A computational model of two-phase, turbulent atmospheric boundary layers containing blowing snow
by Glen Eddy Liston

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Civil Engineering
Montana State University

Abstract:

In blowing and drifting snow, two distinct flow regimes are found. At the surface is the saltation layer which, although very thin, is responsible for the majority of mass transport. Above this layer, the flow can be described as a turbulent, two-phase mixture of air and snow particles. These two layers are highly coupled, and therefore an accurate description of the general snow transport process requires a description of processes occurring within both of these layers.

A physically based computational model of the salient features of blowing and drifting snow in two-dimensional terrain is developed. The model has two distinct parts, one describing the turbulent flow mixture of air and snow, and a second describing the mass transport process and resulting snow accumulation patterns related to the saltation layer. The turbulent flow model consists of a general solution of the time averaged, two-dimensional Navier-Stokes equations, where the k-epsilon turbulence model is used to close the system of equations. The effect of particulates on the turbulent flow field is accounted for by computing the particle concentration field using a convection-diffusion equation and a subsequent modification of the k-epsilon model. The turbulent flow model is coupled to a saltation model and the time evolution of drift development and wind flow fields are computed.

The model suggests that, for the case of precipitating snow, snow particles can be considered a passive additive to the turbulent flow field. Modeled snow accumulation profiles are very similar to published field and experimental data.

A COMPUTATIONAL MODEL OF TWO-PHASE, TURBULENT ATMOSPHERIC BOUNDARY LAYERS CONTAINING BLOWING SNOW

by

Glen Eddy Liston

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Civil Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

March 1991

APPROVAL

of a thesis submitted by

Glen Eddy Liston

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

1 March 91
Date

R L Brown
Chairperson, Graduate Committee

Approved for the Major Department

26 Feb '91
Date

Theodore E. Lang
Head, Major Department

Approved for the College of Graduate Studies

March 1, 1991
Date

Henry L Parsons
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a doctoral degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. I further agree that copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for extensive copying or reproduction of this thesis should be referred to University Microfilms International, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom I have granted "the exclusive right to reproduce and distribute copies of the dissertation in and from microfilm and the right to reproduce and distribute by abstract in any format."

Signature Liston

Date 1 March 1991

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to Dr. Robert L. Brown for his unfailing support and guidance throughout this research effort. His willingness to allow me to develop my own direction and focus on my professional objectives has gone far towards making these past few years very successful. In addition, I would like to thank Drs. Bowers, Brown, Demetriades, Dent, Hansen, and Vogel for their instruction and consultation; they have all contributed to my educational and professional growth and success during my doctoral program at Montana State University.

This work was supported by the Center for High-Elevation Studies, the Department of Civil and Agricultural Engineering, and the Engineering Experiment Station at Montana State University.

TABLE OF CONTENTS

Page

TABLE OF CONTENTS-Continued

LIST OF FIGURES

LIST OF FIGURES-Continued

LIST OF FIGURES-Continued

Figure Page

Appendix Figures

ABSTRACT

In blowing and drifting snow, two distinct flow regimes are found. At the surface is the saltation layer which, although very thin, is responsible for the majority of mass transport. Above this layer, the flow can be described as a turbulent, two-phase mixture of air and snow particles. These two layers are highly coupled, and therefore an accurate description of the general snow transport process requires a description of processes occurring within both of these layers.

A physically based computational model of the salient features of blowing and drifting snow in two-dimensional terrain is developed. The model has two distinct parts, one describing the turbulent flow mixture of air and snow, and a second describing the mass transport process and resulting snow accumulation patterns related to the saltation layer. The turbulent flow model consists of a general solution of the time averaged, two-dimensional Navier-Stokes equations, where the k-epsilon turbulence model is used to close the system of equations. The effect of particulates on the turbulent flow field is accounted for by computing the particle concentration field using a convection-diffusion equation and a subsequent modification of the k-epsilon model. The turbulent flow model is coupled to a saltation model and the time evolution of drift development and wind flow fields are computed.

The model suggests that, for the case of precipitating snow, snow particles can be considered a passive additive to the turbulent flow field. Modeled snow accumulation profiles are very similar to published field and experimental data.

CHAPTER 1

INTRODUCTION

This document describes a study of the physical processes which govern blowing and drifting snow and the snow accumulation and distribution patterns which result from these processes. The study combines techniques from the fields of turbulence modeling, suspended and bed load transport, and computational fluid mechanics to describe the phenomenon of wind transported snow.

Outside our doors in the winter environment, whether it be in the city, the country, or the mountains, snow depth is found to be highly variable as one moves from one place to another. This variability can be caused by several factors, including (1) variations in air temperature and radiation balance, (2) local orographic influences on precipitation and storm movement, (3) the topographic variability of the ground surface, and (4) snow transport due to avalanches. However, when viewed on a small local scale (say 5 to 500 meters) these factors are generally considered secondary processes. The dominant process which produces spatially varying snow depths is the redistribution of snow by wind; this generally occurs during or following precipitation or storm events.

The spatial distribution of seasonal snow is a significant feature of both middle and upper latitude environments. Its influence on both global and local scales is dramatic. On the global scale, the interaction between heat gain at low latitudes, and heat loss at high latitudes, is the primary mechanism driving global atmospheric circulation and weather patterns. Since snow covered surfaces have a much higher albedo (surface reflectivity) than vegetated surfaces, the seasonal snow distribution in high latitudes strongly influences the earth's radiation balance (McFadden and Ragotzkie, 1967). As a consequence, snow distribution is a critical parameter affecting global circulation and climate.

On the local scale, the seasonal snow distribution is an important hydrological parameter affecting farmland irrigation, hydroelectric power, and water supplies for business and domestic use. The snowcover is also utilized as a recreation medium, providing sport for skiers, snowmobilers, and other winter outdoor enthusiasts. The hazardous aspect of snow affects transportation by reducing visibility, forming drifts on roadways, and reducing traction. In addition, hazards such as snow avalanches and snowmelt flooding may pose a threat to both property and life.

In Arctic Alaska, seasonal snow is a dominant feature of the landscape for nine months of each year. Several problems have been identified which relate directly to the spatial variation of this snowcover. In regions lacking sufficient fresh water supply due to lack of runoff or suitable reservoirs, for instance, there is an interest in accumulating snow into large drifts to serve as summer water sources (Slaughter et al., 1975). In addition, the petroleum industry spends millions of dollars annually, removing snow from materiel sites, drill pads, and roads at their arctic drilling locations. Furthermore, Federal regulations designed to protect underlying arctic vegetation stipulate, based on snowcover amount, when travel across the arctic tundra is permissible. Only through knowledge of snow distribution patterns can this determination be made accurately and thus fulfill its intended objective.

