cell length at the west end.
ductdef(2,*) = d1e cell length at the east end.
ductdef(3,*) = d2w width (2 dimension) at the west end.

```

Figure 3-9. Four cell duct

ductdef(4,*) = d2e width (2 dimension) at the east end.
ductdef(5,*) = d3w height (3 dimension) at the west end.
ductdef(6,*) = d3e height (3 dimension) at the east end.
ductdef(7,*) = npw node point number at the west end.
ductdef(8,*) = npe node point number at the east end.
ductdef(9,*) = ncd number of real cells.
ductdef(10,*) = betax mesh expansion factor (west to east).
ductdef(11,*) = duct fitting variable for curved ducts.
ductdef(12,*) = duct fitting variable that depends upon
 ductdef(12,*)

NOTES:

1. The node points npw and npe define the beginning to end (west to east) logical coordinate sense of the duct.
2. Duct specifications are identified by d1w, d2w, d3w and d1e, d2e, d3e and are local to each duct. The positive 1-direction, i.e., d1w and d1e, goes from the west to the east end of the duct. Directions 2 (d2w and d2e) and 3 (d3w and d3e) are in the plane of the duct cross section in such a manner that (d1w, d2w, d3w) and (d1e, d2e, d3e) form right-handed coordinate systems. See Figure 3-9 for graphical explanation.
3. Ducts need not be aligned with a global axis.
4. If d2w > 0.0 and d3w = 0.0, the west end is circular and d2w is interpreted as a diameter; if d2e > 0.0 and d3e = 0.0, the east end is circular and d2e is interpreted as a diameter.
5. If d2w = d3w, the west end is square;
if d2e = d3e, the east end is square.
6. If d2w ≠ d3w, the west end is rectangular;
if d2e ≠ d3e, the east end is rectangular.
7. Exactly two numbers of the set {d1w, d1e, ncd, betax} should be specified; the other two should be set to 0.0.
8. The maximum number of **ductdefs** allowed is 200.
9. **ductdef(11,*)** and **ductdef(12,*)** are described below, after the straight duct example.

Straight duct example. Duct 1 goes from node 1 to node 2, has a constant 100 x 100 cm square cross section, and is to be meshed with 4 uniform cells. Duct 2 goes from node 2 to node 3, has a circular cross section that varies from 100 to 150 cm in diameter, and is to be meshed with 6 expanding cells (100 cm to 125 cm). Duct 3 goes from node 3 to node 4, has a rectangular cross section whose dimensions vary from 100 x 80 to 125 x 120, and is to be meshed with 5 contracting cells (135 cm to 95 cm). The three ducts are defined by

```

ductdef(1,1) = 0.0, 0.0, 100.0, 100.0, 100.0, 100.0,
               1, 2, 4, 1.0, 0, 0,
ductdef(1,2) = 100.0, 125.0, 100.0, 150.0, 0.0, 0.0,
               2, 3, 0, 0.0, 0, 0,
ductdef(1,3) = 135.0, 95.0, 100.0, 125.0, 80.0, 120.0,
               3, 4, 0, 0.0, 0, 0,

```

Curved ducts are defined with the variables **ductdef(11,*)** and **ductdef(12,*)**, which are user-defined input values defined as follows:

- ductdef(11,*)** = Duct fitting variable for curved ducts.
- = 0, implies straight duct. Gradual area change loss will be calculated.
- = 1, implies standard elbow. Total length of the duct, length/diameter (L/D), and gradual area change loss will be calculated.
- = 2, implies a mitre bend. Total length of the duct, L/D, and gradual area change loss will be calculated.
- = 3, implies a 90° pipe bend. Total length of the duct, L/D, and gradual area change loss will be calculated.
- = 4, implies an 180° closed end return bend. Total length of the duct, L/D, and gradual area change loss will be calculated.
- = 5, implies straight duct with gradual area change applied dfit(2,i) cm from the west end of the duct all the way to the east end of the duct, with gradual area change loss.
- = 6, implies straight duct with gradual area change applied dfit(2,i) cm from the west end of the duct to dfit(2,i) cm from the east end of the duct, with gradual area change loss.
- = 7, implies straight duct with sudden expansion area change applied dfit(2,i) cm from the west end of the duct, with area change loss.
- = 10, implies straight duct with no gradual area change loss.

- = 11, implies standard elbow. Only total length of the duct will be calculated.
- = 12, implies a mitre bend. Only total length of the duct will be calculated.
- = 13, implies a 90° pipe bend. Total length of the duct will be calculated.
- = 14, implies an 180° closed end return bend. Only total length of the duct will be calculated.
- = 15, implies straight duct with gradual area change applied $\text{dfit}(2,i)$ cm from the west end of the duct all the way to the east end of the duct, with no gradual area change loss.
- = 16, implies straight duct with gradual area change applied $\text{dfit}(2,i)$ cm from the west end of the duct to $\text{dfit}(2,i)$ cm from the east end of the duct, with no gradual area change loss.
- = 17, implies straight duct with sudden expansion area change applied $\text{dfit}(2,i)$ cm from the west end of the duct, with no area change loss.

For **ductdef(11,*)** = 1 or 11

ductdef(12,*) = alpha = degree of standard elbow, typically 45° or 90°. $\text{dfit}(2,i)$ must be greater than zero and less than 180°.

For **ductdef(11,*)** = 2 or 12

ductdef(12,*) = alpha = degree of mitre bend. Must be greater than zero and less than or equal to 90°.

For **ductdef(11,*)** = 3 or 13

ductdef(12,*) = rcd, radius of curvature/duct diameter. Must be greater than one.

For **ductdef(11,*)** = 4 or 14

ductdef(12,*) = Total length of the duct divided by 2, the distance the duct goes before it returns.

For **ductdef(11,*)** = 5 or 15

ductdef(12,*) = Gradual area change will occur from $\text{dfit}(2,i)$ cm from the west end of the duct to the east end of the duct.

For **ductdef(11,*)** = 6 or 16

ductdef(12,*) = Gradual area change will occur from $\text{dfit}(2,i)$ cm from the west end of the duct to $\text{dfit}(2,i)$ cm from east end of the duct.

For **ductdef(11,*)** = 7 or 17

ductdef(12,*) = Sudden area change will occur at dfit(2,i) cm from the west end of the duct.

Example input for curved ducts:

```

ductdef(1,1) = 0.0, 0.0, 100.0, 100.0, 200.0, 200.0,
               1,      2,      4,      1.0, 0, 0,
ductdef(1,2) = 0.0, 0.0, 100.0, 100.0, 100.0, 100.0,
               2,      3,      4,      1.0, 1, 90.0,
ductdef(1,3) = 0.0, 0.0, 100.0, 100.0, 100.0, 100.0,
               3,      4,      4,      1.0, 2, 45.0,
ductdef(1,4) = 0.0, 0.0, 100.0, 100.0, 100.0, 100.0,
               4,      5,      4,      1.0, 3, 2.0,
ductdef(1,5) = 0.0, 0.0, 100.0, 100.0, 100.0, 100.0,
               5,      6,      4,      1.0, 4, 200.0,
ductdef(1,6) = 0.0, 0.0, 50.0, 50.0, 100.0, 100.0,
               7,      8,      4,      1.0, 11, 30.0,

```

For the example given above,

Duct #	Description
1	Straight pipe. Smooth area change losses will be calculated for duct number 1.
2	Standard elbow with a 90° bend. An effective L/D loss for a standard elbow will be calculated.
3	Mitre bend with a 45° bend. An effective L/D loss for a mitre bend will be calculated.
4	90° bend in which the radius of curvature divided by the duct diameter equals 2. An effective L/D loss for a 90° bend will be calculated.
5	180° bend that goes out 200 cm and then returns 200 cm. An effective L/D loss for a 180° bend will be calculated.
6	Standard 30° elbow. No smooth area change loss will be calculated and no L/D loss will be calculated.

3.4.3 Defining Junctions

Junctions are regular parallelepipeds (shoe boxes). That is, a junction is a single three-dimensional control volume (cell) whose principal directions are aligned with the coordinate axes and are defined by the variable **defjct(4,400)**. Note that **defjct** is

input as part of the `innet` NAMELIST input. Four numbers are required per definition:

```
defjct(1,*) = DX, overall dimension in x-direction.
defjct(2,*) = DY, overall dimension in y-direction.
defjct(3,*) = DZ, overall dimension in z-direction.
defjct(4,*) = node number where junction is located. The junction is centered
                around the node.
```

Junction example: Junction 1 is located at node 2 and has dimensions 100 x 100 x 100 cm. Junction 2 is located at node 3 and has dimensions 125 x 125 x 130 cm. The two junctions are defined by

```
defjct(1,1) = 100.0, 100.0, 100.0, 2,
defjct(1,2) = 125.0, 125.0, 130.0, 3,
```

3.4.4 Network Connection to a 3D Block

Network connections to specific cell surfaces on a 3D block are defined with the variable `nwcx(8,500)`. Eight numbers are required for each definition:

```
nwcx(1,*) = Beginning i mesh index (cell face numbers) of 3D block.
nwcx(2,*) = Ending i mesh index (cell face numbers) of 3D block.
nwcx(3,*) = Beginning j mesh index (cell face numbers) of 3D block.
nwcx(4,*) = Ending j mesh index (cell face numbers) of 3D block.
nwcx(5,*) = Beginning k mesh index (cell face numbers) of 3D block.
nwcx(6,*) = Ending k mesh index (cell face numbers) of 3D block.
nwcx(7,*) = 3D block number.
nwcx(8,*) = Network node number.
```

Note: `nwcx(1-6,*)` must define a single 3D cell face on the boundary of a block.

Example: Consider the connection illustrated in the mesh fragment shown in Figure 3-10. Network node 49 connects to (is coincident with) the west face ($I = 1$) of 3D block number 3 on the cell face bounded by $J = 9$ and 10 and $K = 6$ and 7. This connection is defined with the entry:

```
nwcx(1,1) = 1, 1, 9, 10, 6, 7, 3, 49,
```

3.4.5 Defining Flow Pressure Losses

Flow pressure losses are defined by the following equation [see Theory Manual, Equation (2-113)]:

$$\Delta p = \frac{K \rho u |u|}{2}$$

Figure 3-10. Mesh fragment of a 3D block connected to a 1D network

with the variable **flossdef(6,maxf)**, which is in the NAMELIST group **innet**. Six numbers are required per definition:

- flossdef(1,*)** = I1, first I index.
- flossdef(2,*)** = I2, last I index.
- flossdef(3,*)** = IDUCT duct number.
- flossdef(4,*)** = IBLK block number (not used).
- flossdef(5,*)** = FLKF, user input forward flow loss coefficient equaling K in the above equation (forward implies increasing cell numbers; i. e., in Figure 3-10, forward is left to right). If $FLKF < 0.0$, then an annular flow geometry is assumed. $ABS(FLKF)$ is assumed to be the outer diameter of this geometry. $FLKR$ is assumed to be the inner diameter of this geometry. Appropriate flow losses and wall drags are calculated.
- flossdef(6,*)** = FLKR, user input reverse flow loss coefficient equaling K in the above equation (reverse implies decreasing cell numbers; i. e., in Figure 3-10, reverse is right to left).

Note that if the user inputs $FLKR=0.0$, then $FLKR$ is assumed to be equal to $FLKF$. $FLKR$ cannot be negative. Note that **maxf** is the maximum number of flow loss definitions allowed and is currently set to 20 via a parameter statement in the CINPUT module.

Example inputs:

```
flossdef(1,1) = 2, 3, 4, 1, 2.0, 0.0,
```

```

flossdef(1,2) = 2, 5, 3, 1, 0.5, 1.0,
flossdef(1,3) = 3, 4, 5, 1, -10.0, 9.0,

```

The first **flossdef** given above applies a flow loss factor of 2.0 for both forward and reverse flow at cell edge 3 in duct 4. The second **flossdef** given above applies a forward flow loss coefficient of 0.5 and a reverse flow loss coefficient of 1.0 at cell edges 3, 4, and 5 for duct 3. The third **flossdef** given above applies a forward and reverse flow loss coefficient and wall drag coefficient assuming an annular flow geometry with an outer diameter of 10 cm and an inner diameter of 9 cm.

3.4.6 Defining Wall Friction

Wall friction calculation for the 1D ducts is determined by the **iwshear** input variable in the **innet** NAMELIST group. When **iwshear** = 0, wall shear is not included in 1D ducts, and when **iwshear** = 1, wall shear is included in 1D ducts. Default is **iwshear** = 1.

3.4.7 Defining Filters

Filters are defined with the variable **filtrdef**(7,**maxf**), which is in the **innet** NAMELIST group. Seven numbers are required per definition:

```

filtrdef(1,*) = I1, first I index.
filtrdef(2,*) = I2, last I index.
filtrdef(3,*) = IDUCT duct number.
filtrdef(4,*) = IBLK block number (not used).
filtrdef(5,*) = Filter laminar flow loss coefficient.
filtrdef(6,*) = Filter turbulent flow loss coefficient.
filtrdef(7,*) = Filter effective flow area fraction.

```

Note that **filtrdef**(5,*) and **filtrdef**(6,*) must be positive, and **filtrdef**(7,*) must be greater than zero and less than or equal to one. Note **maxf** is the maximum number of filter definitions allowed and is currently set to 20 via a parameter statement in the CINPUT module.

Example input:

```

filtrdef(1,1) = 2, 4, 2, 1, 2.0, 1.0, 0.20,

```

The example given above places two filters in duct 2, one at cell edge 3 and the other at cell edge 4. Both filters will have a laminar flow loss coefficient of 2.0, a turbulent flow loss coefficient of 1.0, and a filter flow area fraction of 20%.

3.4.8 Defining Dampers

Dampers are defined with the variable **dampdef(8,maxf)**, which is in the **innet** NAMELIST group. Eight numbers are required per definition:

dampdef(1,*) = I1, first I index.
dampdef(2,*) = I2, last I index.
dampdef(3,*) = IDUCT duct number.
dampdef(4,*) = IBLK block number (not used).
dampdef(5,*) = If **dampdef(5,*)** > 0, then **dampdef(5,*)** = ITABLE, the table number for damper movement as a function of time.
dampdef(6,*) = FLK damper flow loss coefficient when damper is fully open.
If **dampdef(5,*)** < 0, then **|dampdef(5,*)|** = rupture disk pressure difference criteria.
dampdef(7,*) = AREADO damper flow area when damper is fully open(cm²).
dampdef(8,*) = FAREAD damper flow area fraction at time = 0.0.

Note that if ITABLE is zero, then the damper flow area fraction at FAREAD will be constant with time. FLK must be positive. AREADO must be positive and nonzero. FAREAD must be greater than or equal to zero and less than or equal to one. ITABLE cannot be larger than the greatest table number input for **dmptb**. Note that **maxf** is the maximum number of damper definitions allowed and is currently set to 20 via a parameter statement in the CINPUT module.

Example input:

```
dampdef(1,1) = 3, 4, 1, 1, 0, 1.0, 10000.0, 1.0,
```

The above example puts a damper model at cell edge 4 in duct 1 with a fully open loss coefficient of 1.0 and a fully open area of 10,000.0 cm², and sets the damper to fully open. The damper definition can also be used to define a rupture disk which fails at a given pressure difference. If **dampdef(5,*)** is negative, then **|dampdef(5,*)|** is the pressure difference which will fail a rupture disk at the location defined by **dampdef**. The following example input would place a rupture disk at cell edge 3 of duct 2, which would fail at a pressure difference of 6000 dynes/cm² with a fully open flow area of 20000 cm².

```
dampdef(1,1) = 2, 3, 2, 1, -6000.0, 20000.0, 0.0,
```

dmptb(2,maxp,maxtb) is the input array in the **innet** NAMELIST group which provides for the time-dependent behavior for dampers. The input for the **dmptb** array is

dmptb(1,i,j) = Time for x-y pair i for table j.

dmptb(2,i,j) = Damper flow area fraction for x-y pair i for table j.

Note that **dmptb(2,i,j)** cannot be less than zero or greater than one.

Example input:

```

;               time(s) flow area fraction
dmptb(1,1,1) = 0.0,      1.0,
                  1.0,      0.5,
                  2.0,      0.0,
dmptb(1,1,2) = 0.0,      0.0,
                  2.0,      1.0,
dampdef(1,1) = 3, 4, 1, 1, 2, 1.0, 10000.0, 0.0,
dampdef(1,2) = 1, 2, 2, 1, 1, 1.0, 10000.0, 1.0,
```

Damper number 1 places a damper at cell edge 4 in duct 1 with a fully open loss coefficient of 1.0 and a damper flow area of 10,000.0 cm²; it will use damper table 2 to start out at time zero with a closed damper, and at 2.0 seconds, the damper will be fully open. Damper 2 puts a damper at cell edge 2 in duct 2 with a fully open loss coefficient of 1.0 and a damper flow area of 10,000.0 cm²; it will use damper table 1 to start out at time zero with a fully open damper, and at 2.0 seconds, the damper will be closed.

Note that **maxp** is the maximum number of points allowed per table and is currently set to 20 via a parameter statement in the CINPUT module. Note that **maxtb** is the maximum number of tables allowed and is currently set to 20 via a parameter statement in the CINPUT module.

3.4.9 Defining Blowers

Blowers are defined with the variable **blowdef(10,maxf)**. Ten numbers are required per definition:

```

blowdef( 1,*) = I1, first I index.
blowdef( 2,*) = I2, last I index.
blowdef( 3,*) = IDUCT duct number.
blowdef( 4,*) = IBLK block number (1 for now).
blowdef( 5,*) = ITBLW table number for blower head versus volumetric flow.
blowdef( 6,*) = ITSPD table number for blower speed vs. time.
blowdef( 7,*) = SPDR, rated blower speed, radians/s.
blowdef( 8,*) = QR, rated blower volumetric flow, cm3/s.
blowdef( 9,*) = HR, rated blower head, cm.
blowdef(10,*) = SPD, blower speed at time = 0.0, radians/s.
```

Note that if ITBLW is zero, then the blower will run at rated conditions and a constant speed of SPD. If ITSPD is zero and ITBLW is not zero, then the blower will run at a constant speed of SPD. SPDR, QR, HR, and SPD cannot be negative. ITBLW cannot be negative or greater than the largest table number for **blwtb** input. ITSPD cannot be negative or greater than the largest table number for SPDTB input.

Note that **maxf** is the maximum number of blower definitions allowed and is currently set to 20 via a parameter statement in the CINPUT module.

Example input:

```
blowdef(1,1) = 1, 2, 5, 1, 0, 0, 40., 100000.0, 200.0,
              40.0,
```

The example given above puts a blower at cell edge 2 in duct 5 that will run with a constant speed of 40 radians/s with a blower head of 200 cm of air.

blwtb(2,maxp,maxtb) is the input array in the innet NAMELIST group which provides the blower head curves. The input for the **blwtb** array is

```
blwtb(1,i,j) = Volumetric flow for x-y pair i for table j, cm3/s.
blwtb(2,i,j) = Blower head for x-y pair i for table j, cm.
```

An example of input for the **blowdef** and **blwtb** arrays is given below:

```

;               Q (cm**3/s)           H (cm)
blwtb(1,1,1) =      0.0,             20000.0,
                100000.0,             5000.0,
                200000.0,             3000.0,
blwtb(1,1,2) =      0.0,             30000.0,
                200000.0,              10.0,
blowdef(1,1) = 1, 2, 5, 1, 2, 0, 40., 100000.0, 200.0,
              40.0,
blowdef(1,2) = 1, 2,10, 1, 1, 0, 40., 100000.0, 200.0,
              40.0,
```

The example given above positions a blower at cell edge 2 in duct 5 and a blower at cell edge 2 in duct 10. Blower number 1 will use the blower head table 2, and blower number 2 will use the blower head table 1. Both blowers will run at a constant speed of 40 radians/s.

Note that **maxp** is the maximum number of points allowed per table and is currently set to 20 via a parameter statement in the CINPUT module. Note that **maxtb** is the

maximum number of tables allowed and is currently set to 20 via a parameter statement in the CINPUT module. **spdtb(2,maxp,maxtb)** is the input array in the innnet NAMELIST group which provides for the time-dependent blower speed tables.

The input for the **spdtb** array is

spdtb(1,i,j) = time for *x-y* pair *i* for table *j*, s.

spdtb(2,i,j) = blower speed for *x-y* pair *i* for table *j*, radians/s.

An example of input for the **spdtb**, **blwtb**, and **blowdef** arrays is given below:

	Time(s)	SPD(radians/s)
spdtb(1,1,1) =	0.0,	0.0,
	100.0,	100.0,
spdtb(1,1,2) =	0.0,	100.0,
	100.0,	0.0,
blwtb(1,1,1) =	0.0,	20000.0,
	100000.0,	5000.0,
	200000.0,	3000.0,
blwtb(1,1,2) =	0.0,	30000.0,
	200000.0,	10.0,
blowdef(1,1) =	1, 2, 5, 1, 2, 1, 40., 100000.0, 200.0,	
	40.0,	
blowdef(1,2) =	1, 2,10, 1, 1, 2, 40., 100000.0, 200.0, 40.0,	

The example given above places a blower at cell edge 2 in duct 5 and a blower at cell edge 2 in duct 10. Blower number 1 will use the blower head table 2, and blower number 2 will use the blower head table 1. Blower number 1 will use blower speed table 1 that goes from a blower speed of zero at time zero to 100.0 radians/s at 100 seconds. Blower number 2 will use blower speed table 2 that goes from a blower speed of 100 radians/s at time zero to zero blower speed at 100.0.

Note that **maxp** is the maximum number of points allowed per table and is currently set to 20 via a parameter statement in the CINPUT module. Note that **maxtb** is the maximum number of tables allowed and is currently set to 20 via a parameter statement in the CINPUT module.

3.5 Checking Geometric Model

Once the mesh has been generated and any walls and obstacles have been defined, the geometry of the computational domain is completely specified. The user may then specify the constituents of the gas mixture to be calculated, impose appropriate initial

and boundary conditions, turn on various desired models, and specify any parameters with regard to running the calculation. However, when setting up a new problem, especially one with a complex geometry, it is often helpful to review the mesh before the actual, desired computation is carried out. Knowing the mesh indices at all computational boundaries and where walls and obstacles are will help minimize errors in defining initial and boundary conditions. This will also make it easy to specify graphical output of the solution at regions of interest, so the calculation can be monitored right from the beginning.

After the input geometry and mesh definition have been read in and processed, GASFLOW writes a file called meshmap that contains a list of all computational cells. Information is given for each cell on its i , j , and k index values, as well as a single index that the code uses for storage in memory (called "master" index, m), and the nature of the cell. The master cell index is related to the logical indices as follows:

$$m = (k-1)*imax*jmax + (j-1)*imax + i$$

In other words, m lists all cells consecutively, going over the i -index first, followed by j , and then k . Note that for multiblock problems, this m is relative to the beginning of the array for each 3D block. For multiblock problems the i , j , k , $iblk$ for a given cell is given by the following formula:

$$m = (k-1)*imax*jmax + (j-1)*imax + i + mboff(iblk)$$

where $mboff(iblk)$ is the offset for each 3D block in the problem described by the `ingf` file.

The fictitious cells beyond the physical domain boundaries are termed *boundary* cells, while a cell within the domain is either a *fluid* or an *obstacle* cell, depending on whether it has been blocked out with a **mob**s definition. Also shown in the file meshmap is whether a fluid cell is open to flow in each of the three directions and the m -index of its neighboring cells in all directions. A section extracted from a meshmap file is shown below:

M	K	J	I	Cell Type	MFLAG	Velocity Comps	----- -J	Neighbors +J	----- -K	----- +K	M
2196	8	1	12	B	24	none	2184	2208	1884	2508	2196
2197	8	2	1	B	24	none	2185	2209	1885	2509	2197
2198	8	2	2	F	6	vw	2186	2210	1886	2510	2198
2199	8	2	3	F	7	uvw	2187	2211	1887	2511	2199
2200	8	2	4	F	7	uvw	2188	2212	1888	2512	2200
2201	8	2	5	F	7	uvw	2189	2213	1889	2513	2201
2202	8	2	6	F	6	vw	2190	2214	1890	2514	2202
2203	8	2	7	O	8	none	2191	2215	1891	2515	2203
2204	8	2	8	O	8	none	2192	2216	1892	2516	2204
2205	8	2	9	O	8	none	2193	2217	1893	2517	2205
2206	8	2	10	F	7	uvw	2194	2218	1894	2518	2206
2207	8	2	11	F	6	vw	2195	2219	1895	2519	2207
2208	8	2	12	B	24	none	2196	2220	1896	2520	2208
2209	8	3	1	B	24	none	2197	2221	1897	2521	2209
2210	8	3	2	F	7	uvw	2198	2222	1898	2522	2210
2211	8	3	3	F	7	uvw	2199	2223	1899	2523	2211
2212	8	3	4	F	7	uvw	2200	2224	1900	2524	2212

Under the Velocity Comps column is information on whether the cell has a velocity component (i. e., whether there can be flow) across the positive cell face in each of the three directions. The cell with an m-index of 2202, for example, is a fluid cell (Cell Type = F), and has velocity components across the positive j- and k-faces (v and w, respectively) but not across the positive i-face (the u-component is not printed), because the next cell in that direction is an obstacle cell (Cell Type for cell 2203 is O). Naturally, all obstacle cells have no velocity components across them. For a fictitious boundary cell (Cell Type = B), there may or may not be flow across any of its faces, depending on the boundary conditions specified there. (In GASFLOW, there are only three types of cells: F, B, and O.) The four columns under Neighbors give the m-indices of adjacent cells in the positive and negative j- and k-directions. Neighbors in the i-direction for an internal cell m have master indices m-1 and m+1. Under the MFLAG column is a number that is used by the code (in binary format) to describe the cell's status, i.e., whether it is open to flow and whether the associated surfaces, if any, have been specified as *no-slip* or *free-slip*, etc. This information, however, is primarily intended for code developers or advanced users who work with the code at the debugger level.

More useful to general code users are graphical displays of the mesh, rather than the tabular listing of each cell as given in the file meshmap. In GASFLOW, five plots of the mesh are generated automatically before the calculation begins. These include 2D slices of the mesh in the i-j, j-k, and i-k planes, as well as 3D perspective views of the mesh from the inside and outside of the computational domain. However, these plots only show the discretization and dimensions of the mesh. Any walls or obstacle cells that may have been defined are not shown on these default plots. The code provides an option for plotting the mesh (and any embedded walls and obstacles) along planes

cutting across all cell faces and cell centers. This is done via the variable **maxcyc** in NAMELIST group **xput**. This input variable normally sets the maximum number of computational cycles or time steps up to which the calculation is allowed to run. However, if **maxcyc** is set to a negative value, the code will not perform any physical calculation but will instead produce a set of plots that show 2D slices of the mesh. How many plots are generated or which views of the mesh are shown depends on the value of **maxcyc**, according to the keys listed in Table 3-1.

As shown in Table 3-1, it is possible to plot six sets of 2D grid plots—one for each *i*, *j*, or *k* cell face location, and one for each *i*, *j*, or *k* cell center location. The set of *i* cell face plots, for example, are *j*-*k* surfaces ($1 \leq j \leq j_{\max}-1$, $1 \leq k \leq k_{\max}-1$) drawn at every *i* mesh line. The set of *k* cell center plots, for example, are *i*-*j* surfaces ($1 \leq i \leq i_{\max}-1$, $1 \leq j \leq j_{\max}-1$) drawn at successive *k*-direction cell center locations. Depending on the value of **maxcyc**, one or more sets can be plotted. An entry of 1 in the table means the set is plotted, while an entry of zero means that the set is not plotted. For example, if only cell center plots of the mesh on all *j*-planes are wanted, then **maxcyc** = -16 should be specified. Cell center plots are often more useful because they show the exact locations of obstacles. At interfaces between fluid and obstacle cells, the cell face plots would be ambiguous.

Table 3-1. Specification of 2D grid plots at cell face and/or cell center locations via input variable `maxcyc` in NAMELIST group `xput`

<code>maxcyc</code>	i Face	j Face	k Face	i Center	j Center	k Center
-1	1	0	0	0	0	0
-2	0	1	0	0	0	0
-3	1	1	0	0	0	0
-4	0	0	1	0	0	0
-5	1	0	1	0	0	0
-6	0	1	1	0	0	0
-7	1	1	1	0	0	0
-8	0	0	0	1	0	0
-9	1	0	0	1	0	0
-10	0	1	0	1	0	0
-11	1	1	0	1	0	0
-12	0	0	1	1	0	0
-13	1	0	1	1	0	0
-14	0	1	1	1	0	0
-15	1	1	1	1	0	0
-16	0	0	0	0	1	0
-17	1	0	0	0	1	0
-18	0	1	0	0	1	0
-19	1	1	0	0	1	0
-20	0	0	1	0	1	0
-21	1	0	1	0	1	0
-22	0	1	1	0	1	0
-23	1	1	1	0	1	0
-24	0	0	0	1	1	0
-25	1	0	0	1	1	0
-26	0	1	0	1	1	0
-27	1	1	0	1	1	0
-28	0	0	1	1	1	0
-29	1	0	1	1	1	0
-30	0	1	1	1	1	0
-31	1	1	1	1	1	0
-32	0	0	0	0	0	1
-33	1	0	0	0	0	1
-34	0	1	0	0	0	1
-35	1	1	0	0	0	1
-36	0	0	1	0	0	1
-37	1	0	1	0	0	1
-38	0	1	1	0	0	1
-39	1	1	1	0	0	1
-40	0	0	0	1	0	1
-41	1	0	0	1	0	1
-42	0	1	0	1	0	1
-43	1	1	0	1	0	1
-44	0	0	1	1	0	1
-45	1	0	1	1	0	1
-46	0	1	1	1	0	1
-47	1	1	1	1	0	1
-48	0	0	0	0	1	1
-49	1	0	0	0	1	1
-50	0	1	0	0	1	1
-51	1	1	0	0	1	1
-52	0	0	1	0	1	1
-53	1	0	1	0	1	1
-54	0	1	1	0	1	1
-55	1	1	1	0	1	1
-56	0	0	0	1	1	1
-57	1	0	0	1	1	1
-58	0	1	0	1	1	1
-59	1	1	0	1	1	1
-60	0	0	1	1	1	1
-61	1	0	1	1	1	1
-62	0	1	1	1	1	1
-63	1	1	1	1	1	1

4 SPECIFICATION OF GAS SPECIES AND PROPERTIES

4.1 Definition of Gas Species

In GASFLOW, the basic thermodynamic properties of all gas species are assumed to be governed by the ideal gas law. The ideal gas equation of state also applies to a multicomponent gas mixture. In other words, Dalton's Law of partial pressures is assumed to be valid. Therefore, within any volume V , we have the following relation for the gas mixture (or for each component):

$$pV = nRT$$

where p is the pressure of the mixture (or partial pressure of a gas component), n is the total number of gram-moles (or number of moles of a gas component), R is the universal gas constant equal to 8.3144 ergs/mole-K, and T is the absolute temperature of the gas mixture. The above relation can also be written in terms of the mass density, ρ , which is given by nM/V , where M is the molecular weight:

$$pM = \rho RT.$$

Therefore, the molecular weight alone is sufficient to define the pressure-density relationship of a gas species.

GASFLOW solves the energy conservation equations in terms of the specific internal energy, i which is related to the absolute temperature, for an ideal gas, by

$$i = \int_{T_{ref}}^T C_v dT' ,$$

(use capital I in eq. above to be consistent with nomenclature of vol I Bottoni)

where C_v is the specific heat capacity at constant volume having units of ergs/g-K, and T_{ref} is a reference temperature. In general, C_v is a function of temperature and one can approximate this function by polynomials of various degrees depending on the accuracy required. GASFLOW gives the user the following options for the calculation of the internal energies:

ieopt = 1	1st order polynomial
ieopt = 2	2nd order polynomial
ieopt = 3	3rd order polynomial
ieopt = 4	4th order polynomial

These fits are provided over two ranges of temperatures, and the user can select which range is more appropriate for the application.

trange = 'low'	T up to 3000 K
trange = 'high'	T up to 5000 K

In addition, the specific heat capacity is required in the evaluation correlations for heat transfer and fluid flow. The user is given the following options for evaluation of the specific heat capacity:

icopt = 0	Derivative of specific internal energy.
icopt = 1	Constant value.
icopt = 2	2nd order polynomial ($T < 750$ K).
icopt = 3	Gordon & McBride approximation.

Note that the conservation equations for mass, energy, and momentum are solved consistently with the user-selected values for **icopt** and **trange**. The recommended selection for **icopt** is **icopt** = 0, which ensures that correlations for heat transfer and fluid flow transport properties are evaluated with a consistent specific heat capacity.

The built-in gas component library in GASFLOW has 25 species with properties given in Table 3-2. GASFLOW does not support user-defined species. The user must choose the species to be calculated from this library.

The input array variable **mat** in NAMELIST group **xput** is used to define the species in a calculation. Any one or all of the species listed in Table 3-2 can be chosen. For example, in a problem involving air, steam, and hydrogen, the input will be

mat = 'air', 'h2o', 'h2' ,

where the character string within each pair of quotes represents the symbol for the corresponding species as given in Table 3-2. The order in which the gas names are listed in the definition of **mat** is arbitrary. However, this order determines the gas component number that identifies each species involved. Therefore, in this example, air is component 1, water vapor is component 2, and hydrogen is component 3 in the gas mixture. These identification numbers will be used in subsequent input specifications where reference to particular components of the mixture is required.

In the example above, we treat air in the gas mixture as a single species, specified as 'air'. In reality, air is itself a mixture consisting of nitrogen, oxygen, and trace amounts of carbon dioxide and inert gases. However, modeling air as a single species

Table 4-1. Properties of gas species available in GASFLOW

Gas Species Common Name	Species Symbol Used in GASFLOW Input
Carbon atoms (soot)	c
Carbon monoxide	co
Carbon dioxide	co2
Hydrogen	h2
Water vapor	h2o
Nitrogen	n2
Nitrous oxide	n2o
Oxygen	o2
Air	air
Argon	ar
Helium	he
Ammonia	nh3
Methane	ch3
Hydroxyl radicals	ho
Hydrogen atoms	h
Hydrogen dioxide	ho2
Nitric oxide	no
Oxygen atoms	o
N-H radicals	nh
H-N-O radicals	hno
Hydrogen Peroxide	h2o2
Amidogen	nh2
Light gas	lg
Xenon	xenon
Liquid water	h2o1

simplifies the input specification and analysis of the calculated results a great deal, as well as reduces the computational time required. The user should follow this approach whenever possible. In problems where nitrogen, oxygen, carbon dioxide, etc., have to be calculated explicitly, such as combustion of hydrogen in oxygen and in nitrous oxide, then air should be specified as consisting of the individual gases at appropriate concentrations. At sea level, the composition of dry air by volume is approximately 78.2% N₂, 20.9% O₂, 0.9% Ar, and 0.03% CO₂. Here, we discuss identifying only the

gases to be involved in the calculation. The concentration, in mole or volume fraction, of each gas component will be specified with the variable **gasdef**, which is discussed in the section on initial and boundary conditions (Section 5).

Some of the species listed in Table 3-2 are not stable molecules. They are included in the gas library because the code (GASFLOW) has been used, and can be used, to study the interaction between the fluid dynamics and *detailed* chemical kinetics of turbulent flame propagation and acceleration. Detailed kinetics of even the "simple" $\text{H}_2\text{-O}_2\text{-H}_2\text{O}$ chemical system involves about 50 reaction steps in which many intermediate reaction products are produced and destroyed. However, because it is intended primarily for practical problems, GASFLOW uses one global reaction to model the entire hydrogen combustion kinetics.

4.2 Definition of Transport Properties

In this section, we discuss how to specify the *physical* transport properties for the gas mixture. These properties determine the rates at which mass, energy, and momentum are transported within the gas by the action of molecular diffusion. (Other mechanisms for mass, energy, and momentum transport include advection and turbulent mixing, both of which depend on the local, instantaneous velocity of the fluid.) In GASFLOW, the diffusion process is modeled by Fick's Law, which states that the diffusive flux is proportional to some gradient quantity that represents a driving potential. The proportionality constant is called the diffusion coefficient. In momentum transfer, the gradient is in the velocity vector, and the diffusion coefficient is the kinematic viscosity, ν . In mass diffusion, the gradient of species density is used, and the diffusion coefficient is called the mass diffusivity, D . For the diffusion of heat, the heat flux is proportional to the product of the temperature gradient and the thermal diffusivity, α . These diffusivities, in general, depend on temperature, mixture composition, and (to a lesser extent) pressure.

If the user selects no temperature dependence for the transport properties, then input specification of ν , D , and α will require three numbers. For the kinematic viscosity ν , the input variable **nu** is used, which has units of cm^2/s . For the mass and thermal diffusivities, we use respectively the nondimensional quantities Sc and Pr (Schmidt and Prandtl numbers) to define them:

$$Sc \equiv \nu/D$$

$$Pr \equiv \nu/\alpha.$$

The Schmidt and Prandtl numbers are represented by the input variables **schmidt** and **prandtl**. All of these variables are in NAMELIST group **xput**.

For example, an input line which reads

nu = 0.2, **prandtl** = 0.7, **schmidt** = 0.4,

specifies constant values of the kinematic viscosity (**nu**) as 0.2 cm²/s, the thermal diffusivity (α) is 0.286 cm²/s, and the mass diffusivity (D) is 0.5 cm²/s. The default value for **nu** is 0.15 cm²/s, while those for **prandtl** and **schmidt** are both 1 (i. e., $\alpha = D = \nu = 0.15$ cm²/s).

Note that the above input is only required or used if the model options requesting calculation of diffusion of momentum, mass, and energy are turned on. These options will be discussed in Section 7.2.

However, if the user requires that the transport properties be functions of the temperature, then the following input options are available:

itopt = 0	Nonmechanistic calculation of the transport properties using input data.	
muoption = 0	Properties are computed from the local density, the input kinematic viscosity nu , and the Prandtl and Schmidt numbers.	
muoption = 1	Properties are computed from the local density, the input kinematic viscosity nu , and constant values for the thermal conductivity and diffusion coefficient.	
muoption = 2	Properties are given by constant values for dynamic viscosity, thermal conductivity, and diffusion coefficient (see Section 7.2).	
itopt $\neq$ 0	Mechanistic calculation of the transport properties, depending on temperature, pressure, and mixture composition.	
itopt = 1	Constant value *	$T < 500$ K
itopt = 2	Linear approximation *	$T < 1000$ K
itopt = 3	Quadratic approximation *	$T < 3000$ K
itopt = 4	Cubic approximation *	$T < 3000$ K
itopt = 5	Quadratic approximation *	$T < 5000$ K
itopt = 6	Using approximation formula from CHEMKIN *	$T < 5000$ K

* regarding temperature dependence for the species' properties

If **itopt** $\neq 0$, no specification of **muoption** is required; it is set to the default value 0. The property fits available in the GASFLOW were obtained from the CHEMKIN computer code (Ref. 1).

5 INITIAL AND BOUNDARY CONDITIONS

GASFLOW solves the Navier-Stokes equations of motion and the energy conservation equations for a fluid in a specified computational domain. The governing equations are time dependent, partial differential equations. To complete the mathematical formulation, we must specify initial and boundary conditions. In problems where heat conduction in solid structures is calculated, the initial and boundary solid temperatures also have to be specified. In this section, we discuss how to define initial and boundary conditions for the *fluid*. Those for the solid thermal structures will be discussed in the next section.

5.1 Specification of Initial Conditions

5.1.1 Fluid Composition and State

Except for a restart run (discussed in Section 9.4), the user must define the pressure, temperature, and composition of the fluid everywhere in the computational domain at the beginning of the calculation. This can be accomplished via the input array variable **gasdef** in NAMELIST group **xput**. Although initial conditions are defined with **gasdef**, the input variable has more general use. For example, the user must define, with **gasdef**, the fluid condition for all fictitious boundary cells that are expected to exchange fluid with adjacent physical cells. (More on that later when we discuss boundary conditions.) The variable is a two-dimensional array. The second index identifies the particular "gas definition." For each **gasdef** specification, there are a minimum of 14 numbers required, which are input through the elements of the first array dimension with the following meaning:

gasdef (1,*)	Beginning i mesh index (cell face number).
gasdef (2,*)	Ending i mesh index (cell face number).
gasdef (3,*)	Beginning j mesh index (cell face number).
gasdef (4,*)	Ending j mesh index (cell face number).
gasdef (5,*)	Beginning k mesh index (cell face number).
gasdef (6,*)	Ending k mesh index (cell face number).
gasdef (7,*)	Block number .
gasdef (8,*)	Pressure (dynes/cm ²) in defined volume. If gasdef (8,*) is less than zero, then the gasdef (8,*) points to the column number in the SORTAM file and the pressure will be obtained from this column in the SORTAM file. If gasdef (8,*) is less than zero and gasdef (8,*) is larger than 1,000,000, then gasdef (8,*) is a packed i,j,k,iblk location and the pressure will be obtained from the cell at i,j,k,iblk .

gasdef (9,*)	Temperature (K) in defined volume. If gasdef (9,*) is less than zero, then the INT(ABS(gasdef (9,*))) points to the column number in the SORTAM file and the temperature will be obtained from this column in the SORTAM file.
gasdef (10,*)	Option flag for specification of gas composition and time-dependent BC. 1 for mass fraction, 2 for volume fraction, or > 9 implies that a time-dependent function for the pressure and temperature will be specified.
gasdef (11,*)	Time(s) at which "gas definition" begins.
gasdef (12,*)	Time(s) at which "gas definition" ends.
gasdef (13,*)	Gas species component number (determined by the order in the gas species list defined by mat). Gas species component can alternatively be specified by its symbol as given in Table 3-2, e. g., 'h2'.
gasdef (14,*)	Mass or volume fraction of above gas species in defined volume. If gasdef (14,*) is less than zero, then the INT(ABS(gasdef (14,*))) points to the column number in the SORTAM file and the mass/volume fraction will be obtained from this column in the SORTAM file.
gasdef (15,*)	Second gas species component number, if needed.
gasdef (16,*)	Mass or volume fraction of second gas species in defined volume, if needed. If gasdef (16,*) is less than zero, then the INT(ABS(gasdef (16,*))) points to the column number in the SORTAM file and the mass/volume fraction will be obtained from this column in the SORTAM file.

From the above, we can see that **gasdef** defines the pressure, temperature, and composition of a specified fluid region. These conditions are imposed on the fluid volume over a specified range of time. Variables **gasdef (15,*)** and beyond are only necessary if the user wants to define a fluid region of multiple gas species. Compositions of up to 23 gas species, 1 liquid component (water liquid), and 1 solid component (carbon soot) may be defined. At least one gas species must be defined, and the sum of all mass or volume fractions defined in each **gasdef** specification must be 1.

The pressure, temperature, and composition for the 1D ducts is also specified with the **gasdef** array. For 1D ducts **gasdef** has the following definition:

gasdef (1,*)	Beginning i mesh index.
gasdef (2,*)	Ending i mesh index.
gasdef (3,*)	IDUCT duct number.
gasdef (4,*)	Not used.

gasdef (5, *)	Not used.
gasdef (6, *)	Not used.
gasdef (7, *)	0 (Must be zero or negative. Used as a flag to indicate gasdef for 1D duct.)
gasdef (8, *)	Pressure dynes/cm ² . If gasdef (8, *) is less than zero, then the INT(ABS(gasdef (8, *))) points to the column number in the SORTAM file and the pressure will be obtained from this column in the SORTAM file.
gasdef (9, *)	Temperature (K). If gasdef (9, *) is less than zero, then the INT(ABS(gasdef (9, *))) points to the column number in the SORTAM file and the temperature will be obtained from this column in the SORTAM file.
gasdef (10, *)	Option flag for specification of gas composition and time-dependent BC: 1 for mass fraction, 2 for volume fraction, or > 9 implies that a time-dependent function for the pressure and temperature will be specified.
gasdef (11, *)	Time(s) at which "gas definition" begins.
gasdef (12, *)	Time(s) at which "gas definition" ends.
gasdef (13, *)	Gas species component number (determined by the order in the gas species list defined by mat). Gas species component can alternatively be specified by its symbol as given in Table 3-2, e. g., 'h2'.
gasdef (14, *)	Mass or volume fraction of above gas species in defined volume. If gasdef (14, *) is less than zero, then the INT(ABS(gasdef (14, *))) points to the column number in the SORTAM file and the mass/volume fraction will be obtained from this column in the SORTAM file.
gasdef (15, *)	Second gas species component number, if needed.
gasdef (16, *)	Mass or volume fraction of second gas species in defined volume, if needed. If gasdef (16, *) is less than zero, then the INT(ABS(gasdef (16, *))) points to the column number in the SORTAM file and the mass/volume fraction will be obtained from this column in the SORTAM file.

Note that at least one definition of **gasdef** is required to fully specify the initial fluid conditions. A common use of **gasdef** is to first specify initial conditions globally, then override them with following definitions for local conditions. Currently, pressure and temperature boundary conditions are only allowed at computational mesh boundaries. Therefore the specification of boundary condition pressures with **gasdef** must currently be only at computational mesh boundaries.

For specification of initial conditions, the beginning and end time should both be set to 0. Consider the following input:

```
mat = 'h2o', 'air',
gasdef(1,1) = 1, 6, 1, 6, 1, 6, 1, 1.013e6, 298.,
              2, 0., 0., 1, 0.1, 2, 0.9
```

which can also be input in the form:

```
mat = 'h2o', 'air',
gasdef(1,1) = 1, 6, 1, 6, 1, 6, 1, 1.013e6, 298.,
              2, 0., 0., 'air', 0.9, 'h2o', 0.1
```

If the coordinate system chosen is Cartesian and the mesh in each of the three directions is the same as that shown on Figure 3-1 (i. e., $i_{max} = j_{max} = k_{max} = 7$), then the above **gasdef** input specifies the initial condition of the fluid throughout the entire physical domain. The fluid is initially composed of 10% water vapor and 90% air by volume at room temperature and atmospheric pressure.

For both 3D blocks and 1D ducts, **gasdef**'s time-dependent functions can be defined for the pressure and temperature. When **gasdef(10,*)** is greater than 9, GASFLOW will use time-dependent functions for the pressure and temperature for the cells specified in the **gasdef** definition. The function type is determined by the ten's digit in the **gasdef(10,*)** input. The constants to be used in the function are determined from the hundreds digit in the **gasdef(10,*)** input. The one's digit for **gasdef(10,*)** is still used as the option flag to indicate whether the input for species is volume fraction or mass fraction. For example, if **gasdef(10,*)** is 321, then the one's digit is 1, the ten's digit is 2, and the hundred's digit is 3. The 1 in the one's place indicates species concentrations will be input as a mass fraction. The 2 in the ten's place indicates a function type of 2 (see list below), and the 3 in the hundred's place indicates that the constants in function type 2 will be obtained from **pfunc(*,3)** and **tfunc(*,3)**.

If **gasdef(10,*)** is greater than 9, let the ten's digit in **gasdef(10,*)** be **ifunc** and the hundred's digit in **gasdef(10,*)** be **icons**. Then the following function types are available for the pressure and temperature, and the constants in these functions are determined from **gasdef(8,*)**, **gasdef(9,*)**, **pfunc(*,icons)**, and **tfunc(*,icons)**. If the function type selected is table look-up, then the user input time versus pressure table is **ptab**, and the user input time versus temperature table is **ttab**.

ifunc = 0 implies constant function.

ifunc = 1 implies linear function, $f = a + b * ts$

ifunc = 2 implies quadratic function, $f = a + b * ts**2$

ifunc = 3 implies cubic function, $f = a + b * ts^{**2} + c * ts^{**3}$
ifunc = 4 implies 4th order function, $f = a + b * ts^{**2} + c * ts^{**3} + d * ts^{**4}$
ifunc = 5 implies table look-up function, $f = \text{table}(tr, f)$
ifunc = 6 implies exponential function, $f = a + b * \exp(tr / c)$
ifunc = 7 implies sinusoidal function, $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$

where,

a = **gasdef**(8,*) or **gasdef**(9,*)
b = **pfunc**(1,icons) or **tfunc**(1,icons)
c = **pfunc**(2,icons) or **tfunc**(2,icons)
d = **pfunc**(3,icons) or **tfunc**(3,icons)
e = **pfunc**(4,icons) or **tfunc**(4,icons)
tr = time - tstart
ts = (time - tstart) / (tend - tstart)
tstart = **gasdef**(11,n)
tend = **gasdef**(12,n)
table(tr,f) = **ptab**(1,maxp,icons) or **ttab**(1,maxp,icons)
maxp = 20, currently

The same function type is used for both pressure and temperature evaluation, but a different set of function constants are available. The user has available the following sets of constants for the functions defined above:

pfunc(4,*) Sets of constants for pressure functions.
ptab(2,*,9) Set of x - y pair tables that define time-dependent pressure boundary condition functions. **ptab**(1,m,n) is the time (s) for m^{th} point of n^{th} pressure table, and **ptab**(2,m,n) is the pressure (dynes/cm²) for m^{th} point of n^{th} pressure table. Currently, the maximum number of points is 20 and the maximum number of tables is 9.
tfunc(4,*) Sets of constants for temperature boundary condition functions.
ttab(2,*,9) Set of x - y pair tables that define time-dependent temperature boundary condition functions. **ttab**(1,m,n) is the time (s) for m^{th} point of n^{th} temperature table, and **ttab**(2,m,n) is the temperature (K) for m^{th} point of n^{th} temperature table. Currently, the maximum number of points is 20 and the maximum number of tables is 9.

Note that for time-dependent tables, linear interpolation is performed between points in the table. The following example illustrates a quadratic increase in pressure from 1 bar to 2 bars from time 100 seconds to 200 seconds. For the example given below, **gasdef**(1,1) will be using the third set of **pfunc** and **tfunc** constants for the quadratic equation. From 100 to 200 seconds, the function for pressure will be

$$P = 10^6 + 10^6((\text{time} - 100) / 100)^2$$

```

pbc(1,1)      = 1, 2, 1, 2, 3, 4, 4, 100.0, 200.0,
gasdef(1,1)    = 1, 2, 1, 2, 3, 4, 4, 1.0e+06, 300.0, 321,
                100.0, 200.0, 'air', 1.0,
pfunc(1,1)     = 100.0,
pfunc(1,2)     = 100.0,
pfunc(1,3)     = 0.0, 1.0e+06,
tfunc(1,1)     = 0.0
tfunc(1,2)     = 0.0
tfunc(1,3)     = 0.0, 0.0,

```

The temperature will not vary but will be held at a constant value of 300 K, since the constants for the temperature function are zero. Note that for the example given above, **gasdef(10,1) = 321**: **iopt** = 1, which implies mass fraction concentrations; **ifunc** = 2, which implies a quadratic function; and **icons** = 3, which implies the third set of constants in the **pfunc** and **tfunc** tables of constants.

The following example illustrates a table look-up for both pressure and temperature. Note that linear interpolation will be performed between points in the table. The second table for both temperature and pressure will be used, and this **gasdef** will be applied from 10.0 to 99,999.0 seconds. Note that the **gasdef** comes on at 10.0 seconds. The times in the table are relative to when the **gasdef** turns on. So the first point in the pressure table is at 50.0 + 10.0 = 60 seconds into the transient. From 10.0 to 60.0 seconds, the first point in the tables will be used (i.e., pressure = 1.1e + 06). Also, note that the times in tables do not have to be same for both the pressure and temperature. The pressure table ends at 200 + 10 = 210 seconds. For time greater than 210 seconds, the last point in the table will be used. For this example, the pressure in bars as a function of time (t) is given below:

```

P = 1.0          for 0 < t < 10
P = 1.1          for 10 < t < 60
P = 1.1 - 0.3*(t-60)/100 for 60 < t < 160
P = 0.8 + 1.2*(t-160)/50 for 160 < t < 210
P = 2.0          for 210 < t < 99999

```

The temperature as a function of time (t) for this example is

```

T = 300          for 0 < t < 60
T = 300 - 5*(t-60)/200 for 60 < t < 260
T = 295 + 15*(t-260)/50 for 260 < t < 310
T = 310          for 310 < t < 99999

```

```

pbc(1,1)      = 1, 2, 1, 2, 3, 4, 4,      0.0, 99999.0,
gasdef(1,1)    = 1, 2, 1, 2, 3, 4, 4, 1.0e+06, 300.0, 1,
                0.0, 10.0, 'air', 1.0,
gasdef(1,2)    = 1, 2, 1, 2, 3, 4, 4, 1.0e+06, 300.0, 251,
                10.0, 99999.0, 'air', 1.0,
ptab(1,1,1)    =      0.0, 1.0e+06,
                100.0, 2.0e+06,
ptab(1,1,2)    =      0.0, 1.1e+06,
                50.0, 1.1e+06,
                150.0, 0.8e+06,
                200.0, 2.0e+06,
ttab(1,1,1)    =      0.0, 300.0,
                100.0, 1000.0,
ttab(1,1,2)    =      0.0, 300.0,
                50.0, 300.0,
                250.0, 295.0,
                300.0, 310.0,

```

If negative values are input for pressure, temperature, and/or composition in the **gasdef** input, then time-dependent values for pressure, temperature, and/or composition can be obtained from the SORTAM file (see Section 5.2.4). When the pressure, temperature, or composition is a negative number in the **gasdef** input, the absolute value of the number points to a column in the SORTAM file. For the following example, the **gasdef** input is

```

gasdef(1,1) = 1, 'im1', 1, 'jm1', 1, 'km1', 1, ; I.C.
              1.00000000e6, 300.0, 1, 0., 0.,
              'n2', 0.80, 'h2', 0.10, 'o2', 0.10,
gasdef(1,2) = 0,      1, 1,      2, 1,      2, 1, ; B.C.
              -2.0, -3.0, 2, 0., 9.e99,
              'n2', -4.0, 'o2', -5.0, 'h2', -6.0,

```

and the SORTAM file is given below.

GF2.1 SORTAM file for ignitor test problem.

```

NCOLS
6
IVVALUES IVTYPES
1 1
0 1
0 1
0 1
0 1
0 1
0 1
time(s) air(g/s) p(dynes/cm^2) t(k) xf(n2) xf(o2) xf(h2)
0.0000E+00 0.0000E+00 1.0000E+06 3.0000E+02 7.9000E-01 2.1000E-01 0.0000E+00
5.0000E+00 1.0000E+01 1.1000E+06 3.1000E+02 7.9000E-01 2.1000E-01 0.0000E+00
1.0000E+01 1.0000E+01 1.1000E+06 3.1000E+02 7.0000E-01 2.0000E-01 1.0000E-01
2.0000E+01 2.0000E+01 1.0000E+06 3.2000E+02 7.0000E-01 2.0000E-01 1.0000E-01
5.0000E+01 2.0000E+01 1.0000E+06 3.0000E+02 7.0000E-01 2.0000E-01 1.0000E-01
5.0000E+02 2.0000E+01 1.0000E+06 3.0000E+02 7.0000E-01 2.0000E-01 1.0000E-01

```

The pressure is given in column two of the SORTAM file, since **gasdef (8,2) = -2**; the temperature is column three in the SORTAM file, since **gasdef (9,2) = -3**; the nitrogen mass fraction is in column four of the SORTAM file, since **gasdef (14,2) = -4**; the oxygen mass fraction is in column five of the SORTAM file, since **gasdef (16,2) = -5**; and the hydrogen mass fraction is in column six of the SORTAM file, since **gasdef (18,2) = -6**.

If the user requires that the pressure in the **gasdef** be obtained from the pressure calculated for a cell inside the mesh, then **gasdef (8,*)** must be input as a negative number with the absolute value of the negative number equal to a packed *i, j, k, iblk* location. A packed *i, j, k, iblk* location is defined as:

$n = i, j, k, \text{iblk packed location} = \text{IJJKKBB}.$
 $i = \text{Number of millions in } n.$
 $j = \text{Number of ten thousands in } n - i * 1000000.$
 $k = \text{Number of hundreds in } n - i * 1000000 - j * 10000.$
 $\text{iblk} = \text{Number of ones in } n - i * 1000000 - j * 10000 - k * 100.$

For example,

gasdef (1,1) = 1,2,3,4,5,6,1,-02030402,300.0,1,0.,1.0e+99,'air',1.0,

The pressure used for this **gasdef** statement will be obtained from the pressure at cell $i = 2, j = 3, k = 4, \text{iblk} = 2$.

5.1.2 Fluid Velocities

For initial fluid velocities, default is that the fluid is initially at rest everywhere in the mesh. However, the user can change the default by setting a constant value for each component of the velocity vector. Then the code will set the initial fluid velocity everywhere in the mesh according to the specified component values. The input variables for defining initial velocity components, in NAMELIST group *xput*, are

ui Initial fluid velocity in *i*-direction (*x* or *r*), cm/s.
vi Initial fluid velocity in *j*-direction (*y* or θ), cm/s.
wi Initial fluid velocity in *k*-direction (*z*), cm/s.

5.2 Specification of Boundary Conditions

GASFLOW offers two methods, one "global" and the other "local," for specifying boundary conditions. These two methods will be described separately.

5.2.1 Global Definition

The first method applies boundary conditions on entire boundaries of the computational domain of all 3D blocks. Each 3D block has six surfaces that bound the three-dimensional mesh discretized by logical indices (i, j, k) . Consequently, the boundary condition on each of these surfaces can be specified through any one of the following variables. These and all other input variables discussed subsequently are in the NAMELIST group `xput`.

ibe	Boundary condition type indicator for the +i (east) boundary.
ibw	Boundary condition type indicator for the -i (west) boundary.
ibs	Boundary condition type indicator for the -j (south) boundary.
ibn	Boundary condition type indicator for the +j (north) boundary.
ibb	Boundary condition type indicator for the -k (bottom) boundary.
ibt	Boundary condition type indicator for the +-k (top) boundary.

These boundary condition types are currently applied uniformly to all 3D blocks in the input model. The boundary conditions on these six boundaries can be specified according to the following key:

Type	Boundary Condition
1	Rigid free-slip wall
2	Rigid no-slip wall
3	Continuative
4	Periodic
5	Specified pressure

Rigid Free-Slip. The default boundary condition is Type 1. Therefore, if no boundary conditions are specified, the code will assume that the entire computational volume is enclosed within rigid, impenetrable walls at which there is free slip, or the gradient of the tangential velocity components is zero. This is the most common boundary condition used. In many practical problems, a large portion of the computational boundaries are solid surfaces (for example, the walls of a room or a containment building), and the mesh resolution is not fine enough to represent the near-wall velocity gradients so the free-slip condition is the best approximation there. This is also the boundary condition at the $-i$ boundary, or at $r = 0$, if cylindrical coordinates are used. This is not a poor numerical representation at the $r = 0$ boundary because that surface has a zero area, hence there is no severe flow limitation caused by the free-slip wall condition at the centerline.

Rigid No-Slip. The no-slip condition, Type 2, is another option with which the user can define a boundary as an impenetrable surface. No slip means that the fluid "sticks" to the solid wall and all velocity components are zero there. (In the current version, the

code does not support moving wall boundary conditions.) This boundary condition is used in problems where the velocity gradients near solid surfaces are important, and the mesh is sufficiently fine to resolve them. For example, if the classical Hagen-Poiseuille flow, i. e., laminar flow through a circular pipe, is to be simulated, then the no-slip condition must be applied at the pipe wall to be able to calculate a parabolic velocity profile in the steady solution.

Continuative. Type 3 is the continuative boundary condition. This condition is usually applied at outflow boundaries, where the fluid is to flow smoothly out of the mesh, causing minimum upstream effects. With this boundary condition, the gradients of pressure, internal energy, density, velocity, etc., across the specified boundary are set to zero.

Periodic. The periodic boundary condition, Type 4, specifies that the fluid conditions at the beginning and ending boundaries in a particular direction are identical. Periodic boundaries must be specified in pairs, i. e., both the + and - boundaries must be specified as periodic. This condition is most commonly used for defining the θ -boundaries when the mesh covers the full 360° in the azimuthal direction. Therefore, if cylindrical coordinates are chosen and the mesh is defined to extend from 0° to 360° , then the following input should be used to specify the appropriate boundary condition at the $-\theta(-j)$ and $+\theta(+j)$ boundaries:

$$\mathbf{ibs} = 4, \mathbf{ibn} = 4$$

The periodic boundary condition is also sometimes used in problems where the computational domain represents part of a much larger physical volume. An example of this is the direct numerical simulation of turbulence. Although only part of the physical volume is modeled, the computational volume chosen is large enough to contain all the relevant scales of motion, so that the flow field calculated is representative of the entire fluid domain and periodic boundary conditions are thus good approximations at the mesh boundaries.

Specified Pressure. The pressure boundary condition, Type 5, specifies the fluid pressure at a particular boundary. The pressure value at the boundary will be that of the fluid in the adjacent fictitious boundary cells. Therefore, for complete specification of the pressure boundary condition, the input array **gasdef** must also be used to give a pressure value in the boundary cells adjacent to the boundary surface. Consider a problem in which $i_{\max} = 11$, $j_{\max} = k_{\max} = 7$. To define a pressure boundary condition at the $+i$ boundary of 1 bar (10^6 dynes/cm²), the user would write the following input:

```
ibe = 5,  
gasdef(1,1) = 10, 11, 1, 7, 1, 7, 1, 1.e6, 300.,
```

1, 0., 9.e99, 1, 1.

The additional information, temperature and gas composition, will be used to define the properties of the fluid flowing into the computational domain across the specified-pressure boundary, if that occurs during the calculation.

5.2.2 Local Definition

In the following, we shall describe an alternative method of defining boundary conditions. The second method of specifying boundary conditions complements the first method by allowing flexibility in imposing the boundary conditions at arbitrary parts of the mesh and within arbitrary time intervals. While the first method applies boundary conditions to the *entire* extreme surfaces of the mesh in each direction at all time, the second method is capable of imposing boundary conditions on selective surfaces, which can be external or internal, over a specified time range. If a surface has boundary conditions defined by both methods, then only that condition defined by the second method will take effect. Therefore, the user can, for instance, define an entire boundary surface with the first, global method, such as specifying Type 2 to indicate a no-slip wall. Then the user can overwrite a portion of that surface with a velocity boundary condition, specified via the second, local method to simulate a wall with an opening through which fluid is injected. Note that the **vbc**, **pbc**, **cbc**, and **mbc** boundary conditions can only currently be applied to the extreme surfaces of the mesh in each direction.

However, only four of the five types of boundary conditions discussed above can be specified with this local method. The periodic boundary condition must still be specified with the first method, i. e., this condition must be imposed on the whole surface of each of the pair of boundaries in a particular direction. The other boundary conditions can be applied to any surface by specifying the appropriate beginning and ending mesh indices in all three directions. Moreover, the continuative, pressure, and velocity boundary conditions can be imposed over a specific time range. This method is also used to specify velocity boundary conditions, which cannot be done with the first method. In addition, the mass flow rate can be specified as a boundary condition. With the mass flow rate specified as a boundary condition, the donor cell density is used to determine the actual velocity that will be specified as a boundary. The input variables required for defining each of the boundary conditions are described below.

Free-Slip and No-Slip Walls. Since any impenetrable surface is *free-slip* by default, there is no need to explicitly request this boundary option. However, the default free-slip condition can be changed to no-slip via the **nslipdef** variable, which requires 8 entries per definition:

nslipdef(1,*) Beginning i mesh index (cell face number).

nslipdef(2,*)	Ending i mesh index (cell face number).
nslipdef(3,*)	Beginning j mesh index (cell face number).
nslipdef(4,*)	Ending j mesh index (cell face number).
nslipdef(5,*)	Beginning k mesh index (cell face number).
nslipdef(6,*)	Ending k mesh index (cell face number).
nslipdef(7,*)	Block number .
nslipdef(8,*)	The side of the surface that is no-slip. Options are
	'lower' negative side,
	'upper' positive side,
	'both' both sides.

In most cases, the beginning and ending i, j, and k mesh indices will define a surface that is coincident with a solid wall, which is to have the no-slip boundary condition. However, **nslipdef** can also be used to specify that all faces of an obstacle volume be no-slip. To do this the user sets **nslipdef(8,*)** to 'both', and specifies the beginning and ending mesh indices that define the volume occupied by the obstacle. Similar to other two-dimensional input array variables such as **walls**, **mobs**, and **gasdef**, the second index of **nslipdef** is used to allow more than one input specification. Consider the following examples:

```

nslipdef(1,1) = 2, 2, 3, 5, 1, 9, 1, 'lower',
nslipdef(1,2) = 3, 3, 3, 5, 1, 9, 1, 'both',
nslipdef(1,3) = 4, 6, 2, 7, 4, 8, 1, 'both'

```

The first definition sets the lower side of the $i = 2$ ($j = 3-5$, $k = 1-9$) surface to no-slip. The second definition sets both the positive and negative sides of the $i = 3$ ($j = 3-5$, $k = 1-9$) surface to no-slip. The last definition sets all the surfaces bounding the volume defined by $i = 4-6$, $j = 2-7$, and $k = 4-8$ to no-slip.

Continuative. The continuative boundary condition can be specified via the variable **cbc**, which requires 9 numbers per definition:

cbc(1,*)	Beginning i mesh index (cell face number).
cbc(2,*)	Ending i mesh index (cell face number).
cbc(3,*)	Beginning j mesh index (cell face number).
cbc(4,*)	Ending j mesh index (cell face number).
cbc(5,*)	Beginning k mesh index (cell face number).
cbc(6,*)	Ending k mesh index (cell face number).
cbc(7,*)	Block number .
cbc(8,*)	Start time(s).
cbc(9,*)	End time(s).

For example, the following input

```
cbc = 21, 21, 1, 15, 1, 15, 1, 0.0, 9.e99
```

will specify that the boundary $i = 21$ has a continuative boundary condition, i. e., gradients of pressure, density, etc., across the boundary are zero. Because of the large end time, which exceeds practically all physical problem time, this boundary condition is effective throughout the calculation.

Pressure. The specified pressure boundary condition can be invoked with the input variable **pbc**:

```
pbc (1, *)   Beginning i mesh index (cell face number).  
pbc (2, *)   Ending i mesh index (cell face number).  
pbc (3, *)   Beginning j mesh index (cell face number).  
pbc (4, *)   Ending j mesh index (cell face number).  
pbc (5, *)   Beginning k mesh index (cell face number).  
pbc (6, *)   Ending k mesh index (cell face number).  
pbc (7, *)   Block number .  
pbc (8, *)   Start time (s).  
pbc (9, *)   End time (s).
```

Similar to the first method, the values of pressure to be specified at the boundary are taken from the fictitious boundary cells, the fluid conditions of which must be defined with **gasdef**. As an example, consider a mesh that is the same as the example used above for illustrating how to globally specify the pressure boundary condition, i. e., a mesh in which $i_{\max} = 11$, $j_{\max} = k_{\max} = 7$. Now the $+i$ boundary (i. e., the surface where $i = i_{\max} - 1 = 10$, refer to Figure 3-1 for convention) is by default a free-slip wall. Suppose there is a hole at the center of the wall that is open to some ambient condition on the outside. Furthermore, the pressure will have a step change from 1 to 2 atmospheres (plus other changes in the ambient condition) at 100 s. The appropriate input would be

```
pbc (1,1)    = 10, 10, 3, 4, 3, 4, 1, 0.0, 9.e99,  
gasdef (1,1) = 10, 11, 3, 4, 3, 4, 1, 1.0132e6, 300.,  
              2, 0., 100., 1, 1.,  
gasdef (1,2) = 10, 11, 3, 4, 3, 4, 1, 2.0265e6, 400.,  
              2, 100., 9.e99, 1, 0.5, 2, 0.5
```

If inflow occurs at the $i = 10$ boundary during the calculation, the properties (pressure, temperature, composition) of the fluid entering will be those defined in the boundary cells by **gasdef**.

Since **gasdef** allows for time-dependent functions for pressure and temperature the example given above can be made to impose a time-dependent pressure that changes from 1 to 2 atms over a 100 second time interval.

```

pbc(1,1) = 10, 10, 3, 4, 3, 4, 1, 0.0, 9.e99,
gasdef(1,1) = 10, 11, 3, 4, 3, 4, 1, 1.0132e6, 300.,
              151, 0., 100., 1, 1.,
ptab(1,1,1) = 0.0, 1.0132e+06,
              100.0, 2.0265e+06,
ttab(1,1,1) = 0.0, 300.0,
              100.0, 300.0,

```

Velocity. The input variable **vbc** can be used to specify velocity boundary conditions. Each definition requires 10 numbers:

vbc (1,*)	Beginning i mesh index (cell face number).
vbc (2,*)	Ending i mesh index (cell face number).
vbc (3,*)	Beginning j mesh index (cell face number).
vbc (4,*)	Ending j mesh index (cell face number).
vbc (5,*)	Beginning k mesh index (cell face number).
vbc (6,*)	Ending k mesh index (cell face number).
vbc (7,*)	Block number.
vbc (8,*)	If vbc (8,*) < 100, then vbc (8,*) is the index in the vvalue array that will define a constant velocity from the start time to the end time for the vbc definition. If vbc (8,*) > 100, then vbc (8,*) points to a time-dependent function which will be used to determine the velocity as a function of time (see example below).
vbc (9,*)	Start time(s).
vbc (10,*)	End time(s).

For 1D duct velocity boundary conditions, the array takes on the following definition:

vbc (1,*)	1 to indicate west end of duct and 2 to indicate east end of duct.
vbc (2,*)	vbc (1,*).
vbc (3,*)	Duct number.
vbc (4,*)	Not used.
vbc (5,*)	Not used.
vbc (6,*)	Not used.
vbc (7,*)	0 or negative. Used as a flag to indicate input for a duct.
vbc (8,*)	If vbc (8,*) < 100, then vbc (8,*) is the index in the vvalue array that will define a constant velocity from the start time to the end time for the vbc definition. If vbc (8,*) > 100, then vbc (8,*)

points to a time dependent function which will be used to determine the velocity as a function of time (see example below).

vbc(9,*) Start time(s).

vbc(10,*) End time(s).

Note that **vbc(8,*)** < 100 does not directly specify what the velocity value is; rather, it specifies an integer that points to the corresponding element in the user input array **vvalue** that stores velocity values. The sign convention used for the velocity is that positive velocity indicates flow in the direction of increasing *i*, *j*, or *k* index. Note that for ducts, the direction of increasing *i* is from the west end to the east end of the duct. The following example illustrates the use of **vbc** and **vvalue** to define inflow and outflow conditions for a mesh in which *imax* = 11, *jmax* = *kmax* = 7:

```
vbc(1,1) = 1, 1, 1, 7, 1, 7, 1, 3, 1.0, 2.0,
vbc(1,2) = 1, 1, 1, 7, 1, 7, 1, 2, 2.0, 5.0,
vbc(1,3) = 10, 10, 1, 7, 1, 7, 1, 2, 2.0, 5.0,
gasdef(1,8) = 0, 1, 1, 7, 1, 7, 1, 1.0132e6, 298.,
                2, 0., 9.e99, 1, 1.,
vvalue = 10., 30., 50., 20.
```

The first two **vbc** definitions specify a velocity of 50 cm/s during the time interval between 1 and 2 s, followed by a lower velocity of 30 cm/s from 2 to 5 s at the -*i* boundary. The third **vbc** definition specifies a velocity of 30 cm/s during the time interval between 2 and 5 s at the +*i* boundary. (Since all boundaries are by default free-slip walls, the -*i* and +*i* boundaries are closed until the beginning time of the respective **vbc** definitions.) Because these velocities are positive, the boundary condition at the -*i* boundary represents an inflow condition, whereas that at the +*i* boundary represents an outflow condition. For inflow conditions, the user must also define the fluid condition in the boundary cells adjacent to the inflow boundary. This is done in the above example with the 8th **gasdef** definition, which states that the incoming fluid is at atmospheric pressure and room temperature, and consists of pure gas component 1. (The gas component number is defined by the order in which the gas species are listed in the definition of the **mat** array.)

If **vbc(8,*)** is larger than 100, then a time-dependent function for the velocity boundary condition will be specified.

When **vbc(8,*)** is larger than 100, then the one's digit (let the one's digit be **ivv**) points to the value in the **vvalue** array to be used for the time = 0.0 constant in the velocity time-dependent functions; the ten's digit (let the ten's digit be **ifunc**) is the function type; and the hundred's digit (let the hundred's digit be **icons**) points to the constants to be used in the **vfunc** constants table. The function types available for

velocity boundary conditions are the same as for the pressure and temperature functions defined in a time-dependent **gasdef** definition.

ifunc = 0 implies constant function.
ifunc = 1 implies linear function, $f = a + b * ts$
ifunc = 2 implies quadratic function, $f = a + b * ts**2$
ifunc = 3 implies cubic function, $f = a + b * ts**2 + c * ts**3$
ifunc = 4 implies 4th order function, $f = a + b * ts**2 + c * ts**3 + d*t**4$
ifunc = 5 implies table look-up function, $f = \text{table}(tr, f)$
ifunc = 6 implies exponential function, $f = a + b * \exp(tr / c)$
ifunc = 7 implies sinusoidal function, $f = a + b*\cos(tr*\pi/c) + d*\sin(tr*\pi/e)$

where,

a = **vvalue(ivv)**
b = **vfunc(1,icons)**
c = **vfunc(2,icons)**
d = **vfunc(3,icons)**
e = **vfunc(4,icons)**
tr = time - **tstart**
ts = (time - **tstart**)/(**tend** - **tstart**)
tstart = **vbc(9,n)**
tend = **vbc(10,n)**
table(tr,f) = **vtab(1,maxp,icons)**

For time-dependent velocity boundary conditions, the user must either supply a table of constants (i.e., **vfunc**) if **ifunc** is not equal to 5 or a table of *x-y* pairs (i.e., **vtab**) if **ifunc** is equal to 5.

vfunc(4,maxtb) Sets of constants for velocity boundary condition functions.
vtab(2,maxp,9) Set of *x-y* pair tables that define velocity boundary condition functions.

Note that with a parameter statement in GASFLOW, **maxp** (i.e., the maximum number of points in a table) is currently set to 20 and **maxtb** (i.e., the maximum number of tables) is set to 20.

The following example illustrates a linear increase in velocity from 100 cm/s to 200 cm/s from time 50 seconds to 150 seconds.

vvalue = 100.0, 250.0, 0.0, -225.0,
vbc(1,1) = 1, 1, 1, 0, 0, 0, 0, 111, 50.0, 150.,
vfunc(1,1) = 100.0,

Mass Flow Rate. The input variable **mbc** can be used to specify mass flow rate boundary conditions. The actual velocity used as the boundary condition is determined from the user-supplied mass flow rate, the donor cell density, and the flow area of the cell edge where the boundary condition is to be supplied, calculated according to the following formula:

$$vbc = mbc / (<\rho>A)$$

where

vbc = Velocity boundary condition.
mbc = Mass flow rate boundary condition.
<ρ> = Donor cell density.
A = Flow area.

Each **mbc** definition requires 10 numbers:

mbc (1, *) Beginning i mesh index (cell face number).
mbc (2, *) Ending i mesh index (cell face number).
mbc (3, *) Beginning j mesh index (cell face number).
mbc (4, *) Ending j mesh index (cell face number).
mbc (5, *) Beginning k mesh index (cell face number).
mbc (6, *) Ending k mesh index (cell face number).
mbc (7, *) Block number .
mbc (8, *) If **mbc (8, *)** < 100, then **mbc (8, *)** is the index in the **mvalue** array that will define a constant mass flow rate from the start time to the end time for the **mbc** definition. If **mbc (8, *)** > 100, then **mbc (8, *)** points to a time-dependent function which will be used to determine the mass flow rate as a function of time (see example given below).
mbc (9, *) Start time(s).
mbc (10, *) End time(s).

For 1D duct velocity boundary conditions, the array takes on the following definition:

mbc (1, *) 1 to indicate west end of duct and 2 to indicate east end of duct.
mbc (2, *) **mbc (1, *)**.
mbc (3, *) Duct number.
mbc (4, *) Not used.
mbc (5, *) Not used.
mbc (6, *) Not used.
mbc (7, *) 0 or negative. Used as a flag to indicate input for a duct.
mbc (8, *) If **mbc (8, *)** < 100, then **mbc (8, *)** is the index in the **mvalue** array that will define a constant mass flow rate from the start time to the

end time for the **mbc** definition. If **mbc(8,*)** > 100, then **mbc(8,*)** points to a time-dependent function which will be used to determine the mass flow rate as a function of time (see example below).

mbc(9,*) Start time(s).

mbc(10,*) End time(s).

Note that **mbc(8,*)** < 100 does not directly specify what the mass flow rate value is; rather, it specifies an integer that points to the corresponding element in the array **mvalue** that stores mass flow rate values. The sign convention used for the mass flow rate is that positive mass flow rate indicates flow in the direction of increasing i, j, or k index. Note that for ducts, i is increasing from the west end to the east end of the duct. The examples for **vbc** and **vvalue** are equivalent to **mbc** and **mvalue** with **vbc** replaced by **mbc** and **vvalue** replaced by **mvalue**.

If **mbc(8,*)** is larger than 100, then a time-dependent function for the mass flow rate boundary condition will be specified. When **mbc(8,*)** is larger than 100, then the one's digit (let the one's digit be **imv**) points to the value in the **mvalue** array to be used for the time = 0.0 constant in the velocity time-dependent functions; the ten's digit (let the ten's digit be **ifunc**) is the function type; and the hundred's digit (let the hundred's digit be **icons**) points to the constants to be used in the **vfunc** constants table. The function types available for mass flow rate boundary conditions are the same as for the pressure and temperature functions defined in a time-dependent **gasdef** definition and for the time-dependent velocity defined in the **vbc** array.

ifunc = 0 implies constant function.

ifunc = 1 implies linear function, $f = a + b * ts$

ifunc = 2 implies quadratic function, $f = a + b * ts**2$

ifunc = 3 implies cubic function, $f = a + b * ts**2 + c * ts**3$

ifunc = 4 implies 4th order function, $f = a + b * ts**2 + c * ts**3 + d*ts**4$

ifunc = 5 implies table look-up function, $f = \text{table}(tr, f)$

ifunc = 6 implies exponential function, $f = a + b * \exp(tr / c)$

ifunc = 7 implies sinusoidal function, $f = a + b*\cos(tr*pi/c) + d*\sin(tr*pi/e)$

where,

a = **mvalue(imv)**

b = **mfunc(1,icons)**

c = **mfunc(2,icons)**

d = **mfunc(3,icons)**

e = **mfunc(4,icons)**

tr = time - tstart

ts = (time - tstart)/(tend - tstart)

```

tstart      = mbc( 9,n)
tend        = mbc(10,n)
table(tr,f) = mtab(1,maxp,icons)

```

For time-dependent mass flow rate boundary conditions, the user must either supply a table of constants (i.e., **mfunc**) if **ifunc** is not equal to 5, or a table of *x-y* pairs (i.e., **mtab**) if **ifunc** is equal to 5.

```

mfunc(4,maxtb)  Sets of constants for mass flow rate boundary condition
                  functions.
mtab(2,maxp,9)  Set of x-y pair tables that define mass flow rate boundary
                  condition functions.

```

Note that with a parameter statement in GASFLOW, **maxp** is currently set to 20 and **maxtb** is set to 20.

The following example illustrates a linear increase in mass flow rate from 1 gm/s to 2 gm/s from time 50 seconds to 150 seconds.