In Antarctica, a continent where more than 95 percent of the land area is covered by a blanket of snow and ice, blowing and drifting snow has been a significant factor affecting all research and exploration conducted there. The United States Antarctic Research Program bases McMurdo and South Pole, for instance, continually have to deal with drifting snow. The continent is a desert with low annual snow accumulation, but since little of it melts, the supply of snow for drifting is literally limitless. At McMurdo Station, the buildings at the near-by air field are virtually buried by drifting snow during the long winter months. Crews spend much of the austral summer digging these buildings out and moving the snow far enough away so it won't induce further drifting. At South Pole Station, where the air temperature never rises above freezing, snow removal is a never ending task. In spite of the crews' efforts, the station is slowly being drifted under.

Snowcover influences not only the activities of people, but also flora and fauna are affected by its distribution. Vegetation growth is influenced by spring and summer snowmelt from drifts formed the previous winter. In addition, winter wheat crops in the northern latitudes require the insulation provided by snowcover to withstand low winter temperatures. The depth of snow dictates where livestock, deer, elk, and moose can feed. Also, microtine rodents living beneath the snow depend on it for protection from predators and from extremes of wind and temperature in the environment above.

Despite the significant role that snow accumulation patterns play in our lives and the world around us, severe deficiencies exist in our ability to describe and model the relevant processes and resulting snow distribution. Here I attempt to fill some of these deficiencies as well as provide a contribution to disciplines interested in describing turbulent-sediment transport.

This research project studies the blowing and drifting snow problem and develops a physically based model describing its significant features. While the current knowledge base is concerned largely with one-dimensional flows, a unique feature of this study is a focus on two-dimensional flows. The current state of understanding and technology clearly indicates the need and capability to make the next step forward and describe this more complex flow configuration. Recent advances in modeling turbulent two-phase flows provide an exciting opportunity to use this information to develop a computational model of snow transport processes.

In blowing and drifting snow, two distinct flow regimes are found. At the surface is the saltation layer. This layer is approximately 5 cm thick and is characterized by particles repeatedly impacting the surface, dislodging additional particles into the air to be brought back to the surface under the influence of gravity. Above this layer, the flow can be described as a turbulent, two-phase mixture of air and snow particles. These two layers are highly coupled, and therefore an accurate description of the general snow transport process relies heavily on the descriptions of motions contained within both of these layers.

A model is developed which describes the turbulent air and snow mixture flowing above the saltation layer. A saltation model is then adopted and the two models are coupled through their common boundary and used to describe the fundamental features of the blowing and drifting snow problem. Features to be described include (1) the flow field of the turbulent air-snow mixture, (2) the snow concentration field within the air-snow mixture, and (3) the resulting snow accumulation profiles. Published experimental results are used to evaluate the computational model.

CHAPTER 2

PREVIOUS WORK ON BLOWING AND DRIFTING SNOW

Past studies of blowing and drifting snow can roughly be broken into two general categories: those studying the physical characteristics of the process, and those modeling the process and effects. Investigations of the physical characteristics of blowing and drifting snow have been largely observational in nature. These studies include measurements of particle size distributions, saltation parameters, flux profiles, and drift formations around natural and created obstructions to the flow.

Detailed reviews of many of these studies can be found in Mellor (1965, 1970), Radok (1977), Male (1980), Kind (1981), and Schmidt (1982a). The results of these reviews and the following cited literature is not summarized here. They are presented as an indication of the information available on this subject.

Particle size distributions have been measured by Budd (1966), Budd et al. (1966), and Schmidt (1981, 1982b, 1984). Vertical flux profiles are described by

Budd et al. (1966), Fohn (1980), Takeuchi (1980), Schmidt (1982b, 1986a), Schmidt et al. (1982), and Schmidt et al. (1984). Sublimation from blowing snow particles is discussed by Dyunin (1967), Schmidt (1972, 1982b), Tabler and Schmidt (1972), Lee (1975), Tabler (1975a), Male (1980), and Benson (1982).

Studies focusing on the saltation process include Bagnold (1941), Mellor and Radok (1960), Jenssen (1963), Owen (1964), Oura (1967), Oura et al. (1967), Kobayashi (1972, 1979), Kind (1976), White and Schluz (1977), Maeno et al. (1979), and Kind and Murray (1982). The influence of snow surface hardness on snow transport is studied by Dyunin (1963), Bagnold (1966), Narita (1978), Schmidt (1980, 1981, 1986a, 1986b), and Martinelli and Ozment (1985).

Measurement of snowdrift profiles in natural topographic catchments include those made by Berg and Caine (1975), Tabler (1975b), Benson (1982), Berg (1986), and Liston (1986). Profiles of snowdrifts formed by snow fences are described by Tabler (1980).

The preceding studies represent a world-wide effort to obtain a better understanding of the physical principles governing drifting snow and the saltation process. They are all observational.

The second classification of the drifting snow studies encompasses those which model the drifting process and/or effects. The complexity of the factors influencing snowdrift patterns formed by structures and terrain features has lead to the simulation of blowing snow and drift formations using wind tunnels and water flumes. Finney (1934) introduced this technique using sawdust and mica in a wind tunnel. In order to assure quantitative scaling of the snowdrifts, certain theoretical requirements must be met. However, these requirements cannot be entirely realized, and consequently compromises must be made if this approach is used (Iversen, 1979, 1980; Anno, 1984a; Kind, 1986). Reduced-scale model experiments, performed outside in natural conditions instead of within a wind tunnel, have been used in an attempt to avoid these restrictive scaling requirements (Tabler and Jairell, 1981; Anno, 1984b).

Empirical models have also been produced. Tabler (1975b) developed a multiple linear regression equation which predicts equilibrium profiles of snowdrifts in topographic catchments. This type of model is generally considered appropriate only for the conditions under which the regression coefficients were originally determined. Similarly, Tabler (1980) developed polynomial regression curves which describe equilibrium snowdrift profiles produced by snow fences.

A computer simulation model which predicts the accumulation of wind-blown snow particles in topographic catchments is presented by Berg and Caine (1975) and Berg (1986). The model comprises a set of mathematical relationships between airflow, topography, and snow particle movement that determine regions where the windspeed is below the threshold required for snow transport and then consequently accumulates particles in such regions.

Recently, computational and physical modeling efforts have been implemented in an attempt to (1) explain the physical processes associated with snow transport and (2) develop predictive tools for these processes. Decker and Brown (1983, 1985) utilized modern mixture theory to study blowing snow in mountainous terrain. This study was aimed at predicting snow deposition patterns in mountainous terrain and determining the nature of the dominant processes governing two phase flow of air and suspended snow particles. Uematsu et al. (1989) developed a two-dimensional finite element model of snowdrift development. They solve the momentum equations assuming a constant eddy or turbulent diffusivity.

The picture that emerges from this review is that the vast majority of previous snow transport studies have been observational in nature, and very little computational modeling of the relevant flow features and processes has been attempted.

CHAPTER 3

MATHEMATICAL MODEL DEVELOPMENT

In this chapter the governing equations and turbulence modeling practices for both single-phase and two-phase flows are presented. The governing equations of fluid motion represent a description of the time and spatial variation of velocity and pressure for the flow of concern. In engineering and atmospheric sciences the flows of interest are almost always turbulent. A turbulent flow is characterized by fluid motion which is eddying, highly random, unsteady, and three-dimensional. These eddies cover a wide spectrum of sizes, ranging from the size of the flow domain to many orders of magnitude smaller, and they also cover a correspondingly wide range of fluctuation frequencies with the high frequencies being associated with the small eddies. Commonly the turbulent fluctuations are removed from the flow description by a suitable averaging of the governing equations. This averaging leads to new terms which contain unknown correlations between fluctuating velocity components. A non-closed system of equations results since the number of unknown variables now exceeds the number of equations. To close the system of equations a turbulence model must be developed which relates the new unknown quantities to the mean flow field.

Turbulence Modeling for Single-Phase Flows

The governing equations of fluid motion are based on the universal conservation laws of mass and momentum. The resulting continuity and momentum equations for a Newtonian fluid are commonly referred to as the Navier-Stokes equations and take the following form for an incompressible fluid (Schlichting, 1979),

Continuity Equation:

$$\frac{\partial U_i}{\partial x_i} = 0 \tag{1}$$

Momentum Equations:

$$\frac{\partial U_i}{\partial t} + U_j \frac{\partial U_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial P}{\partial x_i} + \nu \frac{\partial^2 U_i}{\partial x_j \partial x_j} \tag{2}$$

where U_i is the instantaneous velocity component in the x_i direction, t is time, P is pressure, ρ is the fluid density, and ν is the kinematic viscosity. Here the Einstein summation convention on repeated indices has been applied.

The unsteady Navier-Stokes equations are generally considered to be capable of describing turbulent flow. Direct simulation of the turbulent flow is impractical, however, because the scale of the smallest eddies in turbulent flow is typically 10^{-3} times the size of the flow domain. A common estimation is that a numerical grid containing 10^5 points would be required to resolve just 1 cm^3 of the flow. As a consequence, it is not expected that computers will be large and fast enough to describe the flow field in this manner in the near future.

The common approach used today is to solve for the mean flow field using the equations which have been averaged over a time which is long compared with the turbulence but short when compared to that of the mean flow. In this approach, the instantaneous values of velocity U_i and the pressure P are separated into mean (overbars) and fluctuating (lower case) quantities

$$U_i = \overline{U_i} + u_i \quad , \quad P = \overline{P} + p \tag{3}$$

and the mean quantities are given by

$$\overline{U_i} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} U_i \, dt \quad , \quad \overline{P} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} P \, dt \tag{4}$$

Carrying out the averaging procedure leads to the following equations, where the overbars have been dropped for efficiency, except where averages of products of fluctuating terms are needed for clarity.

Continuity Equation:

$$\frac{\partial U_i}{\partial x_i} = 0 \tag{5}$$

Momentum Equations:

$$\frac{\partial U_i}{\partial t} + U_j \frac{\partial U_i}{\partial x_j} = - \frac{1}{\rho} \frac{\partial P}{\partial x_i} + \frac{\partial}{\partial x_j}\left(\nu \frac{\partial U_i}{\partial x_j} - \overline{u_i u_j} \right) \tag{6}$$

These time averaged Navier-Stokes equations are often called the Reynolds averaged equations after their original developer, Osborne Reynolds, and they govern the mean flow quantities U_i and P. The time averaging has led to new terms $\overline{u_i u_j}$, in the momentum equations which, when multiplied by ρ, are interpreted as stresses associated with the turbulent motion. These turbulent stress components are frequently many orders of magnitude larger than the viscous stresses found in the term containing ν. These new terms must now be modeled or described in terms of the mean flow variables in order to create a closed set of equations.

Existing turbulence models which approximate these terms range from simple models based on Prandtl's mixing-length hypothesis (Schlichting, 1979) to those that use differential transport equations for the individual turbulent stresses and fluxes. Extensive reviews of the various models can be found in Rodi (1982) and Lakshminarayana (1986).

The simple models based on the Prandtl mixing-length hypothesis suffer from several drawbacks. One primary failing is that the hypothesis implies that the turbulence is in local equilibrium, consequently the model is unable to account for the transport and history effects of turbulence. This leads to incorrectly predicting zero turbulence in flow regions of uniform mean velocity where the velocity gradient is zero. Examples of this problem include the prediction of zero turbulence in both the center of a pipe and in the flow behind a uniform grid (Rodi, 1980). It is also worth noting that since it is difficult to prescribe the mixing-length distribution in any but the simplest flows, these models are not applicable to more complex, separating flows.

Efforts to develop a more universal turbulence model has led to what is known as the k-ϵ model of Jones and Launder (1972). This is a two-equation model which uses two partial differential transport equations to describe evolution of the turbulent kinetic energy k, and the dissipation rate ϵ, of the turbulent kinetic energy. The turbulent kinetic energy k, where k is defined to be

$$k = \frac{1}{2}\,\overline{u_i u_i} = \frac{1}{2}\left(\overline{u_1^2}+\overline{u_2^2}+\overline{u_3^2}\right) \tag{7}$$

and a length scale L, which describes the characteristic size of large eddies producing turbulent stresses, are frequently viewed as the primary parameters which allow description of turbulent flows. Since the rate of dissipation ϵ is proportional to $k^{3/2}/L$ (Rodi, 1980), a description of the variation of k and ϵ will allow an indirect description of the characteristic turbulent length scale.

In laminar flows the viscous stresses are proportional to the mean velocity gradients; with the kinematic viscosity ν of the fluid serving as the proportionality constant. To develop the k-ϵ model the turbulent stresses $\overline{u_i u_j}$ are first assumed to be related to the mean velocity gradients by Boussinesq's eddy viscosity hypothesis

$$-\overline{u_i u_j} = \nu_t\left(\frac{\partial U_i}{\partial x_j}+\frac{\partial U_j}{\partial x_i}\right) - \frac{2}{3}\,k\delta_{ij} \tag{8}$$

where ν_t is the turbulent viscosity and δ_{ij} is the Kronecker Delta function. Here the turbulent viscosity ν_t is a property of the flow rather than the fluid, and it may vary greatly over the flow field. Equation 8 contains two new unknowns ν_t and the turbulent kinetic energy k. Since it is difficult to derive an expression directly for ν_t, it was reasoned, by dimensional analysis, that ν_t is proportional to k^2/ϵ where ϵ is the dissipation rate of k (Jones and Launder, 1972). Thus it is assumed that

$$\nu_t = C_\mu \frac{k^2}{\epsilon} \tag{9}$$

where C_μ is a constant. At this stage the problem has been shifted from describing the Reynolds stresses, to describing the distribution of k and ϵ.