```

mvalue      = 1.0,
mbc(1,1)    = 1,  1,  1,  0,  0,  0,  0, 111,  50.0, 150.,
vfunc(1,1)  = 1.0,

```

5.2.3 Diffusion Cutoff and Mass Balance with Source Reservoirs

In the following, we shall describe a method of defining internal boundary conditions to represent the sources of mass, momentum, and energy within the boundaries of the computational mesh. The idea is to generate a reservoir where we can define the fluid condition and velocity so that the fluid from this reservoir can be convected into the mesh.

The procedure is a little complicated and therefore can be best served with an example. The geometry from Figure 3-6 will be used with the addition of the reservoir, conditions in the reservoir, flow conditions from the reservoir to the rest of the computational domain, a method to subtract the reservoir mass and energy out of the normal mass and energy balance, and a method to insure that mass and energy are not diffused from either the reservoir to the computing volume of interest or in the reverse direction.

First, we will define two additional input quantities. The first is used to subtract from the mass and energy balances the mass and energy from the reservoir volume. This can be accomplished via the input array variable **subsofdef** in NAMELIST group **xput**. The variable is a two-dimensional array. The second index identifies the particular "reservoir volume." For each **subsofdef** specification, there are seven numbers

required, which are input through the elements of the first array dimension with the following meaning:

subsofdef (1,*)	Beginning i mesh index (cell face number).
subsofdef (2,*)	Ending i mesh index (cell face number).
subsofdef (3,*)	Beginning j mesh index (cell face number).
subsofdef (4,*)	Ending j mesh index (cell face number).
subsofdef (5,*)	Beginning k mesh index (cell face number).
subsofdef (6,*)	Ending k mesh index (cell face number).
subsofdef (7,*)	Block number .

To prevent undesired diffusion of gases from the reservoir into the computing region of interest and the reverse process as well, we introduce a method of forcing all diffusion coefficients at the reservoir boundary to zero. We define a surface normal to any of the three orthogonal dimensions with logical indices i, j, and k. This is done via the input array variable **zeroddef** in the NAMELIST group **xput**. The array **zeroddef** is 2D with the second index identifying the particular surface definition and the first index specifying seven numbers that are required to define the **zeroddef** surface:

zeroddef (1,*)	Beginning i mesh index (cell face number).
zeroddef (2,*)	Ending i mesh index (cell face number).
zeroddef (3,*)	Beginning j mesh index (cell face number).
zeroddef (4,*)	Ending j mesh index (cell face number).
zeroddef (5,*)	Beginning k mesh index (cell face number).
zeroddef (6,*)	Ending k mesh index (cell face number).
zeroddef (7,*)	Block number.

The asterisk (*) should be replaced by an integer that identifies the particular **subsofdef** and **zeroddef** definitions.

GASFLOW supports 300 definitions each for **subsofdef** and **zeroddef**.

The example is as follows:

1. We assume the same geometry as the example from Figure 3-6.
2. The reservoir is located at cell $i=8$, $j=6$, $k=10$, and walls are constructed around the reservoir such that it is open in the positive x direction.
3. An air mixture of nitrogen and oxygen at 1 bar and 300 K is established throughout the entire computational volume as an initial condition.
4. The reservoir is filled with hydrogen also at 1 bar and 300 K and will remain at that condition for 10 s.
5. Hydrogen from the reservoir is injected into the computational volume at 100 cm/s for a period of 10 s.

6. The mass and energy of the reservoir is subtracted from the overall mass and energy balance using the **subsodef** input variable.
7. Diffusion of hydrogen from the reservoir and nitrogen and oxygen into the reservoir is prohibited by the use of the **zeroddef** input variable.

The input stream is shown here:

```
$xput
.
.
.
mobs= 3,10, 1,11, 5, 8, 1, 1, ; solid obstacle

walls= 2,10, 1,11, 3, 3, 1, 2, ; horizontal wall
       2, 2, 1,11, 6,10, 1, 2, ; vertical wall

       7, 8, 5, 6, 10,10, 1, 2, ; reservoir top wall
       7, 8, 5, 6, 9, 9, 1, 2, ; reservoir bottom wall
       7, 8, 5, 5, 9,10, 1, 2, ; reservoir south wall
       7, 8, 6, 6, 9,10, 1, 2, ; reservoir north wall
       7, 7, 5, 6, 9,10, 1, 2, ; reservoir north wall

holes= 5, 7, 4, 7, 6, 8, 1, 0,0, 0,0, 0,1, ; top hole
       8, 9, 5, 6, 5, 8, 1, 0,0, 0,0, 1,1, ; thru hole
       8, 9, 5, 6, 2, 4, 1, 1,1, 1,1, 1,1, ; wall hole

mat= 'h2', 'n2', 'o2', ; problem components

gasdef(1,1)=1,11, 1,11, 1,11, 1, 1.0e+06, 300.0, 2, 0.0, 0.0,
        'n2', 0.79, 'o2', 0.21, ; initial conditions
gasdef(1,2)=7, 8, 5, 6, 9,10, 1, 1.0e+06, 300.0, 2, 0.0,10.0,
        'h2', 1.00, ; reservoir conditions

vbc(1,1)= 8, 8, 5, 6, 9,10, 1, 1, 0.0,10.0, ; reservoir U

vvalue= 100.0, ; reservoir inflow velocity value (cm/s)

subsodef(1,1)=7, 8, 5, 6, 9,10, 1, ; subtract mass & energy
zeroddef(1,1)=8, 8, 5, 6, 9,10, 1, ; zero reservoir diffusion
.
.
.
$end
```

A display of the geometry including the reservoir is presented in Figure 5-1.

Figure 5-1. Example of applying an internal mass, momentum, and energy source

5.2.4 Boundary Conditions from sortam File

The **vvalue** used in a **vbc** definition can be specified as a time-dependent table if a **sortam** input file is provided by the user. The **sortam** file is a table of velocities, mass flow rates, and/or volumetric flow rates that can be applied at different locations in the GASFLOW mesh. The **sortam** velocities, mass flow rates, and/or volumetric flow rates can be applied at the boundaries of the mesh or at the boundary of an internal mass, momentum, and energy source described in Section 5.2.3. The locations where these time-dependent flow rate boundary conditions will be applied is identified through the **vbc** input as described in Section 5.2.2. The element in the **vvalue** array referenced by the **vbc** will be changed with time according to the input in the **sortam** file. GASFLOW expects a **sortam** file if **sortami** is 1 in the **xput** NAMELIST group. The format of the **sortam** file is described below.

Record #	Description
1	Title, (a80) for the file.

2 Comment, (a80).

3 **ncols**,(*) number of columns of data not counting the first column which is
always time. The column number for the time column is zero.

4 Comment, (a80).

5 **ivvalues**(1), **ivtypes**(1), (*), where **ivvalues**(j) is the **ivvalues** for the
jth column in the sortam table. **ivvalues** is the index pointing to the array
element in the **vvalue** array that will be time-dependent (i.e.
vvalue(**ivvalues**(j)) will be defined by the jth column of data in the
sortam file). An **ivvalues**(j) = 0 implies that the jth column of data in the
sortam file will not be used as a flow rate boundary condition, but is
retained for documentation or for other graphics programs. **ivtypes**(j)
indicates the type of data in the jth column of data in the sortam file.
ivtypes(j) = 0, implies that the jth column of data is a velocity, cm/s.
ivtypes(j) = 1, implies that the jth column of data is a mass flow rate,
gm/s.
ivtypes(j) = 2, implies that the jth column of data is a volumetric flow
rate, cm³/s.

6 **ivvalues**(2), **ivtypes**(2), (*).

4+**ncols** **ivvalues**(**ncols**), **ivtypes**(**ncols**), (*).

5+**ncols** Header for sortam table, (a80).

6+**ncols** TIME, (SORTAB(*,j),j=1,**ncols**), (*), where time is the time in seconds for
the flow rates that will be read into the **sortab** array in columns j = 1
through **ncols**. **sortab** is the container array that holds the columns of
data read in from the sortam file. Note that the first data element read in
after time is counted as column 1.

...

...

...

A partial listing of a sortam file is given below:

GF2.1 SORTAM file for GX6.

NCOLS

5

IVVALUES IVTYPES

1	1
2	1
3	1
5	1
0	1

time(s) r5 h2o(g/s) r6 h2o(g/s) r8 h2o(g/s) h2 (g/s) vreco (cm/s)

0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
3.0000E+01	1.8000E+01	9.0000E+00	9.0000E+00	0.0000E+00	0.0000E+00
2.2680E+03	1.8000E+01	9.0000E+00	9.0000E+00	0.0000E+00	0.0000E+00
2.2980E+03	2.8000E+01	1.4000E+01	1.4000E+01	0.0000E+00	0.0000E+00
3.2400E+03	2.8000E+01	1.4000E+01	1.4000E+01	0.0000E+00	0.0000E+00
3.2700E+03	5.0000E+01	2.1500E+01	2.1500E+01	0.0000E+00	0.0000E+00
5.2920E+03	5.0000E+01	2.1500E+01	2.1500E+01	0.0000E+00	0.0000E+00
5.3220E+03	5.0000E+01	2.9000E+01	2.9000E+01	0.0000E+00	0.0000E+00

In the partial listing of a sortam file given above, you can see that GASFLOW will expect 5 columns of data in addition to the time column. Also, the last or fifth column of flow rate data will be read, but not used. The first four columns of flow rate data are all mass flow rate data. The first three columns of data will be used to set **vvalue(1)**, **vvalue(2)**, and **vvalue(3)**, respectively as a function of time. The fourth column of data will be used to set **vvalue(5)** as a function of time. To find flow rates for time between time points given in the table, linear interpolation is used.

For mass flow rate and volumetric flow rate to be converted to velocity requires a flow area. The flow area is determined from the **vbc** input that refers to the **vvalue** identified for a given column of data in the sortam file. For example, the **vbc** input given below

```

vbc(1,1) = 08, 10, 08, 10, 09, 09, 1, 1, 0., 1.e+99,
vbc(1,2) = 08, 10, 08, 10, 04, 04, 1, 2, 0., 1.e+99,
vbc(1,3) = 08, 10, 16, 18, 04, 04, 1, 3, 0., 1.e+99,
vbc(1,4) = 04, 04, 15, 26, 3, 4, 1, 5, 0., 1.e+99,

```

and the sortam file given above would result in the flow area to be used for the conversion of the first column of mass flow rates to velocity, defined by the z-direction flow area on the top face of the following i, j, k cells:

i	j	k
9	99	
10	99	
9	10	9
10	10	9

The density required to convert the mass flow rate to a velocity will be obtained from the cell identified in the **vbc** input as the last i, j, k cell in the **vbc** definition. For the example given above, the last cell in the **vbc** definition is at i=10, j=10, k=9. The gas species convected into the GASFLOW mesh by the sortam flow rate boundary conditions will be of the normal donor cell densities and species. Therefore, the

locations where the **vbc** velocities are convecting from, must be defined with the appropriate **gasdef**, **subsodef**, and **zeroddef** (See Section 5.2.3).

6 DEFINITION OF SOLID HEAT STRUCTURES

In Section 3.3, we discussed how to define solid walls and obstacles in the computational mesh (with the variables **walls** and **mobs** in NAMELIST group **xput**) to restrict the fluid flow path. Our convention is that within the mesh, walls are surfaces and obstacles are volumes which the fluid is not allowed to penetrate. If heat transfer is invoked (by setting the variable **ihflag** = 1 in NAMELIST group **rheat**), then all the defined walls and obstacles, as well as all the closed computational boundaries, will exchange heat with the fluid cells across the solid-fluid interfaces. A time-dependent heat-conduction equation is solved for each solid structure, with an implicit scheme (backward Euler **teta** = 1.0) or a semi-implicit scheme (Crank Nicholson **teta** = 0.5). As an approximation, which greatly improves computational efficiency and speed, we assume that heat conduction is one-dimensional, i. e., heat conducts only in a direction perpendicular to the interface between the solid and fluid. (In other words, if three orthogonal faces of an obstacle are exposed to fluid, then the code solves a 1D heat-conduction equation for each of the directions independently.) Another simplification is that the solid properties (conductivity, density, and heat capacity) have negligible dependence on temperature changes.

For the purpose of heat-conduction calculations, we distinguish the solid surfaces where energy exchange with the fluid occurs into two types: wall heat structures and slab heat structures, depending on the depth of solid material behind the surfaces. In addition, we can include so-called distributed heat sinks, which are planar walls of a given volume thickness and material. These sinks do not occupy any fluid volume; one can think of them as having an area per unit flow volume. They are exposed to the fluid on the negative side of the sink. The inside of the sink (the positive side of the sink structure) can be simulated with various boundary conditions that will be explained later.

6.1 Wall/Slab Heat Structures

A solid surface is a *wall* surface if the depth of solid material behind it is *thin* and the other side of the solid is also exposed to fluid. In general a wall heat structure is two-sided, with its temperature profile determined by the adjacent fluid cell temperatures on both sides and by its heat capacity and conductivity. There are two cases in which we have wall heat structures:

1. All impenetrable surfaces defined by **walls** in NAMELIST group **xput** will be considered wall heat structures, because by definition, these are infinitely thin surfaces between adjacent fluid cells. (However, as discussed later, these surfaces will be assigned some effective thickness for the heat-conduction calculation.)

2. For obstacle mesh cells (defined by **mob**s in NAMELIST group **xput**) surrounded by fluid cells on opposite sides, whether the heat structure type is a wall depends on the thickness between the two opposite sides which are exposed to fluid. If the thickness is smaller than **slabthk**, an input variable in NAMELIST group **rheat**, then the code treats the opposing surfaces as the two sides of a wall heat structure. If the thickness is greater than or equal to **slabthk**, then each of the opposing surfaces will be treated as a slab surface. Figure 6-1 illustrates some possibilities for obstacle surfaces treated either as belonging to a wall or to a slab heat structure.

A solid surface belongs to a *slab* heat structure if the solid material behind it is thicker than the **slabthk** value input by the user. In the default option, a slab is considered infinitely thick so that within the problem time scale, the heat or temperature wave due to exchange with the fluid never penetrates deep enough to affect the temperature profile near its back side. Therefore, if its backside is also exposed to fluid, then the backside surface will be treated as belonging to a separate slab heat structure, and the temperature distribution within each slab will only be affected by the temperature of the fluid in contact with its front side. The initial option has been extended to also define slabs with boundary conditions on the back side. This will be discussed in a later part. There are two cases where we have slab heat structures:

1. All boundaries of the computational domain not open to flow will be treated as slabs (see Figure 6-1), if a material number is specified for the boundary material (**matbdy** > 0). If **matbdy** is set to zero, the boundary of the computational domain is adiabatic.

2. For obstacles (defined by **mob**s and associated with a material number > 0), each surface exposed to fluid will be treated as an independent slab surface if the depth behind the surface is greater or equal to the input variable **slabthk**. If the thickness of the obstacle as defined by **mob**s is less than **slabthk**, then it is treated as a wall. Refer to Figure 6-1 for some examples.

Figure 6-1. Illustration of an obstacle surface belonging to a wall heat structure or to a slab heat structure, depending on the value of **slabthk**. Included are nonflow boundaries (of the mesh) treated as slab surfaces. Also shown are walls from wall statements.

Walls can overlap slabs and a slab can be replaced by a wall at these locations with the boundary conditions defined for the **walldf** overlaps the slab.

Note that the user does not directly define obstacles as slab or wall heat structures. The code automatically determines the heat structure type of surfaces corresponding to closed computational boundaries, which are generally treated as slabs, and the heat structure type of the solid structures defined by **walls** and **mobs**. The user provides the thickness of the slab via the input variable **slabthk**.

6.2 Configuration Control for Slabs and Walls

There are situations where the user wants more control over the mesh boundaries and either does not want to apply adiabatic conditions or does not want the same material on the entire boundary. One solution is to put walls on top of the boundary cells. This procedure can also be applied to obstacles that are located inside the mesh where one wants better control over the conditions on material with the same heat capacity on all all-sides of the obstacle or boundary cells. Refined control can be obtained by use of the **slabdef** statements (see Section 6.6), which allow the user to specify different

conditions on each side of obstacles and boundaries but with a tight coupling so that all slabs in each of the x -, y -, z - planes have the same conditions. Another approach that gives full flexibility is to apply the following convention implemented in the code: any wall that is put on top of an obstacle (no matter how thick that obstacle is and no matter whether the obstacle could become a wall itself) deactivates the slab structure on this side and replaces it with a wall structure for which properties and boundary conditions can be selected with a larger degree of freedom. The implemented scheme for the definition of slabs and walls is better explained in Figure 6-2.

The heat-conducting structures are defined in subroutine `setheat`. Three loops, one for the y - z plane, one for the x - z plane, and one for the x - y plane (planes 1, 2, and 3), examine each computational cell from three directions. If an obstacle cell starts next to a fluid cell, the other side, where the fluid cell is found next to the obstacle cell, is searched, and the thickness of the obstacle is defined. When a material number 0 is assigned to this obstacle, it is not counted as a heat-conducting structure. This also holds for boundary cells where the material number is set by `matbdy`. If a material number > 0 is assigned, the code checks whether the obstacle thickness d exceeds the input value `slabthk`. Boundary cells are automatically assigned the input slab thickness `slabthk` or the plane-dependent input values from the `slabdef` statement. Prior to assigning each side of the obstacle in the searched plane its own slab index `islab`, the code checks whether a wall statement has been made before that covers the negative (west, south, bottom) or the positive side (east, north, top) of the obstacle (the negative side of the structure is where the x -, y -, or z -coordinate have their origin). If in addition, a wall has been input for this particular side, this wall statement and the boundary conditions associated with it control the thermal behavior on this side of the obstacle. For a wall that is located on top of the positive side of the obstacle (see Figure 6-2), one can input the boundary conditions on the negative wall side (temperature time history, heat flux, adiabatic, etc.), which bypasses the slab boundary conditions that are globally set to be the same for all slabs. The remaining sides of the obstacle are simulated independently from this.

Figure 6-2. Definition of heat-conducting structures for obstacles and boundary cells

If the obstacle thickness is less than **slabthk**, the code checks whether any side of the obstacle in this plane is already defined by an input wall statement. If so, the input wall statement with its boundary conditions controls the definition of the obstacle, where it is already defined by a wall statement. The other side of the thin obstacle is disregarded for heat exchange in this case. If the other side of the thin obstacle in this particular plane also participates in the heat exchange, it must be defined by an additional wall statement. If no input wall statement exists, the thin obstacle is identified with an additional wall and treated like a normal wall with fluid conditions on both sides.

In the example in Figure 6-3 for a box, $d > \text{slabthk}$ causes the definition of six independent slabs on all sides of the box. The number of slabs is reduced to five if a wall statement defines a wall on one side of the box. For the thin slab, one wall is defined from the west to the east side of the obstacle; the remaining four sides of this obstacle are treated as slabs. If a wall statement for the east side defines this side of the structure, then the west side is not simulated as a heat-conducting structure unless it is defined by an additional wall statement. Heat conduction into the remaining four sides is again simulated with four independent slabs.

Figure 6-3. Example of the definition of slabs and walls on a dice obstacle and on a thin obstacle with thickness d in the x -direction

6.3 Heat Conduction in Wall Heat Structures

Regardless of whether a heat structure is a slab or wall, the conduction calculation requires some information about physical properties and a spatial dimension. The spatial dimensions of obstacle cells are defined by the mesh. However, for surfaces defined by **walls**, the user will have to input an effective physical thickness, even though mathematical surfaces between adjacent fluid cells in the mesh have no thickness. Furthermore, the user must define the material in the wall. These definitions are accomplished through the 8th element of the **walls** array in NAMELIST group **xput** and through the **wallddef** array in NAMELIST group **rheat**:

walls(8,*)	Integer to identify the type of wall through the wallddef array that stores the material identification number and thickness for each wall type (see Section 3.3.1).
wallddef(1,*)	Material identification number. Options are 1 concrete, 2 steel, 3 superconducting material. > 3 Use table wallddef(1,*) for element by element input of thermal conductivity and ρC_p . See also wltsink and rcptsink .
wallddef(2,*)	Thickness of wall (cm).
wallddef(3,*)	Sets boundary condition (BC) for first fluid/wall surface (BC#1). wallddef(3,*) = 0.0 implies a fluid-wall heat exchange (default); wallddef(3,*) > 0.0 implies a constant wall temperature boundary condition with $T = \text{wallddef}(3,*)$; and wallddef(3,*) < 0.0 implies an adiabatic wall boundary condition.
wallddef(4,*)	Sets BC#2 for last fluid/wall surface. wallddef(4,*) = 0.0 implies a fluid-wall heat exchange (default); wallddef(4,*) > 0.0 implies a constant wall temperature boundary condition with $T = \text{wallddef}(4,*)$; and wallddef(4,*) < 0.0 implies an adiabatic wall boundary condition.
wallddef(5,*)	δx for the first node in the wall. wallddef(5,*) = 0.0 implies a uniform mesh spacing for heat-conduction nodes, and wallddef(5,*) > 0.0 implies a variable mesh spacing for heat-conduction nodes, where wallddef(5,*) = δx for the surface heat-conduction node at both surfaces of the wall.
wallddef(6,*)	Fraction of wall area from mesh surface that is used for heat transfer.
wallddef(7,*)	Flag for further specification of BC on negative side of wall: = 0 no further modification;

> 0 gives table number from **surftab** that specifies time-dependent surface temperature, with the initial temperature at $t=0$ taken from **walldf(3,*)**;
 $= -1$ applies heat flux **walldf(9,*)** and/or heat transfer with coefficient **walldf(10,*)** and applies fluid temperature **walldf(3,*)** on negative side.
walldf(8,*) Flag for further specification of BC on positive side of wall:
 $= 0$ no further modification;
 > 0 gives table number from **surftab** that specifies time-dependent surface temperature, with the initial temperature at $t=0$ taken from **walldf(4,*)**;
 $= -1$ applies heat flux from **walldf(9,*)** and/or heat transfer with coefficient **walldf(10,*)** and applies fluid temperature **walldf(4,*)** on positive side.
walldf(9,*) Heat flux [erg/cm²-s] applied as BC by **walldf(7,*)** or **walldf(8,*)**. Positive flux means add heat to the wall (i.e., condensation), which is the same convention as in the fluid wall condensation/vaporization heat flux from GASFLOW.
walldf(10,*) Heat-transfer coefficient [erg/cm²-s- K] applied as BC by **walldf(7,*)** or **walldf(8,*)**.

GASFLOW offers a choice among three solid conducting materials with the following properties:

Material Number	Material Name	Density (g/cm ³)	Specific Heat Capacity (ergs/g-K)	Thermal Conductivity (ergs/s-cm-K)
0	no structure	disregard structure in heat conduction		
1	concrete	2.40	1.004×10^7	2.0×10^5
2	steel	7.85	4.904×10^6	5.0×10^6
3	superconductor	10.0	1.000×10^9	1.0×10^{20}

Because the code will calculate the obstacle thicknesses, the user only has to input the material which the obstacle is made of. This can be done via the 8th element of the **mobs** array (see Section 3.3.2):

mobs(8,*) Material identification number. Options are
 0 disregard heat conduction,
 1 concrete,
 2 steel, and
 3 superconducting material.

If a zero value is specified for the material number, this particular structure is not included in the heat-conduction and heat-transfer simulation; it is counted neither as a slab nor a wall for heat transfer but only serves as a wall or an obstacle for the flow simulation. When heat transfer is turned on (**ihflag** = 1), the zero material number allows the user to selectively leave out certain structures from heat exchange (for instance walls from rupture disks which cannot be easily removed otherwise) but still consider those with nonzero material numbers.

The code also allows the specification of additional heat-conducting materials, which can be defined explicitly in the input. Every material number >3 requires a table input for the thermal conductivity and the product of ρC_p . This table input must then be specified for walls, slabs, and sinks (see definition below) according to the input quantities **nhteslab**, **nhtesink**, and **nhthewall**, which are heat-conducting elements in slabs, walls, and sinks, respectively. This table input allows the simulation of composite structures (i.e., a liner on top of a concrete structure) in which thermal conductivities and heat capacities can vary from node to node. Note that these tables must always be specified for all structure types even though some material numbers may be applied only in a wall. The extended options above **walldf(6,*)** require the specification of the following additional data in block rheat:

ntotmat	Total number of structure materials. (Default =3.) Read new properties only for materials 3+ imat to ntotmat because components 1 to 3 are automatically defined. ntotmat must be <7.
wltabslab(*,imat)	Thermal conductivity for material imat for each of the nhteslab elements of the slab structure. imat must be >3. Materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials. (Maximum: 29 elements per structure.)
wltabwall(*,imat)	Thermal conductivity for material imat for each of the nhthewall elements of the wall structure, imat must be >3. Materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials. (Maximum: 29 elements per structure.)
wltabsink(*,imat)	Thermal conductivity for material imat for each of the nhtesink elements of the sink structure. imat must be >3. Materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials. (Maximum: 29 elements per structure.)
rcptabslab(*,imat)	ρC_p for material imat for each of the nhteslab elements of the slab structure. imat must be >3. Materials 1 to 3 are automatically filled with constant values for concrete, steel,

	and superconducting materials. (Maximum: 29 elements per structure.)
rcptabwall(*,imat)	ρC_p for material imat for each of the nhtewall elements of the wall structure. imat must be >3. Materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials. (Maximum: 29 elements per structure.)
rcptabsink(*,imat)	ρC_p for material imat for each of the nhtesink elements of the sink structure. imat must be >3. Materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials. (Maximum: 29 elements per structure.)
surftab(2,j,i)	Pair of time [s] and temperature [K] at time t_j for temperature table i . Maximum number of points per table is 50 and the maximum number of tables is 30. (The problem time must never exceed the maximum of the table time.)
twall0	Initial wall surface temperature on the wall side for which walldef does not define the surface temperature (walldef(3,*) and/or walldef(4,*) = 0). If twall0 < 0, then the surface temperature from the adjacent fluid node is applied on the undefined sides of the wall.

Note that one must not input the fluid conditions and/or heat flux for the side of the wall. This would overspecify the problem and the code would automatically stop with a message. It is possible however to specify the temperature on both sides of the wall, thus defining a thermal boundary condition for fluid heating or cooling. Defining a wall with specified fluid conditions on one side or making the boundary condition adiabatic on one side makes sense only if the input conditions apply to a side of the wall that does not face a fluid. This could be the inner side of a wall on top of an obstacle or a wall side facing the boundary of the computational mesh. The code is currently set up in such a way that any wall on top of a slab replaces that particular slab, which allows quite a flexible definition of the structure boundary conditions.

The initial temperature profile across the wall is evaluated from the surface temperatures on the two sides of the wall, taking **twall0**, the temperature of the adjacent GASFLOW fluid node, or the input surface temperature [**walldef(3,*)** and/or **walldef(4,*)**], whichever applies. If one side of the wall is adiabatic, the code initiates a flat temperature profile across the wall with the wall surface temperature on the fluid side. The steady-state profile includes the effect of varying thermal conductivities. A flat steady-state temperature profile under nonadiabatic conditions on any wall side requires input so that either **twall0** > 0 or **twall0** < 0.0

with a uniform fluid temperature on both sides of the wall, leaving the input values `walldf(3,*)` and `walldf(4,*)` zero.

When specifying a heat flux and/or heat-transfer coefficient with a fluid temperature on one side of the wall, the code determines the wall surface temperature on this side from the input data and calculates a steady-state heat flux through the wall from this data. The surface temperature on the other (fluid) side is applied according to the fluid or `twall0` specification. The surface temperature on the fluid side and the calculated steady-state heat flux are used to calculate the steady-state temperature profile across the wall again, accounting for varying thermal conductivities from layered structures, if the wall material chosen is greater than 3.

When cylindrical coordinates are chosen (`cyl = 1`), heat conduction through walls in a radial direction (walls in plane 1) is calculated in cylindrical coordinates under both steady-state and transient conditions.

Examples:

1. Wall of 10 cm thickness with 29 heat-conducting elements, fluid on both sides, and composite material. (All but the 15th structure element is made of concrete; the 15th structure element is made of steel.) Use backward Euler scheme (`teta = 1.0 = default`). Use of dynamic mesh expansion with a small surface node of 0.01 cm on both sides of the wall (example shown in Figure 6-4 for 10 elements only).

Sinks or slabs positive side adiabatic

Sinks or walls positive side nonadiabatic

Figure 6-4. Noding scheme for 10 heat-conducting elements with small surface nodes and internal mesh expansion in sinks, slabs, and walls. Element number 6, counting from the negative side, is shaded. Input defines thickness of surface nodes.

```

(Test1 input) .
gasdef(1,1) = 1 , 04, 1 , 02, 1 , 04, 1 ,
              1.0e6, 400., 2, 0., 0.,
              'air', 1.,
walls(1,1) = 3,3, 1,2, 1,2, 1, 1,$end
$rrheat
teta          = 1.0 ; default
ihtflag       = 1,
ntotmat       = 4,
nhteslab      = 20,
nhtesink      = 10,
nhtewall      = 29,
twall0        = -1,

walldef(1,1)  = 4, 10., 0.0, 300.0, 0.01,
wltabwall(1,4) = 14*2.e+5, 5.e+6, 14*2.e+5,
wltabslab(1,4) = 10*2.e+5, 5.e+6, 9*2.e+5,
wltabsink(1,4) = 5*2.e+5, 5.e+6, 4*2.e+5,
rcptabwall(1,4)=14*6.25924e+7, 3.84964e+7, 14*6.25924e+7,
rcptabslab(1,4)=10*6.25924e+7, 3.84964e+7, 9*6.25924e+7,
rcptabsink(1,4)=5*6.25924e+7, 3.84964e+7, 4*6.25924e+7,
$end

```

In Figure 6-5, we present the steady-state temperature distribution for this example.

2. Concrete wall on top of an obstacle with specified heat flux of 10^{10} erg/cm²-s on the obstacle side of the wall and fluid conditions on the fluid side. Solve with Crank Nicholson scheme (**teta** = 0.5).

```

$xrput
.
(Test2 input) .
gasdef(1,1) = 1 , 04, 1 , 02, 1 , 04, 1 ,
              1.0e6,400., 2, 0., 0.,
              'air', 1.,
mobs(1,1)   = 2,3, 1,2, 2,3, 1, 1,
walls(1,1)  = 2,2, 1,2, 2,3, 1, 1,
.
$end

$rrheat

```

Figure 6-5. Steady-state solution for Example 1, the 10 cm thick composite structure made up of a sandwich of concrete, steel, and concrete

```

.
.
ihtflag      = 1,
teta         = 0.5, ; Crank Nicholson
nhtewall     = 29,
twall0       = -1.,

walldef(1,1) = 1, 10., 0., 1., 0.01, 1.0, 0, -1, 1.e+10,
.
.

```

\$end

In Example 2, a steady-state temperature profile is calculated across the wall with the surface temperature 400 K on the fluid side. The steady-state wall surface temperature on the obstacle side is determined from the input heat flux. Note that the fluid conditions on the obstacle side must be specified by a number greater than zero if the heat flux is to continue to be applied during the transient also. If the heat flux is only applied to set up a steady-state temperature profile, one can set **walldef(4,1)** to zero.

In this case, a steady-state profile is set up, but the obstacle side of the wall is defaulted to adiabatic conditions.

3. If one wants to start from a flat profile of 400 K across the wall before applying the input heat flux, one can use the **walldf** statement only, with the boundary temperature on the obstacle side specified to be equal to the fluid temperature in the startup and with the heat flux specified only in the restart.

```
(Test3 start)
  walldf(1,1) = 1, 10., 0., 400., 0.01,
(Test3 restart)
  walldf(1,1) = 1, 10., 0., 400., 0.01,1.,0,-1,1.e+10,
```

We have applied this procedure in Example 3 to check the transient heat-conduction solution from GASFLOW with analytical results for a 10 cm steel and concrete wall. Figure 6-6 compares the predicted surface temperature vs. time development on the obstacle side with analytical predictions. Results from a calculation of the same transient with 10 heat-conduction nodes are also displayed in Figure 6-6, which documents that this discretization is not sufficient for this strong heat flux.

4. The same wall but with input of a temperature transient from Table 3-1 of the **surftab** array for the surface of the positive side, simulated with the input:

\$xput

```
.
(Test4 input) .
twfin          = 1.,
gasdef(1,1)    = 1 , 04,  1 , 02,  1 , 04,  1 ,
                1.0e6,400., 2, 0., 0.,
                'air', 1.,
mobs(1,1)      = 2,3,  1,2,  2,3,  1, 1,
walls(1,1)     = 2,2,  1,2,  2,3,  1, 1,
```

\$end

\$rheat

```
.
.
ihtflag        = 1,
teta           = 0.5,
```

Figure 6-6. GASFLOW calculated the wall surface temperature on the positive side and calculated an analytical solution for 1D heat conduction through a 10 cm wall of steel or concrete starting at 400 K isothermal condition, applying a constant surface temperature of 400 K on the negative side and a constant heat flux of 1000 W/cm² on the positive side. Results are for 29 and 10 heat-conduction elements and internal mesh expansion with equal surface elements of 0.01cm. No noticeable difference between Crank Nicholson (**teta** = 0.5) and backward Euler (**teta** = 01.0).

```

nhtewall      = 29,
twall0        = -1.,
walldef(1,1)  = 1, 10., 0., 400., 0.01, 0.9, 0, 2,
surftab(1,1,2) = 0., 500.,
               1., 800.,

```

\$end

In this example, the wall surface temperature on the positive side increases linearly from 500 K to 800 K within 1 s. The steady-state profile and the steady-state surface temperature on the obstacle side have been set to 400 K. The table values are only applied during the transient. Thus this input assumes that the surface temperature will jump to the initial value of 500 K at the beginning of the transient for a flat internal

profile of 400 K from the steady-state initialization. Note that the problem time **twfin** must be within the maximum time specified in **surftab**. We have additionally decreased the wall area for the heat transfer by a factor of 0.9 relative to the wall area defined by the mesh surface.

6.4 Heat Conduction in Sink Heat Structures

In some practical problems where the computational mesh is not fine enough to represent the details of all internal structures, it is desirable to have the capability of modeling the heat-transfer effects of these "subgrid" structures. That is, we need a method of modeling heat transfer between the fluid in a computational cell and the structures embedded within the cell. We accomplish this by defining a third type of heat structure, which we call distributed heat sinks.

Sinks are heat structures defined by the user which are assumed to be distributed within the fluid cells. Each sink is characterized by the simple model illustrated in Figure 6-7. Similar to the other heat structure types, 1D heat-conduction is calculated across the sink structure thickness. Both sides of a sink heat structure are exposed to the same fluid cell, and it is assumed that the structure temperature profile is symmetric about the centerline, so that only conduction in half the structure thickness needs to be calculated. Definition of sink heat structures is done with the input array **sinkdef** in NAMELIST group **rheat**:

sinkdef(1,*)	Beginning i mesh index (cell face number).
sinkdef(2,*)	Ending i mesh index (cell face number).
sinkdef(3,*)	Beginning j mesh index (cell face number).
sinkdef(4,*)	Ending j mesh index (cell face number).
sinkdef(5,*)	Beginning k mesh index (cell face number).
sinkdef(6,*)	Ending k mesh index (cell face number).
sinkdef(7,*)	Block number .
sinkdef(8,*)	Material identification number. Options are 1 concrete, 2 steel, 3 superconducting material. > 3 Use sinkdef(8,*) as the table number for element by element input of thermal conductivity and ρC_p . See also wltabsink and rcptabsink .
sinkdef(9,*)	Total material volume volsink (cm ³).
sinkdef(10,*)	Average material thickness avgthick (cm).
sinkdef(11,*)	Sets BC for sink/fluid surface (BC#1). sinkdef(11,*) = 0.0 implies that the BC#1 will be sink-fluid heat exchange

Figure 6-7. Schematic representation of the distributed heat sink used in GASFLOW

(default). **sinkdef(11,*)** > 0.0 implies a wall temperature boundary condition of $T = \text{sinkdef}(11,*)$ on the fluid side.

sinkdef(12,*)

Sets the sink centerline BC (BC#2). **sinkdef(12,*)** = -1.0 implies an adiabatic BC (default) will be applied at the sink centerline. **sinkdef(12,*)** > 0.0 implies a temperature boundary condition will be applied at the sink centerline of $T = \text{sinkdef}(12,*)$.

sinkdef(13,*)

δx for the first node in the sink. **sinkdef(13,*)** = 0.0 implies uniform mesh spacing for heat-conduction nodes. **sinkdef(13,*)** > 0.0 implies variable mesh spacing for heat-conduction nodes, with **sinkdef(13,*)** = δx of first heat-conduction node if internal BC is adiabatic and

sinkdef(13,*) = δx on outer and inner sink surface nodes if internal BC is nonadiabatic.

sinkdef(14,*) Flag for further specification of BC on negative side:
 = 0 no further modifications (default);
 > 0 gives table number from **surftab** that specifies time-dependent surface temperature. The initial temperature at $t=0$ is taken from **sinkdef(11,*)**.

sinkdef(15,*) Flag for further specification of BC on positive side (centerline) of slab:
 = 0 no further modification (default);
 > 0 gives table number from **surftab** that specifies time-dependent surface temperature, with the initial temperature at $t=0$ taken from **sinkdef(12,*)**;
 = -1 applies heat flux from **sinkdef(16,*)** and/or heat transfer with coefficient **sinkdef(17,*)** and fluid temperature **sinkdef(12,*)** on positive side (positive heat flux means add energy to the sink).

sinkdef(16,*) Heat flux [erg/cm²-s] applied as BC by **sinkdef(15,*)**.

sinkdef(17,*) Heat-transfer coefficient [erg/cm²-s-K] applied as BC by **sinkdef(15,*)**.

Note that each sink definition can cover a fluid region (specified by the starting and ending i, j, and k mesh indices) consisting of multiple fluid cells. If such is the case, then the code will distribute the sink material to each fluid cell according to the cell volume, i. e., a fluid cell having twice the volume of another one will get twice as much sink material. According to our model, depicted in Figure 6-6, specifying the volume and thickness of the sink material will also give the surface area through which heat exchange with the fluid occurs.

The following relation exists between the sink volume volsink, the input thickness avgthick, and the total surface to which the fluid is exposed:

$$\begin{aligned} \text{thickness} &= \text{avgthick}/2 \\ \text{area} &= \text{volsink} / \text{thickness} \end{aligned}$$

thickness denotes the actual thickness of the sink structure in the heat-conduction solution.

Example. In the example given below, the first sink heat structure is of material-type steel with positive (central) BC defined by a surface heat-transfer coefficient and a fluid temperature and applies 10 elements for heat conduction and a small surface node of 0.01 cm on both sides. The second sink heat structure applies surface temperature table 1 for temperature transient on the positive (central side).

\$xput

```
(Test5 input) .  
twfin          = 100.,
```

\$end

\$rheat

```
ihflag         = 1,  
nhtesink       = 10,  
tsink0         = -1.,  
  
sinkdef(1,1)   = 2, 4, 1, 2, 1, 2, 1, 2,  
                8000., 10., 0.0,  
                873., 0.01, 0, -1, 0., 10000.,  
  
sinkdef(1,2)   = 2, 3, 1, 2, 2, 3, 1, 2, 4000., 10.,  
                0.0, 400., 0.01, 0, 1,  
  
sinkdef(1,3)   = 2, 4, 1, 2, 2, 3, 1, 2,  
                4000., 10.,  
                0.0, -1.0, 0.01,  
  
surftab(1,1,1) = 0.0, 500.,  
                1., 800.,  
  
$end
```

Regarding distributed heat sinks, the first **sinkdef** assigns sinks to two fluid cells with (i, j, k) indices of (3, 2, 2) and (4, 2, 2). (Refer to Figure 3-1 for the cell numbering convention used in GASFLOW.) The sink material is steel because **sinkdef(8,1) = 2**. If the two cells have the same size, then the distributed heat sink in each cell will have a volume of 4000 cm³, a thickness of 5 cm, and a total surface area 800 cm². The second **sinkdef** specifies only one fluid cell, (3, 2, 3), and the heat sink in that cell also has a volume of 4000 cm³ and a thickness of 5 cm. The third sink is adiabatic at the sink centerline. In this case, the dynamic mesh expansion only defines the small input node of 0.01cm on the fluid side and monotonically increases the mesh size to the inner boundary at 5 cm thickness (see also Figure 6-4). The third sink partially overlaps the fluid node of the second sink. Because sinks do not directly interfere with each other, one can specify more than one sink statement to apply to the same fluid node. All sinks have the surface temperatures on their negative sides (fluid sides) specified by the fluid

temperature because no **tsink0** > 0 has been specified in the input. The surface temperature at the positive (inner) sink side of the first **sinkdef** statement is calculated with the temperature from an outside fluid and the heat-transfer coefficient of 10,000 ergs/cm**2 K specified in the input. The steady-state profile is set up with a linear increase from the temperature on the fluid side to the calculated surface temperature on the inner side of the sink.

6.5 Other Heat Structure Input

The heat-conduction equations are solved by finite difference. The finite differences are applied to a one-dimensional mesh that is either planar or cylindrical depending upon the value of **cy1** (**cy1** = 0 implies planar conduction and **cy1** = 1 implies cylindrical conduction in the i-direction and planar conduction in the j- and k-directions). The user can specify the number of nodes used for each type of heat structure. Note that all heat structures of the same type will have the same number of nodes. The input variables for this purpose are in NAMELIST group **rheat** and are explained below:

nhtesink Number of discretized elements to be used for calculating heat conduction in a sink heat structure.
Default = 2. (Max < 30.)

nhtewall Number of discretized elements to be used for calculating heat conduction in a wall heat structure.
Default = 2. (Max < 30.)

nhteslab Number of discretized elements to be used for calculating heat conduction in a slab heat structure.
Default = 2. (Max < 30.)

In GASFLOW, all solid heat structures can be defined to have a uniform temperature distribution in the beginning of a problem. These initial heat structure temperatures can be specified with the following input variables in NAMELIST group **rheat**:

tsink0 Initial temperature in sink heat structures (K).
Default = 300.

twall0 Initial temperature in wall heat structures (K).
Default = 300.

tslab0 Initial temperature in slab heat structures (K).
Default = 300.

If any initial temperature is set negative, then the corresponding heat structures are assumed to be in thermal equilibrium with the contacting fluid cells. For example, if **tsink0** = -1 and the initial fluid temperature is 298 K, then a sink heat structure will also have a temperature of 298 K in the beginning of the calculation. This default initialization of the structure temperatures can change when the boundary conditions specify different temperatures on both sides of the structures or when heat transfer from external fluid conditions or heat fluxes are specified on one side of the structure at the onset of the analysis.

6.6 Heat Conduction in Slab Heat Structures (Boundary Cells)

The slab temperature boundary conditions for all blocks and all boundary cells can either be input as a single number (see below) or as the **slabdef** array which allows a slab's thickness, initial temperatures, boundary conditions, and material type to be dependent upon the block number and upon whether the boundary cell is on the east, west, north, south, top, or bottom boundary. **tslabbc** and **slabdef** are described below:

Variable	Definition
tslabbc	Inner slab temperature boundary condition (default -9.123) for all blocks and boundaries (uniform). Can be overridden individually for each of the 6 slab planes using slabdef . tslabbc < 0 implies adiabatic BC. tslabbc > 0.0 implies fixed temperature BC on the inside of the slab, with $T(bc) = \text{tslabbc}$.
tslab0	Initial slab temperature (default 300 K) can be overridden using slabdef . If tslab0 < 0, use temperature of the adjacent fluid node to define initial slab temperature. A flat temperature profile with tslab0 between the fluid side and the inner slab side is initiated. If tslabbc > 0 and islablin = 1, a linear steady-state profile from the fluid surface to tslabbc at the inner surface of the slab is set up. If tslabbc > 0, islablin = 1, and hslablin > 0, a linear profile between the slab surface temperature on the fluid side and the inner slab temperature tslabbc is set up. The slab surface temperature on the fluid side is then calculated from steady-state fluid to slab heat transfer on the fluid side with tslab0 being the fluid temperature and hslablin the heat-transfer coefficient on the fluid side of the slab. tslabbc is

slabdef(25,*)	the surface temperature on the inside (positive side) of the slab. Sets tslab0 , tslabbc , and slabthk for specific boundaries and blocks. Conditions on the east boundary generally apply to the slabs on the west side of all obstacles. Correspondingly, the west side boundary conditions are applied to all slabs on the east side of obstacles, and so on similarly for the other four planes.
slabdef(1,n)	Block number.
slabdef(2,n)	Thickness of slabs on east boundary.
slabdef(3,n)	Thickness of slabs on west boundary.
slabdef(4,n)	Thickness of slabs on north boundary.
slabdef(5,n)	Thickness of slabs on south boundary.
slabdef(6,n)	Thickness of slabs on top boundary.
slabdef(7,n)	Thickness of slabs on bottom boundary.
slabdef(8,n)	Initial temperature for slabs on east boundary (applied instead of tslab0).
slabdef(9,n)	Initial temperature for slabs on west boundary(applied instead of tslab0).
slabdef(10,n)	Initial temperature for slabs on north boundary(applied instead of tslab0).
slabdef(11,n)	Initial temperature for slabs on south boundary(applied instead of tslab0).
slabdef(12,n)	Initial temperature for slabs on top boundary(applied instead of tslab0).
slabdef(13,n)	Initial temperature for slabs on bottom boundary(applied instead of tslab0).
slabdef(14,n)	Temperature BC for slabs on east boundary(applied instead of tslabbc).
slabdef(15,n)	Temperature BC for slabs on west boundary(applied instead of tslabbc).
slabdef(16,n)	Temperature BC for slabs on north boundary(applied instead of tslabbc).
slabdef(17,n)	Temperature BC for slabs on south boundary(applied instead of tslabbc).
slabdef(18,n)	Temperature BC for slabs on top boundary(applied instead of tslabbc).
slabdef(19,n)	Temperature BC for slabs on bottom boundary(applied instead of tslabbc).
slabdef(20,n)	Material type for slabs on east boundary(applied instead of matbdy).

slabdef(21,n)	Material type for slabs on west boundary(applied instead of matbdy).
slabdef(22,n)	Material type for slabs on north boundary(applied instead of matbdy).
slabdef(23,n)	Material type for slabs on south boundary (applied instead of matbdy) .
slabdef(24,n)	Material type for slabs on top boundary(applied instead of matbdy).
slabdef(25,n)	Material type for slabs on bottom boundary(applied instead of matbdy).
islablin	Flag to set an approximate linear initial temperature profile in each slab cell based on temperature of slab BC#2 and fluid cell contiguous with BC#1 (default is 0, no linear initial temperature profile).
hslablin	Heat-transfer coefficient on slab BC#1. Used in evaluating the linear slab cell temperature profile (default is 1000).
dxslabc	dxslabc = 0.0 results in uniform mesh spacing for all concrete slabs. dxslabc $\neq$ 0 results in a variable mesh spacing with for all concrete slabs. The first δx for each concrete slab will be dxslabc (default is zero).
dxslabs	dxslabs = 0.0 results in uniform mesh spacing for all nonconcrete slabs. dxslabs $\neq$ 0 results in a variable mesh spacing with for all nonconcrete slabs. The first δx for each nonconcrete slab will be dxslabs (default is 0).
matbdy	Material type (uniform) for the mesh boundary slabs in all blocks. Can be overridden for specific blocks and boundaries using slabdef . matbdy = 0 results in no boundary slabs.

A material number > 0 specified for a boundary of the computational mesh implies that this particular boundary side is simulated as a slab of the material represented by the specified number. If nothing more is specified, a flat temperature profile with **tslab0** (default 300 K) is initiated and the inner side of the slab is maintained adiabatic. If **tslab0** < 0, a flat profile with the adjacent fluid temperature is specified and an adiabatic BC is applied at the inside. If a value greater than zero is input for **tslabbc**, the BC on the inner (positive) surface of the slab uses a constant surface temperature **tslabbc**. The analysis still starts from a flat profile with **tslab0**, which can cause a sudden jump at the inner slab surface if **tslabbc** is specified different from **tslab0**. Only if **islablin** is set to 1 is a linear steady-state profile set up between the slab surface temperature **tslab0** on the fluid side and the inner slab surface temperature

tslabbc. If **hslablin** is also specified, the slab surface temperature on the fluid side is calculated with a steady-state fluid slab heat transfer assuming that **tslab0** is the fluid temperature on the fluid side and **tslabbc** is the surface temperature on the inner side of the slab.

The **slabdef** statements allow different definitions for each of the six boundaries of the computational mesh of the block they are defined for. But they apply at the same time also to the obstacle planes inside. The conditions for the computational boundary on the east side define the west side of the boundary slab and are also used to simulate the west side of the obstacle slabs inside. In the same way, the conditions for the top boundary define the bottom side of the top boundary slabs and the bottom side of the internal obstacle slabs.

Examples:

1. Obstacle slab made of steel with specified inner temperature of 400 K and with the initial temperature taken from the adjacent fluid node. To avoid a jump in temperature at the inner surface, we set the inner slab surface temperature equal to the initial fluid temperature.

\$xput