An exact equation for the Reynolds stresses $\overline{u_i u_j}$ can be derived from the Navier-Stokes equations. To do this, the time averaged momentum equations (Equation 6) are subtracted from the time dependent Navier-Stokes equations (Equation 2) for both velocity components i and j. The equation for u_i is then multiplied by u_j and the equation for u_j by u_i, and the two resulting equations are added. Time averaging this last equation, and assuming the viscous diffusion term is negligible in high Reynolds number flows, yields (Hinze, 1975),

$$\frac{\partial \overline{u_i u_j}}{\partial t} + U_l \frac{\partial \overline{u_i u_j}}{\partial x_l} = -\frac{\partial}{\partial x_l}\left(\overline{u_l u_i u_j}\right) - \frac{1}{\rho}\left(\frac{\partial \overline{u_j p}}{\partial x_i} + \frac{\partial \overline{u_i p}}{\partial x_j}\right)$$
$$- \overline{u_i u_l}\frac{\partial U_j}{\partial x_l} - \overline{u_j u_l}\frac{\partial U_i}{\partial x_l} + \overline{\frac{p}{\rho}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)} - 2\nu\overline{\frac{\partial u_i}{\partial x_l}\frac{\partial u_j}{\partial x_l}} \qquad (10)$$

When these three equations for the three normal stresses (i=j=1,2,3) are summed, an exact equation for the turbulent kinetic energy is obtained (Rodi, 1984),

$$\frac{\partial k}{\partial t} + U_i \frac{\partial k}{\partial x_i} = -\frac{\partial}{\partial x_i}\left(\overline{u_i\left(\frac{u_j u_j}{2} + \frac{p}{\rho}\right)}\right) - \overline{u_i u_j}\frac{\partial U_i}{\partial x_j}$$
$$- \nu\overline{\frac{\partial u_i}{\partial x_j}\frac{\partial u_i}{\partial x_j}} \qquad (11)$$

In this equation the rate of change of k is balanced by convective transport due to the mean flow, diffusive transport due to velocity and pressure fluctuations, production of turbulent kinetic energy due to the combination of Reynolds stresses and mean velocity gradients, and dissipation of k by the transfer of kinetic energy into heat.

For this k equation to be applied to the momentum equations, the additional unknown terms in this equation must be modeled. Applying the Boussinesq eddy viscosity hypothesis, the term describing the production of k due to the interaction of Reynolds stresses and velocity shear becomes

$$-\overline{u_i u_j}\frac{\partial U_i}{\partial x_j} = \nu_t\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)\frac{\partial U_i}{\partial x_j} \tag{12}$$

The diffusive transport term is modeled as

$$-\left(\overline{u_i\left(\frac{u_j u_j}{2} + \frac{p}{\rho}\right)}\right) = \frac{\nu_t}{\sigma_k}\frac{\partial k}{\partial x_i} \tag{13}$$

where σ_k is a constant. The last term describes the product of the molecular kinematic viscosity and the fluctuating vorticity, or the rate of viscous dissipation of energy ϵ,

$$\nu\overline{\frac{\partial u_i}{\partial x_j}\frac{\partial u_i}{\partial x_j}} = \epsilon \tag{14}$$

The final, most common form of the turbulent kinetic energy transport equation is

$$\underset{\text{(i)}}{\frac{\partial k}{\partial t}} + \underset{\text{(ii)}}{U_i\frac{\partial k}{\partial x_i}} = \underset{\text{(iii)}}{\frac{\partial}{\partial x_i}\left(\frac{\nu_t}{\sigma_k}\frac{\partial k}{\partial x_i}\right)} + \underset{\text{(iv)}}{\nu_t\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)\frac{\partial U_i}{\partial x_j}} - \underset{\text{(v)}}{\epsilon} \tag{15}$$

where term (i) represents the rate of change of k, term (ii) describes convective transport of k, term (iii) is the diffusive transport, term (iv) describes the production of k through shear, and term (v) is the rate of viscous dissipation.

An exact transport equation for ϵ can also be derived from the Navier-Stokes equations for high Reynolds number flows (Hanjalic and Launder, 1972),

$$\frac{\partial \epsilon}{\partial t} + U_i \frac{\partial \epsilon}{\partial x_i} = -2\nu\left(\overline{\frac{\partial u_j}{\partial x_l}\frac{\partial u_i}{\partial x_l}} + \overline{\frac{\partial u_l}{\partial x_i}\frac{\partial u_l}{\partial x_j}}\right)\frac{\partial U_i}{\partial x_j} - 2\nu\overline{\frac{\partial u_j}{\partial x_l}\frac{\partial u_i}{\partial x_l}\frac{\partial u_j}{\partial x_i}} - 2\nu^2\overline{\left(\frac{\partial^2 u_j}{\partial x_i \partial x_l}\right)^2} - \frac{\partial}{\partial x_i}\overline{u_i \epsilon'} \tag{16}$$

where ϵ' describes the fluctuating dissipation rate. Again there have been new terms introduced which must be modeled in order to obtain a closed set of equations (Hanjalic and Launder, 1972). The first term on the right is a generation term which is modeled as

$$-2\nu\left(\overline{\frac{\partial u_j}{\partial x_l}\frac{\partial u_i}{\partial x_l}} + \overline{\frac{\partial u_l}{\partial x_i}\frac{\partial u_l}{\partial x_j}}\right) = C_{1\epsilon}\,\epsilon\,\frac{\nu_t}{k}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) \tag{17}$$

where $c_{1\epsilon}$ is a modeling constant. The second and third terms are modeled together in the form

$$2\nu\overline{\frac{\partial u_j}{\partial x_l}\frac{\partial u_i}{\partial x_l}\frac{\partial u_j}{\partial x_i}} + 2\nu^2\overline{\left(\frac{\partial^2 u_j}{\partial x_i \partial x_l}\right)^2} = C_{2\epsilon}\frac{\epsilon^2}{k} \tag{18}$$

where $c_{2\epsilon}$ is a constant. The fourth term describes the diffusion of ϵ due to velocity fluctuations and becomes

$$-\overline{u_i\epsilon'} = \frac{\nu_t}{\sigma_\epsilon}\frac{\partial\epsilon}{\partial x_i} \tag{19}$$

where σ_ϵ is a constant. The most common form of the modeled ϵ equation is

$$\begin{aligned}\frac{\partial\epsilon}{\partial t} + U_i\frac{\partial\epsilon}{\partial x_i} &= \frac{\partial}{\partial x_i}\left(\frac{\nu_t}{\sigma_\epsilon}\frac{\partial\epsilon}{\partial x_i}\right)\\ &+ C_{1\epsilon}\epsilon\frac{\nu_t}{k}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)\frac{\partial U_i}{\partial x_j} - C_{2\epsilon}\frac{\epsilon^2}{k}\end{aligned} \tag{20}$$

where each term has a meaning similar to those found in the k equation.