```

      .
      (Test6 input) .
      gasdef(1,1)    = 1 , 04,  1 , 02,  1 , 04,  1 ,
                      1.0e6, 400.,  2, 0., 0.,
                      'air', 1.,

```

```

      mobs(1,1)      = 2,3, 1,2, 2,3, 1, 2,
      .

```

\$end

\$rheat

```

      .
      ihtflag        = 1,
      nhteslab       = 20,
      tslab0         = -1.,
      tslabbc        = 400.,
      slabthk        = 30.,
      .

```

\$end

2. Obstacle in block 1 made of concrete with specified inner temperature of 400 K in the west slab, 500 K in the east slab, and 600 K in the top, bottom, south, and north slab. Initial fluid temperature is 300 K. The surface node on the fluid side has a size of 0.01 cm. A steady-state profile is set up with a heat-transfer coefficient of 1000 erg/cm²-s-K on the fluid sides of the six slabs that bound the obstacle. The boundary cells are considered adiabatic (material numbers for boundaries of the mesh all set to zero).

\$meshgn

```
(Test7 input) .  
iblock      = 1,  
  
xgrid       = 0., 50., 100.0, 150.,  
ygrid       = 0., 100.0,  
zgrid       = 0., 50., 100.0, 150.,  
.  
.  
.
```

\$end

\$xput

```
gasdef(1,1)  = 1 , 04, 1 , 02, 1 , 04, 1 ,  
              1.0e6,400., 2, 0., 0.,  
              'air', 1.,  
  
mobs(1,1)    = 2,3, 1,2, 2,3, 1, 1,  
.  
.  
.
```

\$end

\$rheat

```
ihflag       = 1,  
nhteslab     = 20,  
slabdef(1,1) = 1, 6*30., 6*-1., 400., 500.,  
              4*600.,6*0,  
  
dxslabc      = 0.01,  
islablin     = 1,  
hslablin     = 1000.,  
.  
.  
.
```

slabthk = 30.,

\$end

Because the obstacle extends to the mesh boundary on the south and north side, no slabs are defined for the south and north side of this obstacle. The heat-conduction equations for this example are solved for four slabs only.

Note that the user has the option to choose different obstacle materials and different materials for each boundary of the block. The code is currently not set up, however, to specify different sizes of the surface nodes for different obstacles or to apply other than the specified boundary conditions. These are defined for each plane within one block but must be used in each obstacle at this particular plane. It is thus not possible to use different inner boundary conditions for different obstacles in the same plane. Note also that the decision whether an obstacle is considered a slab or a wall is always made based on the input parameter **slabthk** only. Therefore, this parameter must be specified together with **slabdef (2-7,1)**, which applies only to the mesh boundaries. But it has been shown before that defining walls on top of an obstacle side allows the user to vary the boundary conditions on any desired obstacle side and thus to overcome the restrictions that currently exist in the simulation of slab heat structures.

6.7 Background for Defining Steady-State Temperature Profiles

The steady-state temperature profile in the heat-conducting structures can either be flat or have an initial gradient. If one side of the structure is modeled as adiabatic, the initial temperature profile is always flat and defined from the surface temperature of the opposite side. Steady-state temperature profiles are only simulated for nonadiabatic boundary conditions on either side of the structure. Different thermal conductivities for composite layers must be accounted for in the initial profile, if a gradient is simulated. The temperature gradient across the structure is calculated from a steady-state heat flux q''_{ss} for planar walls or linear heat rating $\dot{q}_{ss}$ for cylindrical structures.

Planar wall

$$T_j = \begin{cases} T_1 & \text{for } j = 1 \\ T_{j-1} - \frac{\delta x_j}{k_j} q''_{ss} & \text{for } j = 2, NHTE \\ T_{NHTE+1} & \text{for } j = 1 \end{cases} \quad (6-1)$$

Cylindrical wall

$$T_j = \begin{cases} T_1 & \text{for } j=1 \\ T_{j-1} - \frac{\ln(x_j / x_{j-1})}{k_j} \dot{q}_{ss} & \text{for } j=2, NHTE \\ T_{NHTE+1} & \text{for } j=1 \end{cases} \quad (6-2)$$

The steady-state heat ratings and the surface temperatures depend on the selected boundary conditions.

6.7.1. Fluid Conditions from GASFLOW Define Both Sides (Applies Only to Walls)

A steady-state temperature gradient with the respective fluid temperatures T_{fe} and T_{fw} on the east and west side of the wall is implemented with consideration of composite structures. The steady-state heat flux $\dot{q}_{ss}$ through a planar wall is

$$\dot{q}_{ss} = - \frac{(T_{NHTE+1} - T_1)}{\sum_{j=2}^{NHTE+1} \frac{\delta x_j}{k_j}} \quad (6-3)$$

For cylindrical walls, a steady-state linear heat flux $\dot{q}_{ss}$ is calculated from

$$\dot{q}_{ss} = - \frac{(T_{NHTE+1} - T_1)}{\sum_{j=2}^{NHTE+1} \frac{\ln(x_j / x_{j-1})}{k_j}} \quad (6-4)$$

The surface temperatures, T_1 and T_{NHTE+1} from Equations (6-1) and (6-2), are set to the respective fluid temperatures

$$T_1 = T_{fw} \quad (6-5)$$

and

$$T_{NHTE+1} = T_{fe} \quad (6-6)$$

because there is no steady-state heat-transfer coefficient from GASFLOW available prior to the start of the analysis. The heat flux and one surface temperature are then used to determine the steady-state temperature profile from Equations (6-1) and (6-2), respectively.

6.7.2. Steady-State Fluid Conditions Input for the Negative or Positive Side of the Structure

GASFLOW allows input of some outer fluid condition (outside of the GASFLOW simulation) on one structure side by specifying an outer fluid temperature, a heat-transfer coefficient, and a heat flux. This applies to the center of a sink but is also possible on either side of a wall. If the fluid conditions are specified on the positive side (positive side implies the high number side of the heat structure or right side where increasing node numbers go from left to right), the steady-state heat flux for planar geometry is calculated from.

$$q_{ss}'' = - \frac{(T_{fe} - T_1) + q_{ce} / h_e}{\frac{1}{h_e} + \sum_{j=2}^{NHTE+1} \frac{\delta x_j}{k_j}} \quad (6-7)$$

For the same conditions, the correlation for the linear heat rate in cylindrical coordinates is

$$q_{ss}' = - \frac{(T_{fe} - T_1) + q_{ce} / h_e}{\frac{1}{x_{NHTE+1} h_e} + \sum_{j=2}^{NHTE+1} \frac{\ln(x_j / x_{j-1})}{k_j}} \quad (6-8)$$

In these correlations T_{fe} , q_{ce} , and h_e are the input values for the fluid temperature, the steady-state heat flux, and the convective heat-transfer coefficient on the east side. Note that the heat flux going into the wall on either side is assigned a positive sign in GASFLOW. This means that q_{ce} has to be entered with a negative sign because the heat flux on the east/positive/right side must have the sign of the positive x -direction also. Using the total heating rate, the structure surface temperature on the east/positive/right side is defined as

$$T_{NHTE+1} = \begin{cases} T_{fe} + \left(\frac{q_{ss}''}{h_e} + \frac{q_{ce}}{h_e} \right) & \text{for planar wall} \\ T_{fe} + \left(\frac{q_{ss}'}{x_{NHTE+1} h_e} + \frac{q_{ce}}{h_e} \right) & \text{for cylindrical wall} \end{cases} \quad (6-9)$$

When the fluid conditions are defined on the west/negative/left side of the structure, the heat flux and linear heat rating have the correlations for planar structures

$$q_{ss}'' = - \frac{(T_{NHTE+1} - T_{fw}) - q_{cw} / h_w}{\frac{1}{h_w} + \sum_{j=2}^{NHTE+1} \frac{\delta x_j}{k_j}} \quad (6-10)$$

and for cylindrical structures

$$q_{ss}' = - \frac{(T_{NHTE+1} - T_{fw}) - q_{cw} / h_w}{\frac{1}{x_1 h_c} + \sum_{j=2}^{NHTE+1} \frac{\ln(x_j / x_{j-1})}{k_j}} \quad (6-11)$$

The structure surface temperatures on the west/negative/left side are then calculated from Equations (6-10) and (6-11) as

$$T_1 = \begin{cases} T_{fw} - \left(\frac{q_{ss}''}{h_w} - \frac{q_{cw}}{h_w} \right) & \text{for planar wall} \\ T_{fw} - \left(\frac{q_{ss}'}{x_1 h_w} - \frac{q_{cw}}{h_w} \right) & \text{for cylindrical wall} \end{cases} \quad (6-12)$$

The calculated surface temperature and the heating rates for these boundary conditions are applied to calculate the steady-state temperature profile using Equation (6-1) or (6-2).

The convention for calculating steady-state profiles in slabs is slightly different. In this case, with **islablin** = 1 and **hslablin** > 0 (as explained in Section 6.6), the initial fluid temperature is specified on the negative side of the slab and is only considered as an initial condition. The surface temperature **tslabbc** is defined on the positive side and held constant as a boundary condition. The slab surface temperature on the negative side is then adjusted to be consistent with the steady-state heat flux.

6.7.3. Direct Input of Steady-State Heat Flux on the Negative or Positive Side of the Structure

One can also specify only the heat flux and the surface temperature on one side of the structure. In this case, the steady-state surface temperature on the side of the heat flux is consistently defined for this heat flux using the given surface temperature on the opposite side. Equations (6-1) and (6-2) are then applied again with the known surface temperatures to determine the steady state temperature profiles in the different structure elements. If the heat flux and one surface temperature are specified, then the surface temperature on the opposite side is calculated consistently. It can no longer be

set by input if a steady-state temperature profile is assumed to exist prior to the analysis. The input heat fluxes q_{ce} and q_{cw} determine the heat flux for planar walls by

$$q_{ss}'' = \begin{cases} q_{cw} & \text{when flux is specified on negative side} \\ -q_{ce} & \text{when flux is specified on positive side} \end{cases} \quad (6-13)$$

For cylindrical walls, the linear heat rate is determined from the input heat fluxes by

$$q_{ss} = \begin{cases} x_1 q_{cw} & \text{when flux is specified on negative side} \\ -x_{NHE+1} q_{ce} & \text{when flux is specified on positive side} \end{cases} \quad (6-14)$$

The input currently requires the user to specify (?) the steady-state surface temperature on the opposite side as the heat flux. The missing surface temperature on the heat flux side is then calculated from Equation (6-3) or (6-4), respectively, with the following correlation:

$$T_{NHE+1} - T_1 = - \begin{cases} q_{ss}'' \sum_{j=2}^{NHE+1} \frac{\delta x_j}{k_j} & \text{for planar walls} \\ q_{ss} \sum_{j=2}^{NHE+1} \frac{\ln(x_j / x_{j-1})}{k_j} & \text{for cylindrical walls} \end{cases} \quad (6-15)$$

Instead of inputting the surface temperature on the opposite side, it would be possible to specify the steady-state surface temperature on the same side as the heat flux. But this condition would require prior hand calculations to arrive at the desired surface temperature on the fluid side; therefore, it has not been put into the code.

6.8 Heat Fluxes into Slabs, Walls, and Sinks

The heat rate absorbed in the different structures is evaluated in subroutine `outheat` and printed for each slab, wall, and sink at the user-specified time interval `prt dt`. This subroutine also evaluates the heat fluxes which the structure takes out of the GASFLOW fluid cells. These heat fluxes are summed up for each structure type and are also included in the general plot output from tape 11. They can be applied in an overall energy balance if the system is closed. The transient temperature changes are also evaluated separately in the surface elements of both structure sides. This allows the calculation of transient heat fluxes associated with boundary conditions that only specify structure surface temperatures. The heat rate absorbed in the structure is evaluated from the rate of temperature change at each temperature node, and the heat absorption rate equals the sum of the heat fluxes on both surfaces of the structure, which is an integral check that the transient evaluation of the structure temperatures does indeed conserve energy.

6.9 Balancing of Heat Structure Surfaces

The code automatically sums up all structure surfaces for slabs, walls, and sinks, split up into concrete, steel, and total surface. This information is printed out to the terminal. Additionally, by setting the input parameter in rheat to

```
lprarea =1,
```

a data set area is defined at the start of the calculation under unit 99 that contains the following: all affected m nodes

m	type	mat	thickness	area
4601	1	1	5.0000E+01	1.4000E+04
4661	1	1	5.0000E+01	1.4000E+04;

the type (1=slab, 2=sink, 3=wall); the material number; the thickness applied for heat conduction; and the surface area of the respective structure. This information can be applied from other programs for a room wise balancing of structure surfaces as needed in the comparison of GASFLOW heat structure input to corresponding input from lumped parameter models.

7 PHYSICAL MODEL OPTIONS

7.1 Body Forces

The momentum conservation equations solved by GASFLOW include body forces, or forces given by the product of the fluid density and a constant acceleration. The most common body force is that due to gravity. Because the user can orient the computational mesh arbitrarily with respect to the gravity vector, the code by default sets the body force acceleration term to zero in all three directions. To specify the acceleration vector due to gravity, the user has to define the values of the following variables in NAMELIST group `xput`:

- gx** Acceleration due to gravity in the *i*- (*x*- or *r*-) direction (cm/s^2).
Default = 0.
- gy** Acceleration due to gravity in the *j*- (*y*- or θ -) direction (cm/s^2).
Default = 0.
- gz** Acceleration due to gravity in the *k*- (*z*-) direction (cm/s^2).
Default = 0.

For example, in a problem where the *z*-direction is vertically upward, normal earth gravity is activated with the following input:

gx = 0., **gy** = 0., **gz** = -980.

Besides specifying the gravity term, the code offers the option of starting a calculation with a pressure gradient in the fluid that is in equilibrium with its own body weight. This option is specified through the **ihystat** variable in NAMELIST group `xput`:

- ihystat** Option flag for imposing an initial hydrostatic pressure gradient in the fluid cells according to the acceleration components **gx**, **gy**, and **gz**: 1 means ON; 0 means OFF (default).

7.2 Diffusion of Mass, Energy, and Momentum

In GASFLOW, transport due to gradient diffusion is modeled by Fick's law-type fluxes in the conservation equations for mass, energy, and momentum. The default is no gradient diffusion. The user must explicitly specify them using the following input variables in NAMELIST group `xput`:

- idiffme** Option flag for mass and energy diffusion:
1 means ON; 0 means OFF (default).

idiffmom Option flag for momentum diffusion:
1 means ON; 0 means OFF (default).

Note that these option flags apply to the diffusion terms, which include both molecular and turbulent transport. However, by default, the code does not calculate turbulent diffusion; the user must explicitly specify that model option using the **tmodel** input variable (see discussion on turbulence in the next section).

For momentum diffusion, if **idiffmom** is set to 1 and no turbulence model is activated (**tmodel** = 'none'), there is an additional option of how the molecular diffusion coefficient is calculated. The user can specify whether the input dynamic viscosity, μ (in units of poise or g/cm s), or the input kinematic viscosity, $\nu \equiv \mu/\rho$ (cm²/s), is to be used for calculating the diffusional momentum fluxes or viscous stresses. This is done via the following variables in NAMELIST **xput**:

muoption	Option specifying whether the input variable nu or cmug is to be used for viscous stress calculation: 1 means nu is used (default); 2 means cmug is used.
nu	Kinematic viscosity, ν (cm ² /s). Default = 0.15.
cmug	Dynamic viscosity, μ (g/cm s). Default = 1.8×10^{-4} .

For the diffusion of mass species and energy, the code uses the input variables **schmidt** and **prandtl**, respectively, if the user has selected the **itopt** = 0 option, to determine the appropriate diffusion coefficient. See Section 4.2 for information concerning temperature-dependent transport properties.

7.3 Turbulence

The Navier-Stokes equations are nonlinear, and when the flow speed exceeds certain criteria (as measured by the Reynolds or Grashof number), they become unstable in the sense that the solution (fluid velocity, pressure, temperature, etc.) exhibits oscillatory or fluctuating behavior. These fluctuations can be calculated directly if the computational mesh is so fine and the time advancement increments are so small that even the smallest eddies (scales of motion) can be resolved. However, in practically all problems, this is not the case. Furthermore, in most problems, there is no need to trace the *exact* behavior of every fluid element at every instant. It is often adequate to calculate the fluid behavior averaged over some time interval and space region that are large compared with the turbulent fluctuation scale, but small compared with the problem transient time scale. (Such averaging procedure is called Reynolds averaging.)

Therefore, the equations solved in GASFLOW are actually Reynolds averaged, with the dependent variables being mean quantities after the turbulent fluctuations are removed. However, the fluctuations cannot be simply ignored. The equations retain terms that

are averages of products like $u'v'$ and $u'\phi'$, where u' , v' are fluctuating velocity components and ϕ' is a fluctuating scalar quantity such as internal energy or a mass species concentration. These terms represent transport fluxes due to turbulent fluctuations and must be determined by some turbulence model that relates these fluxes to the calculated quantities.

In GASFLOW, all turbulent fluxes are modeled like a molecular diffusion process. Turbulent momentum fluxes (stresses) are modeled as

$$\tau_{t,ij} \equiv \mu_t \partial u_i / \partial x_j,$$

where $\partial u_i / \partial x_j$ is the velocity gradient tensor. This is the so-called Boussinesq's hypothesis, which states that turbulent transport can be modeled as a diffusion process with an eddy viscosity, μ_t . In GASFLOW, we further assume that the turbulent diffusion coefficients for mass and energy are both the same as that for momentum and are given by μ_t / ρ , where ρ is the fluid density.

Therefore the task of modeling turbulence is reduced to determining the eddy viscosity. In GASFLOW, two turbulence models are available. The choice can be specified via the input character variable **tmodel** in NAMELIST group xput:

tmodel Symbol designating the type of turbulence model to be used:
 'none', no turbulence model, i.e., only molecular diffusion;
 'alg', algebraic turbulence model;
 'ke', κ - ϵ turbulence model.

Note that the turbulence model invoked via **tmodel** is only in effect when diffusion calculations have also been specified, i.e., when **idiffmom** and/or **idiffme** has been set to 1.

7.3.1 Algebraic Model

The algebraic turbulence model used in GASFLOW assumes that the turbulent viscosity can be written as

$$\mu_t = c_\mu \rho \sqrt{k} \ell ,$$

where κ is the turbulent kinetic energy (per unit mass), ρ is the fluid density, ℓ is a turbulent length scale, and c_μ is a constant coefficient. In this model, it is assumed that the turbulent kinetic energy is a specified fraction of the fluid mean kinetic energy, i.e.,

$$k = f \cdot \frac{1}{2} (u^2 + v^2 + w^2) .$$

The parameters to be input for the algebraic turbulence model are c_μ , ℓ , and f , which are represented by the following variables in NAMELIST group xput:

cmu	Constant c_μ for the algebraic turbulence model. Default = 0.05.
clength	Length scale, ℓ , for the algebraic turbulence model (cm). Default = 30.48.
fracke	Fraction of the mean kinetic energy, f , used in the algebraic turbulence model to determine the turbulent kinetic energy. Default = 0.1.

7.3.2 The κ - ϵ Model

In the κ - ϵ model, it is assumed that the turbulent viscosity can be written as

$$\mu_t = c_\mu \rho \kappa^2 / \epsilon ,$$

where κ is the turbulent kinetic energy (per unit mass), ρ is the fluid density, ϵ is the rate of dissipation of turbulent kinetic energy, and c_μ is a constant coefficient. The turbulent kinetic energy is determined from solution of its transport equation:

$$\begin{aligned} & \frac{\partial(\rho\kappa)}{\partial t} + \nabla \cdot (\rho\kappa\mathbf{u}) \\ &= \nabla \cdot \left(\frac{\mu_t}{\sigma_\kappa} \nabla \kappa \right) + \tau : \nabla \mathbf{u} + \mu \alpha \mathbf{g} \cdot \nabla T - \rho \epsilon + \kappa S_\kappa \end{aligned}$$

The terms on the right hand side are, in order, diffusional transport, production by viscous stresses, production by buoyancy, viscous dissipation, and generation from other sources. Similarly, the rate of dissipation, ϵ , is determined from solution of its transport equation:

$$\begin{aligned} & \frac{\partial(\rho\epsilon)}{\partial t} + \nabla \cdot (\rho\epsilon\mathbf{u}) \\ &= \nabla \cdot \left(\frac{\mu_t}{\sigma_\epsilon} \nabla \epsilon \right) + C_1 \frac{\epsilon}{\kappa} \tau : \nabla \mathbf{u} + C_1 \frac{\epsilon}{\kappa} \mu \alpha \mathbf{g} \cdot \nabla T - C_2 \rho \frac{\epsilon^2}{\kappa} + \epsilon S_\epsilon . \end{aligned}$$

The following variables in NAMELIST group xput are used to set the κ - ϵ model parameters:

cmuke	c_μ used in the κ - ϵ turbulence model. Default = 0.09.
c1ke	c_1 used in the κ - ϵ turbulence model. Default = 1.44.
c2ke	c_2 used in the κ - ϵ turbulence model. Default = 1.92.
sigmak	σ_κ used in the κ - ϵ turbulence model. Default = 1.
sigmae	σ_ϵ used in the κ - ϵ turbulence model. Default = 1.3.

Because the κ - ϵ model solves time-dependent transport equations for both κ and ϵ , the user must define initial conditions for these quantities. The **turbdef** array described below is used for defining initial and boundary conditions for the κ - ϵ model and is in the NAMELIST group **xput**.

turbdef(1,*)	Beginning i mesh index (cell face number).
turbdef(2,*)	Ending i mesh index (cell face number).
turbdef(3,*)	Beginning j mesh index (cell face number).
turbdef(4,*)	Ending j mesh index (cell face number).
turbdef(5,*)	Beginning k mesh index (cell face number).
turbdef(6,*)	Ending k mesh index (cell face number).
turbdef(7,*)	Block number.
turbdef(8,*)	Integer pointer to location in tkeval array for value of turbulent kinetic energy.
turbdef(9,*)	Integer pointer to location in epsval array for value of turbulence dissipation rate, ϵ .
turbdef(10,*)	Integer pointer to location in sclval array for value of turbulence length scale.
turbdef(11,*)	Time (s) at which "turbulence definition" begins.
turbdef(12,*)	Time (s) at which "turbulence definition" ends.

The user must define a valid pointer for the array **tkeval** in **turbdef(8,*)**, and a valid pointer for the array **epsval** in **turbdef(9,*)**. The input in **turbdef(10,*)** will be ignored. The following input illustrates the use of **turbdef** to define an initial condition and a boundary condition for solving the κ and ϵ transport equations:

```

turbdef(1,1) = 1, 50, 1, 32, 1, 45, 1, 2, 1, 0, 0.0, 0.0,
turbdef(1,2) = 0, 1, 2, 5, 4, 6, 1, 1, 2, 0, 0.0, 1.e9,

tkeval       = 1.e3, 0.0,
epsval       = 0.0, 800.,

```

The first **turbdef** defines an initial condition for κ and ϵ . The values of both κ and ϵ are initially 0. The second **turbdef** specifies a boundary condition at the $-i$ boundary, between j -indices of 2 and 5, and between k -indices of 4 and 6. The boundary values of κ and ϵ specified are $1000 \text{ cm}^2/\text{s}^2$ and $800 \text{ cm}^2/\text{s}^3$, respectively.

7.4 Combustion

GASFLOW models the combustion of hydrogen in air with the following single, global reaction:

where the coefficients a, b, c, d, e, f , and g are generally referred to as the stoichiometric coefficients. The reaction rate, $\dot{\omega}$, is determined from a modified Arrhenius law, where

$$-\frac{1}{a} \frac{dc_{h_2}}{dt} = -\frac{1}{b} \frac{dc_{o_2}}{dt} = +\frac{1}{e} \frac{dc_{h_2o}}{dt} = \dot{\omega} ,$$

and $a = 2, b = 1$, and $e = 2$.

Computationally, during each time cycle, the change in molar density of the fuel, δc_{h_2} is calculated implicitly first when the fuel-oxidizer mixture is fuel lean. Then the molar densities at the advanced time level $n+1$ $c_{h_2}^{n+1}$, $c_{o_2}^{n+1}$, and $c_{h_2o}^{n+1}$ are determined respectively as

$$c_{h_2}^{n+1} = c_{h_2}^n + \delta c_{h_2} , \quad c_{o_2}^{n+1} = c_{o_2}^n + \frac{b}{a} \delta c_{h_2} , \quad \text{and} \quad c_{h_2o}^{n+1} = c_{h_2o}^n - \frac{e}{a} \delta c_{h_2} .$$

When the mixture is fuel rich, then the molar density change for the oxidizer δc_{o_2} is calculated first, and the molar densities are determined as

$$c_{o_2}^{n+1} = c_{o_2}^n + \delta c_{o_2} , \quad c_{h_2}^{n+1} = c_{h_2}^n + \frac{a}{b} \delta c_{o_2} , \quad \text{and} \quad c_{h_2o}^{n+1} = c_{h_2o}^n - \frac{e}{b} \delta c_{o_2} .$$

This will ensure that nonphysical negative species densities will not result from the chemical reaction. In addition to changing the reactant and product species densities, the reaction rate is used to compute the increase in energy due to the combustion process.

This global reaction and the hydrogen recombiner model are the only chemistry models in GASFLOW. The reaction rate parameters and heat of reaction are fixed in the code. Therefore, the user only has to specify the following variable, in NAMELIST group `xput`, in order to turn on the hydrogen combustion model in the calculation:

iburn Option flag for the hydrogen burn model:
1 means ON; 0 means OFF (default).

To activate the hydrogen recombiner model, the **ircomb** input variable must be 1 and **rcombdef** input described in Section 7.7.3 must be provided.

7.5 Heat Transfer

Although the energy conservation equation is solved by the code under all circumstances, heat transfer between the fluid and any solid structures are, by default, not calculated. Therefore, in a problem that, for instance, involves hydrogen combustion (**iburn** = 1) but the heat-transfer model is not explicitly requested, a temperature rise will still be calculated, which is only valid if the process is adiabatic or if the problem time scale is fast compared with the heat-transfer time scale.

To activate heat-transfer and steam-condensation calculations, using models based on the argument of analogies between momentum, heat, and mass transfer, the user must specify the following variable in NAMELIST group **rheat**:

ihthflag Option flag to turn on heat-transfer and condensation calculations:
1 means ON; 0 means OFF (default).

The user should also refer to Section 6, on defining heat structures, for complete specification of a heat-transfer problem.

7.6 Aerosol Model

To set up the GASFLOW Lagrangian discrete particle dynamics models, an understanding of the input parameters and their function in the model's is essential. In Appendix F the particle input parameters are listed and defined with default values. These are described in more detail in the following sections focusing on the four major functions of the particle model. These functions are particle initialization, transport, deposition, and entrainment. In addition, the particle cloud model is described. Each section presents and describes the input parameters, presents associated variables and arrays and presents in outline form the setup procedure for the model function. Next, descriptions of the parameter statements, restart procedure, and graphic and tabular output are given. A sample problem setup and output are discussed.

Generally, it is impossible, because of computer storage limitations and costs, to track each particle in an aerosol cloud. The total mass of the aerosol cloud must be represented by a finite number of discrete particles. In the following description of input parameters, the terms *real* and *simulation* are often used. The term *real* is used in reference to the particle number, mass, etc., that are being modeled. The term *simulation* refers to the discrete particles that are being used in the numerical model to simulate the *real* particles being modeled. Each *simulation* particle may represent thousands of *real* particles of the same size and density.

7.6.1 Description of Particle Initialization Input Parameters

The particulate materials are characterized by class and size. The material class corresponds to a material density. Each material class may have multiple particle sizes. The input parameter **pmass(lc,ls)** is the total mass of each particle class and size that is to be input over the total problem time. The class material density is set in **prho(lc)** and the particle diameters are specified for each class and size by **pdiamin(lc,ls)**. From this input data the number of *real* particles, **npreal(lc,ls)**, for each class and size and the total number of *real* particles, **npsum**, are computed. Next, the volumes, **volp(lc)**, within the computational mesh into which each particle class is to be set are determined from the input parameters specifying the maximum and minimum *x*-, *y*-, and *z*-coordinates of the particle volumes; these are **xpe(lc)**, **xpw(lc)**, **ypn(lc)**, **yps(lc)**, **zpt(lc)**, and **zpb(lc)**.

Generally, the total number of *real* particles, determined from **pmass(lc,ls)**, **prho(lc)**, and **pdiamin(lc,ls)**, would be prohibitively large to compute efficiently. The user selects the total number of *simulation* particles to be input and transported by the code. This input parameter is **npinpt** (number particles input total). From the total number of simulation particles input, **npinpt**, the number of *real* particles for each class and size, **npreal(lc,ls)**, and the total number of *real* particles, **npsum**, the code computes the number of *simulation* particles for each class and size **npsim(lc,ls)**. Using the information available, the *real* mass represented by each *simulation* particle, **rmpp(lc,ls)**, is computed.

The input parameter **twpinp(lc)** designates the initial time when *simulation* particles of each class are to be input into the system. The total number of particles for each class may all be set in the mesh initially or may be injected as a function of time. This choice is governed by the input parameters **tinjt(lc)** (total injection time) and **pinpdt(lc)** (particle input delta-*t*). To inject the particles as a function of time, set the initial time when particles are to be injected, **twpinp(lc)**, the total injection time, **tinjt(lc)**, and the time interval between particle injections, **pinpdt(lc)**. From this input data the code computes the number of injection times and the number of *simulation* particles for each class and size, **npinj(lc,ls)**, to be injected at the appropriate intervals. To inject a single volume of particles, input the time of injection, **twpinp(lc)**, and values for **tinjt(lc)** and **pinpdt(lc)** such that the ratio **tinjt(lc)/pinpdt(lc)** is less than 1.0. The code will set the number of injection times, **ninjts(lc)**, to 1.0. The value of **pinpdt(lc)** also must be greater than the problem time.

When restarting a problem from a previous particle run, set the parameter **twpinp(lc)** to a value greater than the final problem time to negate the injection of additional particles. If injection of particles is wanted at the time determined in the previous run,

do not set a value for **twpinp(1c)** in the restart input parameter file. The correct value will be read from the restart tape file.

The variables **itpcl**, **itpsz**, and **mzp** are included in the xput NAMELIST group and are used to set array sizes in the particle model. Therefore, whenever **solatype** is greater than zero, these three inputs must be supplied in the xput NAMELIST group. A calculation can be started with **solatype** equal to zero and then restarted with **solatype** greater than zero. The restart must include the three inputs **itpcl**, **itpsz**, and **mzp**. Once these three inputs have been used, all subsequent restarts must include the same values for these input variables.

The following aerosol input parameters are in the xput NAMELIST group:

itpcl	Total number of particle classes.
itpsz(1c)	Number of initial particle sizes in each particle class.
mzp	Total number of simulation particles allowed in this calculation.
solatype	solatype = 0.0 implies a hydrodynamic calculation without the particle model. solatype = 1.0 implies a hydrodynamic calculation with the particle model. solatype = 2.0 implies a particle-model-only calculation with gas velocities fixed in time.

The following input parameters are in the parts NAMELIST group:

ipblkin(1c)	Mesh block in which particle class, index 1c, is initially located.
ipclin(1c)	Particle class number input.
npinpt	Total number of <i>simulation</i> particles to be input.
pinpdt(1c)	Time interval between particle inputs for each particle class (s).
pdiamin(1c,1s)	Particle diameter input for each class and size (cm).
pmass(1c,1s)	Total <i>real</i> particle mass of each class and size (g).
prhoin(1c)	Material density input for each particle class (g/cm ³).
tinjt(1c)	Total injection time for each particle class (s).
twpinp(1c)	Time when to input particles of each class (s).
xpe(1c)	Maximum x-coordinate of initial particle block for each particle class (cm).
xpw(1c)	Minimum x-coordinate of initial particle block for each particle class (cm).
ypn(1c)	Maximum y-coordinate of initial particle block for each particle class (cm).
yps(1c)	Minimum y-coordinate of initial particle block for each particle class (cm).
zpb(1c)	Minimum z-coordinate of initial particle block for each particle class (cm).

zpt(1c) Maximum z-coordinate of initial particle block for each particle class (cm).

The following inputs are required:

1. Particle material density, particle size, and total mass of particulate material:

prhoin(1c) **pdiamin(1c,1s)** **pmass(1c,1s)**

2. Particle input volume and location:

xpw(1c) **xpe(1c)**
yps(1c) **ypn(1c)**
zpb(1c) **zpt(1c)**

3. Particle injection time and rate:

twpinp(1c) Initial time when to input particles.
tinjt(1c) Total injection time.
pinpdt(1c) Time increment between particle injections.

4. Total number of simulation particles to be input over the entire injection time:

npinpt

5. Determine the total number of simulation particles to be simulated, the number of classes, and the number of different sizes in each class:

mshp
itpcl
itpsz

The following is the procedure for the preparation of input for particle initialization:

1. Define the location of the particles to be input for each class and size. The mesh block, **ipblkln(1c)**, in which the particles of class **1c** are placed, has a default value of 1. Specify the rectangular volume in the computational mesh into which the *simulation* particles are to be placed by inputting the maximum and minimum *x*-, *y*-, and *z*-coordinates of the particle volume for each class; these are, respectively, **xpe(1c)**, **xpw(1c)**, **ypn(1c)**, **yps(1c)**, **zpt(1c)**, and **zpb(1c)**.
2. Define the number of aerosols to be modeled. Set the number of different particle materials, i.e., particle classes, **itpcl**. Specify the number of particle sizes to be modeled in each of these classes, **itpsz(1c)**. **itpcl** has a default of 1, whereas there is

no default for *itpsz*. The particle class number, **ipclin(1c)**, is currently set by default and does not need to be changed.

3. Characterize the *real* particle material to be modeled. Specify the total material mass of each class and size, **pmass(1c,1s)**. Set the density of the material of each class, **prho1n(1c)**, and specify the particle diameter for each class and size, **pdiamin(1c,1s)**. (These same particle densities and diameters are used for calculating the *simulation* particle dynamics in the code particle transport model.)

4. Define the *simulation* particle injection number and mode. Set the total number of *simulation* particles, which includes all the particles of all classes and sizes to be input over all time, **npinpt**. Select the initial time and duration for which particles of each class will be injected by setting the time when to input particles, **twpinp(1c)**, and the total injection time for each class **tinjt(1c)**. Set the time interval between particle injections during this period, **pinpdt(1c)**. To inject a single volume of particles, i.e, a one-time injection, input the time of injection, **twpinp(1c)**. Set the value of **pinpdt(1c)** to a value greater than the final problem time. Select a value for **tinjt(1c)** such that the ratio **tinjt(1c) / pinpdt(1c)** is less than 1.0.

To inject into the computational domain a selected *simulation* particle concentration each computational cycle, a nearly constant computational time step is necessary for this to result in a nearly constant particle concentration input. For particle number concentration input, the *simulation* particle mass is assumed equal to the *real* particle mass, **pmass(1c,1s)**. The standard input parameters are set with the exception of **npinpt** and **pmass(1c,1s)**, which must be computed by the user.

To initialize the particle input in this manner, choose the input parameters **xpe(1c)**, **xpw(1c)**, **ypn(1c)**, **yps(1c)**, **zpt(1c)**, and **zpb(1c)**, and compute the particle input volume. Choose a *simulation* particle number concentration and compute the number of particles to be injected each computational time step. Next, choose input parameters **tinjt(1c)** and **pinpdt(1c)** and compute the total number of injection times. Use this information to compute the input parameter **npinpt**, the total number of particles input. (If this number is prohibitively large, the number concentration or the input volume must be reduced.) Finally, select the particle size and density and compute the input parameter **pmass(1c,1s)**, the particle mass of each class and size to be input over the total problem time.

In more detail, the setup procedure is as follows:

1. Select the input volume dimension in the inflow velocity direction to be compatible with $U_{in} dt$ (cm).

2. Select the particle number concentrations, $\text{cpinj}(lc, ls)$, for each particle class and size (no. of particles/cm³).

3. Compute particle input volume, $\text{volp}(lc)$, from $\text{xpe}(lc)$, etc., for each class (cm³).

4. Compute the number of particles injected each injection time for each class and size,

$$\text{npinj}(lc, ls) = \text{cpinj}(lc, ls) * \text{volp}(lc).$$

5. Select a maximum injection time, $\text{tinjt}(lc)$ (s), and the particle injection time interval, $\text{pinpdt}(lc)$ (s).

6. Compute the number of injection times anticipated in the total problem time,

$$\text{ninjts}(lc) = \text{tinjt}(lc) / \text{pinpdt}(lc).$$

7. Compute the number of particles for each class and size,

$$\text{npsim}(lc, ls) = \text{npinj}(lc, ls) * \text{ninjts}(lc)$$

8. Sum the total number of particles in each class and size. This gives the input parameter npinpt , which is the total number of particles to be injected.

9. Select the particle size, $\text{pdiamin}(lc, ls)$ (cm), and the particle density, $\text{prhoin}(lc)$ (g/cm³).

10. Compute the mass of each *simulation* particle class and size injected. The product of this and the number of particles for each class and size is the input parameter $\text{pmass}(lc, ls)$,

$$\text{pmass}(lc, ls) = [(\pi/6) * \text{pdiamin}(lc, ls)^3 * \text{prhoin}(lc)] \\ * \text{npsim}(lc, ls) \text{ (g).}$$

7.6.2 Description of Particle Transport Input Parameters

If $\text{imarker} = 1$, the particles input are used as marker particles and moved at the local fluid velocity. Most of the particle physics is bypassed. The default value of imarker is 0. In this case, the complete particle dynamics algorithm is executed.

The particle transport model includes the inertial, drag, and gravitational forces. In cylindrical coordinates ($\text{cyl} = 1$ in NAMELIST group xput), the centrifugal force also is modeled. The drag coefficient in the Stokes flow region is a function of the flow Reynolds number and, consequently, of the relative particle-fluid velocity. This requires

an iterative procedure (the Newton-Raphson method is used) to simultaneously solve for the drag coefficient and the particle velocity. **niterp** is input to set the maximum number of iterations. Generally, the smaller the particle size, the greater the number of iterations required. For example, for 0.5- μm -diameter particles of unit density, it requires about 40 iterations to compute an accurate terminal settling velocity. It requires only 20 iterations for a 100- μm , unit-density particle to compute the correct terminal settling velocity. The default value for **niterp** is set to 30. The slip correction factor, **scfacin(mxpc1, mxpsz)**, reduces the drag force for small particles and should be used for particles with diameters less than about 1.0 μm . The value is always 1.0 or greater, and is 1.165 for 1.0- μm -diameter particles and 2.888 for 0.1- μm -diameter particles. The default value for **scfacin(mxpc1, mxpsz)** is 1.0.

The turbulent diffusion of the particles is characterized by the particle turbulent diffusion coefficient, **tdcp**. Note that there are currently three models in GASFLOW for the turbulent diffusion coefficient of the particles (see input description below). The numerical model for turbulent diffusion computes the diffusion velocity components for each particle. In this model, the fluid velocity components ($u_f(n)$, $v_f(n)$, and $w_f(n)$) at the locations of each particle are the sum of the mean velocity components and the random turbulent velocity component. The basic idea in determining the random turbulent velocity is to consider the particle as a point source that diffuses for a time Δt . The probability of where the particle is likely to move is given by a Gaussian distribution, with a width distribution determined by the standard deviation $4 \Delta t \text{tdcp}$. A random number generator selects the location actually used within the distribution. A default value of 0.0 is set.

The coefficient of kinematic viscosity, **visf**, is set by default to the value of **nu** input in NAMELIST group **xput** and typically does not need to be changed. Similarly, the default value of 2.0 for **wmax**, the maximum argument of the error function used in the inverse error function table, does not need to be changed. The input parameter **inpvol** is not used for the Lagrangian particle setup. It may be used in the future for the combined Lagrangian-Eulerian scheme.

The following input parameters for the particle transport model are in the parts NAMELIST group.

imarker	Marker particle flag: if imarker = 1, particles are moved at the local fluid velocity (most of the particle physics is bypassed); if imarker = 0, the complete particle dynamics algorithm is executed. Default = 0.
inpvol	Input parameter to choose actual or fictitious volume assigned to each particle: if inpvol = 0, actual volume is computed; if inpvol = 1, $pvol = \text{delx} * \text{dely} * \text{delz}$ of cell in which particle is located.

niterp	Number of iterations for the Newton iteration cycle for determining drag-induced particle velocity. Default = 30.
scfacin(mxpcl, mxpsz)	Cunningham slip correction factor for each particle class and size. Default = 1.0.
tdcp	Turbulent diffusion coefficient for particles (cm^2/s). If tdcp is greater than zero, then tdcp is used for the turbulent diffusion coefficient for particles. If tdcp = -1.0, then $Sc\mu/\rho$ is used for the turbulent diffusion coefficient for particles, where μ is the total viscosity of the gas mixture including the turbulent viscosity. If tdcp = -2.0, then the turbulent kinetic energy is used to determine the turbulent diffusion coefficient if tmodel = 'ke' or the turbulent viscosity is used to determine the turbulent diffusion coefficient if tmodel = 'alg'.
visf	Coefficient of kinematic viscosity of fluid = ν (cm^2/s).
wmax	Maximum argument of error function used in inverse error function table. Default = 2.0.

The procedure for the particle transport model setup is as follows:

1. Set up particle input as described in Section 7.6.1.
2. Select the value of **imarker**. The default value of 0 utilizes the full particle dynamics algorithm.
3. If the particle diameters are $1.0\ \mu\text{m}$ or less, set the slip correction factor, **scfacin(mxpcl, mxpsz)**. The default value is 1.0.
4. If particle turbulent diffusion is to be modeled, set the coefficient **tdcp** to an estimated value. The default value is $0.0\ \text{cm}^2/\text{s}$.
5. Choose a value for **niterp**, typically between 10 and 40, depending on the particle size and accuracy required (default = 30).
6. Use the default values for **inpvol**, **visf**, and **wmax**.

7.6.3 Description of Particle Deposition Model Input

The default value of the particle deposition flag, **ipdep(1c)**, is 0. If no particle adhesion upon impact for particle class **1c** is to be modeled, no further input is necessary. At the other extreme, if all particles adhere upon impact, an **ipdep(1c)** value of 2 is input. A deposition model based on the particle material properties (and

some stochastic considerations) is invoked if **ipdep(1c)** is set equal to 1. For example, if the first class of particle material (class number 1) is a liquid mist, then **ipdep(1) = 2** is appropriate because the impacting material will most likely not bounce. Then, if the second particle class is a harder material and the deposition model is to be used to choose whether a particle bounces or adheres on impact, **ipdep(2) = 1** is input.

The deposition model uses the Dahneke theory of rebound to compute a critical rebound velocity, at which there is a 50% probability of bounce. (The code randomly chooses which particles inside a window bracketing the critical rebound velocity actually bounce.) This critical rebound velocity is a function of the impacting particle mass and energy and the coefficient of restitution. The particle energy is a function of the Hamaker constant, **hca**, the particle diameter, **pdiamin(1c, 1s)**, the separation distance of the particle and substrate, **sdz**, and particle bulk mechanical properties, **bmck(1c)**, which are a function of the material Poisson ratio, **poisrt(1c)**, and Young's modulus, **yngmod(1c)**.

The Hamaker constant has been determined from theory and experiments for many particle-surface combinations. The values range from about 5.0×10^{-13} to 5.0×10^{-12} erg. A typical value of 1.0×10^{-12} erg is used for the default value of **hca**. The equilibrium separation distance, **sdz**, is estimated to be 4.0×10^{-8} cm. This may vary according to surface roughness. The default value for **sdz** is this estimated value. However, the influence of surface roughness can be qualitatively modeled by increasing the value of **sdz** to the estimated roughness height.

The coefficient of restitution, **core**, has been experimentally determined for many particle-surface combinations. The values typically range from 0.90 to 0.99 for a hard particle impinging onto a hard surface. The input parameter **core** has a default value of 0.96. Experimental data shows the trend for the coefficient of restitution to reach this maximum value at a given incident velocity and then almost immediately the ratio of rebound velocity to incident velocity begins to decrease as the incident velocity increases. This phenomenon is programmed into the code and does not need any input data.

Values for the mechanical properties, Poisson ratio, **poisrt(1c)**, and Young's modulus, **yngmod(1c)**, are in appropriate handbooks. The Poisson ratio is always less than 0.5. It ranges from 0.42 for gold to 0.16 for fused quartz. The default value is 0.29, which is the value for steel. Young's modulus ranges from 2.10×10^{12} dynes/cm² for steel (default value) to 3.24×10^{10} dynes/cm² for plexiglass.

Deposition is in a real sense a stochastic process that follows the general trend of the theoretical and empirical models developed from and compared with experimental data. Because of this, it is a reasonable assumption that some small, unknown percentage of the particles that impact a surface will adhere. To account for this, the

parameter **depperc(1c)** is input. The default value is set to 5.0%. If the substrate is dirty or there are other reasons to believe a smaller or larger percentage of the particles will adhere on impact, this value can be changed.

In general, the harder the particle and surface, the greater the surface roughness, the larger the particle, and the greater the velocity, the more likely it is that bounce will occur. Bounce is not a problem for liquids or easily deformed material, such as tar.

The following input parameters are in the parts NAMELIST group:

core	Coefficient of restitution of particle material.
depperc(1c)	Percentage of particles that immediately deposit upon impact.
hca	Hamaker constant (ergs).
ipdep(1c)	Particle deposition flag input for each class: if ipdep(1c) = 0, no adhesion; = 1, adhesion determined by deposition model; = 2, all impacting particles adhere.
ndxpd	Number of longitudinal sections in x-direction in which particle deposition (only on bottom of mesh) is monitored: if < 1, no plot.
pdiamin(1c,1s)	Particle diameter input for each class and size (cm).
poisrt(1c)	Poisson ratio of particle class material.
prho1n(1c)	Material density input for each particle class (g/cm ³).
sdz	Separation distance of particle and substrate (cm).
ynghmod(1c)	Young's modulus of particle material (dynes/cm ²).

The following are the setup procedures for the particle deposition model:

1. Set up particle input as described in Section 7.6.1, Particle Initialization, and in Section 7.6.2, Particle Transport. This includes the input for the diameter of each class and size particle, **pdiamin(1c,1s)**, and the particle material density for each particle class, **prho1n(1c)**.
2. Select a value of **ipdep(1c)** for each particle class being modeled. There is no deposition of particles when the default value of 0 is set.
3. Use the default values for the Hamaker constant, **hca**, the particle-substrate separation distance, **sdz**, and the coefficient of restitution, **core**. These input parameters may be changed if there is a reason to do so.
4. Input appropriate material property values, Poisson ratio, **poisrt(1c)**, and Young's modulus, **ynghmod(1c)**, for all particle classes for which the deposition model is used, that is, when **ipdep(1c)** = 1. The default values are those for steel.

5. Use the default value (5%) for the percentage of particles that deposit on impact, **depperc(1c)**; or, considering the condition of the substrate and any other pertinent information, set it to a better guess.

6. If a plot is wanted of the total particle mass deposited on the "floor" of the mesh (i.e., the bottom face of the $k = 2$ layer of cells) as a function of distance in the x -direction, set the input parameter **ndxpd** to the number of sections into which the distance from the mesh minimum x -coordinate to the maximum x -coordinate is to be divided and in which the deposition is to be monitored. If this information is not wanted, do not set this parameter, and the default value is such that this computation and plot will be skipped.

7. If time-history plots are wanted for the computational mesh cell faces onto which particles have deposited, input the necessary information according to instructions in Section 9.1.7, Graphics and Tabular Particle Data Output.

7.6.4 Description of Entrainment Input Parameters

The default value for the particle entrainment flag, **intrn**, is 0. If particle entrainment is not to be modeled, no further input is required. If particle entrainment is to be modeled, set **intrn** to 1, and the entrainment model is used to suspend deposited particles when the fluid velocity equals or exceeds the computed particle threshold suspension velocity.

Criteria for determining the fluid velocity at which a particle initially at rest on a surface will become suspended is determined from a force balance equation that includes gravity, adhesion, fluid lift, drag, and friction forces. A particle adhering to a surface will be dislodged when the removal forces equals or exceeds the force of particle adhesion. This force balance equation is modified by experimental data that takes into account the effects of particle interactions with other particles, and the, usually, nonspherical particle shape. The gravitational, buoyant, and adhesive forces depend only on the physical properties of the particle and gas densities, and they are independent of the gas stream velocity. The adhesive force is the van der Waals inter-surface molecular force and is a function of the Hamaker constant, **hca**, the particle diameter, **pdiamin(1c,1s)**, the separation distance of the particle and the substrate, **sdz**, and particle bulk mechanical properties, **bmck(1c)**, which is a function of the material Poisson ratio, **poisrt(1c)**, and Young's modulus, **yngmod(1c)**. These parameters are described in the discussion of particle deposition. The drag and lift forces depend on the gas stream velocity. The friction force is proportional to the coefficient of sliding friction and is set to a value of 0.45, which is the value suggested from experimental studies.

To compute the lift and drag forces, the velocity at the center of the stationary particle, which is typically in the viscous sublayer, must be known. The law-of-the-wall equation, which characterizes the fluid velocity in the turbulent boundary layer near the wall, is used to calculate the characteristic shear velocity and, subsequently, the velocity at the center of the particle. A no-slip wall boundary condition is assumed. In addition, the velocity gradient across the surface boundary layer must be known to compute the lift force. These computations require an estimate of the boundary layer thickness at the location of the deposited particle. If the input parameter **dbl** is set to a value less than 0, the code will compute a boundary layer thickness as a function of a Reynolds number and the distance along the wall from the point at which the turbulent fluid initially contacts the wall. It is assumed that this is the point at which the fluid enters the computational mesh. However, if this model is not compatible with the problem setup, an estimate of a constant boundary layer thickness may be set by the parameter **dbl**.

The input parameters **cdrf** and **sdzrf** are multipliers for the computed drag coefficient and the usual surface-substrate separation distance, respectively, that allow a qualitative representation of the roughness of the particle and substrate surfaces. The default values are 10.0 and 100.0, respectively. These values were determined by comparison of computations with one set of experimental data and require further evaluation.

The following input parameters are in the parts NAMELIST group:

intrn	Input parameter to choose particle entrainment option: = 0, no particle pickup; = 1, pickup option is on. Default = 0 and no more input is required.
dbl	Boundary layer thickness. Default = 10 cm.
cdrf	Aerodynamic drag coefficient roughness factor. Default = 10.
sdzrf	Particle-substrate separation distance roughness factor. Default = 100.

The procedure for the entrainment model setup is given below:

1. Set up particle input and deposition input as described in Sections 7.6.1, 7.6.2, and 7.6.3. This includes the input for the diameter of each class and size particle, **pdiamin(1c,1s)**, and the material property values, Poisson ratio, **poisrt(1c)**, and Young's modulus, **ynmod(1c)**, for all particle classes for which the entrainment model is used. The default values for the Hamaker constant, **hca**, and the particle-substrate separation distance, **sdz**, are set.
2. Select a value of **intrn**. There is no entrainment of particles for the default value of 0. Input a value of 1 to turn on the entrainment model.

3. Set **dbl** to -1.0 for the code to compute the boundary layer thickness, or set an estimate of a constant boundary layer thickness. The default value is 10.0 cm.
4. Use the default values of **cdrf** and **sdzrf**. These may be changed if better estimates are known.

7.6.5 Description of Particle Cloud Model Input Parameters

The concept of the discrete computational *simulation* particle representing a multitude of *real* particles, all of which are located at the same point in space as the *simulation* particle, is extended in the particle cloud model. This model permits each *simulation* particle to represent a multitude of *real* particles that disperse as a Gaussian cloud.

The default value of the particle cloud model flag, **icloud**, is 0. If the particle cloud model is not to be used, no further input is necessary. If the cloud model is to be used (**icloud** = 1), then a value for the particle cloud diffusion coefficient, **pcdc**, is chosen. The default value is 0. An estimated value for this input parameter is the value of the turbulent diffusion coefficient, **tdcp**.

The cloud model is designed to be used with particle monitors at selected locations in the computational domain. The total number of monitors is specified by the input parameter **ntmntr**, which has a default value of 0. The selected number of monitors must not be greater than the maximum number of monitors, **mntmx**, which is set in a parameter statement. The current value is 20. The monitor locations are specified by the input parameters **xm**, **ym**, and **zm** for each monitor.

Time-history plots of the *real* particle mass detected by monitors is plotted using the procedure described in Section 9.1.3.

The input parameters are in the **parts** NAMELIST group:

icloud	Particle cloud model flag: if icloud = 0, cloud model off; =1, cloud model is on. Default = 0.
ntmntr	Total number of particle cloud monitors.
pcdc	Particle cloud diffusion coefficient. If pcdc is greater than zero, it is used for the particle cloud diffusion coefficient. If pcdc = -1.0, then the particle cloud diffusion coefficient is equal to $Sc\mu/\rho$, where μ is the total viscosity of the gas mixture including the turbulent viscosity. Default = 0.
xm	x-coordinate of particle cloud monitors.
ym	y-coordinate of particle cloud monitors.
zm	z-coordinate of particle cloud monitors.

The following is the setup procedure for the particle cloud model:

1. Choose **icloud** = 1 to turn on the particle cloud model.
2. Set **pcdc** to an estimated value.
3. Choose the total number of monitors, **ntmntr**, and their locations, **xm**, **ym**, and **zm**.
4. Specify in the graphics input parameter **pthp** 'pmntr' for time-history plots of the *real* particle cloud mass at each monitor. (See Section 91.7.)

7.6.6 Particle Model Restart

Setting the NAMELIST group xput input parameter **tddt** to the appropriate time interval between tape dumps will generate a **gfdx** file from which to restart a run. A common procedure in running the aerosol model is to set **solatype**, also in NAMELIST group xput, to a value of 0.0, for a fluid solution only, and generating a steady-state fluid flow field. Then, **solatype** = 2.0 is set for particle transport only, and particles are injected into this steady flow field. Parameters that may need changing when restarting from a tape dump include

twfin	maxcyc
solatype	nrsdump
pthpt0	twpinp

7.7 Special Containment Models

In order to accurately model modern nuclear containments, it was found that three special models were necessary. The three models (i.e., ignitor, sump, and recombiner models) are described below.

7.7.1 Ignitor Model

The ignitor model is simple, but effective. At user-specified locations GASFLOW first checks the gas composition to determine if the mixture is combustible, based on the Shapiro Diagram for hydrogen, oxygen, and steam mixtures. The locations where the ignitor model is applied are defined with the input array **ignitdef**, which is in the xput NAMELIST group. The parameter **ignitdef** is described below:

ignitdef(1,*)	i cell index.
ignitdef(2,*)	j cell index.
ignitdef(3,*)	k cell index.

ignitdef(4,*)	Block number.
ignitdef(5,*)	Ignitor type: = 0, continuous ignitor; > 0, spark ignitor.

The asterisk (*) should be replaced by an integer that identifies the particular **ignitdef** definition.

When **ignitdef(5,*)** is greater than zero, then the ignitor types are defined in the input array **spxidef** and the input variable **spxigdt**, which are also in the xput NAMELIST group. Input array **spxidef** and input variable **spxigdt** are defined below:

spxigdef(1,igtyp)	Time when ignitor is first turned on.
spxigdef(2,igtyp)	Time when ignitor is turned off.
spxigdef(3,igtyp)	Time interval between sparks.
spxigdt	Time of the spark duration. Default = 0.001 s.

In the **spxigdef** definitions, **igtyp** is the ignitor type number defined by **ignitdef(5,*)**. The maximum number of ignitor definitions (i.e., **ignitdef(*,n)**) and the maximum number of ignitor types are both currently set with a parameter statement to 300.

For example, let's consider three *continuously* operating ignitors in block 1 at the following locations:

1st ignitor: (i = 12, j = 41, k = 17, number of block = 1)
 2nd ignitor: (i = 21, j = 14, k = 06, number of block = 1)
 3rd ignitor: (i = 31, j = 22, k = 25, number of block = 1)

and three *sparking* ignitors in block 2 at locations:

4th ignitor: (i = 21, j = 4, k = 71, number of block = 2)
 5th ignitor: (i = 12, j = 14, k = 60, number of block = 2)
 6th ignitor: (i = 13, j = 22, k = 25, number of block = 2),

where ignitors 4 and 6 start operating at time = 0.0, with a 10-s period, operating until 1000.0 s, while ignitor 5 starts operating at 2000.0 s with a 60.0-s period, operating until 8000.0 s.

The input stream would be input as follows:

\$xput