In engineering applications the empirical constants most frequently used in these equations are: $C_\mu = 0.09$, $C_{1\epsilon} = 1.44$, $C_{2\epsilon} = 1.92$, $\sigma_k = 1.0$, $\sigma_\epsilon = 1.3$ (Launder and Spalding, 1974). These constants have been found to successfully predict mean flow characteristics for a large variety of flows including flows having a free surface and those involving recirculation. Over the past 15 years the k-ϵ model has been tested extensively and has found success for a variety of different flows, including free shear flows, wall boundary layers, flows involving recirculation, and duct flows (Rodi, 1982).

Hanjalic and Launder (1980) proposed that an additional term be added to increase the rate of irrotational deformations in promoting turbulent energy transfer. Rodi (1985) developed a buoyancy-extended k-ϵ model which is used to model horizontal shear layers with stable stratification. This paper is essentially a simplification of the transport equation model of Gibson and Launder (1978).

A further step in turbulence model complexity is the second-order closure schemes which employ transport equations to describe the individual stresses and fluxes. This eliminates the assumption of local isotropy found in the k-ϵ model. Although these models contain fewer assumptions and are much more general than the simpler models, they also contain additional terms which must be modeled based on approximations and assumptions of flow properties, some of which are still unable to be measured. In addition, the large number of partial differential equations involved make these models computationally cumbersome. At this stage, these very complex models are considered to be subjects of turbulence research and are not sufficiently refined for application to practical problems. The stress and flux transport equations can be reduced, however, to algebraic equations which still maintain many of the important characteristics of the more complex equations.

These algebraic equations are then combined with the k and ϵ equations to simulate the turbulent stresses (Lakshminarayana, 1986).

A completely different approach is called large eddy simulation. In this scheme the large scale turbulence, which is responsible for the majority of the flow behavior, is simulated numerically while a model is used to describe the small scale turbulence structure (Peyret and Taylor, 1983). This computationally tedious method also shows much promise but is still largely in its developmental stages.

Turbulence Modeling for Two-Phase Flows

As early as 1945 research showed that adding a dilute suspension of particles to a turbulent fluid increases the flow rate under a given pressure gradient (Sproull, 1961). Hetsroni and Sokolov (1971) concluded that the particles suppress the turbulence in the dissipation range of small eddies and that the process is dependent upon both particle size and concentration. Moderrass et al. (1984) showed that, depending on particle loading, the turbulent kinetic energy can be reduced as much as 50 percent, thereby increasing the carrier fluid velocity. This process is again attributed to a decrease in turbulence dissipation resulting from particle-fluid interaction.

Two common approaches are taken to describe fluid-particle flows, the trajectory approach and the two-fluid approach. In the trajectory approach the turbulent flow field is first calculated assuming single phase flow. Then particle velocities and trajectories are calculated using the single-phase flow field. A major shortcoming of this is the assumption that the presence of particles does not influence the motion of the fluid. This results in what is known as a one-way coupling of the phases. This method can be extended by recalculating the flow field with the inclusion of the calculated particle behavior. The particle trajectories are again calculated based on the flow field and this process is repeated until the flow field does not change with subsequent iterations. In this manner the trajectory approach is able to include the two-way coupling which commonly characterizes two-phase flows (Crowe, 1982). Two-way coupling is also included when using the two-fluid approach. This approach regards the fluid and particulate phases as two interacting fluids which are governed by their own mass balance and momentum balance equations.

In this study the author will consider only dilute particle suspensions where the particulate phase volume fraction is small (of order 10^{-3}), and particle to particle interactions can be considered negligible. For this case, the governing equations of motion can be described as taking the same form as the previously introduced Navier-Stokes and Reynolds averaged equations, where an additional source term F_p has been added to the right side of the momentum equations to account for the force exerted upon the fluid by the particles (Marble, 1970).

Two-phase turbulence closure modeling is a relatively new field of study. Danon et al. (1977) proposed a one-equation, one-way coupled model for a very lightly loaded flow. Melville and Bray (1979) produced an algebraic model based on the Prandtl mixing-length hypothesis for moderately loaded jets. Genchev and Karpunov (1980) developed a mixing-length model which includes an extra term which takes into account the particle influence on turbulent motion. Their results are not compared to experimental data. Elghobashi and Abou-Arab (1983) presented a complex, two-equation closure scheme for two-phase flows which attempts to resolve some of the weaknesses of the schemes of Danon et al. and Genchev and Karpunov. Decker and Brown (1985) modeled the turbulent fluctuations of the particulate phase based on the mixing-length hypothesis and applied it to a one-way coupled mixture of air and snow.

One of the most advanced and extensively tested modeling efforts has been that of Chen (1983) and Chen and Wood (1984; 1985). It is assumed that the particulate phase volume fraction is much less than unity and made up of spherical particles of uniform size.

Their model is based on the two-equation k-ϵ turbulence model described previously. Here an additional term, which results from including a particle-fluid interaction force, is added to each of the k and ϵ turbulence equations. It is assumed that the particles follow the mean flow, but that on the high frequency turbulent fluctuation level, the particles do not exactly follow the fluid and the resulting slip is a hydrodynamic drag force F_p which can be described by Stokes law and takes the form,

$$F_p = \frac{\rho_p(V_i - U_i)}{t^*} \tag{21}$$

where U_i and V_i represent the fluid and particle velocities, respectively. ρ_p is the particulate phase density, and $t^* = d^2\rho_s/18\mu$ is the characteristic response time of the particles; d is the particle diameter, ρ_s is the density of the solid particulate material, and μ is the fluid viscosity. This approach appears to be valid for the dilute mixture considered here. Experimental data by Popper et al. (1974) and Yuu et al. (1978) suggest that, to a first approximation, we can assume that the particles follow the mean motion but not necessarily the turbulent fluctuations.

A derivation of the turbulence kinetic energy equation with the inclusion of the Stokes drag force in the momentum equations produces an additional term in the k equation which takes the form (Chen and Wood, 1984),

$$\frac{\rho_p}{\rho}\,\frac{\left(\overline{u_i v_i - u_i u_i}\right)}{t^*} \tag{22}$$

where v_i is the fluctuating particulate velocity. This new term represents additional dissipation resulting from particle slip at the turbulent fluctuation level. This term is modeled by letting

$$-\left(\overline{u_i v_i - u_i u_i}\right) = 2\,k\left(1 - \exp\left(-B_k \frac{t^*}{t_e}\right)\right) \tag{23}$$

where $t_e = 0.165\ k/\epsilon$, and B_k is a constant set equal to 0.0825 (Chen and Wood, 1985).

The dissipation rate equation for ϵ will also have an additional term

$$\frac{2}{t^*}\,\frac{\rho_p}{\rho}\left(\nu\,\overline{\frac{\partial u_i}{\partial x_j}\left(\frac{\partial v_i}{\partial x_j}-\frac{\partial u_i}{\partial x_j}\right)}\right) \qquad (24)$$

which is modeled as

$$-\left(\nu\,\overline{\frac{\partial u_i}{\partial x_j}\left(\frac{\partial v_i}{\partial x_j}-\frac{\partial u_i}{\partial x_j}\right)}\right) = \epsilon\left(1-\exp\left(-B_\epsilon\frac{t^*}{\tau}\right)\right) \qquad (25)$$

where $\tau = (\nu/\epsilon)^{1/2}$ is the Kolmogorov time scale frequently used to describe high frequency eddies and B_ϵ is a modeling constant.

These modeled terms, when added to the right side of the appropriate k and ϵ equations, produce additional dissipation which reduce turbulent velocity fluctuations.

Since the two additional particulate terms depend on the dimensionless particulate phase loading ρ_p/ρ, an additional transport equation must be solved to determine the particle concentration field. This equation accounts for particulate mass conservation and takes the following form (Rodi, 1984),

$$\frac{\partial w}{\partial t} + U_i\,\frac{\partial w}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\frac{\nu_t}{\sigma_t}\,\frac{\partial w}{\partial x_i}\right) \qquad (26)$$

where the particulate concentration $w = \rho_p/\rho$, and σ_t is the turbulent Schmidt number = 0.5 for planar flows (Reynolds, 1976).

Saltation Modeling

Saltation is a process in which snow particles are transported close to the ground, undergoing repeated impacts with the snow covered surface. Initially loose snow particles are entrained into the air stream by the wind. After having gained momentum from the wind they fall back to the ground to strike other snow particles on the snow covered surface. Most of the time the impacting particle rebounds from the surface and is accelerated by the wind before it again falls to the ground. Since the impact with the surface is inelastic, the rebound velocity is necessarily less than the impact velocity. Since the surface is rough, on a scale comparable to the particle size, the impacting particle rebounds at an angle different than the impact angle. In order to sustain saltation, the average vertical rebound velocity must equal the incident vertical velocity. If it is less, the vertical velocity will die out after several impacts and the particle will cease to saltate. During impact a particle may also dislodge additional snow particles from the snow surface. While some of these particles may then join the saltation process, others may roll or bounce over short distances to form a layer of reptating particles. As more particles are added to the airstream, the speed of the wind in the saltating layer is reduced. This reduces the impact velocity of the saltating particles which in turn decreases the average vertical rebound velocity. In addition, as the impact velocity decreases, the effectiveness of an impacting particle at ejecting more particles into the flow decreases. Thus, snow particles begin to settle out of the saltating layer. As the particles start to settle out, the wind deceleration is reversed, allowing more particles to be entrained. An equilibrium is eventually reached for a given free stream airflow, and only a certain number of particles are maintained in the saltating layer.

Measurements above a horizontal surface show a very rapid decrease in mass flux with height through the saltation layer. In fact, Kind (1981), using data provided by Oura (1967) and Kobayashi (1972), reported that approximately 90 percent of the total saltating mass flux is contained within the first 3 cm above the surface.

The presence of saltation is dependent upon the relative magnitudes of the shear strength of the snow surface and the shear stress at the surface produced by the wind above. In order for snow particles to begin to move, some threshold shear stress must be exceeded. This threshold value is dependent upon such factors as temperature, grain size, extent of grain bonding, and surface hardness. Kind (1981) lists experimental studies which have found threshold shear velocity values ranging from 0.1 m/s for light dry snow, to 0.4 m/s for old hardened snow.

One saltation model which has found considerable favor in the literature was developed by Iversen et al. (1975) based on similitude arguments. Schmidt (1982a) has shown this model to successfully reproduce field data collected by several researchers. The model takes the form

$$Q_s = C\left(\frac{\rho}{g}\right)\left(\frac{V_s}{U_{*t}}\right)U_*^2\,(U_* - U_{*t}) \tag{27}$$

where Q_s is the total mass transport rate per unit lateral dimension with units of kg/(m s), ρ is the fluid density, g is the gravitational acceleration, V_s is the settling velocity of a snow particle, U_* is the friction or shear velocity, U_{*t} is the threshold shear velocity, and C is a constant set equal to 1.0.

Since the majority of the mass flux is confined to the area very near the ground, and a computation of the turbulent flow field above the surface will provide values of the surface shear velocity, the computed upper flow field will be used to drive the saltation model. Here the turbulent layer's sole purpose is to drive the saltation model which is responsible for the greatest contribution of snow transport.

As presented, the coupling between the turbulent and saltation layers has been assumed passive or one-way, i.e., the presence or lack of saltation does not affect the solution of the turbulent flow field. Initially, it appears that the two-way coupling which occurs in the natural system is only a secondary process. Consequently, the one-way coupling approximation will be assumed valid.

The saltation model provides the vertically integrated rate of snow transport, $Q_s(x)$, within that layer. Knowledge of the particulate concentration and velocity fields allow computation of the vertically integrated rate of snow transport within the turbulent flow layer, $Q_t(x)$, where

$$Q_t(x) - \int U(x,z)\ w(x,z)\ dz \tag{28}$$

Changes in the bed surface level (snow accumulation or drift formation) result from changes in Q_t and Q_s with x. This change in surface level is described by the vertically integrated mass continuity equation,

$$\frac{\partial h}{\partial t} + \frac{1}{\rho_b} \frac{\partial}{\partial x} (Q_t + Q_s) = 0 \qquad (29)$$

where h is the surface level with respect to a horizontal datum, t = time, and ρ_b is the bulk density of snow. The accumulation of mass, and subsequent new boundary configuration, necessitates a recomputation of the turbulent flow field.