```
.  
.
.
ignitdef =    12, 41, 17,  1,  0,  
                21, 14,  6,  1,  0,  
                31, 22, 25,  1,  0,  
                21, 14, 71,  1,  1,  
                12, 14, 60,  1,  2,  
                13, 22, 25,  1,  1,
```

```
spxigdef =      0.0, 1000.0, 10.0,  
                2000.0, 8000.0, 60.0,
```

```
spxigdt  =      0.01,  
.
.
.
```

\$end

NOTE: In this case, the spark duration variable **spxigdt** is specified a different value than the default value of 0.001 s.

7.7.2 Sump Model

GASFLOW allows the user to specify a sump, which is basically a water film that has a constant thickness and a time-dependent temperature. We will again demonstrate the use of the sump model by example, but first we define some additional input variables and discuss some restriction for this version of GASFLOW.

First we will specify the additional input quantities. The first is used to define a constant water film thickness for the sump. This can be accomplished via the input array variable **cfilmdef** in NAMELIST group rheat. The variable is a two-dimensional array. The second index identifies the particular sump "area". For each **cfilmdef** specification, there are eight numbers required, which are input through the elements of the first array dimension with the following meaning:

cfilmdef (1,*)	Beginning i mesh index (cell face number).
cfilmdef (2,*)	Ending i mesh index (cell face number).
cfilmdef (3,*)	Beginning j mesh index (cell face number).

cfilmdef(4,*)	Ending j mesh index (cell face number).
cfilmdef(5,*)	Beginning k mesh index (cell face number).
cfilmdef(6,*)	Ending k mesh index (cell face number), <u>must</u> be the same as cfilmdef(5,*) .
cfilmdef(7,*)	Block number.
cfilmdef(8,*)	Film thickness (must be > 0.0 and < 100.0 cm).

The asterisk (*) should be replaced by an integer that identifies the particular **cfilmdef** definitions.

GASFLOW supports 500 definitions for **cfilmdef**.

The temperature of the sump, **sumptemp**, as a function of time, **sumptime**, can be controlled, and input in either degrees centigrade or kelvins. This can be accomplished via the input variable **nsumppts** and the array variables **sumptemp** and **sumptime** in NAMELIST group rheat. The definitions are as follows:

nsumppts	Number of sump temperatures in the sumptemp and sumptime arrays. Absolute value of nsumppts must be less than 100. If nsumppts < 0, then sumptemp values are in °C. If nsumppts > 0, then sumptemp values are in K.
sumptemp	Sump temperatures < 100 values.
sumptime	Times for the sump temperatures < 100 values.

GASFLOW supports a table of 99 paired temperature vs. time values to describe the sump temperature in either degrees centigrade or kelvins.

When specifying a sump, the user must

1. Define a horizontal wall (**walls**) that coincides exactly with the sump location. This wall cannot be located on the computational boundary nor coincide and share an obstacle edge.
2. Define a constant film thickness for the sump with the **cfilmdef** input variable that exactly coincides with the defined wall from 1.
3. Make certain that 'h2o' is listed in the problem composition, **mat**, and additionally 'h2o1' if the two-phase homogeneous equilibrium model (HEM) is to be used.
4. Input a sump temperature as a function of time through the paired single-dimensioned arrays **sumptemp** and **sumptime**.

An example is as follows:

1. We assume the same geometry as the example from Figure 3-6.
2. The sump is located at cell edge $k = 2$ and covers the entire bottom of the computation domain, i.e., $1 < i < 11$ and $1 < j < 11$, and a wall is constructed to indicate this sump.
3. The HEM is activated.
4. A constant film thickness is established to exactly coincide with the wall from 2.
5. Eight paired temperature and time values controlling the sump temperature in degrees centigrade are defined.

The input stream is shown here:

\$xput

mobs= 3,10, 1,11, 5, 8, 1, 1, ; solid obstacle

walls= 2,10, 1,11, 3, 3, 1, 2, ; horizontal wall
2, 2, 1,11, 6,10, 1, 2, ; vertical wall

1,11, 1,11, 2, 2, 1, 2, ; sump surface

holes= 5, 7, 4, 7, 6, 8, 1, 0,0, 0,0, 0,1, ; top hole
8, 9, 5, 6, 5, 8, 1, 0,0, 0,0, 1,1, ; thru hole
8, 9, 5, 6, 2, 4, 1, 1,1, 1,1, 1,1, ; wall hole

mat= 'h2', 'n2', 'o2', 'h2o', 'h2ol', ; components -> HEM

cfilmdef= 1,11, 1,11, 2, 2, 1, 5.0, ; sump film thickness

\$end

\$rheat

nsumppts= -8, ; sump paired temperature and time values in C

sumptemp= 25.0, 37.0, 50.0, 45.0, ; sump temperature
60.0, 62.0, 63.0, 63.0, ; sump temperature


```

sumptime=    0.0, 2800.0, 6200.0, 9480.0, ; sump time
            12600.0, 14400.0, 18000.0, 30000.0, ; sump time
.
.
.
$end

```

A display of the geometry including the sump is presented in Figure 7-1.

7.7.3 Recombiner Model

Recombiner boxes of the Nucelar Ingenieur Service (NIS) and Siemens design are modeled in the current version of GASFLOW. In addition, the Gesellschaft für Reaktorsicherheit (GRS) model for a single recombiner foil is included for completeness. We provide an example of the recombiner input, but first we'll discuss the details of the input variables.

Figure 7-1. Example of a sump

The location in the *i, j* plane for each recombiner is given in the **rcombdef** array described below. The **rcombdef** array is in the rheat NAMELIST group.

If **rcombdef(1,*)** = I1, **rcombdef(3,*)** = J1, and **rcombdef(5,*)** = K1, then the location of the recombiner in the *i, j* plane is at I1+1, J1+1. The energy source term due to the recombiner will be at location I1+1, J1+1, K1+1. The location of the cell for which the inlet conditions for the recombiner are determined will be at I1+1, J1+1, K1+1+offset, where offset is given by **rcombdef(10,*)**, and the velocity boundary condition for the recombiner flow rate will be specified at the z-face of cell I1+1, J1+1, K1+1+offset.

Therefore, to use inlet compositions from cells below the recombiner, offset must be negative. The inlet compositions are only used in the correlations for volumetric flow through the recombiner. For the Siemens type FR-90/1 designs, the offset is zero, so the composition, recombination, and energy release due to recombination are located in the catalytic reaction volume. For the GRS foil, the reaction volumes are all fluid cells with common cell faces to the foil.

rcombdef(1,*)	Beginning <i>i</i> mesh index (cell face number).
rcombdef(2,*)	Ending <i>i</i> mesh index (cell face number).
rcombdef(3,*)	Beginning <i>j</i> mesh index (cell face number).
rcombdef(4,*)	Ending <i>j</i> mesh index (cell face number).
rcombdef(5,*)	Beginning <i>k</i> mesh index (cell face number).
rcombdef(6,*)	Ending <i>k</i> mesh index (cell face number).
rcombdef(7,*)	Block number.
rcombdef(8,*)	Actual flow area into the recombiner irrespective of the computational mesh (not used with Siemens correlations 5 <= type <= 9)
rcombdef(9,*)	Recombiner type. = 1, implies NIS recombiner model with chemistry based on compositions at the location identified by the offset level, and forced volumetric flow through the recombiner based on the correlation for the NIS recombiner; = 2, implies Siemens recombiner model with chemistry based on compositions at the location identified by the offset level, and forced volumetric flow through the recombiner based on the correlation for the Siemens recombiner; = 3, implies Siemens recombiner model with chemistry based on compositions in the catalytic reaction volume according to the correlations for Siemens Type FR-90/1 recombiners; = 4, implies GRS recombiner foil model with chemistry based on compositions in the fluid cells with

common faces to the specified foil (currently not in the code)

=5, implies Siemens correlation for box type

FR-90/1-100, this box was used also in GX tests

=6, implies Siemens correlation for box type

FR-90/1-150

=7, implies Siemens correlation for box type

FR-90/1-320

=8, implies Siemens correlation for box type

FR-90/1-960

=9, implies Siemens correlation for box type

FR-90/1-1500

rcombdef (10,*)

Offset.

rcombdef (11,*)

Threshold value for hydrogen concentration. Hydrogen concentration must be above this value before recombiner will operate. If **rcombdef (11,*)** is zero and **rcombdef (9,*)** is one, then **rcombdef (11,*)** will be set to 0.008. If **rcombdef (11,*)** is zero and **rcombdef (9,*)** is two, then **rcombdef (11,*)** will be set to 0.031. If **rcombdef (11,*)** is zero and **rcombdef (9,*)** is ≥ 5 then **rcombdef (11,*)** will be set to 0.02. If **rcombdef (11,*)** is less than zero, then **rcombdef (11,*)** is the time in seconds when the recombiner will begin to operate.

rcombdef (12,*)

Time constant. If the time constant is less than or equal to zero for recombiner type 1, then the time constant is set to 1800.0 s. If the time constant is less than or equal to zero for recombiner type 2 or type ≥ 5 , then the time constant is set to 100 s. The time constant is not used for recombiner types 3 and 4.

The asterisk (*) should be replaced by an integer that identifies the particular recombiner definition.

GASFLOW supports 500 definitions for recombiners.

When specifying a recombiner, the user must

1. Define at least hydrogen, oxygen, and water vapor in the problem composition.
2. Define the recombiner box or foil.
3. Define the catalytic reaction volume, the type of recombiner, offset for the NIS and Siemens forced volumetric flow rate models, and other recombiner characteristics defined above.

An example is as follows:

1. We assume the same geometry as the example from Figure 3-2, but without the internal obstacles and walls other than the recombiner geometries.
2. Introduce a NIS recombiner box with 1-m² inflow area (see 1 in Figure 7-2).
3. Introduce a Siemens recombiner box of volumetric flow type with 200-cm² inflow area (see 2 in Figure 7-2).
4. Introduce a Siemens Type FR-90/1 recombiner box of reaction volume type with 0.96-m² inflow area (see 3 in Figure 7-2).
5. Introduce a GRS catalytic foil with 5-m² surface area on each side of the foil.

The input stream is shown here:

\$xput

```

.
.
.
walls= 9,10, 6, 6, 6, 9, 1, 2, ;NIS north wall (type 1)
        10,10, 5, 6, 6, 9, 1, 2, ;NIS east wall (type 1)
        9, 9, 5, 6, 6, 9, 1, 2, ;NIS west wall (type 1)
        9,10, 5, 5, 6, 9, 1, 2, ;NIS south wall (type 1)

        3, 4, 6, 6, 2, 8, 1, 2, ;Siemens north wall (type 2)
        4, 4, 5, 6, 2, 8, 1, 2, ;Siemens east wall (type 2)
        3, 3, 5, 6, 2, 7, 1, 2, ;Siemens west wall (type 2)
        3, 4, 5, 5, 2, 8, 1, 2, ;Siemens south wall (type 2)
        3, 4, 5, 6, 8, 8, 1, 2, ;Siemens top wall (type 2)

        8, 9, 6, 6, 2, 5, 1, 2, ;Siemens north wall (type 3)
        9, 9, 5, 6, 2, 5, 1, 2, ;Siemens east wall (type 3)
        8, 8, 5, 6, 2, 5, 1, 2, ;Siemens west wall (type 3)
        8, 9, 5, 5, 2, 5, 1, 2, ;Siemens south wall (type 3)

        6, 6, 4, 8, 7,10, 1, 2, ;GRS foil model (type 4)

```

```

mat= 'h2', 'n2', 'o2', 'h2o', 'h2ol', ; components -> HEM
.
.
.

```

\$end

\$rheat

```
.  
.
rcombdef= 9,10, 5, 6, 7, 8, 1, 1.0e+04,; NIS (type 1)
          1,   -2,   0,   0,           ; NIS (type 1)

          3, 4, 5, 6, 4, 5, 1, 2.0e+02,; Siemens (type 2)
          2,   -3,   0,   0,           ; Siemens (type 2)

          8, 9, 5, 6, 3, 4, 1, 9.6e+03,; Siemens (type 3)
          3,   0,   0,   0,           ; Siemens (type 3)

          5, 7, 4, 8, 7,10, 1, 5.0e+04,; GRS foil (type 4)
          4,   0,   0,   0,           ; GRS foil (type 4)
.  
.

```

\$end

A display of the geometry including the four types of recombiners is presented in Figure 7-2.

Figure 7-2. Examples of recombining types and setup for a vertical cut through the computational mesh at $j = 6$. Type 1 is a NIS forced volumetric flow recombining, type 2 is a Siemens forced volumetric flow recombining, and type 3 is a Siemens FR-90/1 design.

8 OPTIONS ON NUMERICAL SOLUTION PROCEDURE

8.1 Pressure Iteration

The numerical algorithm used in GASFLOW for solving the coupled fluid mass, momentum, and energy equations includes an implicit pressure iteration phase, which enables the code to simulate compressible flows without the computational time step being limited by the speed of sound. The implicit, iterative calculation is the solution of matrix equations arising from discretization of Poisson-type pressure equations by the preconditioned conjugate residual (PCR) method (see the Theory and Computational Model). The PCR algorithm is constructed such that the true solution is obtained after a *finite* number of iterations. However, that number is roughly the total number of computational cells, so in order to complete most practical problems in a reasonable amount of computer time, we have to specify certain error acceptance criteria to terminate the iteration procedure and move on to the next calculation phase. Note that the matrix solution has to be performed at each time cycle, so it is important to keep the iteration numbers reasonably low for complex, long transient problems.

The accuracy of an iterative solution is indicated by a residual vector r , which would be zero if the solution is exact. In GASFLOW, the iteration will stop if all the components in r are less than a specified error value. This convergence criterion or tolerance, ϵ , as defined for the actual matrix equations in the code, is a dimensionless quantity. However, to allow flexibility in controlling the iteration procedure for a wide range of problems, the code provides the following input variable in NAMELIST group `xput`:

epsi0 Initial value of ϵ . Default = 1×10^{-5} .

The user can also specify the maximum number of iterations after which the iteration stops and the calculation continues, regardless of whether the current matrix solution satisfies the convergence criterion. This is done via the following variable in NAMELIST group `xput`:

itmax Maximum number of iterations (per time cycle) allowed.
Default = 20.

The code prints out iteration and time-step information for each time cycle to a file called `cyclinfo`. (The same information is output to the terminal at a specified frequency. Section 9 describes how to set this frequency.) The following is part of a `cyclinfo` file that illustrates the information reported:

TIME	CYCLE	PITER	DELT	DMAX	EPSI	CODE (IBLK	I	J	K)
1.000E-02	1	33	1.000E-02	8.543E-06	1.000E-05	MX	1	1	1

2.000E-02	2	48	1.000E-02	6.030E-06	1.000E-05	MX	1	1	1	1
3.000E-02	3	45	1.000E-02	7.538E-06	1.000E-05	MX	1	1	1	1
4.000E-02	4	39	1.000E-02	7.900E-06	1.000E-05	MX	1	1	1	1
5.000E-02	5	41	1.000E-02	9.453E-06	1.000E-05	MX	1	1	1	1
6.000E-02	6	40	1.000E-02	8.151E-06	1.000E-05	MX	1	1	1	1
7.000E-02	7	40	1.000E-02	6.678E-06	1.000E-05	MX	1	1	1	1
8.000E-02	8	39	1.000E-02	8.235E-06	1.000E-05	MX	1	1	1	1
9.000E-02	9	38	1.000E-02	9.919E-06	1.000E-05	MX	1	1	1	1
1.000E-01	10	39	1.000E-02	8.282E-06	1.000E-05	MX	1	1	1	1

Here the first 10 computational time cycles are reported. The TIME column gives the problem time (s), and DELT is the time-step size. EPSI is the value of ϵ , and it can be seen that it remains unchanged. PITER is the number of pressure iterations that have been carried out. The pressure matrix solution in each cycle may or may not have converged. Whether convergence has been achieved is indicated by comparing EPSI with DMAX, which represents the largest component in the residual vector r . In this example, convergence has been achieved in every cycle. If the pressure iteration does not converge but the maximum error is still relatively large, the calculation will stop and the run will have to be repeated with a smaller time-step size. The printout under CODE provides the user with information on what is controlling the time-step size. An explanation for the CODE output is given below:

- CU The fluid Courant limit based on the x -direction velocity is controlling the time-step size.
- CV The fluid Courant limit based on the y -direction velocity is controlling the time-step size.
- CW The fluid Courant limit based on the z -direction velocity is controlling the time-step size.
- DA Energy diffusion is controlling the time-step size.
- DD Species mass diffusion is controlling the time-step size.
- DN Momentum diffusion is controlling the time-step size.
- IG Energy source term is controlling the time-step size.
- IN Indicates either the initial time-step size or that the user has fixed the time-step size with $\text{autot} = 0.0$.
- IT Pressure iteration is controlling the time-step size.
- MX Time-step size is at **deltmax**.
- NA Nothing controlling the time-step size.
- PJ The particle injection time-step size is controlling the time-step size.
- PU The particle velocity in the x -direction is controlling the time-step size.
- PV The particle velocity in the y -direction is controlling the time-step size.
- PW The particle velocity in the z -direction is controlling the time-step size.

The cell controlling the time-step size is identified under the IBLK, I, J, K headings. If a 1D duct component is controlling the time-step size, then iblk, j, and k will be printed as zero, and i will be the 1D duct cell number relative to the beginning of the 1D duct cells at duct 1, cell 1.

8.2 Time-Step Control

Because GASFLOW solves the time-dependent conservation equations, a calculation proceeds in finite time steps (also called cycles) until the problem end time or the specified maximum number of cycles has been reached. The end time and maximum number of cycles are defined by the following input variables (all variables discussed in this section are in NAMELIST group xput):

twfin	Time (s) at which the problem is finished. Default = 10.
maxcyc	Maximum allowable number of cycles. Default = 10.

Hence, if the user does not specify the above variables, the calculation will stop at a problem time of 10 s or after 10 cycles have been carried out, whichever occurs first.

How fast a problem can be completed depends on the time-step size Δt chosen. The initial, minimum, and maximum values of Δt to be used are defined with the following input variables:

delt0	Beginning time step size (s). Default = 0.02.
deltmin	Minimum time step size (s). Default = 1×10^{-4} .
deltmax	Maximum time step size (s). Default = 1×10^{30} .

The maximum and minimum values are used to define a range within which Δt can be varied during the calculation. Hence, the maximum Δt allowed is **deltmax**, and if Δt goes below **deltmin**, the problem will terminate. GASFLOW has an algorithm for adjusting the time increment during a calculation, or the user can force the code to use a fixed time-step size.

The numerical method used in GASFLOW treats most physical processes implicitly in time, except the advection and diffusion terms. Explicit treatment of these terms leads to the fact that the solution procedure is only stable if the time-step size Δt satisfies both the Courant criterion and the diffusion limit. To ensure numerical stability, the code limits Δt as follows:

$$\Delta t \leq \frac{1}{4} \min \left(\frac{\Delta_i}{u_i}, \frac{\Delta_j}{u_j}, \frac{\Delta_k}{u_k} \right) \quad (\text{Courant Limit})$$

$$\Delta t \leq 0.9 \left(\frac{\Delta_{\min}^2}{6 v_{\text{eff}}} \right) \quad (\text{Diffusion Limit}).$$

For the Courant limit, Δ_j and u_j , etc., are local cell spacing and velocity in all three directions. For the diffusion limit, v_{eff} is the effective diffusivity in a cell, and $\Delta_{\min}$ is the smallest dimension of that cell. The limit is applied, of course, only if the diffusion option is turned on (**idiffmom** and/or **idiffme** set to 1). In a problem where all diffusion processes and turbulence modeling are turned on, the effective diffusivity used in the diffusion limit will be the sum of the turbulent diffusivity and the largest of the molecular diffusivities for mass, energy, and momentum.

GASFLOW by default will try to adjust Δt to achieve maximum efficiency while satisfying the stability limits. The code increases or decreases Δt for the next computational cycle according to the number of pressure iterations required for convergence in the current cycle. If the number is greater than the input variable **itdowndt** (default = 50), then Δt will be decreased by 2%. If the iteration number is less than **itupdt**, then Δt will be increased by 2% in the next cycle.

If the user wants a fixed time-step size, the following input variable should be specified:

autot Option flag for automatic control of time-step size Δt :
 1.0 means Δt is adjusted by code during calculation (default);
 0.0 means the input **delt0** is used for Δt throughout
 calculation.

8.3 Advection Scheme

Each of the conservation equations solved by GASFLOW contains a convective flux term $\nabla \cdot (\phi \mathbf{u})$, where ϕ is the conserved quantity (mass density, internal energy, or momentum) and $\mathbf{u}$ is the velocity vector. The default numerical method for discretizing this term in space is the first-order donor-cell method. The donor-cell scheme is simple and fast, and does not suffer from the spurious oscillations caused by some higher-order schemes. However, the solution obtained has larger numerical diffusion error than those given by higher-order schemes. Therefore, the code provides an alternative advection scheme, which was originally developed by van Leer (see Ref. 2). The van Leer scheme is a second-order, slope-limiting method that has the *monotone* property. (A monotone scheme does not have the unphysical undershoots and overshoots exhibited by many higher-order methods.) In a problem where numerical diffusion errors need to be minimized, the more sophisticated van Leer scheme should be

substituted for the default simple donor-cell method to calculate the convective fluxes. This can be done via the following input variable in NAMELIST group xput:

ifv1 Option flag for turning on the van Leer advection scheme:
 0 means the donor-cell method will be used (default);
 1 means van Leer scheme will be used.

9 OUTPUT AND RESTART

9.1 Graphical Outputs

A typical problem solved by GASFLOW usually involves many computational cells and time steps. To analyze the calculation results by studying printed tables of numbers is useful in special cases but inefficient in many situations. Therefore, graphical display of results has been the primary focus of the GASFLOW code development effort with regard to user outputs. Graphical outputs have the advantage of showing data in a compact and comprehensive form so that important trends and phenomena can be easily identified.

The following is a description of various plotting capabilities available in the code. Most input variables regarding graphical outputs are in NAMELIST group `grafic`. One exception is the time interval between plots, which control the plotting frequency. This variable is in NAMELIST group `xput`:

pltdt Time interval (s) between successive sets of 1D profile, 2D contour, and 2D and 3D vector plots, if such plots are requested.
Default = 1.

All input variables discussed in the rest of this section are in NAMELIST group `grafic`, except where otherwise noted.

Sample plots are not given in this section. However, in Section 11, Sample Problems, extensive examples are shown of the graphics generated by GASFLOW. The plot file generated is called `pgf`.

9.1.1 Basic Plots

GASFLOW always produces five basic plots (each on a separate page or frame) of the mesh, and places them in the beginning of the plot file. These plots show, in order, 2D j-k, i-k, and i-j views and 3D "internal" and "external" perspective views of the mesh. In addition, toward the end of the plot file and in front of all *requested* time-history plots are five basic time-history plots. The first plot is the total mass time-history, and it appears on a single page. The next four are energy time-history plots, which all appear on one page. The quantities plotted are total internal energy, total kinetic energy, total energy loss due to steam condensation, and total internal plus kinetic energy. The basic time-history plots will be generated if the problem end time, **twfin**, is greater than **thdt**, the time interval at which time-history data are written (see discussion of time-history plots below).

9.1.2 Geometry Plots

In Section 3.3, we discuss how to use GASFLOW to just plot the mesh plus all defined walls and obstacles. This is done by setting **maxcyc** in NAMELIST group **xput** to a negative number. The set of geometry plots obtained depends on the actual value of the negative number according to Table 3-1.

9.1.3 Time-History Plots

Time-history plots of selected solution variables can be requested with the following input array variable:

- thp(1,*)** For 3D plot variables, *i* mesh index (cell number or cell face number). For 1D duct plot variables, the cell number or cell face number relative to the beginning of the duct.
- thp(2,*)** For 3D plot variables, *j* mesh index (cell number or cell face number). For 1D duct plot variables the duct number.
- thp(3,*)** *k* mesh index (cell number or cell face number).
- thp(4,*)** Block number.
- thp(5,*)** Solution variable to be plotted. Choose one of the character strings (enclosed in single quotes) given in Table 9-1.
- thp(6,*)** Gas species name (symbol representing one of the species defined by **mat** in NAMELIST group **xput**) enclosed in single quotes. This variable is used only if **thp(5,*)** has been set to 'rsn', 'mf', or 'vf'. Instead of a character string representing the species name, a component number (based on the order in which the species is defined in the **mat** array) can alternatively be entered here.

Table 9-1. Solution variables available for plotting

Symbol	Quantity to be plotted
'pn'	Pressure.
'rn'	Mixture density.
'rsn'	Species density.
'cmn'	Cell mass.
'sien'	Specific internal energy.
'un'	i- (x or r) velocity component.
'vn'	j- (y or θ) velocity component.
'wn'	k- (z-) velocity component.
'tk'	Fluid temperature, K.
'mf'	Species mass fraction.
'vf'	Species volume fraction.
'vmag'	Velocity magnitude.
'mdotx'	Mass flow rate in i- (x or r) direction.
'mdoty'	Mass flow rate in j- (y or θ) direction.
'mdotz'	Mass flow rate in k- (z) direction.
'vdotx'	Volume flow rate in i- (x or r) direction.
'vdoty'	Volume flow rate in j- (y or θ) direction.
'vdotz'	Volume flow rate in k- (z) direction.
'delpx'	Pressure difference in i- (x or r) direction.
'delpy'	Pressure difference in j- (y or θ) direction.
'delpz'	Pressure difference in k- (z) direction.
'delt'	Time-step size (does not depend on spatial location).
'diffp'	Cell pressure minus an ambient or reference pressure defined by pamb0 (default = 1.01325×10^6 dynes/cm ²) in NAMELIST group xput.
'pstag'	Stagnation pressure ($\rho u ^2/2$) minus pamb0 .
'mu'	Effective (molecular and turbulent) viscosity.
'nu'	Effective viscosity divided by fluid density.
'tke'	Turbulent kinetic energy, only valid if tmodel has been set to 'ke'.
'eps'	Rate of dissipation of turbulent kinetic energy, only valid if tmodel has been set to 'ke'.
'tsat'	Saturation temperature, K.
'psat'	Saturation pressure, dynes/cm ² .
'rh'	Percent relative humidity.
'tc'	Fluid temperature, °C.
'diffT'	Fluid temperature minus the saturation temperature.
'insht'	Instrument cooling heat removal.

The second dimension in the **thp** array allows more than one definition of time-history plot request, and the first dimension consists of six elements that define a particular time-history plot. The variables **thp(1,*)**, **thp(2,*)**, and **thp(3,*)** are i-, j-, and k-indices, respectively, that define the spatial location where a solution quantity is to be plotted as a function of time. The logical indices can either represent cell number or cell face number, depending on the quantity being plotted. The reason for this is that in GASFLOW components of velocity, mass flow rate, volume flow rate, and pressure gradient are defined at cell faces in the corresponding direction, while all scalar quantities such as densities, pressure, temperature, etc., are defined at cell centers. (See Figure 3-1 for cell numbering convention.) Consider the following examples:

```

thp(1,1) = 4, 8, 2, 1, 'sien', 0,
thp(1,2) = 3, 4, 5, 1, 'un', 0,
thp(1,3) = 3, 4, 5, 1, 'vn', 0,
thp(1,4) = 3, 4, 5, 1, 'vdotz', 0,
thp(1,5) = 3, 4, 5, 1, 'mf', 'h2o',
thp(1,6) = 4, 8, 2, 1, 'vf', 1,
thp(1,7) = 3, 2, 0, 0, 'un', 0,

```

The first **thp** definition asks for the time-history plot of internal energy at cell (4,8,2). The fifth time-history plot is that of the mass fraction of water vapor at cell (3,4,5). The sixth time-history plot is that of the volume fraction of fluid component 1 (component identification numbers are determined by the order in which the species are listed in the definition of the **mat** array in NAMELIST group **xput**) at cell (4,8,2). The second, third, and fourth time-history plots are those of components of vector quantities, and therefore the location should indicate a cell face. For example, the second plot is that of the *i* velocity component at the *i* = 3 face of the cell with a *j*-index of 4 and a *k*-index of 5. Similarly, the fourth plot is that of the volume flow rate in the *z*-direction at the *k* = 5 face of the cell with an *i*-index of 3 and a *j*-index of 4. The seventh time-history plot is for the velocity at the third cell face of the second duct.

To conserve space, four time-history plots are grouped on each page or frame. The data to be plotted are first stored in binary files called **pth1**, **pth2**, ... (one file for four time histories), then read back in and plotted after the calculation is completed. The frequency at which time-history data are written and subsequently plotted is controlled by the following input variable:

thdt Time interval (s) at which time-history data are written and plotted.
 Default = 1×10^{100} .

The time-history plots by default have grid lines for both the horizontal (time) and vertical (solution quantity) axes. To turn the grid lines off, the user should set the following input variable:

gline On/off switch for grid lines on time-history plots. The options are 'on' (default) or 'off'.

Besides time histories of the fluid solution quantities given in Table 9-1, the user can also plot the surface temperature of a solid heat structure as a function of time. This is done via the following input variable:

htthp(1,*)	i-index of fluid cell in contact with the heat structure.
htthp(2,*)	j-index of fluid cell in contact with the heat structure.
htthp(3,*)	k-index of fluid cell in contact with the heat structure.
htthp(4,*)	Block number.
htthp(5,*)	Heat structure type. Choices are 'slab', slab heat structure; 'sink', sink heat structure; 'wall', wall heat structure.
htthp(6,*)	The side of the fluid cell which coincides with the solid surface whose temperature is to be plotted. This entry is only used if htthp(5,*) has been set to 'slab' or 'wall', because sink structures are assumed to be distributed in the fluid cell. Choices are 'east', +i side of fluid cell; 'west', -i side of fluid cell; 'north', +j side of fluid cell; 'south', -j side of fluid cell; 'top', +k side of fluid cell; 'bottom', -k side of fluid cell.

Of course heat-transfer calculations have to be invoked (by setting **ihtflag** = 1 in NAMELIST group rheat) for these definitions to be effective. To illustrate the use of **htthp** definitions, consider the following examples:

```
htthp = 9, 5, 7, 1, 'slab', 'top',
        2, 4, 8, 1, 'sink', 0,
        4, 5, 4, 1, 'wall', 'east',
```

The first heat-transfer time-history plot is the temperature at the surface of the slab heat structure that coincides with the +k face of the fluid cell (9,5,7). The second plot is that of the surface temperature of the distributed sink heat structure in fluid cell (2,4,8). The

third plot will show the time history of the surface temperature of the wall heat structure that is on the +i side of fluid cell (4,5,4).

9.1.4 Profile Plots

The user can plot all of the solution variables listed in Table 9-1 (except 'delt') as a function of any one of the three spatial coordinates through an arbitrary region of the mesh. To define the line along which the profile of the quantity of interest is to be plotted, GASFLOW uses the concept of *points*. A line parallel to any of the axes can be defined by two points with the same spatial coordinates in two directions. For example, points with mesh indices (3,4,1) and (3,4,10) define the line going from $k = 1$ to $k = 10$, at $i = 3$ and $j = 4$. (As described in the following paragraphs, 2D contour plots and 2D and 3D vector plots also need points to define the region over which the solution quantity is plotted.) Points for plotting purposes can be defined with the following input variable:

pnt(1,*)	i mesh index.
pnt(2,*)	j mesh index.
pnt(3,*)	k mesh index.
pnt(4,*)	Block number.

Note that the first dimension of the array **pnt** contains four elements to define the point location, and the second dimension identifies the point. Once the points have been defined, the user can specify what the 1D profile plots are via the following input variable:

p1d(1,*)	Identification number of the first point (the point number is the second index of the corresponding pnt definition).
p1d(2,*)	Identification number of the second point (the point number is the second index of the corresponding pnt definition).
p1d(3,*)	Solution variable whose 1D profile is to be plotted. Choose one of the symbols (enclosed in single quotes) listed on Table 9-1, except 'delt'.
p1d(4,*)	Gas species name (symbol representing one of the species defined by mat in NAMELIST group xput) enclosed in single quotes. This variable is used only if p1d(3,*) has been set to 'rsn', 'mf', or 'vf'. Instead of a character string representing the species name, a component number (based on the order in which the species is defined in the mat array) can alternatively be entered here.

Note that the first point should not have higher mesh index values than the second point, or an error will result. Consider the following input, which illustrates how to use point definitions to define 1D profile plots:

```

pnt(1,1) = 3, 4, 1, 1,
pnt(1,2) = 3, 4, 10, 1,
pnt(1,3) = 2, 6, 7, 1,
pnt(1,4) = 15, 6, 7, 1,

p1d(1,1) = 1, 2, 'pn', 0
p1d(1,2) = 1, 2, 'rsn', 'h2',
p1d(1,3) = 3, 4, 'tk', 0,

```

Here four points are defined, with the first two and the last two points being "colinear" pairs. Therefore the two pairs of points, 1 and 2, and 3 and 4, can be used to define 1D profile plots. The first profile plot is that of the fluid pressure along the line going from point 1 to point 2. The second profile plot is that of the hydrogen species density along the same line. The third profile plot is that of the fluid temperature along the line defined by points 3 and 4.

Similar to time-history plots, the code by default puts vertical and horizontal grid lines on 1D profile plots. This can be changed via the following input variable:

glinep1d On/off switch for grid lines on 1D profile plots. The options are 'on' (default) or 'off'.

In problems involving heat transfer, the user can request plotting of the temperature profile in the solid heat structure via the following input variable:

```

ht1dp(1,*)    i-index of fluid cell in contact with the heat structure.
ht1dp(2,*)    j-index of fluid cell in contact with the heat structure.
ht1dp(3,*)    k-index of fluid cell in contact with the heat structure.
ht1dp(4,*)    Block number.
ht1dp(5,*)    Heat structure type. Choices are
               'slab', slab heat structure;
               'sink', sink heat structure;
               'wall', wall heat structure.
ht1dp(6,*)    The side of the fluid cell which is in contact with the heat
               structure whose temperature profile is to be plotted. This
               entry is only used if ht1dp(5,*) has been set to 'slab'
               or 'wall', because sink structures are assumed to be
               distributed in the fluid cell. Choices are
               'east',    +i side of fluid cell;
               'west',    -i side of fluid cell;
               'south',   +j side of fluid cell;
               'north',   -j side of fluid cell;

```

```

'bottom', +k side of fluid cell;
'top',    -k side of fluid cell.

```

For slab and wall heat structures, the temperature profile along the entire depth of the structure is plotted. For sink heat structures, only half of the profile is plotted, because it is assumed in the calculation that the temperature distribution is symmetric about the centerline. Consider the following input:

```

ht1dp = 9, 5, 7, 1, 'slab', 'top',
        2, 4, 8, 1, 'sink', 0,
        4, 5, 4, 1, 'wall', 'east',

```

The first heat-transfer 1D profile plot is that of the temperature in the slab heat structure that is in contact with the +k face of fluid cell (9,5,7). The second plot is that of the surface temperature profile inside the distributed sink heat structure in fluid cell (2,4,8). The third plot will show the temperature distribution within the wall heat structure that is on the +i side of fluid cell (4,5,4).

9.1.5 Contour Plots

It is often useful to plot the contour of a solution quantity on a plane (for example, to identify "hot spots" in certain calculations). Two-dimensional contour plots are defined in basically the same way as 1D profile plots. Two points with the same mesh index in one direction (i.e., a pair of so-called "coplanar" points) are used to define the plane where data are to be taken for the contour plot. Once some (coplanar) points have been defined, contour plots can be requested via the following input variable:

c2d(1,*)	Identification number of the first point (second index of the corresponding pnt definition).
c2d(2,*)	Identification number of the second point (second index of the corresponding pnt definition).
c2d(3,*)	Solution variable for the 2D contour plot. Choose one of the symbols (enclosed in single quotes) listed on Table 9-1, except 'delt'.
c2d(4,*)	Gas species name (symbol representing one of the species defined by mat in NAMELIST group xput) enclosed in single quotes. This variable is used only if c2d(3,*) has been set to 'rsn', 'mf', or 'vf'. Instead of a character string representing the species name, a component number (based on the order in which the species is defined in the mat array) can alternatively be entered here.

Similar to 1D profile plots, the two points specified in **c2d(1,*)** and **c2d(2,*)** should be chosen such that the mesh index values increase in the direction from the first to the second point, or an error will occur. The following input illustrates how to define points and how to specify 2D contour plots:

```
pnt(1,1) = 3, 1, 1, 1,
pnt(1,2) = 3,12, 10, 1,
pnt(1,3) = 2, 1, 7, 1,
pnt(1,4) = 15,12, 7, 1,

c2d(1,1) = 1, 2, 'pn', 0
c2d(1,2) = 1, 2, 'rsn', 'h2',
c2d(1,3) = 3, 4, 'tk', 0,
```

The first two points have the same i-index, so they define a plane normal to the i-direction, at $i = 3$, ranging from $j = 1$ to 12 and from $k = 1$ to 10. The third and fourth points have the same k-index, so they define a plane normal to the k-direction, at $k = 7$, ranging from $i = 2$ to 15 and $j = 1$ to 12. The first contour plot is that of the fluid pressure on a plane defined by points 1 and 2. The second contour plot is that of the hydrogen species density on the same plane. The third profile plot is that of the fluid temperature on the plane defined by points 3 and 4.

The contour lines are plotted in color. GASFLOW uses seven colors; these are, in the order of decreasing contour values, red, yellow, green, cyan, blue, magenta, and black. The user can specify the number of contour levels via the following input variable:

ncontur Number of contour levels. Default = 7.

9.1.6 Velocity Vector Plots

Using the concept of points, as discussed above for profile and contour plots, the user can also specify velocity vector plots. There are two types of vector plots available. Two-dimensional velocity vector plots show the velocity magnitude and direction on a plane defined by two coplanar points. Three-dimensional velocity vector plots show the velocity magnitude and direction in a volume, which can be specified by defining two points that locate its diagonal vertices. The length of the shaft, the size of the arrowhead, and the color of the vector are made proportional to the velocity magnitude. Therefore, the vectors in regions with tiny velocities will show up as black dots.

To specify 2D velocity vector plots, the user should define the following:

v2d(1,*) Identification number of the first point (second index of the corresponding **pnt** definition).

- v2d(2,*)** Identification number of the second point (second index of the corresponding **pnt** definition).
- v2d(3,*)** Flag for frame advance. This option (if set to 0) can be used to overlay the vector plot with the next plot for special presentation. However, it is advised that this flag be set to 1 so that the vector plot will appear by itself on a single frame.

Note that the two points should have the same mesh index in one direction, and the second point should have larger coordinates than the first, or an error will occur. The following is an example showing the use of **pnt** and **v2d** to define a 2D velocity vector plot:

```
pnt(1,1) = 1, 1, 2, 1,
pnt(1,2) = 9, 9, 2, 1,

v2d(1,1) = 1, 2, 1,
```

In this example, velocity vectors will be plotted on the plane $k = 2$, ranging from $i = 1$ to 9, $j = 1$ to 9.

To specify 3D velocity vector plots, the user should define the following:

- v3d(1,*)** Identification number of the first point (second index of the corresponding **pnt** definition).
- v3d(2,*)** Identification number of the second point (second index of the corresponding **pnt** definition).
- v3d(3,*)** Flag for frame advance. This option (if set to 0) can be used to overlay the vector plot with the next plot for special presentation. However, it is advised that this flag be set to 1 so that the vector plot will appear by itself on a single frame.
- v3d(4,*)** Number of the 3D view coordinates definition, i.e., second index of the corresponding **viewcrds** definition, if **viewcrds** has been specified. If there is no definition of **viewcrds**, this number should be set to 1, in order to use the default 3D viewing coordinates, which are calculated by the code based on the dimensions of the mesh.

Note that the two points specified above should be chosen such that the second point has higher coordinate values than the first, or an error will occur. The following is an example showing the use of **pnt** and **v3d** to define a 3D velocity vector plot:

```
pnt(1,1) = 1, 1, 1, 1,
pnt(1,2) = 9, 9, 9, 1,
```

v3d(1,1) = 1, 2, 1, 1,

In this example, velocity vectors will be plotted in a volume ranging from $i = 1$ to 9, $j = 1$ to 9, $k = 1$ to 9. Because **v3d(4,1)** is set to 1 (and there is no definition of **viewcrds**), the viewing coordinates for the 3D plot will be determined by the code based on the dimensions of the computational domain.

If there are too many computational cells in the region defined by either **v2d** or **v3d**, the velocity vectors, which by default are plotted at each cell location, will appear cluttered. In this case, the user can use the following input variable to reduce the number of vectors that are plotted:

iinc	Increment in i-index between adjacent vectors. Default = 1.
jinc	Increment in j-index between adjacent vectors. Default = 1.
kind	Increment in k-index between adjacent vectors. Default = 1.

For example, to plot velocity vectors at every *other* cell location, one should set **iinc**, **jinc**, and **kind** all to 2.

9.1.7 Graphic and Tabular Particle Data Output

Two kinds of graphic output are used to interpret the results of particle computations. One of these displays particles plotted in a perspective view plot of the computational domain. Any combination of particle classes may be selected for each plot. The three-dimensional volume of particles is integrated along each line of sight and plotted onto the two-dimensional plane. A varied selection of eye points from which to view the mesh and particles at selected times in the transport of the particles can give a qualitative interpretation of the particle behavior. Another useful graphic display is the time-history plots of particle number, mass, volume fraction, mass fraction, and mass deposited in selected mesh cells. In addition, the time-history data is written on an output tape, PTHDATA, and is available for examination.

The input parameter definitions are given in Appendix D, which describes NAMELIST group **grafic** input. However, a summary listing will be given here of the input parameters specifically used in the plots of perspective views of particles.

The following input is for perspective views of particles and is in the **grafic** NAMELIST group:

ippka(np)	Selects the package (i.e., particle class, size, or deposition plane) to be plotted. The input corresponds to the array mpac definition, given below. Default = 1.
------------------	---

ipview(np)	Viewpoint for particle plot, i.e., viewpoint 'nv' defined in viewcrds . Default = 1. (Input value corresponds with the array <code>pplt(2,100)</code>)
nap(np)	Number of film frame advances between particle plots 'np'. Default = 1.
npplts	The number of particle packages (i.e., classes or deposition planes) to be plotted. Default = 0.
pplt(2,100)	Perspective plot viewpoint, i.e., the viewpoint 'nv' defined in viewcrds . (Corresponds with ipview and is used for perspective plots in general.)
viewcrds(6,nv)	3D view description, object center and eye coordinates: xco, yco, zco, xeye, yeye, zeye. Default = 1.0.

The following input is for time-history particle plots and is in the `grafic` NAMELIST input group:

gline	Indicator for grid line backgrounds on plots: 'on' default or 'off'.
thdt	Time increments between time-history data points. Default = 10^{100} s.
pthp(1,*)	i index of fluid cell.
pthp(2,*)	j index of fluid cell.
pthp(3,*)	k index of fluid cell.
pthp(4,*)	Block number.
pthp(5,*)	Particle data to be plotted: 'pnc', particle number concentration; 'pmc', particle mass concentration; 'pmt', total mass in fluid cell; 'pmf', particle mass fraction; 'pvf', particle volume fraction; 'pmd', particle mass deposited; 'pmntr', particle cloud mass detected at each monitor.
pthp(6,*)	Particle class 0, all classes; >0, particle class number.
pthp(7,*)	Particle size number 0, all sizes for class pthp(6,*) ; >0, particle size number.
pthp(8,*)	Particle mass deposited on cell faces (for 'pmd' only): 0, all deposited particles; >0, particles deposited on one of the six possible cell faces as defined below: 11, deposited on east face of cell; 21, deposited on west face of cell; 12, deposited on north face of cell;

pthp(9,*)

pthpt0

22, deposited on south face of cell;

13, deposited on top face of cell;

23, deposited on bottom face of cell.

Monitor number for particle monitor (for 'pmntr' only) = particle cloud monitor number to be plotted when pthp(5,*) = 'pmntr'.

Time-history plot initial time (used for runs from restart tapes).

The procedure to set up perspective view particle plots:

1. Input the selected viewpoints, **viewcrds**. The object center coordinates should be the *x*-, *y*-, *z*-coordinates at the center of the mesh. The eye points selected should position the viewpoint several mesh dimensions from the mesh center.

2. Select the number of particle plots wanted, **npplts**.

3. Select the particle class number and the class size number or the cell faces with deposited particles that are to be plotted, **ippka**. For the particle class and size, the input number is a two-digit integer. The rightmost digit corresponds to the particle class number. The leftmost digit corresponds to the class size number. For example, to plot class 2, size 3 particles the input would be **ippka(1) = 32**. To plot all particles of all classes and sizes set **ippka(1) = 0**. The cell face flags are those defined in the array **mpac(n)**. The input value for **ippka** is the value of **mpac(n) + 100**. For example, to plot all the particles deposited on the "floor," i.e., the bottom face of each cell in the *k* = 2 plane of the mesh, input **ippka(1) = 123**. This would also plot all particles deposited on the top side of an obstacle.

4. Select the perspective view for each plot from the **viewcrds** input. This number is set into both **ipvew** and **pplt**. For the second perspective view being defined, if **viewcrds** 3 is chosen, **ipvew(2) = 3** and **pplt(1,2) = 3**.

5. The number of film frames advanced, **nap**, is typically 1, which is the default value. However, if the frames are to be cut and mounted for slides, advancing 2 frames between each plot is advisable.

The procedure to set up time-history plots:

1. Choose whether or not the plots will have grid lines, **gline**. The default is 'on'.

2. Select the time interval between data points, **thdt**.

3. Select the time-history plot initial time, **pthpt0**. (Note that the last character in this parameter is a zero.) This has a default value of 0.0, but can be set to the restart time when a steady-state flow field has been established with **solatype** = 0, and a restart with **solatype** = 2 is being used to initialize particle input. For example, if restarting from a tape with a dump time of 1.32 s, the particle time-history plots will start at time 0 if **pthpt0** is set to 1.32.

4. Select the particle time-history data to be plotted, **pthp**, and input using the description of the input parameter given above. For example, to plot the total mass of particle class 2, size 3 in fluid cell $i = 5$, $j = 4$, $k = 12$, block #1, enter

```
pthp(1,1) = 5, 4, 12, 1, 'pmt', 2, 3, 0,
```

When particle cloud mass is plotted and **pthp(*,5)** is 'pmntr', the other **pthp** input is not required; however it is recommended for plot identification.

9.1.8 Graphic Display of Detonation Criterion

A new methodology has been developed (Ref. 3) to evaluate the safety of ignitor implementation in complex containment geometries. The method consists of the following steps:

1. determination of bounding H₂/steam sources,
2. high-resolution analysis of the three-dimensional transport calculation,
3. evaluation of the detonation potential at the time of ignition,
4. optimization of the ignitor system such that only early ignition and nonenergetic combustion occurs, and
5. modeling of the continuous deflagration process during H₂ release.

In order to evaluate this new methodology and determine detonation possibilities in complex-shaped rooms for complicated geometries with GASFLOW, we have implemented a generalized input in the NAMELIST input block **xput**. GASFLOW will automatically plot for each room defined by the following input (1) the characteristic h₂ cloud dimension, (2) the detonation cell size, (3) the detonation parameter, and (4) the equivalence ratio.

To activate this new methodology, the user may define multiple rooms in several ways with the **iroomdef** two-dimensional array. Each room segment volume is defined by eight entries in the **iroomdef** input array.

iroomdef(1,*)	Beginning i mesh index (cell face number).
iroomdef(2,*)	Ending i mesh index (cell face number).
iroomdef(3,*)	Beginning j mesh index (cell face number).

iroomdef(4,*)	Ending j mesh index (cell face number).
iroomdef(5,*)	Beginning k mesh index (cell face number).
iroomdef(6,*)	Ending k mesh index (cell face number).
iroomdef(7,*)	Block number.
iroomdef(8,*)	Actual room number:
	> 0 implies positive volume;
	< 0 implies negative volume.

The asterisk (*) should be replaced by an integer that identifies the particular **iroomdef** definition.

GASFLOW supports 300 definitions of **iroomdef** and 25 separate and distinct different rooms.

The utility of the **iroomdef** input is best demonstrated by an example or two as follows:

Consider a two-dimensional computational mesh that has 9 fluid cells in the x-direction and 5 fluid cells in the y-direction (Figure 9-1). There is a complex-shaped room in this mesh shown by the obstacles.

Figure 9-1. Example problem to demonstrate the utility of the **iroomdef** input option

One way to define this room is

\$xput

```
.  
.  
.  
mat= 'h2', 'n2', 'o2', 'h2o', 'h2ol', ; components -> HEM
```

```
mobs=      1,  2,   1,  3,   1,  2,   1,  1,  
            2,  4,   1,  2,   1,  2,   1,  1,  
            3,  5,   4,  6,   1,  2,   1,  1,  
            7,  8,   1,  3,   1,  2,   1,  1,
```

```
iroomdef=  1,  3,   3,  6,   1,  2,   1, +1,  
            4,  7,   1,  2,   1,  2,   1, +1,  
            2,  7,   2,  3,   1,  2,   1, +1,  
            8, 10,   1,  3,   1,  2,   1, +1,  
            3, 10,   3,  4,   1,  2,   1, +1,  
            5, 10,   4,  6,   1,  2,   1, +1,
```

```
.  
.  
.  
$end
```

Another way to define this same room is

\$xput

```
.  
.  
.  
mat= 'h2', 'n2', 'o2', 'h2o', 'h2ol', ; components -> HEM
```

```
mobs=      1,  2,   1,  3,   1,  2,   1,  1,  
            2,  4,   1,  2,   1,  2,   1,  1,  
            3,  5,   4,  6,   1,  2,   1,  1,  
            7,  8,   1,  3,   1,  2,   1,  1,
```

```
iroomdef=  1, 10,   1,  6,   1,  2,   1, +1,  
            1,  2,   1,  3,   1,  2,   1, -1,  
            2,  4,   1,  2,   1,  2,   1, -1,  
            3,  5,   4,  6,   1,  2,   1, -1,  
            7,  8,   1,  3,   1,  2,   1, -1,
```

\$end

As another example, let's consider the same computational domain, but make the single room into three rooms by the vertical walls at $i = 4$ and $i = 7$ (shown by the heavy thick lines in Figure 9-2).

Input this geometry as follows:

\$xput

```
mat= 'h2', 'n2', 'o2', 'h2o', 'h2ol', ; components -> HEM
```

```

mobs=      1,  2,   1,  3,   1,  2,   1,  1,
           2,  4,   1,  2,   1,  2,   1,  1,
           3,  5,   4,  6,   1,  2,   1,  1,
           7,  8,   1,  3,   1,  2,   1,  1,

```

```

walls=      4,  4,   3,  4,   1,  2,   1,  1,
           7,  7,   3,  5,   1,  2,   1,  1,

```

Figure 9-2. Another example problem to demonstrate the utility of the **iroomdef** input option

```

iroomdef=      1,  4,  2,  6,  1,  2,  1, +1,
                1,  2,  2,  3,  1,  2,  1, -1,
                3,  4,  4,  6,  1,  2,  1, -1,

                4,  7,  1,  4,  1,  2,  1, +2,
                5,  7,  4,  6,  1,  2,  1, +2,

                7, 10,  1,  6,  1,  2,  1, +3,
                7,  8,  1,  3,  1,  2,  1, -3,

$end

```

9.2 Printed Output

In addition to graphical outputs, GASFLOW provides printed outputs for each calculation. The text file `meshmap`, which was discussed in Section 3.4, contains cell-number, cell-status (i.e., whether each cell is a fluid or boundary cell, or if its faces are open to flow, etc.), and neighbor-list information. Another printed output file is `cyclinfo`, which lists iteration and time-step information at each computational time cycle. This file was discussed in Section 8.1. A third printed output file prints out the input NAMELIST variables used in the code, and is called `tape16`.

The main printed output file is `gfout`. In the beginning of the file, the code version number and the date of the run are printed. Then the values of main input variables are listed, followed by tables showing mesh coordinates and cell spacing (edge-to-edge and center-to-center). The plotting output specifications are then echoed. Next, the calculated fluid velocity (all three components), pressure, and density at each cell are listed at selected time intervals. This time interval is defined by the following variable in NAMELIST group `xput`:

```

prtdt    Time interval (s) between printing of the fluid solution field (all
            velocity components, pressure, and density) to the output file
            gfout. Default = 1000.

```

Because in most 3D problems the listing of fluid solution at all cells can be quite long, the default printed output interval has been chosen to be reasonably large (1000 s) to avoid unintended, excessively long output listing.

The `gfout` file also prints out the time, cycle number, and the file name (`gfdn`, where n is an integer) whenever a restart dump file is written. At the end of `gfout`, the total central processing unit (CPU) time used is reported, as well as the per-cell, per-cycle CPU time.

9.3 Output to Terminal

Besides graphical and text file outputs, GASFLOW also writes output to a terminal (or the FORTRAN standard output unit that, under a UNIX-type operating system, can be "piped" to a specified file). This output is intended to help the user monitor the calculation as it is being carried out. Any error messages will also be given here. After some banner messages that include identification of code version, the time-step and pressure iteration information is printed. The information given is the same as that in the `cyclinfo` file (see Section 8.1); however, instead of printing at every computational time cycle, the terminal output is printed at a selected frequency, which is defined by the following input variable in NAMELIST group `xput`:

itttfreq Number of cycles between printing of time-step and pressure iteration information to the terminal. Default = 20.

When calculation is finished, the code prints to the terminal the same timing information as in the output file `gfout` (discussed at the end of the above section). In addition, it reports the number of restart dump files written and the number of pages (or frames) generated in the plot file `pgf`.

9.4 Restart

Because GASFLOW is capable of solving complex, large problems, it may take a large amount of computer time to finish a problem. Therefore, the code provides a restart capability so that a long calculation can be divided into a series of shorter runs. A restart dump file is always produced at the end of each run. However, the user can specify that additional restart files be written at selected time intervals. This is done via the following input variable in NAMELIST group `xput`:

tddt Time interval (s) at which restart dump files are written.
Default = 10.

Therefore, one restart dump file, called `gfd1`, will be written if the problem end time (specified by **twfin** in NAMELIST group `xput`) is less than **tddt**. If **twfin** is larger than **tddt**, then `gfd1` will be the restart file written at time **tddt**. The next restart files, `gfd2`, `gfd3`, etc., will be written at times that are multiples of **tddt**. Hence, the restart file that contains the final solution will have the name `gfdn`, where *n* is the total number of restart files produced.

To specify that a run is to begin from the solution stored in a restart dump file, the user should define the following variable in NAMELIST group `xput`:

nrsdump Number that appears in the name of the restart dump file
that is to be read in. For example:
0, new problem, not a restart run (default);
1, read from restart file gfd1;
2, read from restart file gfd2.

10 GENERAL USER GUIDANCE

Before setting up a complex problem, it is always helpful for the user to run some similar but simpler problems first. Doing so will allow the user to quickly gain insight to the problem and verify the majority of the input deck. Common ways of simplifying a problem include the following:

1. Use a coarser mesh. In a heat-transfer problem, also coarsen the nodalization in all heat conducting solids.
2. Use default physical models, which are normally the simplest options.
3. Reduce the problem time, i.e., **twfin**.
4. Relax the pressure iteration error criterion, i.e., increase **epsi0**.

Through a series of runs in which complexities are added successively, the user will become more familiar with the problem, which should help in analyzing results from the final calculations.

For large-scale, long-running problems, it is advisable to use the restart capability of the code to break the problem into a series of shorter runs. The user should check the calculations with extensive graphical display of the solution both as a function of time and space. Note that the code can be used as a graphics postprocessor for the data in a restart dump file, which is always written at the end of a run.

In case a calculation gives unphysical results, the user should review the input deck. Most problems arise from incorrectly specifying initial and boundary conditions. The user should ask herself or himself the following:

1. Have all fluid cells in the entire domain been given the correct initial conditions via defining the **gasdef** array appropriately?
2. Are there any open boundaries across which inflow can occur? If so, do the boundary cells have the appropriate fluid conditions defined via **gasdef**?
3. Does reducing Δt (by decreasing **delt0** and/or **deltmax**) give the same unphysical results?
4. Does tightening the pressure iteration convergence criterion (i.e., reducing **epsi0**) give the same unphysical results?

Finally, the user should note that GASFLOW always solves the time-dependent conservation equations. A *steady-state* calculation option is not available in the code. However, this should not prevent the user from solving steady-state (time-independent) problems with the code. The initial conditions in such a calculation will constitute an initial guess, and each time cycle will represent an iteration toward the steady solution. Time-history plots of the relevant solution quantities will indicate if and when steady-state is attained. (Even in codes which provide the steady-state calculation option,

"false time stepping" is sometimes used on a particular equation to improve convergence if there is knowledge of the time scale over which the variable changes. In some high-speed compressible or multiphase flow problems, it may be necessary to solve the steady-state problem as a transient one, with small time steps in the beginning.)

11 SAMPLE PROBLEMS

Note that all figures showing sample input decks and output plots, Figures 11-1 to 11-17 are placed at the end of this section.

11.1 Von Kármán Vortex Street

The von Kármán vortex street problem is discussed in the Assessment Manual. Here, we show the input listing and some of the graphical outputs obtained for one case of the calculations. This is the case of flow past a rectangular block at a Reynolds number of 30. This Reynolds number is less than the critical value (around 40) above which vortex streets are formed behind the flow obstacle.

Figure 11-1 lists the input used in the calculation. Note that the van Leer advection scheme is chosen to reduce numerical diffusion errors. The positive and negative j mesh boundaries (which model the wind-tunnel wall, if this calculation is compared to an experiment) are free-slip. However, surfaces of the obstacle are specified with no-slip velocity boundary conditions (**nslipdef**). Note also definition of a perturbed inlet velocity during the initial 1 s. Figure 11-2 shows the mesh used, and Figures 11-3 and 11-4 are selected output plots that show some of the computed results.

11.2 Hydrogen Burn

This sample problem illustrates how to set up a premixed hydrogen-air combustion problem. Figure 11-5 shows the input listing for this problem. Note that no boundary conditions are specified here, so the default free-slip wall condition is used for all boundaries. The initial condition is a stagnant, premixed hydrogen-air mixture (10% hydrogen, 19% oxygen, and 71% nitrogen by volume) at atmospheric pressure and 300 K. The computational domain is two-dimensional, 100 cm long (x -direction, discretized into 20 cells), and 250 cm high (y -direction, 50 cells). All computational cells are uniform in size, 5 cm $\times$ 5 cm. Three compartments, which are defined by a **walls** and a **mobs** definition, are modeled, as can be seen in Figure 11-6. The **walls** definition specifies an infinitely thin partition between the upper and middle compartments. This partition extends from $x = 0$ to 75 cm, leaving an opening of 25 cm between the two compartments. The **mobs** definition specifies a physical barrier 5 cm thick separating the middle and lower compartments. This barrier extends from $x = 25$ to 100 cm, so that fluid flow between the two compartments is only allowed between x -coordinates of 0 and 25 cm.

It is assumed that the mixture is ignited at a location on the "left" side of the domain ($x = 0$) and 50 cm from the bottom. The activation energy, which comes from the ignition source, is simulated by defining the initial temperature of the cell where ignition occurs

Input File Listing.

Flow past block, Re = 30

HMS-93

TSA-8 Lam

HMS-vk.block

NOTES: Obstacle (2cm tall, 1cm thick) in channel 24cm tall, 20 cm long.

Obstacle to channel height ratio is 1/12

Variable mesh spacing: 2 zones in x and 3 zones in y.

Number of real cells = 44 x 44 x 1.

\$Id: von_Kármán .ingf,v 1.3 1997/06/18 09:57:14 jspore Exp \$

\$Log: von_Kármán .ingf,v \$

Revision 1.3 1997/06/18 09:57:14 jspore

Fixed itdowndt and itupdt input.

\$innet

\$end

\$xput

```
nrsdump = 0,
ifvl = 1, ; Turn on van Leer advection scheme
autot = 1.0,
delt0 = 0.025,
deltmin = 1.000e-04,
deltmax = 1.000e-00,
epsi0 = 1.000e-05,
epsimax = 1.000e-05,
epsimin = 1.000e-05,
gz = -000.0,
iobpl = 1,
itdowndt = 499,
itupdt = 498,
itmax = 500,
lpr = 1,
maxcyc = 900056,
ittyfreq = 100,
nu = 0.153,
muoption = 1,
pltdt = 50.0,
prtdt = 40.0,
twfin = 40.0,
tddt = 500.0,
velmx = 1.5,
ibb = 1,
ibn = 1,
ibs = 1,
ibw = 1,
ibe = 1,
ibt = 1,

idiffmom = 1,

nslipdef(1,1) = 5, 9, 19, 19, 1, 2, 1, 'both',
nslipdef(1,2) = 5, 9, 27, 27, 1, 2, 1, 'both',
nslipdef(1,3) = 5, 5, 19, 27, 1, 2, 1, 'both',
nslipdef(1,4) = 9, 9, 19, 27, 1, 2, 1, 'both',
```

Figure 11-1. Input listing for Von Kármán Vortex Street test problem

```

vvalue = 2.3, 1.2, ; Inlet velocity, Perturbed velocity

half vbc(1,1) = 01, 01, 1, 23, 1, 02, 1, 1, 0.0, 2.5e+03, ; Inlet, upper
half vbc(1,2) = 01, 01, 23, 45, 01, 02, 1, 2, 0.0, 1.0e+00, ; Perturb., lower
half vbc(1,3) = 01, 01, 23, 45, 01, 02, 1, 1, 1.0, 2.5e+03, ; Inlet, lower

pbc(1,1) = 45, 45, 1, 45, 01, 02, 1, 00.0, 2.5e+03,

mat = 'air',

gasdef(1,1) = 1, 45, 1, 45, 1, 2, 1,
             1.0132500000e6, 300.00, 1, 0., 0.,
             'air', 1.00000,

gasdef(1,2) = 0, 1, 1, 45, 1, 2, 1,
             1.0132500000e6, 300.00, 1, 0., 2.5e+03,
             'air', 1.00000,

gasdef(1,3) = 45, 46, 1, 45, 1, 2, 1,
             1.0132500000e6, 300.00, 1, 0., 2.5e+03,
             'air', 1.00000,

mobs = 5, 9, 19, 27, 1, 2, 1,

```

\$end

M E S H

\$meshgn

```

iblock = 1,
nkx = 02,
xl(1) = 0.0, xc(1) = 0.0, nxl(1) = 0, nxr(1) = 32, dxmn(1) = 1000.0,
xl(2) = 8.0, xc(2) = 8.0, nxl(2) = 0, nxr(2) = 12, dxmn(2) = 0.3,
xl(3) = 20.0,

nky = 03,
yl(1) = 0.0, yc(1) = 9.0, nyl(1) = 10, nyr(1) = 0, dymn(1) = 0.3,
yl(2) = 9.0, yc(2) = 9.0, nyl(2) = 0, nyr(2) = 24, dymn(2) = 9999.9,
yl(3) = 15.0, yc(3) = 15.0, nyl(3) = 0, nyr(3) = 10, dymn(3) = 0.3,
yl(4) = 24.,

nkz = 01,
zl(1) = 0.0, zc(1) = 0.0, nzl(1) = 0, nzz(1) = 01, dzmn(1) = 1000.0,
zl(2) = 10.0,

```

\$end

G R A P H I C S

\$grafic

Figure 11-1. Input listing for Von Kármán Vortex Street test problem (cont)

```

igrid = 1,

thdt = 0.05,

thp(1,1) = 18, 3, 2, 1, 'vn', 0,
thp(1,2) = 18, 6, 2, 1, 'vn', 0,
thp(1,3) = 36, 3, 2, 1, 'vn', 0,
thp(1,4) = 36, 6, 2, 1, 'vn', 0,
thp(1,5) = 41, 3, 2, 1, 'vn', 0,
thp(1,6) = 41, 6, 2, 1, 'vn', 0,
thp(1,7) = 45, 3, 2, 1, 'vn', 0,
thp(1,8) = 45, 6, 2, 1, 'vn', 0,
thp(1,9) = 17, 4, 2, 1, 'un', 0,
thp(1,10) = 17, 6, 2, 1, 'un', 0,
thp(1,11) = 17,24, 2, 1, 'un', 0,
thp(1,12) = 37, 4, 2, 1, 'un', 0,
thp(1,13) = 37, 6, 2, 1, 'un', 0,
thp(1,14) = 37,24, 2, 1, 'un', 0,
thp(1,15) = 42, 6, 2, 1, 'un', 0,
thp(1,16) = 42,24, 2, 1, 'un', 0,

pnt(1,1) = 1, 1, 2, 1,
pnt(1,2) = 45, 45, 2, 1,
pnt(1,3) = 9, 1, 2, 1,
pnt(1,4) = 9, 45, 2, 1,
pnt(1,5) = 21, 1, 2, 1,
pnt(1,6) = 21, 45, 2, 1,
pnt(1,7) = 37, 1, 2, 1,
pnt(1,8) = 37, 45, 2, 1,
pnt(1,9) = 42, 1, 2, 1,
pnt(1,10) = 42, 45, 2, 1,
pnt(1,11) = 45, 1, 2, 1,
pnt(1,12) = 45, 45, 2, 1,
           9, 15, 2, 1,
           37, 31, 2, 1,

v2d(1,1) = 1, 2, 1,
           13,14, 1,

iinc = 2,
jinc = 2,
$end

```

P A R T I C L E S

```

$parts

$end
$rheat

$end
$special
$end

```

Figure 11-1. Input listing for Von Kármán Vortex Street test problem (cont)

Figure 11-2. The GASFLOW mesh for calculating flow field behind a rectangular block

Figure 11-3. Velocity vectors for flow around flat plate

Figure 11-4. Time history velocity in the Y-direction at $i=18, j=3, k=2$ for flow around flat plate

N6 jrt

HMS-Burn

NOTES: 2-D domain 100 cm x 250 cm with $\text{deltax} = \text{deltay} = \text{deltaz} = 5$ cm
 Number of fluid cells = $20 \times 50 \times 1$ for the
 coordinate dimension x, y, and z, respectively.

Two horizontal walls separating computational domain.

Top wall defined by "walls" definition.

Bottom wall defined by "mobs" definition.

 \$xput

```

    iburn = 1,                      ; Turn on hydrogen combustion

    idiffmom = 1,                   ; Momentum diffusion on
    idiffme  = 1,                   ; Mass and energy diffusion on
    nu       = 0.150 ,
    prandtl  = 0.70,
    schmidt  = 0.25,

    tmodel   = 'alg',               ; Algebraic turbulence model

    autot    = 1.0,
    delt0    = 00.01000 ,
    deltmin  = 1.000e-06,
    deltmax  = 1.000e-02,

    epsi0    = 1.000e-05,
    epsimax  = 1.000e-05,
    epsimin  = 1.000e-05,

    gy       = -980.0,              ; Gravity in -y direction

    itdowndt = 100,
    itupdt   = 100,
    itmax    = 500,

    lpr      = 1,
    ittyfreq = 100,
    pltdt    = 0.50,
    prtdt    = 500.0,
    tddt     = 100.0,
    velmx    = 1.5,

    maxcyc   = 10000,
    twfin    = 5.000,               ; Total problem time 5 seconds

    mobs     = 6, 21, 20, 21, 1, 2, 1, 0, ; Obstacle definition

    walls    = 1, 16, 36, 36, 1, 2, 1, 0, ; Wall definition

    mat = 'h2', 'o2', 'n2', 'h2o',

; Initial Condition Throghout domain:
    gasdef(1,1) = 1 , 21, 1 , 51, 1 , 2, 1 ,
                  1.013500e6, 300.0, 2, 0., 0.,
                  'h2', 0.1000000, 'o2', 0.1900000, 'n2', 0.7100000,

; Initial Condition at Ignition Location:
    gasdef(1,2) = 1 , 02, 11 , 12, 1 , 2, 1 ,
                  1.013500e6, 1000.0, 2, 0., 0.,
                  'h2', 0.1000000, 'o2', 0.1900000, 'n2', 0.7100000,

```

\$end

Figure 11-5. Input listing for the hydrogen burn sample problem


```

-----
M E S H
-----
$meshgn

  iblock = 1,

  nkx=1,
    xl(1)= 0.0, xc(1) = 0.000 , nxl(1)= 0, nxr(1)=20 , dxmn(1)= 9999.,
    xl(2)= 100.0,
  nky=1,
    yl(1)= 0., yc(1) = 0.000 , nyl(1)= 0, nyx(1)=50 , dymn(1)= 9999.,
    yl(2)= 250.0,

  nkz=1,
    zl(1)=0.0000, zc(1) = 0.0000, nzl(1)= 0, nzx(1)= 1 , dzmn(1)= 9999.,
    zl(2)= 5.,

$end
-----
G R A P H I C S
-----
$graphic

  thdt = 0.050,

  thp = 10, 25, 2, 1, 'pn', 0,           ; pressure time history
        10, 25, 2, 1, 'tk', 0,           ; temperature time history
        10, 25, 2, 1, 'vf', 'h2',       ; H2 volume fraction t. h.
        10, 25, 2, 1, 'vf', 'h2o',     ; H2O vol. frac. t. h.

; Define two points that would cover the entire physical x-y domain.
  pnt(1, 1) = 1, 1, 2, 1,
              21,51, 2, 1,

  v2d      = 1,02, 1,           ; velocity vector plot on plane
              ; defined by points 1 and 2.

  c2d      = 1,02, 'tk' , 0,    ; temperature contour plot on plane
              ; defined by points 1 and 2.