CHAPTER 4

TWO-PHASE TURBULENT FLOW MODEL FOR BLOWING SNOW

Governing Equations

In the previous chapter the ground work was set for developing a model for application to blowing and drifting snow. As a rigorous test case for the model the flow over a two-dimensional obstruction, such as an infinitely long solid wall 2 meters high and aligned perpendicular to the wind direction, will be considered (Figure 1). Developing the governing equations for this problem will begin with the Reynolds averaged equations. Since the flow of interest will contain significant zones of recirculation, boundary layer approximations will not apply and the Reynolds equations must be solved in their full form. In addition to the two-dimensional restriction, only steady state flows will be considered. As such, the previously introduced Reynolds equations and the k and ϵ turbulence model equations will apply with the neglect of the appropriate terms. Also, as noted previously, since the kinematic viscosity ν is

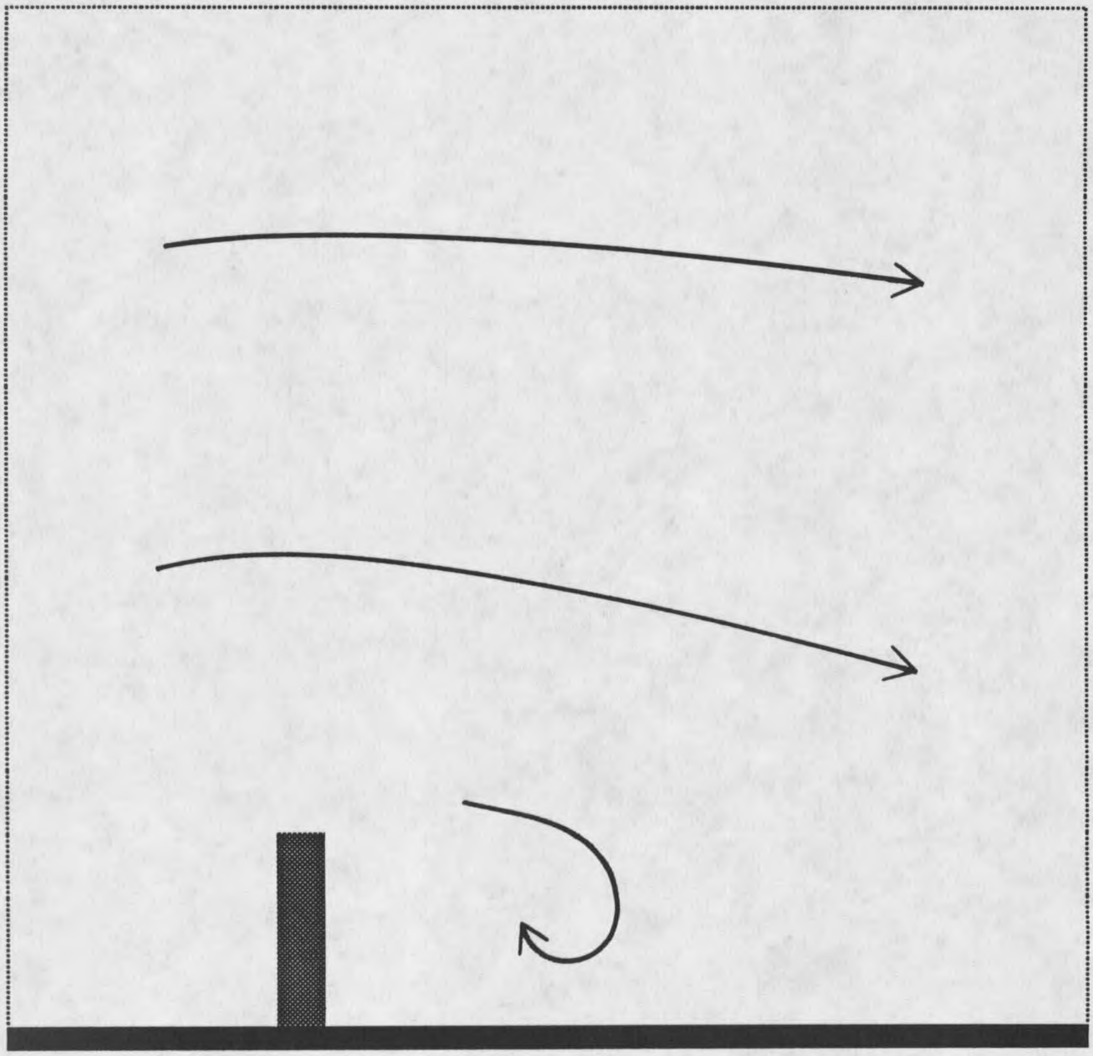

Figure 1. Computational domain boundary configuration.

typically several orders of magnitude smaller than the turbulent viscosity ν_t, the term containing ν in the time averaged momentum equations (Equation 6), will be neglected.

To establish the exact form that the momentum equations will take, Equation 8 must be introduced into Equation 6 to eliminate the $\overline{u_i u_j}$ term. When doing this it is noted that the Kronecker Delta term in Equation 8 is a pressure-like term which can be absorbed into the momentum equation pressure term. The diffusion terms in the momentum equations now take the form

$$\frac{\partial}{\partial x_j}\left(\nu_t\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)\right) \tag{30}$$

This can be expanded for the dimensions x and z, respectively, in the horizontal and vertical directions. By applying the continuity equation (Equation 5) and the property that in a discrete difference scheme products are interchangeable, the term becomes

$$\frac{\partial}{\partial x}\left(\nu_t \frac{\partial U}{\partial x}\right) + \frac{\partial}{\partial z}\left(\nu_t \frac{\partial U}{\partial z}\right) \tag{31}$$

for the x momentum equation with velocity U. A similar term is found for the z momentum equation with velocity V. Combining this information and expanding the other indexed terms in the continuity equation and momentum equations leads to three of the equations comprising the turbulent flow model. They will be listed with the other equations at the end of this chapter.

The k and ϵ equations include a production term, resulting from shear, of the form

$$\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) \frac{\partial U_i}{\partial x_j} \tag{32}$$

When this is expanded in a manner similar to the term in the momentum equation, this term becomes

$$\left(\frac{\partial U}{\partial z} - \frac{\partial V}{\partial x}\right)^2 \tag{33}$$

Together the k and ϵ equations contain five modeling constants, c_μ, $c_{1\epsilon}$, $c_{2\epsilon}$, σ_k, and σ_ϵ which must be determined before the equations can be solved (Launder and Spalding, 1974). Values for these constants were listed in the previous chapter. How the constants were obtained is discussed by Rodi (1984). The value of $c_{2\epsilon}$ was computed based on measurements of the decay rate of k behind grids. Values for the diffusion constants σ_k and σ_ϵ were obtained by a computer optimization procedure which involved applying the model to several laboratory shear flows and adjusting the constants to achieve the best overall fit with experimental results. The final two constants, c_μ and $c_{1\epsilon}$, are evaluated by considering local equilibrium shear layers occurring near a wall. For atmospheric flows, as opposed to engineering or hydraulic flows, the values of $c_\mu = 0.03$ and $c_{1\epsilon} = 1.16$ have been suggested in the literature (Sutton et al., 1986). Since these constants agree well with atmospheric measurements (Lumley and Panofsky, 1964; Frost et al., 1975), I will adopt these atmospheric values.