$end
-----
P A R T I C L E S
-----
$parts

$end
-----
H E A T   T R A N S F E R   &   C O N D E N S A T I O N
-----
$rheat

$end
-----
S P E C I A L
-----
$special

$end

```

Figure 11-5. Input listing for the hydrogen burn sample problem (cont)

Figure 11-6. Temperature contours at 0.5 s. Note the rising hot plume

to 1000 K. The problem time is 5 s. The hydrogen burn model is, of course, activated (**iburn** = 1). All diffusion models, as well as the algebraic turbulence model, are turned on. These and other code options are annotated in the input listing as shown in Figure 11-5. Figures 11-6 to 11-10 show some of the calculated results in temperature contour, velocity vector, and time-history plots.

11.3 Particle Model Sample Problem

The problem was run in two steps. First, the 10 x 5 x 5 cell mesh (*x*-, *y*-, *z*-direction, respectively) was set up according to the instructions given in Section 7.6. The wall boundary conditions are free-slip and the inflow and outflow boundaries are set by specified velocity boundary conditions. There are no internal obstacles or walls in the

Figure 11-7. Velocity vectors at 0.5 s showing upward motion near the ignition location

Figure 11-8. Temperature contours at 1.0 s. The hydrogen flame is split between the lower and middle compartment

Figure 11-9. Velocity vectors at 1.0 s, showing fluid motion spreading to other locations from the ignition point

Figure 11-10. Temperature contours at 2.0 s. The flame has spread to most areas, except in regions below the bottom and top compartments.

mesh. A quasi-steady-state flow of air without particle input was achieved by setting the input parameter **solatype** = 0.0 and running to a problem time of 0.99 s. An x-direction velocity of 30.0 cm/s was chosen to accommodate the particle behavior for this sample problem. To demonstrate the injection of particles into a computational domain, a jet of air with a velocity of 45.0 cm/s in the z-direction was injected through a 1.0-cm² area in the bottom of the mesh at cell location $i = 2, j = 3$.

The second step was to input particles into the steady-state flow and to compute their behavior to a problem time of 1.5 s, which is a time of 0.51 s from particle initialization. Two classes of particles were input. Class 1 particle density is 1.0 g/cm³ and there are two particle sizes, 1.0 μm and 100.0 μm diameter. These particles were injected through the mesh floor at the location described above at 0.00140-s time intervals, which is each computational time step. Particle class 2 is one size, and the particle diameter is 40.0 μm and the density is 5.2 g/cm³. Class 2 particles were all set in the mesh in the selected

volume at time zero of the particle run. A total of 15,000 particles was chosen for this problem, comprising 10,000 class 1, size 1 particles, 3,000 class 1, size 2 particles, and 2,000 class 2 particles. The class 1 particles were injected over a period of 0.5 s. This information was used with the instructions given in the particle initialization procedure in Section 7.6 to compute the total mass, **pmass(lc,ls)**, for each class and size.

To initiate the particle run, the parameters **twfin**, **solatype**, and **nrstdump**, in NAMELIST group **xput**, were changed. The particle input data in NAMELIST group **parts** may be included in the input data file **ingf** when the **solatype** = 0.0 steady-state flow run is being made, although it is not read at that time.

The particle model sample problem input is given in Figure 11-11.

```
GASFLOW Particle Sample Problem
N6 bdn
PARTICLE TEST 050693
NOTES:
  - Cartesian mesh 10x5x5 cells

$Id: part_samp_rst.ingf,v 1.3 1997/12/03 19:36:18 jspore Exp $
$Log: part_samp_rst.ingf,v $
Revision 1.3 1997/12/03 19:36:18 jspore
Made particle plot input consistent with version 2.1.1.0.

Revision 1.2 1997/06/18 12:07:23 jspore
Added RCS header.

$innet
$end
-----
X P U T
-----
$xput

msp      = 15000,
itpcl    = 2,
itpsz    = 2, 1,
delt0    = 1.000e-04,
deltmin  = 1.000e-08,
deltmax  = 1.000e-02,
epsi0    = 1.000e-06,
epsimax  = 1.000e-06,
epsimin  = 1.000e-06,
gz       = -980.0,
iobpl    = 1,
itdowndt = 50,
itupdt   = 49,
itmax    = 100,
lpr      = 1,
maxcyc   = 50000,
```

Figure 11-11. Aerosol test input deck

```

ittyfreq = 25,
nu        = 0.1500,
pltdt     = 1.000e-01,
prtdt     = 2.000e+10,
twfin     = 1.500e+00,
tddt      = 1.000e+10,
velmx     = 1.5,
ibb       = 1,
ibn       = 1,
ibs       = 1,
ibw       = 1,
ibe       = 1,
ibt       = 1,
ui        = 000.0,
solatype  = 2.0,
nrsdump   = 1,

mat = 'air',

gasdef(1,1) = 1, 11, 1, 6, 1, 6, 1,
              1.01325e+06, 295.0, 1, 0.0, 0.0,
              'air', 1,
gasdef(1,2) = 0, 1, 1, 6, 1, 6, 1,
              1.01325e+06, 295.0, 1, 0.0, 1.0e+06,
              'air', 1,
gasdef(1,3) = 11, 12, 1, 6, 1, 6, 1,
              1.01325e+06, 295.0, 1, 0.0, 1.0e+06,
              'air', 1,
gasdef(1,4) = 2, 3, 3, 4, 0, 1, 1,
              1.01325e+06, 295.0, 1, 0.0, 1.0e+06,
              'air', 1,

vvalue = 30.0, 45.0, 31.8,

vbc(1,1) = 1, 1, 1, 6, 1, 6, 1, 1, 0.0, 100.0,
vbc(1,2) = 2, 3, 3, 4, 1, 1, 1, 2, 0.0, 100.0,
vbc(1,3) = 11, 11, 1, 6, 1, 6, 1, 3, 0.0, 100.0,

$end
-----
      M E S H
-----
$meshgn

iblock = 1,

      nkx = 1,
      xl(1) = 0.000, xc(1) = 5.000 , nxl(1)= 5, nxr(1)= 5,
      dxmn(1) = 1.000,
      xl(2) = 10.00,

      nky = 1,
      yl(1) = 0.000, yc(1) = 0.000, nyl(1)= 0, nyr(1)= 5 ,

```

Figure 11-11 (cont). Aerosol test input deck


```

dymn(1) = 1.000,
  yl(2) = 5.000,

  nkz = 1,
  zl(1) = 0.000, zc(1) = 0.000 , nzl(1)= 0, nzc(1)= 5 ,
  dzmn(1) = 1.000,
  zl(2) = 5.000,

$end
-----
      G R A P H I C S
-----
$grafic

pnt(1,1) = 1, 1, 2, 1,
pnt(1,2) = 1, 4, 1, 1,
pnt(1,3) = 11, 4, 6, 1,
pnt(1,4) = 11, 6, 2, 1,

pnt(1,5) = 1, 1, 6, 1,
pnt(1,6) = 11, 6, 6, 1,

v2d(1,1) = 1, 4, 1,
v2d(1,2) = 2, 3, 1,
v2d(1,3) = 5, 6, 1,

viewcrds(1,1) = 5.0, 2.50, 2.50, 5.0, -100.0, 25.0,
viewcrds(1,2) = 5.0, 2.50, 2.50, -40.0, -50.0, 15.0,
viewcrds(1,3) = 5.0, 2.50, 2.50, 2.5, 100.0, 30.0,
viewcrds(1,4) = 5.0, 2.50, 2.50, 110.0, 2.5, 40.0,

  npplts = 9,

  ipview(1) = 1,
  ippka(1,1) = 1,1,
  nap(1) = 1,

  ipview(2) = 1,
  ippka(1,2) = 1,2,
  nap(2) = 1,

  ipview(3) = 1,
  ippka(1,3) = 2,1,
  nap(3) = 1,

  ipview(4) = 1,
  ippka(1,4) = 0,
  nap(4) = 1,

  ipview(5) = 2,
  ippka(1,5) = 0,
  nap(5) = 1,

  ipview(6) = 4,
  ippka(1,6) = 0,
  nap(6) = 1,

```

Figure 11-11 (cont). Aerosol test input deck

```

ipview(7) = 1,
ippka(1,7) = 1,1,
nap(7) = 0,

ipview(8) = 1,
ippka(1,8) = 1,2,
nap(8) = 0,

ipview(9) = 1,
ippka(1,9) = 2,1,
nap(9) = 1,

thdt      = 0.025,
gline     = 'off',
pthpt0    = 1.000,
pthp(1, 1) = 3, 4, 02, 1, 'pnc' , 1, 1, 0,
pthp(1, 2) = 3, 4, 02, 1, 'pnc' , 1, 2, 0,
pthp(1, 3) = 11, 4, 3, 1, 'pnc' , 1, 0, 0,
pthp(1, 4) = 4, 4, 02, 1, 'pmd' , 1, 2, 23,
pthp(1, 5) = 11, 4, 2, 1, 'pmd' , 0, 0, 23,
pthp(1, 6) = 10, 4, 2, 1, 'pmd' , 2, 0, 23,
pthp(1, 7) = 11, 4, 2, 1, 'pnc' , 2, 0, 0,

```

Send

P A R T I C L E S

\$parts

```

niterp = 20,
tdcp   = 0.1,

intrn = 0,
npinpt(1,1) = 5000, ; particle clase 1 size 1
npinpt(1,2) = 5000, ; particle clase 1 size 2
npinpt(2,1) = 5000, ; particle clase 2 size 1

ipdep(1)      = 2,
ipblkkin(1)   = 1,
xpw(1)        = 1.25,          xpe(1) = 1.75,
yps(1)        = 2.25,          ypn(1) = 2.75,
zpb(1)        = 0.000,         zpt(1) = 4.280e-02,
prhoin(1)     = 1.0,
pdiamin(1,1)  = 1.000e-04,      pmass(1,1) = 5.236e-09,
pdiamin(1,2)  = 1.000e-02,      pmass(1,2) = 0.001571,
twpinp(1)     = 1.0,           tinjt(1) = 0.50,
pinpdt(1)     = 1.400e-03,      scfacin(1,1) = 1.165,

ipdep(2)      = 1,
ipblkkin(2)   = 1,
xpw(2)        = 0.00,          xpe(2) = 1.00,
yps(2)        = 2.00,          ypn(2) = 3.00,
zpb(2)        = 3.5,           zpt(2) = 4.5,
prhoin(2)     = 5.2,
pdiamin(2,1)  = 4.000e-03,      pmass(2,1) = 3.485e-04,

```

Figure 11-11 (cont). Aerosol test input deck

```

        twpinp(2)    = 1.0,          tinjt(2) = 00.01,
        pinpdt(2)    = 100.0,

$end

-----
      H E A T  T R A N S F E R   V A R I A B L E S
-----

$ Rheat

$end

-----
      P L O T   G R I D S
-----

$special

$end

```

Figure 11-11 (cont). Aerosol test input deck

The graphic output for this sample problem was chosen to show how the graphics in GASFLOW can be used to interpret the results of a run. The perspective plots show the location of the particle classes and sizes plotted separately and together at selected times. The particles in each class and size may be plotted in a different color. More than one particle class may be plotted on the same plot frame by setting **nap** = 0 for all but the last class being plotted. However, the number of particles plotted that is printed on the plot will be for the last class plotted only. Also, the particle class is not printed on the plot. This should not be a problem because the plots are in the order of the plot input data.

Figure 11-12 shows the perspective view plot of the particles for both classes and both sizes at a time of 1.1 s. This is the first time step of the injection run. Figure 11-13 shows the class 1, size 1 particles plotted at 1.2 s. Some of these particles have deposited just down stream of the injection port, and others have moved with the mainstream gas velocity. The deposition flag for this particle class is 2, so all the particles adhere to the surface on impact. The class 2 particles, shown in Figure 11-14, have dispersed somewhat at this time and have reached their terminal settling velocity as they fall through the primarily *x*-direction mainstream flow field. The perspective view in these figures is number 1 in the input deck. Figure 11-15 is perspective view 2 and is at a time of 1.5 s. Some of the class 1 particles have continued to deposit soon after their injection into the mesh and others are following the mainstream flow. The class 2 particles have virtually all settled to the mesh floor and deposited. The class 2 deposition flag is set to 1, so the deposition model determines if the particle adheres on impact.

Time-history plots are shown in Figures 11-16 and 11-17. The particle number concentration in the cell into which the class 1, size 1 and size 2, particles are injected is

plotted in the upper two plots of Figure 11-16. Of course, total particle mass in the cell or some other property could have been selected also. The lower left plot in Figure 11-17 is the particle number concentration of both sizes of the class 1 particles in the last cell before the mesh exit plane ($i = 11, j = 4, k = 3$), from which the particle concentration that is being fluxed across the exit boundary can be calculated. Similarly, in Figure 11-17, the upper left plot is the deposited particle mass of all particle classes adjacent to the outflow boundary in the $i = 11, j = 4, k = 2$ cell. The upper right plot in Figure 11-17 is the class 2 particle mass deposited in one cell ($i = 10, j = 4, k = 2$, bottom face) on the floor of the mesh one cell back from the outflow boundary.

Figure 11-12. All particles at Time = 1.1 seconds

Figure 11-13. Class 1, size 1 particles at 1.2 seconds

Figure 11-14. Class 2 particles at 1.2 seconds

Figure 11-15. All particles at 1.5 seconds

Figure 11-16. Time history plots for particle concentrations and deposition

Figure 11-17. Time history plots for particles

12 REFERENCES

1. R. J. Kee, J. A. Miller, and T. H. Jefferson "CHEMKIN: A General-Purpose, Problem-Independent, Transportable, FORTRAN Chemical Kinetics Code Package," Sandia National Laboratories Report SAND80-8003, 1980.
2. B. van Leer, "Towards the Ultimate Conservation Difference Scheme IV. A New Approach to Numerical Convection," *Journal of Computational Physics*, **23**: 276-299.
3. W. Breitung, S. B. Dorofeev, and J. R. Travis, "A Mechanistic Approach to Safe Ignitor Implementation for Hydrogen Mitigation," OECD Workshop on the Implementation of Hydrogen Mitigation Techniques, May 13-15, 1996, AECL-11762 NEA/CSNI/R (96)8, Winnipeg, Manitoba, Canada.

APPENDIX A

SUMMARY OF VARIABLES IN NAMELIST GROUP `xput`

Variable	Default	Description	Restart
<code>areardef(*,*)</code>	none	Defines fractional mesh areas at openings (plane surface).	yes
<code>areardef(1,*)</code>	none	Beginning i mesh index (cell face number).	yes
<code>areardef(2,*)</code>	none	Ending i mesh index (cell face number).	yes
<code>areardef(3,*)</code>	none	Beginning j mesh index (cell face number).	yes
<code>areardef(4,*)</code>	none	Ending j mesh index (cell face number).	yes
<code>areardef(5,*)</code>	none	Beginning k mesh index (cell face number).	yes
<code>areardef(6,*)</code>	none	Ending k mesh index (cell face number).	yes
<code>areardef(7,*)</code>	none	Block number.	yes
<code>areardef(8,*)</code>	none	Fraction of the i, j, k plane open for flow. If <code>areardef(8,*)</code> is less than zero and <code>areardef(9,*)</code> equals zero, then no flow loss is included at these locations. If <code>areardef(8,*)</code> is less than zero and <code>areardef(10,*)</code> equals zero, then no laminar drag loss is included at the locations specified by this <code>areardef</code> definition.	yes
<code>areardef(9,*)</code>	0.0	User input flow loss coefficient. If <code>areardef(9,*)</code> is zero, then orifice loss coefficient is calculated by GASFLOW. If <code>areardef(9,*)</code> is zero and <code>areardef(8,*)</code> is less than zero, then no flow loss is calculated for the locations specified by this <code>areardef</code> definition.	yes
<code>areardef(10,*)</code>)	0.0	Hydraulic diameter to be used for the laminar drag loss. If <code>areardef(10,*)</code> is zero, then no laminar drag loss is calculated by GASFLOW for the locations specified by this <code>areardef</code> definition.	yes
<code>areardef(11,*)</code>)	64.0	Coefficient in the laminar drag loss correlation. Can be used to model noncircular geometries.	yes

Variable	Default	Description	Restart
areardef(12,*)	1.0	Fraction of the cell-center-to-cell-center distance of the neighboring cells over which the laminar drag loss model should be applied.	yes
autot	1.0	Automatic time-step control flag: 1.0 means ON; 0.0 means OFF (fixed time-step size). See Section 8.2.	
c1ke	1.44	Parameter for κ - ϵ turbulence model. See Section 7.3.2.	
c2ke	1.92	Parameter for κ - ϵ turbulence model. See Section 7.3.2.	
cbc(*,*)	none	Continuative boundary condition definition array. See Section 5.2.2.	
cbc(1,*)	none	Beginning i mesh index (cell face number).	
cbc(2,*)	none	Ending i mesh index (cell face number).	
cbc(3,*)	none	Beginning j mesh index (cell face number).	
cbc(4,*)	none	Ending j mesh index (cell face number).	
cbc(5,*)	none	Beginning k mesh index (cell face number).	
cbc(6,*)	none	Ending k mesh index (cell face number).	
cbc(7,*)	none	Block number (must be set to 1).	
cbc(8,*)	none	Start time (s).	
cbc(9,*)	none	End time (s).	
clength	30.48	Length scale for algebraic turbulence model. See Section 7.3.1.	
cmu	0.05	Constant for algebraic turbulence model. See Section 7.3.1.	
cmug	1.8E-04	Constant value of dynamic viscosity, to be used with muoption = 2. See Section 7.2.	
cmuke	0.09	Parameter for κ - ϵ turbulence model. See Section 7.3.2.	
cyl	0.0	Coordinate system option: 0.0 means Cartesian; 1.0 means cylindrical.	

Variable	Default	Description	Restart
delt0	0.02	Initial time increment size. See Section 8.2.	
deltmax	1.0E+30	Maximum allowable time increment size. See Section 8.2.	
deltmin	1.0E-04	Minimum allowable time increment size. Run will be terminated if this exceeds the time-step size calculated by code. See Section 8.2.	
epsi0	1.0E-05	Initial pressure iteration error criterion. See Section 8.1.	
epsimax	1.0E-03	Not used. Input is still available to provide backward compatibility with old input decks.	
epsimin	1.0E-06	Not used. Input is still available to provide backward compatibility with old input decks.	
epsval(*)	none	Array to store values for turbulence dissipation rate. Used with turbdef . See Section 7.3.2.	
esdef(*,*)	none	Volumetric energy source definition array.	
esdef(1,*)	none	Beginning i mesh index (cell face number).	
esdef(2,*)	none	Ending i mesh index (cell face number).	
esdef(3,*)	none	Beginning j mesh index (cell face number).	
esdef(4,*)	none	Ending j mesh index (cell face number).	
esdef(5,*)	none	Beginning k mesh index (cell face number).	
esdef(6,*)	none	Ending k mesh index (cell face number).	
esdef(7,*)	none	Block number.	
esdef(8,*)	none	Volumetric energy source (ergs/cm ³) added to the cells defined by this esdef definition.	
esdef(9,*)	none	Time (s) at which volumetric energy source begins.	
esdef(10,*)	none	Time (s) at which volumetric energy source ends.	

Variable	Default	Description	Restart
fractke	0.1	Fraction of mean kinetic energy, used to calculate turbulent kinetic energy in algebraic turbulence model. See Section 7.3.1.	
gasdef(*,*)	none	Gas definition array, used for defining initial and boundary conditions for fluid. See Section 5.1.1.	
gasdef(1,*)	none	Beginning i mesh index (cell face number).	
gasdef(2,*)	none	Ending i mesh index (cell face number).	
gasdef(3,*)	none	Beginning j mesh index (cell face number).	
gasdef(4,*)	none	Ending j mesh index (cell face number).	
gasdef(5,*)	none	Beginning k mesh index (cell face number).	
gasdef(6,*)	none	Ending k mesh index (cell face number).	
gasdef(7,*)	none	Block number.	
gasdef(8,*)	none	Pressure (dynes/cm ²) in defined volume. If gasdef(8,*) is less than zero, then the INT(ABS(gasdef(8,*))) points to the column number in the SORTAM file and the pressure will be obtained from this column in the SORTAM file. If gasdef(8,*) is less than zero and gasdef(8,*) is larger than 1,000,000, then it is a packed i, j, k, iblk location and the pressure will be obtained from the pressure in cell i, j, k, iblk.	
gasdef(9,*)	none	Temperature (K) in defined volume. If gasdef(9,*) is less than zero, then the INT(ABS(gasdef(9,*))) points to the column number in the SORTAM file and the temperature will be obtained from this column in the SORTAM file.	
gasdef(10,*)	none	Option flag for specification of gas composition: 1 for mass fraction, 2 for volume fraction, > 9 implies a time-dependent function for the pressure and temperature will be specified.	
gasdef(11,*)	none	Time (s) at which "gas definition" begins.	
gasdef(12,*)	none	Time (s) at which "gas definition" ends.	

Variable	Default	Description	Restart
gasdef(13,*)	none	Gas species component number (determined by the order in the gas species list defined by mat). Gas species component can alternatively be specified by its symbol as given in Table 3-2, e.g., 'h2'.	
gasdef(14,*)	none	Mass or volume fraction of above gas species in defined volume. If gasdef(14,*) is less than zero, then the INT(ABS(gasdef(14,*))) points to the column number in the SORTAM file and the mass/volume will be obtained from this column in the SORTAM file.	
gasdef(15,*)	none	Second gas species component number, if needed.	
gasdef(16,*)	none	Mass or volume fraction of second gas species in defined volume, if needed. If gasdef(16,*) is less than zero, then the INT(ABS(gasdef(16,*))) points to the column number in the SORTAM file and the mass/volume will be obtained from this column in the SORTAM file.	
gasdef(40,*)	none	Last index in first dimension is 40; therefore, compositions of a maximum of $(40 - 12)/2 = 14$ species can be defined.	
gx	0.0	Acceleration due to gravity in the i- (x- or r-) direction (cm/s^2).	
gy	0.0	Acceleration due to gravity in the j- (y- or θ -) direction (cm/s^2).	
gz	0.0	Acceleration due to gravity in the k- (z-) direction (cm/s^2).	
holes(*,*)	none	Variable array that puts holes into existing obstacles. See Section 3.3.3.	no
holes(1,*)	none	Beginning i mesh index (cell face number).	no
holes(2,*)	none	Ending i mesh index (cell face number).	no
holes(3,*)	none	Beginning j mesh index (cell face number).	no
holes(4,*)	none	Ending j mesh index (cell face number).	no
holes(5,*)	none	Beginning k mesh index (cell face number).	no

Variable	Default	Description	Restart
holes(6,*)	none	Ending k mesh index (cell face number).	no
holes(7,*)	none	Block number.	no
holes(8,*)	none	Flag for fluxing on the beginning i mesh face holes(1,n): = 0 implies no fluxing allowed; = 1, fluxing allowed.	no
holes(9,*)	none	Flag for fluxing on the ending i mesh face holes(2,n): = 0 implies no fluxing allowed; = 1, fluxing allowed.	no
holes(10,*)	none	Flag for fluxing on the beginning j mesh face holes(3,n): = 0 implies no fluxing allowed; = 1, fluxing allowed.	no
holes(11,*)	none	Flag for fluxing on the ending j mesh face holes(4,n): = 0 implies no fluxing allowed; = 1, fluxing allowed.	no
holes(12,*)	none	Flag for fluxing on the beginning k mesh face holes(5,n): = 0 implies no fluxing allowed; = 1, fluxing allowed.	no
holes(13,*)	none	Flag for fluxing on the ending k mesh face holes(6,n): = 0 implies no fluxing allowed; = 1, fluxing allowed.	no
ibb	1	Boundary condition indicator for -k (bottom) mesh boundary. See Section 5.2.1. Options are 1, rigid free-slip; 2, rigid no-slip; 3, continuative; 4, periodic; 5, specified pressure.	
ibe	1	Boundary condition indicator for +i (east) mesh boundary. See ibb description.	
ibn	1	Boundary condition indicator for +j (north) mesh boundary. See ibb description.	
ibs	1	Boundary condition indicator for -j (south) mesh boundary. See ibb description.	
ibt	1	Boundary condition indicator for +k (top) mesh boundary. See ibb description.	
iburn	0	Option flag for hydrogen combustion: 0 means OFF; 1 means ON.	

Variable	Default	Description	Restart
ibw	1	Boundary condition indicator for -i (west) mesh boundary. See ibb description.	
icopt	0	Method for computing the specific heat c_p : = 0, derivative of internal energy function; = 1, constant value ($T < 500$ K); = 2, second-degree polynomial ($T < 750$ K); = 3, Gordon & McBride approximation.	no
idiffme	0	Option flag for mass and energy diffusion: 0 means OFF; 1 means ON. See Section 7.2.	
idiffmom	0	Option flag for momentum diffusion: 0 means OFF; 1 means ON. See Section 7.2.	
ieopt	1	Polynomial fit for specific internal energy: = 1, first-degree polynomial; = 2, second-degree polynomial; = 3, third-degree polynomial; = 4, fourth-degree polynomial.	no
ifvl	0	Option flag to activate van Leer advection scheme: 0 means donor-cell method; 1 means van Leer method. See Section 8.3.	
ignitdef(*,*)	none	Ignitor definition array. See Section 7.7.1.	
ignitdef(1,*)	none	i mesh index (cell centered) of ignitor.	yes
ignitdef(2,*)	none	j mesh index (cell centered) of ignitor.	yes
ignitdef(3,*)	none	k mesh index (cell centered) of ignitor.	yes
ignitdef(4,*)	none	Block number of ignitor.	yes
ignitdef(5,*)	none	= 0, for continuously operating ignitors; > 0, for periodic sparking ignitors.	yes
ihystat	0	Option flag for imposing hydrostatic pressure gradient in fluid cells according to acceleration components gx , gy , and gz : 0 means OFF; 1 means ON. See Section 7.1.	

Variable	Default	Description	Restart
iroomdef(*,*)	none	Defines i, j, k region for evaluation of detonation potential from $d/7\lambda$ criterion. For each iroomdef , the following information is plotted from unit 11: burnable cloud size d, average hydrogen density of burnable cloud, detonation cell size for average hydrogen density in the mixture, $d/7\lambda$ (see Section 7.7).	yes
iroomdef(1,*)	none	Beginning i mesh index (cell face number).	no
iroomdef(2,*)	none	Ending i mesh index (cell face number).	no
iroomdef(3,*)	none	Beginning j mesh index (cell face number).	no
iroomdef(4,*)	none	Ending j mesh index (cell face number).	no
iroomdef(5,*)	none	Beginning k mesh index (cell face number).	no
iroomdef(6,*)	none	Ending k mesh index (cell face number).	no
iroomdef(7,*)	none	Block number.	no
iroomdef(8,*)	none	Actual room number: = + implies positive volume; = - implies negative volume (i.e., subtracts obstacles).	no
itdowndt	50	Number of pressure iterations at which time-step size is reduced by 2%. If the number of pressure iterations is larger than itdowndt , then the time-step size is reduced by 2%. See Section 8.2.	
itmax	20	Maximum number of pressure iterations per cycle. See Section 8.2.	
itopt	0	Method for computing the molecular transport properties, dynamic viscosity, thermal conductivity, and mass diffusion coefficient: = 0, nonmechanistic using input data (then muoption applies); = 1, constant value ($T < 500$ K); = 2, linear ($T < 1000$ K); = 3, quadratic ($T < 3000$ K); = 4, cubic ($T < 3000$ K); = 5, quartic ($T < 5000$ K); = 6, functions are evaluated from the CHEMKIN library.	no

Variable	Default	Description	Restart
itpcl	1	Total number of particle classes.	
itpsz (mxpcl1)	none	Number of initial particle sizes in each particle class.	
ittyfreq	20	Frequency of printing iteration/cycle information to terminal. See Section 9.3.	
itupdt	100	Number of pressure iterations at which time-step size is increased by 2%. If the number of pressure iterations is larger than itdowndt , then the time-step size is reduced by 2%. See Section 8.2.	
lpr	1	Control flag for printing to gfout: = 0 results in reduced output sent to gfout file; = 1 results in full output sent to gfout file.	
mat (*)	none	List of gas species symbols (enclosed in single quotes) for specifying gas components. See Section 4.1.	
maxcyc	10	Maximum number of cycles allowed. See Section 8.2. If set to negative, no fluid calculation will be done, but instead a set of mesh plots are produced, according to the keys in Table 3-1. See Section 3.5.	
mbc (*,*)	none	Mass flow rate boundary condition array. See Section 5.2.2.	
mbc (1,*)	none	Beginning i mesh index (cell face number).	
mbc (2,*)	none	Ending i mesh index (cell face number).	
mbc (3,*)	none	Beginning j mesh index (cell face number).	
mbc (4,*)	none	Ending j mesh index (cell face number).	
mbc (5,*)	none	Beginning k mesh index (cell face number).	
mbc (6,*)	none	Ending k mesh index (cell face number).	
mbc (7,*)	none	Block number.	
mbc (8,*)	none	Element of mvalue that will define the velocity value. Also used as a flag to determine functional dependence of specified velocity.	
mbc (9,*)	none	Start time (s).	

Variable	Default	Description	Restart
mbc(10,*)	none	End time (s).	
mfunc(*,*)	none	Set of constants for the mass flow rate boundary condition time-dependent functions. See Section 5.2.2.	
mfunc(1,*)	none	The constant b in the following functions: $f = a + b * ts$ $f = a + b * ts**2$ $f = a + b * ts**2 + c * ts**3$ $f = a + b * ts**2 + c * ts**3 + d * t**4$ $f = a + b * \exp(tr / c)$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
mfunc(2,*)	none	The constant c in the following functions: $f = a + b * ts**2 + c * ts**3$ $f = a + b * ts**2 + c * ts**3 + d * t**4$ $f = a + b * \exp(tr / c)$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
mfunc(3,*)	none	The constant d in the following functions: $f = a + b * ts**2 + c * ts**3 + d * t**4$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
mfunc(4,*)	none	The constant e in the following function: $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
mobs(*,*)	none	Mesh obstacle definition array. See Section 3.3.2.	
mobs(1,*)	none	Beginning i mesh index (cell face number).	
mobs(2,*)	none	Ending i mesh index (cell face number).	
mobs(3,*)	none	Beginning j mesh index (cell face number).	
mobs(4,*)	none	Ending j mesh index (cell face number).	
mobs(5,*)	none	Beginning k mesh index (cell face number).	
mobs(6,*)	none	Ending k mesh index (cell face number).	
mobs(7,*)	none	Block number.	

Variable	Default	Description	Restart
mobs(8,*)	none	Material identification number: 1 means concrete; 2 means steel; 3 means superconductor. If $i > 3$, use table i for element-by-element input for thermal conductivity and ρC_p . See also wltabslab , rcptabslab ($i > 3$ used for layered structures). Only used if ihflag = 1. See Section 6.2.	
mnp	1	Total number of simulation particles allowed in this calculation.	
mtab(*,*,*)	none	Time-dependent mass flow rate boundary condition table. See Section 5.2.2.	
mtab(1,ip,it)	none	Time (s) for the ip^{th} point in the it^{th} table.	
mtab(2,ip,it)	none	Mass flow rate (g/s) for the ip^{th} point in the it^{th} table.	
muoption	0	Controls how to determine transport properties when itopt is set to 0: = 0, properties are computed from the local density and input kinematic viscosity, nu , and the Prandtl and Schmidt numbers with prandtl = 1.0 and schmidt = 1.0; = 1, properties are computed from the local density and input kinematic viscosity, nu , and constant values for the thermal conductivity and diffusion coefficient; = 2, properties are computed from constant values for the dynamic viscosity, thermal conductivity, and diffusion coefficient.	no
mvalue(*)	none	Array to store values for mass flow rates, used with mbc . See Section 5.2.2.	
nrsdump	0	Number that appears in the name of the restart dump file to be read in. See Section 9.4. Negative or zero values indicate a new run.	
nslipdef(*,*)	none	Array to define no-slip surfaces. See Section 5.2.2.	
nslipdef(1,*)	none	Beginning i mesh index (cell face number).	
nslipdef(2,*)	none	Ending i mesh index (cell face number).	

Variable	Default	Description	Restart
nslipdef(3,*)	none	Beginning j mesh index (cell face number).	
nslipdef(4,*)	none	Ending j mesh index (cell face number).	
nslipdef(5,*)	none	Beginning k mesh index (cell face number).	
nslipdef(6,*)	none	Ending k mesh index (cell face number).	
nslipdef(7,*)	none	Block number (must be set to 1).	
nslipdef(8,*)	none	Side of the surface that is no-slip. Options: 'lower' means negative side; 'upper' means positive side; 'both' means both negative and positive sides.	
nu	0.0	Kinematic viscosity, ν (cm ² /s). See Section 7.2.	
pamb0	1.0E+06	Ambient pressure value for plotting purposes. See Section 9.1.3.	
pbc(*,*)	none	Array for defining pressure boundary conditions. See Section 5.2.2.	
pbc(1,*)	none	Beginning i mesh index (cell face number).	
pbc(2,*)	none	Ending i mesh index (cell face number).	
pbc(3,*)	none	Beginning j mesh index (cell face number).	
pbc(4,*)	none	Ending j mesh index (cell face number).	
pbc(5,*)	none	Beginning k mesh index (cell face number).	
pbc(6,*)	none	Ending k mesh index (cell face number).	
pbc(7,*)	none	Block number (must be set to 1).	
pbc(8,*)	none	Start time (s).	
pbc(9,*)	none	End time (s).	
pfunc(*,*)	none	Set of constants for the pressure time-dependent functions. See Section 5.1.1.	

Variable	Default	Description	Restart
pfunc(1,*)	none	The constant b in the following functions: $f = a + b * ts$ $f = a + b * ts^2$ $f = a + b * ts^2 + c * ts^3$ $f = a + b * ts^2 + c * ts^3 + d * t^4$ $f = a + b * \exp(tr / c)$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
pfunc(2,*)	none	The constant c in the following functions: $f = a + b * ts^2 + c * ts^3$ $f = a + b * ts^2 + c * ts^3 + d * t^4$ $f = a + b * \exp(tr / c)$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
pfunc(3,*)	none	The constant d in the following functions: $f = a + b * ts^2 + c * ts^3 + d * t^4$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
pfunc(4,*)	none	The constant e in the following function: $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
pltdt	1.0	Time interval (s) between successive sets of 1D profile, 2D contour, and 2D and 3D velocity vector plots. See Section 9.1.	
prandtl	1.0	Fluid Prandtl number, used to determine thermal diffusivity. See Section 4.2.	
prtdt	1000	Time interval (s) between printing of the fluid solution field (all velocity components, pressure, and density) to file gfout.	
ptab(*,*,*)	none	Time-dependent pressure table. See Section 5.1.1.	
ptab(1,ip,it)	none	Time (s) for the ip^{th} point in the it^{th} table.	
ptab(2,ip,it)	none	Pressure (dynes/cm ²) for the ip^{th} point in the it^{th} table.	
schmidt	1.0	Fluid Schmidt number, used to determine mass diffusivity. See Section 4.2.	
sclval(*)	none	Array to store values for turbulence length scale. Used with turbdef . See Section 7.3.	
sigmae	1.3	Parameter for κ - ϵ turbulence model. See Section 7.3.2.	

Variable	Default	Description	Restart
sigmak	1.0	Parameter for κ - ϵ turbulence model. See Section 7.3.2.	
solatype	0.0	Option flag to activate particle transport model: =0.0 implies normal hydrodynamic calculation with no particle transport; =1.0 implies particle transport calculation with fixed fluid velocities and no hydrodynamic calculation; =2.0 implies combined hydrodynamic and particle transport calculations.	
sortami	0	Flag to indicated whether or not a SORTAM input file will be used in this calculation: = 0, no SORTAM input file will be read for this calculation; = 1, a SORTAM input file will be read for this calculation.	yes
sgxigdef(*,*)	none	Spark ignitor definition	
sgxidef(1,*)	none	Time when ignitor is first turned on	
sgxidef(2,*)	none	Time when ignitor is turned off	
sgxidef(3,*)	none	Time interval between sparks	
sgxidt	0.001	Time of spark duration	
subsodef(*,*)	none	Reservoir source region excluded from mass balance.	yes
subsodef(1,*)	none	Beginning i mesh index (cell face number).	yes
subsodef(2,*)	none	Ending i mesh index (cell face number).	yes
subsodef(3,*)	none	Beginning j mesh index (cell face number).	yes
subsodef(4,*)	none	Ending j mesh index (cell face number).	yes
subsodef(5,*)	none	Beginning k mesh index (cell face number).	yes
subsodef(6,*)	none	Ending k mesh index (cell face number).	yes
subsodef(7,*)	none	Block number.	yes
tddt	10	Time interval (s) at which restart dump file are written. See Section 9.4.	

Variable	Default	Description	Restart
tfunc(*,*)	none	Set of constants for the temperature time-dependent functions. See Section 5.1.1.	
tfunc(1,*)	none	The constant b in the following functions: $f = a + b * ts$ $f = a + b * ts^{**2}$ $f = a + b * ts^{**2} + c * ts^{**3}$ $f = a + b * ts^{**2} + c * ts^{**3} + d * t^{**4}$ $f = a + b * \exp(tr / c)$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
tfunc(2,*)	none	The constant c in the following functions: $f = a + b * ts^{**2} + c * ts^{**3}$ $f = a + b * ts^{**2} + c * ts^{**3} + d * t^{**4}$ $f = a + b * \exp(tr / c)$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
tfunc(3,*)	none	The constant d in the following functions: $f = a + b * ts^{**2} + c * ts^{**3} + d * t^{**4}$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
tfunc(4,*)	none	The constant e in the following function: $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
tkeval(*)	none	Array to store values for turbulent kinetic energy. Used with turbdef . See Section 7.3.	
tmodel	'none'	Turbulence model options: 'none' means no turbulence model; 'alg' means algebraic model (see Section 7.3.1); 'ke' means κ - ϵ model (see Section 7.3.2).	
trange	'low'	Range of applicability of fits to internal energy: 'low', for temperatures up to 3000 K; 'high', for temperatures up to 5000 K.	no
ttab(*,*,*)	none	Time-dependent temperature table. See Section 5.1.1.	
ttab(1,ip,it)	none	Time (s) for the ip^{th} point in the it^{th} table.	
ttab(2,ip,it)	none	Temperature (K) for the ip^{th} point in the it^{th} table.	
turbdef(*,*)	none	Array to define turbulence quantities for model. See Section 7.3.2.	
turbdef(1,*)	none	Beginning i mesh index (cell face number).	

Variable	Default	Description	Restart
turbdef(2,*)	none	Ending i mesh index (cell face number).	
turbdef(3,*)	none	Beginning j mesh index (cell face number).	
turbdef(4,*)	none	Ending j mesh index (cell face number).	
turbdef(5,*)	none	Beginning k mesh index (cell face number).	
turbdef(6,*)	none	Ending k mesh index (cell face number).	
turbdef(7,*)	none	Block number (must be set to 1).	
turbdef(8,*)	none	Integer pointer to location in tkeval array for value of turbulent kinetic energy.	
turbdef(9,*)	none	Integer pointer to location in epsval array for value of turbulent kinetic energy.	
turbdef(10,*)	none	Integer pointer to location in sc1val array for value of turbulent kinetic energy.	
turbdef(11,*)	none	Start time (s).	
turbdef(12,*)	none	End time (s).	
twfin	10	Problem end time (s).	
ui	0.0	Initial velocity in i-direction (cm/s). See Section 5.1.2.	
vbc(*,*)	none	Array for defining velocity boundary conditions. See Section 5.2.2.	
vbc(1,*)	none	Beginning i mesh index (cell face number).	
vbc(2,*)	none	Ending i mesh index (cell face number).	
vbc(3,*)	none	Beginning j mesh index (cell face number).	
vbc(4,*)	none	Ending j mesh index (cell face number).	
vbc(5,*)	none	Beginning k mesh index (cell face number).	
vbc(6,*)	none	Ending k mesh index (cell face number).	
vbc(7,*)	none	Block number (must be set to 1).	

Variable	Default	Description	Restart
vbc(8,*)	none	Element of vvalue that will define the velocity value. Also used as a flag to determine functional dependence of specified velocity.	
vbc(9,*)	none	Start time (s).	
vbc(10,*)	none	End time (s).	
velmx	2.0	Scaling factor for velocity vector plots (multiplies internally scaled vectors).	
vfunc(*,*)	none	Set of constants for the velocity boundary condition time-dependent functions. See Section 5.2.2.	
vfunc(1,*)	none	The constant b in the following functions: $f = a + b * ts$ $f = a + b * ts^{**2}$ $f = a + b * ts^{**2} + c * ts^{**3}$ $f = a + b * ts^{**2} + c * ts^{**3} + d * t^{**4}$ $f = a + b * \exp(tr / c)$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
vfunc(2,*)	none	The constant c in the following functions: $f = a + b * ts^{**2} + c * ts^{**3}$ $f = a + b * ts^{**2} + c * ts^{**3} + d * t^{**4}$ $f = a + b * \exp(tr / c)$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
vfunc(3,*)	none	The constant d in the following functions: $f = a + b * ts^{**2} + c * ts^{**3} + d * t^{**4}$ $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
vfunc(4,*)	none	The constant e in the following function: $f = a + b * \cos(tr * \pi / c) + d * \sin(tr * \pi / e)$	
vi	0.0	Initial velocity in j-direction (cm/s). See Section 5.1.2.	
vtab(*,*,*)	none	Time-dependent velocity boundary condition table. See Section 5.2.2.	
vtab(1,ip,it)	none	Time (s) for the ip^{th} point in the it^{th} table.	
vtab(2,ip,it)	none	Velocity (cm/s) for the ip^{th} point in the it^{th} table.	
vvalue(*)	none	Array to store values for velocity, used with vbc . See Section 5.2.2.	

Variable	Default	Description	Restart
walls(*,*)	none	Wall definition array. See Section 3.3.1.	
walls(1,*)	none	Beginning i mesh index (cell face number).	
walls(2,*)	none	Ending i mesh index (cell face number).	
walls(3,*)	none	Beginning j mesh index (cell face number).	
walls(4,*)	none	Ending j mesh index (cell face number).	
walls(5,*)	none	Beginning k mesh index (cell face number).	
walls(6,*)	none	Ending k mesh index (cell face number).	
walls(7,*)	none	Block number.	
walls(8,*)	none	Integer to identify the type (material and effective thickness) of wall through the walldf array. Only used if ihflag = 1. A value of 0 takes this wall out of heat-transfer considerations even when ihflag is set to 1. See Section 6.1.	
wi	0.0	Initial velocity in k-direction (cm/s). See Section 5.1.2.	
zeroddef(*,*)	none	Zero diffusion boundary at source reservoirs (plane surface).	yes
zeroddef(1,*)	none	Beginning i mesh index (cell face number).	yes
zeroddef(2,*)	none	Ending i mesh index (cell face number).	yes
zeroddef(3,*)	none	Beginning j mesh index (cell face number).	yes
zeroddef(4,*)	none	Ending j mesh index (cell face number).	yes
zeroddef(5,*)	none	Beginning k mesh index (cell face number).	yes
zeroddef(6,*)	none	Ending k mesh index (cell face number).	yes
zeroddef(7,*)	none	Block number.	yes

APPENDIX B

SUMMARY OF VARIABLES IN NAMELIST GROUP **meshgn**

Summary of Variables in NAMELIST Group **meshgn**

(Refer to Section 3 for detailed explanation of variables.)

Variable	Default	Description
dxmn (*)	none	Array to store minimum cell size (cm) in i-direction for each submesh. Used for automatic mesh generation.
dymn (*)	none	Array to store minimum cell size (cm) in j-direction for each submesh. Used for automatic mesh generation.
dzm (*)	none	Array to store minimum cell size (cm) in k-direction for each submesh. Used for automatic mesh generation.
iblkxp(*,*)	none	Multiblock connectivity information array. Not used in GASFLOW.
iblock	none	Block identification number.
nkx	none	Number of submesh in i-direction.
nky	none	Number of submesh in j-direction.
nkz	none	Number of submesh in k-direction.
nxl (*)	none	Array to store number of cells on the -i side for each submesh. Used for automatic mesh generation.
nxr (*)	none	Array to store number of cells on the +i side for each submesh. Used for automatic mesh generation.
nyl (*)	none	Array to store number of cells on the -j side for each submesh. Used for automatic mesh generation.
nyr (*)	none	Array to store number of cells on the +j side for each submesh. Used for automatic mesh generation.
nzl (*)	none	Array to store number of cells on the -k side for each submesh. Used for automatic mesh generation.

Variable	Default	Description
nzr(*)	none	Array to store number of cells on the +k side for each submesh. Used for automatic mesh generation.
xc(*)	none	Array to store location (cm) of edge of smallest cell in i-direction. Used for automatic mesh generation.
xgrid(*)	none	Array to store mesh coordinates in i-direction. Used for direct mesh definition.
x1(*)	none	Array to store i-coordinate (cm) of starting location of a submesh, which is the same as the ending location of the previous submesh. Used for automatic mesh generation.
yc(*)	none	Array to store location (cm) of edge of smallest cell in j-direction. Used for automatic mesh generation.
ygrid(*)	none	Array to store mesh coordinates in j-direction. Used for direct mesh definition.
y1(*)	none	Array to store j-coordinate (cm) of starting location of a submesh, which is the same as the ending location of the previous submesh. Used for automatic mesh generation.
zc(*)	none	Array to store location (cm) of edge of smallest cell in k-direction. Used for automatic mesh generation.
zgrid(*)	none	Array to store mesh coordinates in k-direction (cm). Used for direct mesh definition.
z1(*)	none	Array to store k-coordinate (cm) of starting location of a submesh, which is the same as the ending location of the previous submesh. Used for automatic mesh generation.

APPENDIX C

SUMMARY OF VARIABLES IN NAMELIST GROUP **rheat**

Variable	Default	Description	Restart
cbulkrlx	0.1	Relaxation factor for bulk condensation (s^{-1}). See Section 7.5.	yes
cfilmdef(*,*)	none	Area that defines the sump surface with constant film thickness. This area must also be defined as a horizontal wall with free space underneath, i.e., define as holes. See Section 7.7.2.	yes ??
cfilmdef(1,*)	none	Beginning i mesh index (cell face numbers).	yes??
cfilmdef(2,*)	none	Ending i mesh index (cell face numbers).	yes??
cfilmdef(3,*)	none	Beginning j mesh index (cell face numbers).	yes??
cfilmdef(4,*)	none	Ending j mesh index (cell face numbers).	yes??
cfilmdef(5,*)	none	Beginning k mesh index (cell face numbers).	yes??
cfilmdef(6,*)	none	Ending k mesh index (cell face numbers).	yes??
cfilmdef(7,*)	none	Block number (must be set to 1).	yes??
cfilmdef(8,*)	none	Constant film thickness (must be less than 100 cm).	yes??
cons1	4.0	Multiplication factor for the heat-transfer coefficient (applies to convection and condensation/evaporation).	yes
crelax	0.01	Relaxation factor for depleting liquid H_2O (s^{-1}). See also iliq .	yes
dxslabc	0.0	> 0. results in variable mesh spacing for all concrete slabs. The first dx for each concrete slab will be dxslabc ; = 0, use equal mesh spacing in concrete slab.	no
dxslabs	0.0	> 0. results in variable mesh spacing for all steel slabs. The first dx for each steel slab will be dxslabs ; = 0, use equal mesh spacing in steel slab.	no

Variable	Default	Description	Restart
filmth	0.5	> 0, maximum film thickness from condensation (cm); = 0, no vaporization allowed; < 0, a liquid film is initiated on all slab, wall, and sink surfaces with a thickness of filmth .	no
hslablin	1000	Heat-transfer coefficient on slab BC#1, used in evaluating the linear slab cell temperature profile with tslabbc as slab surface temp. on the positive side BC#2 and tn0(m) or tslab0 as fluid temp. on BC#1.	
icond	1	Flag for condensation on structures: 1 means ON; 0 means OFF.	yes
ienh	1	Film-model enhancement from Bird Stewart Lightfoot: 1 means ON; 0 means OFF; so far only for film condensation.	yes
ifcca	1	Flag for Chilton-Colburn analogy: 1 means ON; 0 means OFF. See Section 7.5.	yes
ihtflag	0	Option flag to activate heat transfer and steam condensation: 1 means ON; 0 means OFF.	no
iliq	1	Flag for depletion of H ₂ O liquid, see crelax : 1 means ON; 0 means OFF.	yes
ircomb	0	= 0, no recombiners; = 1, use recombiners.	yes ??
islablin	0	Flag to set an approximate linear initial temperature profile in each slab boundary cell, based on the temperature of slab BC#2 and fluid cell contiguous with BC#1: = 0 implies no linear profile (i.e., slab initial temperature is constant and equals fluid temperature or user input initial slab temperature); = 1 implies a linear profile with fluid temperature on negative side, or initial slab temperature (tslab0). Option applies to all slabs in one block in the same way.	
iwh2o	0	Flag for vaporization of film: 1 means ON; 0 means OFF.	yes

Variable	Default	Description	Restart
lprarea	0	Flag for output of heat structure surfaces into file areas.	yes
matbdy	1	Material type (uniform) for the mesh boundary: = 0 results in no boundary slabs with an adiabatic boundary of the computational mesh.	no
nhtesink	2	Number of heat-conduction elements in a sink heat structure (< 100).	no
nhteslab	2	Number of heat-conduction elements in a slab heat structure (< 100).	no
nhtewall	2	Number of heat-conduction elements in a wall heat structure (< 100).	no
nsumppts	none	 nsumppts = number of components in temperature vs. time table for the sump (see also sumptemp and sumptime): > 0, sumptemp is in kelvins; < 0, sumptemp is in degrees centigrade.	yes
ntotmat	3	Total number of structure materials. Read new properties only for materials 3 + imat to ntotmat because components 1 to 3 are automatically defined.	
rcombdef(*,*)	none	Array for the definition of recombiner location. See Section 7.7.3.	yes??
rcombdef(1,*)	none	Beginning i mesh index of recombiner cells (cell face numbers).	yes??
rcombdef(2,*)	none	Ending i mesh index of recombiner cells (cell face numbers).	yes??
rcombdef(3,*)	none	Beginning j mesh index of recombiner cells (cell face numbers).	yes??
rcombdef(4,*)	none	Ending j mesh index of recombiner cells (cell face numbers).	yes??
rcombdef(5,*)	none	Beginning k mesh index of recombiner cells (cell face numbers).	yes??
rcombdef(6,*)	none	Ending k mesh index of recombiner cells (cell face numbers).	yes??

Variable	Default	Description	Restart
<code>rcombdef(7,*)</code>	none	Block number.	yes??
<code>rcombdef(8,*)</code>	none	Recombiner area specified for region i, j, k (for recombiner type 1 or 2); projected flow area through the recombiner (for recombiner type 3 or 4). (not used for recombiner type 5 to 9)	yes??
<code>rcombdef(9,*)</code>	none	Recombiner type: = 1, NIS granulate recombiner (active ventilation option); = 2, Siemens recombiner (active ventilation option). =3 Siemens Reco FR90/1-100 with reaction rate from GX tests with buoyancy calculated from reaction rate =4 GRS type Reco (not currently in the code) =5 Siemens Reco type FR90/1-100 with reaction rate from Siemens =6 Siemens Reco type FR90/1-150 =7 Siemens Reco type FR90/1-320 =8 Siemens Reco type FR90/1-960 =9 Siemens Reco type FR90/1-1500	yes??
<code>rcombdef(10,*)</code>	-1	Number of levels below recombiner zone where hydrogen is sampled.	yes??
<code>rcombdef(11,*)</code>	0.008/ 0.031	Threshold of hydrogen concentration above which the recombiner starts operating (default of 0.008 is for NIS reco, 0.031 for Siemens reco with GX corr. and 0.02 for Siemens reco with Siemens correlations). If negative number is used for <code>rcombdef(11,*)</code> absolute value specifies the starting time of the recombiner operation.	yes??
<code>rcombdef(12,*)</code>	1800/ 100	Time constant to approach asymptotic flow rate in recombiner (default 1800 s for NIS reco, 100 s for Siemens reco).	yes??
<code>rcptabsink(*,*)</code>		Array that specifies ρC_p in sinks element by element (<code>nhtesink</code> components must be filled).	
<code>rcptabsink(*,imat)</code>		ρC_p (Density times specific heat) table for material <code>imat</code> for all sink elements: <code>imat</code> must be > 3; materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials.	

Variable	Default	Description	Restart
rcptabslab(*,*)		Array that specifies P.Cp in slabs element by element (nhteslab components must be filled).	
rcptabslab(*,imat)		P.Cp table for material imat for all slab elements: imat (must be > 3; materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials).	
rcptabwall(*,*)		Array that specifies P.Cp in walls element by element (nhtewall components must be filled).	
rcptabwall(*,imat)		P.Cp table for material imat for all wall elements: imat must be > 3; materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials.	
rholiqmx	2.e-6	Lower liquid density for liquid depletion (rainout) model (g/cm ³).	yes
sinkdef(*,*)	none	Array to define distributed heat sinks. See Section 6.4.	no
sinkdef(1,*)	none	Beginning i mesh index (cell face number).	no
sinkdef(2,*)	none	Ending i mesh index (cell face number).	no
sinkdef(3,*)	none	Beginning j mesh index (cell face number).	no
sinkdef(4,*)	none	Ending j mesh index (cell face number).	no
sinkdef(5,*)	none	Beginning k mesh index (cell face number).	no
sinkdef(6,*)	none	Ending k mesh index (cell face number).	no
sinkdef(7,*)	none	Block number.	no
sinkdef(8,*)	none	Material identification number: 1 means concrete; 2 means steel; 3 means superconductor. If i > 3, use table i for element-by-element input for thermal conductivity and P.Cp See also wltabsink , rcptabsink (i > 3 used for layered structures).	no
sinkdef(9,*)	none	Total material volume (cm ³).	no
sinkdef(10,*)	none	Average material thickness (cm).	no

Variable	Default	Description	Restart
sinkdef(11,*)	0.0	Sets BC for fluid/sink surface: = 0.0 implies sink fluid heat exchange; > 0.0 implies a given sink temperature on the fluid side (i.e., a cooler).	no
sinkdef(12,*)	none	Sets BC at the sink centerline: > 0.0 implies a constant temperature boundary; < 0.0 implies an adiabatic temperature boundary.	
sinkdef(13,*)	0.0	> 0.0 Δx for the fluid/surface node of the sink with dynamic mesh expansion for heat-conduction nodes; = 0.0 implies uniform mesh spacing for heat conduction.	
sinkdef(14,*)	0	Flag for further specification of BC on negative side: = 0, no further modification; = $i > 0$ gives table number from surftab that specifies time-dependent surface temperature, the initial temperature at $t = 0$ is taken from sinkdef(11,*) ; = -1 applies constant heat flux from sinkdef(16,*) and/or heat transfer with coefficient sinkdef(17,*) and fluid temperature sinkdef(11,*) on negative side (positive heat flux means add energy to wall).	
sinkdef(15,*)	0	Flag for further specification of BC on positive side (centerline) of slab: = 0, no further modification; = $i > 0$ gives table number from surftab that specifies time-dependent surface temperature, the initial temperature at $t = 0$ is taken from sinkdef(12,*) ; = -1 applies constant heat flux from sinkdef(16,*) and/or heat transfer with coefficient sinkdef(17,*) and fluid temperature sinkdef(12,*) on positive side (positive heat flux means add energy to wall).	
sinkdef(16,*)	0	Heat flux (ergs/cm ² s) applied as BC by sinkdef(14,*) or sinkdef(15,*) .	
sinkdef(17,*)	0	Heat-transfer coefficient (ergs/cm ² s K) applied as BC by walldf(7,*) or walldf(8,*) .	
slabdef(25,*)		Sets tslab0 , tslabbc , and slabthk for specific boundaries and blocks.	

Variable	Default	Description	Restart
<code>slabdef(1,*)</code>		Block number.	
<code>slabdef(2,*)</code>		Thickness of slabs on east boundary.	
<code>slabdef(3,*)</code>		Thickness of slabs on west boundary.	
<code>slabdef(4,*)</code>		Thickness of slabs on north boundary.	
<code>slabdef(5,*)</code>		Thickness of slabs on south boundary.	
<code>slabdef(6,*)</code>		Thickness of slabs on top boundary.	
<code>slabdef(7,*)</code>		Thickness of slabs on bottom boundary.	
<code>slabdef(8,*)</code>		Initial temperature for slabs on east boundary.	
<code>slabdef(9,*)</code>		Initial temperature for slabs on west boundary.	
<code>slabdef(10,*)</code>		Initial temperature for slabs on north boundary.	
<code>slabdef(11,*)</code>		Initial temperature for slabs on south boundary.	
<code>slabdef(12,*)</code>		Initial temperature for slabs on top boundary.	
<code>slabdef(13,*)</code>		Initial temperature for slabs on bottom boundary.	
<code>slabdef(14,*)</code>		Temperature BC for slabs on east boundary.	
<code>slabdef(15,*)</code>		Temperature BC for slabs on west boundary.	
<code>slabdef(16,*)</code>		Temperature BC for slabs on north boundary.	
<code>slabdef(17,*)</code>		Temperature BC for slabs on south boundary.	
<code>slabdef(18,*)</code>		Temperature BC for slabs on top boundary.	
<code>slabdef(19,*)</code>		Temperature BC for slabs on bottom boundary.	
<code>slabdef(20,*)</code>		Material type for slabs on east boundary.	
<code>slabdef(21,*)</code>		Material type for slabs on west boundary.	
<code>slabdef(22,*)</code>		Material type for slabs on north boundary.	
<code>slabdef(23,*)</code>		Material type for slabs on south boundary.	

Variable	Default	Description	Restart
slabdef(24,*)		Material type for slabs on top boundary.	
slabdef(25,*)		Material type for slabs on bottom boundary.	
slabthk	100.0	Thickness (cm) of a slab heat structure. Also used to determine whether obstacle is a wall or slab heat structure. See Section 6.1. Still determines whether inner obstacles are slabs or walls; slabdef statements can overwrite only the thickness of the boundary slabs.	no
sumptemp(*)	none	Sump temperature nsumppts components: in kelvins for nsumppts > 0; in degrees centigrade for nsumppts < 0.	yes
sumptime(*)	none	Time for sump temperature table (s).	yes
surftab(2,*,*)		Surface temperature vs. time table referenced by walldef(7,*) or walldef(8,*) and by sinkdef(14,*) or sinkdef(15,*) , respectively.	
surftab(2,j,i)		Pair of time (s) and surface temperature (K) at time for table i: $j_{\max} \leq 50$, $i_{\max} \leq 30$ 1 (the problem time must never exceed the maximum of the table time).	
teta	1.0	Heat-conduction integration flag: = 1.0, backward Euler; = 0.5, Crank-Nicholson.	yes
tsink0	300.0	Initial temperature (K) of sink heat structures. Negative values indicate temperature of adjacent fluid cell is to be used.	no
tslab0	300.0	Initial temperature (K) of slab heat structures. Negative values indicate temperature of adjacent fluid cell is to be used throughout the slab. If tslabbc is set, it overwrites the outer boundary temperature irrespective of tslab0 .	no
tslabbc	0.0	Slab temperature boundary condition for all blocks and boundaries away from the fluid node, can be overridden using slabdef : < 0. implies adiabatic BC; > 0. implies fixed outer boundary temperature; = 0. (not input) then tslabbc is determined from tslab0 and kept constant on the boundary.	no

Variable	Default	Description	Restart
twall0	300.0	Initial temperature (K) of wall heat structures. Negative values indicate temperature of adjacent fluid cell is to be used.	no
walldf(*,*)	none	Wall type definition array. See Section 6.2.	
walldf(1,*)	none	Material identification number: 1 means concrete; 2 means steel; 3 means superconductor. If $i > 3$ use table i for element-by-element input for thermal conductivity and PCp. See also wltabwall , rcptabwall ($i > 3$ used for layered structures).	
walldf(2,*)	none	Effective thickness of wall (cm).	
walldf(3,*)	0.0	Sets BC#1, i.e., defines conditions on the negative side of the wall: = 0.0 implies a fluid-wall heat exchange (default); > 0.0 implies a constant wall temperature $T = \text{walldf}(3,*)$; < 0.0 implies an adiabatic wall condition.	
walldf(4,*)	0.0	Sets BC#2, i.e., defines conditions on the positive side of the wall: = 0.0 implies a fluid-wall heat exchange (default); > 0.0 implies a constant wall temperature $T = \text{walldf}(4,*)$; < 0.0 implies an adiabatic wall condition.	
walldf(5,*)	0.0	Δx for the node size on the two surfaces of the wall with dynamic mesh expansion symmetrical to the midplane of the node for the given wall thickness; = 0.0 implies a uniform mesh spacing.	
walldf(6,*)	1.0	Fraction of wall area from mesh plane that is used for heat transfer.	

Variable	Default	Description	Restart
walldf(7,*)	0	Flag for further specification of BC on negative side: = 0, no further modification; = $i > 0$ gives table number from surftab that specifies time-dependent surface temperature, the initial temperature at $t = 0$ is taken from walldf(3,*) ; = -1 applies constant heat flux from walldf(9,*) and/or heat transfer with coefficient walldf(10,*) and fluid temperature walldf(3,*) on negative side (positive heat flux means add energy to wall). Only apply fuel conditions to one side of the wall; i.e., do not set walldf(8,*) to -1 when walldf(7,*) is -1.	
walldf(8,*)	0	Flag for further specification of BC on positive side: = 0, no further modification; = $i > 0$ gives table number from surftab that specifies time-dependent surface temperature, the initial temperature at $t = 0$ is taken from walldf(4,*) ; = -1 applies constant heat flux from walldf(9,*) and/or heat transfer with coefficient walldf(10,*) and fluid temperature walldf(4,*) on positive side (positive heat flux means add energy to the wall). For restrictions, see above.	
walldf(9,*)	0	Heat flux (ergs/cm ² s) applied as BC by walldf(7,*) or walldf(8,*) , positive heat flux means add energy to wall.	
walldf(10,*)	0	Heat-transfer coefficient (ergs/cm ² s K) applied as BC by walldf(7,*) or walldf(8,*) .	
wltsink(*,*)		Array that specifies thermal conductivity in sinks element by element (nltsink components must be filled).	
wltsink(*,imat)		Thermal conductivity table for material imat for all sink elements: imat must be > 3 ; materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials.	
wltslab(*,*)		Array that specifies thermal conductivity in slabs element by element (nltslab components must be filled).	

Variable	Default	Description	Restart
<code>wltabslab(*,imat)</code>		Thermal conductivity table for material imat for all slab elements: imat must be > 3; materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials.	
<code>wltabwall(*,*)</code>		Array that specifies thermal conductivity walls element by element (nhtewall components must be filled).	
<code>wltabwall(*,imat)</code>		Thermal conductivity table for material imat for all wall elements: imat must be > 3; materials 1 to 3 are automatically filled with constant values for concrete, steel, and superconducting materials.	

APPENDIX D

SUMMARY OF VARIABLES IN NAMELIST GROUP **grafic**

Variable	Default	Description
c2d(*,*)	none	Array for defining 2D contour plots. See Section 9.1.5.
c2d(1,*)	none	Identification number for first point (second index in pnt).
c2d(2,*)	none	Identification number for second point (second index in pnt).
c2d(3,*)	none	Solution variable. Choose from list in Table 9-1.
c2d(4,*)	none	Gas species name or component number. Required only if c2d(3,*) is 'rsn', 'mf', or 'vf'.
gline	'on'	Option flag for drawing grid lines on time-history plots: 'on' means grid lines drawn; 'off' means grid lines not drawn.
glinepld	'on'	Option flag for drawing grid lines on 1D profile plots: 'on' means grid lines drawn; 'off' means grid lines not drawn.
ht1dp(*,*)	none	Array for defining heat structure temperature profile plots. See Section 9.1.4.
ht1dp(1,*)	none	i-index of fluid cell in contact with heat structure.
ht1dp(2,*)	none	j-index of fluid cell in contact with heat structure.
ht1dp(3,*)	none	k-index of fluid cell in contact with heat structure.
ht1dp(4,*)	none	Block number (must be set to 1).
ht1dp(5,*)	none	Heat structure type. Choices are 'slab', slab heat structure; 'sink', sink heat structure; 'wall', wall heat structure.

Variable	Default	Description
htldp(6,*)	none	Side of fluid cell in contact with heat structure (not needed for sink heat structure). Choices are 'east', +i side of fluid cell; 'west', -i side of fluid cell; 'north', +j side of fluid cell; 'south', -j side of fluid cell; 'top', +k side of fluid cell; 'bottom', -k side of fluid cell.
htthp(*,*)	none	Array for defining heat structure surface temperature time-history plots. See Section 9.1.3.
htthp(1,*)	none	i-index of fluid cell in contact with heat structure.
htthp(2,*)	none	j-index of fluid cell in contact with heat structure.
htthp(3,*)	none	k-index of fluid cell in contact with heat structure.
htthp(4,*)	none	Block number (must be set to 1).
htthp(5,*)	none	Heat structure type. Choices are 'slab', slab heat structure; 'sink', sink heat structure; 'wall', wall heat structure.
htthp(6,*)	none	Side of fluid cell in contact with heat structure (not needed for sink heat structure). Choices are 'east', +i side of fluid cell; 'west', -i side of fluid cell; 'north', +j side of fluid cell; 'south', -j side of fluid cell; 'top', +k side of fluid cell; 'bottom', -k side of fluid cell.
igrd	1	Option flag for drawing grid box (axes) on 2D contour and vector plots: 1 means ON; 0 means OFF.
iinc	1	i-direction cell increment between velocity vectors.
ippka(*)	1	Particle type array for particle plots.
ipvew(*)	1	Viewpoint array for particle plots.
jinc	1	j-direction cell increment between velocity vectors.
kinc	1	k-direction cell increment between velocity vectors.

Variable	Default	Description
nap	1	Number of film frames advanced between particle plots.
ncontur	7	Number of contour levels. See Section 9.1.5.
npplts	0	Number of particle plots.
p1d(*,*)	none	Array for defining 1D profile plots. See Section 9.1.4.
p1d(1,*)	none	Identification number for first point (second index in pnt).
p1d(2,*)	none	Identification number for second point (second index in pnt).
p1d(3,*)	none	Solution variable. Choose from list in Table 9-1.
p1d(4,*)	none	Gas species name or component number. Required only if p1d(3,*) is 'rsn', 'mf', or 'vf'.
pnt(*,*)	none	Array to define points for profile, contour, and vector plots. See Section 9.1.4.
pnt(1,*)	none	i-mesh index.
pnt(2,*)	none	j-mesh index.
pnt(3,*)	none	k-mesh index.
pnt(4,*)	none	Block number (must be set to 1).
pplt(2,100)	none	Perspective plot input designating view coordinates (viewcrds) chosen.
pthp(*,*)	none	Array for defining particle time-history plots.
pthp(1,*)	none	i index for fluid cell.
pthp(2,*)	none	j index for fluid cell.
pthp(3,*)	none	k index for fluid cell.
pthp(4,*)	none	Block number.
pthp(5,*)	none	Particle data plot option (see Section 9.1.7).
pthp(6,*)	none	Particle class number (see Section 9.1.7).

Variable	Default	Description
pthp(7,*)	none	Particle size number (see Section 9.1.7).
pthp(8,*)	none	Particle has deposited face (see Section 9.1.7).
pthp(9,*)	none	Particle cloud monitor number.
pthpts	0.0	Time-history plot offset for particle plots.
thdt	10 ¹⁰⁰	Time interval (s) at which time-history data are stored. See Section 9.1.3.
thp(*,*)	none	Array for defining time-history plots. See Section 9.1.3.
thp(1,*)	none	i-mesh index (cell number or cell face number).
thp(2,*)	none	j-mesh index (cell number or cell face number).
thp(3,*)	none	k-mesh index (cell number or cell face number).
thp(4,*)	none	Block number (must be set to 1).
thp(5,*)	none	Solution variable. Choose from list in Table 9-1.
thp(6,*)	none	Gas species name or component number. Required only if thp(5,*) is 'rsn', 'mf', or 'vf'.
v2d(*,*)	none	Array for defining 2D vector plots. See Section 9.1.6.
v2d(1,*)	none	Identification number for first point (second index in pnt).
v2d(2,*)	none	Identification number for second point (second index in pnt).
v2d(3,*)	none	Film advance flag: 0 means NO; 1 means YES.
v3d(*,*)	none	Array for defining 3D vector plots. See Section 9.1.6.
v3d(1,*)	none	Identification number for first point (second index in pnt).
v3d(2,*)	none	Identification number for second point (second index in pnt).
v3d(3,*)	none	Film advance flag: 0 means NO; 1 means YES.
v3d(4,*)	none	Number of 3D view coordinates definition (second index of corresponding viewcrds definition).

Variable	Default	Description
<code>viewcrds(*,*)</code>	none	Array for defining 3D viewing coordinates.
<code>viewcrds(1,*)</code>	none	i object coordinate (cm).
<code>viewcrds(2,*)</code>	none	j object coordinate (cm).
<code>viewcrds(3,*)</code>	none	k object coordinate (cm).
<code>viewcrds(4,*)</code>	none	i eye point coordinate (cm).
<code>viewcrds(5,*)</code>	none	j eye point coordinate (cm).
<code>viewcrds(6,*)</code>	none	k eye point coordinate (cm).

APPENDIX E

SUMMARY OF VARIABLES IN NAMELIST GROUP **innet**

Variable	Default	Description
blowdef (*,*)	none	Array to define blower components. See Section 3.4.9.
blowdef (1,*)	none	First <i>i</i> index.
blowdef (2,*)	none	Last <i>i</i> index.
blowdef (3,*)	none	Duct number.
blowdef (4,*)	none	Block number (not used).
blowdef (5,*)	none	Table number for blower head vs. volumetric flow. If zero, then blower will run at rated conditions.
blowdef (6,*)	none	Table number for the blower speed vs. time table. If zero and blowdef (5,*) is not zero, then the blower will run at a constant speed, given by blowdef (10,*).
blowdef (7,*)	none	Rated blower speed (radians/s).
blowdef (8,*)	none	Rated blower volumetric flow (cm ³ /s).
blowdef (9,*)	none	Rated blower head (cm).
blowdef (10,*)	none	Blower speed at time = 0.0 (radians/s).
blwtb (*,*,*)	none	Blower head tables. See Section 3.4.9.
blwtb (1, <i>ip</i> , <i>it</i>)	none	Volumetric flow (cm ³ /s) for the <i>ip</i> th point in the <i>it</i> th table.
blwtb (2, <i>ip</i> , <i>it</i>)	none	Blower head (cm) for the <i>ip</i> th point in the <i>it</i> th table.
cpnt (*,*)	none	<i>x</i> -, <i>y</i> -, <i>z</i> -coordinates (cm) in space for each node point. See Section 3.4.1.
cpnt (1, <i>i</i>)	none	<i>x</i> -coordinate (cm) in space for <i>i</i> th node point.
cpnt (2, <i>i</i>)	none	<i>y</i> -coordinate (cm) in space for <i>i</i> th node point.
cpnt (3, <i>i</i>)	none	<i>z</i> -coordinate (cm) in space for <i>i</i> th node point.

Variable	Default	Description
dampdef(*,*)	none	Array to define damper components. See Section 3.4.8.
dampdef(1,*)	none	First i index.
dampdef(2,*)	none	Last i index.
dampdef(3,*)	none	Duct number.
dampdef(4,*)	none	Block number (not used).
dampdef(5,*)	none	Table number for damper movement as a function of time. If table number is zero, then a constant value is used for damper flow area fraction and is given by dampdef(8,*) .
dampdef(6,*)	none	Damper flow loss coefficient when fully open.
dampdef(7,*)	none	Damper flow area when fully open (cm ²).
dampdef(8,*)	none	Damper flow area fraction at time = 0.0.
defjct(*,*)	none	Array that defines junctions where more than one duct connects. See Section 3.4.3.
defjct(1,*)	none	Overall dimension in the x-direction.
defjct(2,*)	none	Overall dimension in the y-direction.
defjct(3,*)	none	Overall dimension in the z-direction.
defjct(4,*)	none	Node point that junction is centered around.
dmptb(*,*,*)	none	Damper flow area fraction table. See Section 3.4.8.
dmptb(1,ip,it)	none	Time (s) for the ip th point in the it th table.
dmptb(2,ip,it)	none	Damper flow area fraction for the ip th point in the it th table.
ductdef(*,*)	none	Array that defines the 1D ducts for the ventilation system model. See Section 3.4.2.
ductdef(1,*)	none	Cell length at west end (cm).
ductdef(2,*)	none	Cell length at east end (cm).
ductdef(3,*)	none	Cell width at west end (cm).

Variable	Default	Description
ductdef(4,*)	none	Cell width at east end (cm).
ductdef(5,*)	none	Cell height at west end (cm). If zero is input, then it is assumed that the cell width is the diameter of a circular duct.
ductdef(6,*)	none	Cell height at east end (cm). If zero is input, then it is assumed that the cell width is the diameter of a circular duct.
ductdef(7,*)	none	Node point number at west end.
ductdef(8,*)	none	Node point number at east end.
ductdef(9,*)	none	Number of real cells.
ductdef(10,*)	none	Mesh expansion factor from west to east.
ductdef(11,*)	none	dfit(1,*) duct fitting type. See Section 3.4.2.
ductdef(12,*)	none	dfit(2,*) duct fitting variable. See Section 3.4.2.
filtrdef(*,*)	none	Array to define filter components. See Section 3.4.7.
filtrdef(1,*)	none	First i index.
filtrdef(2,*)	none	Last i index.
filtrdef(3,*)	none	Duct number.
filtrdef(4,*)	none	Block number (not used).
filtrdef(5,*)	none	Filter laminar flow loss coefficient.
filtrdef(6,*)	none	Filter turbulent flow loss coefficient.
filtrdef(7,*)	none	Filter effective flow area (cm ²).
flossdef(*,*)	none	Array to define flow losses in the 1D ducts. See Section 3.4.5.
flossdef(1,*)	none	First i index.
flossdef(2,*)	none	Last i index.
flossdef(3,*)	none	Duct number.
flossdef(4,*)	none	Block number (not used).

Variable	Default	Description
flossdef(5,*)	none	Forward flow loss coefficient. If the forward flow loss is less than zero, then it is assumed that flossdef(5,*) is the outer diameter of an annulus and that flossdef(6,*) is the inner diameter. Appropriate flow losses and wall drag coefficients will be calculated for this geometry.
flossdef(6,*)	none	Reverse flow loss coefficient. If zero, then reverse flow loss coefficient is set equal to the forward flow loss coefficient. If flossdef(5,*) is less than zero, then flossdef(6,*) is the inner diameter of annular geometry and will be used to calculate appropriate flow losses and wall drags.
iwshear	1	Wall shear flag: = 0 implies no wall shear is included in the 1D ducts; = 1 implies wall shear is included in the 1D ducts.
nwcx(*,*)	none	Array that defines 1D duct to 3D block connections. See Section 3.4.4.
nwcx(1,*)	none	Beginning i mesh index (cell face numbers) of 3D block.
nwcx(2,*)	none	Ending i mesh index (cell face numbers) of 3D block.
nwcx(3,*)	none	Beginning j mesh index (cell face numbers) of 3D block.
nwcx(4,*)	none	Ending j mesh index (cell face numbers) of 3D block.
nwcx(5,*)	none	Beginning k mesh index (cell face numbers) of 3D block.
nwcx(6,*)	none	Ending k mesh index (cell face numbers) of 3D block.
nwcx(7,*)	none	3D block number.
nwcx(8,*)	none	Network node number.
spdtb(*,*,*)	none	Blower speed tables. See Section 3.4.9.
spdtb(1,ip,it)	none	Time (s) for the i_p^{th} point in the i_t^{th} table.
spdtb(2,ip,it)	none	Blower speed (radians/s) for the i_p^{th} point in the i_t^{th} table.

APPENDIX F

SUMMARY OF VARIABLES IN NAMELIST GROUP **parts**

The **parts** NAMELIST group is necessary only for problems where particle dynamics are to be modeled or where tracer particles are wanted. It contains all of the particle input parameters, except those for the particle graphics output and **m****sp**, **i****tp****cl**, and **i****tps****z**. The particle graphics parameters are input in the NAMELIST group **grafic**; **m****sp**, **i****tp****cl**, and **i****tps****z** are input as part of the **xput** NAMELIST group. NAMELIST group **parts** is selected when **solatype** = 1 or 2. It is located in and called from subroutine **RPARTS**. All of the arrays and variables in NAMELIST group **parts** are in named common **CPARTS**.

The following input parameters are to be included in the GASFLOW input file **ingf**. They are to follow **\$parts** and precede **\$end**. In each case, the **\$** is in column 2.

Variable	Default	Description
cdrf	10.0	Drag coefficient roughness factor.
core	0.96	Coefficient of restitution of particle material.
dbl	10.0	Boundary layer thickness (cm).
depperc	5.0	Percentage of particles that immediately deposit upon impact.
hca	1.0e-12	Hamaker constant (ergs).
icloud	0	Particle cloud model flag: = 0, particle cloud model is off; = 1, cloud model is on.
imarker	0	Marker particle flag: = 1, particles are moved at the local fluid velocity (most of the particle physics is bypassed); = 0, the complete particle dynamics algorithm is executed.
inpvol	0	Input parameter to choose actual or fictitious volume $pvol = \Delta x \Delta y \Delta z$ for cell in which particle is located; = 0, actual; = 1, fictitious.
intrn	0	Input parameter to choose particle entrainment option: = 0, no particle pickup; = 1, pickup option is on.

Variable	Default	Description
ipblkln	1	Mesh block in which particle is initially located.
ipclln	1	Particle class number input for each class.
ipdep	0	Particle deposition flag input for each class: = 0, no adhesion; = 1, adhesion determined by deposition model; = 2, all impacting particles adhere.
ndxdp	-1	Number of longitudinal sections in x-direction in which particles deposition is monitored. Default = -1, deposition not monitored or plotted.
niterp	30	Number of iterations for the Newton iteration cycle for determining drag-induced particle velocity.
npinp	1000	Total number of <i>simulation</i> particles to be input.
ntmtr	0	Total number of particle cloud monitors.
pcdc	0	Particle cloud diffusion coefficient (cm ² /s).
pdiamin	1.0	Particle diameter input for each class and size (cm).
pinpdt	1.0e+06	Time interval in seconds between particle inputs for each class.
pmass	1.0	Total <i>real</i> particle mass of each class and size.
poisrt	0.29	Poisson ratio of particle material class.
prho	1.0	Material density input for each particle class (g/cm ³).
scfacin	1.0	Cunningham slip correction factor for each particle class and size.
sdz	4.0e-08	Separation distance of particle and substrate (cm).
sdzrf	100.0	Separation distance roughness factor.
tdcp	0.0	Particle turbulent diffusion coefficient (cm ² /s).
tinjt	0.01	Total injection time for each particle class (s).
twpinp	0.0	Time when to input particles of each class (s).

Variable	Default	Description
visf	nu	Coefficient of kinematic viscosity of fluid (= nu) (cm ² /s).
wmax	2.0	Maximum argument of error function used in inverse error function table.
xm	0.0	x-coordinate of particle cloud monitor (cm).
xpe	0.0	Maximum x-coordinate of initial particle block for each particle class (cm).
xpw	0.0	Minimum x-coordinate of initial particle block for each particle class (cm).
ym	0.0	y-coordinate of particle cloud monitor (cm).
yngmod	2.10e+12	Young's modulus of particle material (dynes/cm ²).
ypn	0.0	Maximum y-coordinate of initial particle block for each particle class (cm).
yps	0.0	Minimum y-coordinate of initial particle block for each particle class (cm).
zm	0.0	z-coordinate of particle cloud monitor (cm).
zpb	0.0	Minimum z-coordinate of initial particle block for each particle class (cm).
zpt	0.0	Maximum z-coordinate of initial particle block for each particle class (cm).

APPENDIX G

SAMPLE INPUT DECK WITH MINIMUM DATA REQUIRED

To set up a GASFLOW problem, one must, at the minimum, define a mesh, specify what fluid species are involved, and prescribe any appropriate initial and boundary conditions. In addition desired model options have to be activated. Many of the input variables used by GASFLOW have default values. For example, the default boundary condition is free-slip rigid wall, and the default initial velocity is zero. However, variables like **mat**, **gasdef**, and those in NAMELIST group meshgn (for mesh generation) have no defaults. Therefore, for a problem in which the fluid is initially at rest and is enclosed by free-slip solid boundaries, the minimum input would be that required to define the fluid species (**mat**), the initial fluid thermodynamic condition (**gasdef**), and the mesh (NAMELIST group meshgn). An input deck that has such minimum required data is shown in Figure G-1, which should help the new user to set up a simple problem quickly.

The fluid in the problem is air, which is initially at 300 K and 1×10^6 dynes/cm² pressure. Because the problem specifies no inflow or outflow and does not activate any physical models (such as heat transfer and gravity), the uniform pressure, temperature, and velocity fields should persist as the calculation advances in time. (In this case, the initial condition is the steady solution.) The accuracy of the calculation (measured by deviation of the velocity from zero, for example) is controlled by the pressure iteration convergence criterion (**epsi0**, default = 1×10^{-5}) and by the maximum iteration number allowed per cycle (**itmax**, set to 40 in this problem). The default initial time-step size (**delt0**) is 0.02 s, and the problem end time (**twfin**) is specified as 1.0 s. By default, automatic time-step control is chosen. Specification of the graphic NAMELIST variables is not strictly required, but contour and vector plots are graphics that are commonly asked for. With these problem specifications, the maximum velocity magnitude at the end of the calculation is 4×10^{-6} cm/s.

```

Fictitious Problem with Minimum Input Data          HMS-93
TSA-8 Lam
HMS-Play
NOTES: 2-D domain 100 cm x 250 cm with deltax = deltax = deltax = 5 cm
       Number of fluid cells = 20 x 50 x 1 for the
       coordinate dimension x, y, and z, respectively.
       This problem is a closed box of pure air experiencing no artificial
       perturbation. The solution should show no deviation from the
       initial conditions (300 K, 1 bar, no flow) as problem time increases.
-----
                M A I N      I N P U T
-----
$innet
$end
$xput
  twfin    = 1.0,      ; Default problem finish time is 10.0, so this
                      ; definition is not necessary for the code
                      ; to run. However, default pltdt (plotting
                      ; interval) is 1.0, so we set the problem time
                      ; to be the same so that only one set of
                      ; contour and vector plots are produced.
                      ; Default delto is 0.02, so there will be
                      ; a few cycles carried out.

  itmax     = 40,      ; Default is 20, which is not enough to get
                      ; an accurate solution for this problem, as
                      ; the mesh is not trivially small.

  maxcyc    = 9999999, ; Set maximum time cycle to large number
                      ; to ensure problem end time is reached.
                      ; Default maximum number of cycles is 10.

  mat       = 'air',

; Initial Condition Throughout domain:

  gasdef(1,1) = 1, 21, 1, 51, 1, 2, 1,
               1.000e6, 300.0, 2, 0., 0.,
               'air', 1.0,

$end
-----
                M E S H      G E N E R A T I O N
-----
$meshgn
  iblock = 1,

  nkx=1,
  x1(1)= 0.0, xc(1) = 0.000, nx1(1)= 0, nxx(1)=20, dxmn(1)= 9999.,
  x1(2)= 100.0,
  nky=1,
  y1(1)= 0., yc(1) = 0.000, nyl(1)= 0, nyr(1)=50, dymn(1)= 9999.,
  y1(2)= 250.0,

  nkz=1,
  z1(1)=0.0000, zc(1) = 0.0000, nz1(1)= 0, nzz(1)= 1., dzmn(1)= 9999.,
  z1(2)= 5.,

$end
-----
                G R A P H I C S
-----
$grafic
  thdt = 0.1      ; Want to get the basic time history plots, because
                  ; default value for thdt (interval between time
                  ; history data are written and plotted) is 1.e100.
                  ; With thdt = 0.1, and twfin = 0.5, each basic time
                  ; history plot will have six data points, including
                  ; beginning and end times.

; Define two points that would cover the entire physical x-y domain.
  pnt(1, 1) = 1, 1, 2, 1,
             21, 51, 2, 1,

  v2d      = 1, 02, 1,      ; velocity vector plot on plane
                        ; defined by points 1 and 2.

  c2d      = 1, 02, 'tk' , 0, ; temperature contour plot on plane
                        ; defined by points 1 and 2.

$end

$rheat
$end
$parts
$end
$special
$end

```

Figure G-1. Input deck showing minimum set of data required to run GASFLOW