The exact form of the particulate source terms in the k and ϵ equations must also be determined. To do this, representative atmospheric profiles of velocity, turbulent kinetic energy, and dissipation rate must be established.

A wide range of experimental and theoretical studies have shown that wind profiles within a surface layer of nearly hydrostatically neutral stratification can be closely approximated by a logarithmic equation. This is commonly expressed in the form (Holton, 1979),

$$U = \frac{U_*}{\kappa} \ln\left(\frac{z}{z_0}\right) \qquad (34)$$

where U_* is the shear velocity, z is the height above the surface, z_0 is the roughness length which is dependent upon the roughness of the surface, and $\kappa = 0.4$ is von Karman's constant. A roughness length of $z_0 =$ 0.1 cm will be used as a representative value for snow surfaces (Sorbjan, 1989).

In a homogeneous, neutrally stratified boundary layer the rate of dissipation above the viscous sublayer is given by

$$\epsilon = \frac{U_*^3}{\kappa z} \tag{35}$$

(Panofsky and Dutton, 1984; Sorbjan, 1989). This is also consistent with experimental data (Wyngaard and Cote, 1971). Frost et al. (1975) provided atmospheric profiles of turbulent kinetic energy k for the same conditions,

$$k = \frac{U_*^2}{\sqrt{C_\mu}} \tag{36}$$

This agrees with field measurements by Lumley and Panofsky (1964). These profiles of U, k, and ϵ given by Equations 34, 35, and 36 can all be computed upon choosing an appropriate value for U at a specified height z. For reference, in mid-latitude regions, typical windspeeds experienced during snow transport events fall in the range 10 to 15 m/s measured at a height of 10 m (Kind, 1981).

The form that the particulate source terms in the k and ϵ equations take is strongly dependent upon the relative magnitudes of time scales t^*, t_e, and τ. To compute t^* for a spherical ice grain in air, note that $\rho_s = 910$ kg/m^3, $\nu = 13.3e{-6}$ m^2/s, and $\rho = 1.29$ kg/m^3 for a pressure of one atmosphere and temperature of 0° C. Applying a typical wind velocity of 10 m/s at 10 m, and computing the values of these time scales shows that for snow particles having diameters $d \geq 0.5$ mm, $t^* >> \tau$. This analysis also shows that t^* and t_e are of the same order of magnitude for this particle size range, and that only when $d \leq 0.1$ mm is $t^* << t_e$. The implication of these results is that, for this study, a restriction of $d \geq 0.5$ mm allows the neglecting of the exponential term in the ϵ equation particulate source term (Equation 25), while the k equation particulate source term (Equation 23) must be used in its full form.

The particle concentration equation (Equation 26) is valid as stated with the exception that a constant particle settling velocity V_s will be added to the vertical convection term. Here the author has assumed that the particle concentration field is convected downward at a velocity additional to the vertical velocity of the fluid. A survey of published data suggests that $V_s = 0.75$ m/s is a representative still air fall velocity for snow particles of interest in this study (Mellor, 1965; Kind, 1981; Schmidt, 1982a).

Boundary Conditions

To solve the governing equations, boundary conditions must be applied at all boundaries of the computational domain for all the dependent variables. For this problem conditions U, V, P, k, ϵ, and w must all be specified in some form. A representative flow domain is depicted in Figure 1. The flow domain contains an unrestricted inflow, outflow, and top boundary as well as a solid lower boundary composed of horizontal surfaces and vertical walls. The appropriate boundary conditions are defined as follows.

Inflow boundary:

$V = 0$

U is given by Equation 34

$\frac{\partial(\ )}{\partial n} = 0$ for P

k is given by Equation 36

ϵ is given by Equation 35

w is set equal to a constant, $0 \le w \le 1.0$

Upper boundary:

$$\frac{\partial(\ )}{\partial n} = 0 \text{ for } U,\ V,\ P,\ k, \text{ and } \epsilon$$

w is set equal to a constant, $0 \leq w \leq 1.0$

Outflow boundary:

$$\frac{\partial(\ )}{\partial n} = 0 \text{ for } U,\ V,\ P,\ k,\ \epsilon, \text{ and } w$$

Horizontal surfaces:

U = logarithmic

V = 0

k and ϵ are functions of the surface shear stress through the friction velocity U_*

$$\frac{\partial(\ )}{\partial n} = 0 \text{ for } P \text{ and } w$$

Vertical surfaces:

U = 0

V = 0

$$\frac{\partial(\ )}{\partial n} = 0 \text{ for } P,\ k, \text{ and } \epsilon$$

w = 0

where n is the direction perpendicular to the boundary.

The horizontal and vertical surface type boundary conditions warrant additional discussion. The pressure boundary condition will be discussed in Chapter 5. In developing the turbulence equations, the laminar flow within the viscous sublayer near a wall has been neglected. Turbulence models have been developed which account for these effects, but since the gradients encountered in the near-wall region are typically very steep, and consequently computationally expensive to resolve, these models are not commonly used. A far more common scheme is to choose a computational domain which is slightly removed from the solid boundary and apply semi-empirical wall functions to the gap between the solid boundary and computational boundary.

The logarithmic profile given by Equation 34 is used in this study for the U velocity boundary condition on all horizontal surfaces including the top of the wall. U is set to zero on vertical walls. The V component of velocity, which is perpendicular to horizontal surfaces, is set equal to zero, and considered negligibly small on vertical surfaces. To apply the log-wall boundary condition, a profile given by Equation 34 is assumed to exist between the solid surface at $z = 0.0$, a grid point on the computational domain boundary at $z = z_1$, and the first interior grid point at $z = z_2$. The resulting two equations for U_1 and U_2 can be equated through their common U_* and solved to yield a boundary condition of the form

$$U_1 = \frac{\ln\left(\frac{z_1}{z_0}\right)}{\ln\left(\frac{z_2}{z_0}\right)} U_2 \qquad (37)$$

The boundary conditions for k and ϵ are also related to the friction or shear velocity U_* and are given by Equations 36 and 35, respectively, where z in the ϵ equation is now given by $z = z_1$. These boundary conditions can also be obtained by noting that in the region near the wall the Reynolds stresses are nearly constant (Rodi, 1984). Here convection and diffusion is small and the production due to shear and viscous dissipation terms are balanced.

For the particle concentration equation the horizontal surface boundary condition can be obtained by applying the concentration equation at that surface. Since the only diffusion influence present there is due to the viscosity of the fluid, and the only convective component existing is due to the particle settling velocity, the equation reduces to the boundary condition stated above. For the case of vertical surfaces, since both convective and diffusive processes are negligible or nonexistent there, the concentration there cannot rise above zero.

Model Summary

Combining the above information yields the coupled, highly non-linear system of equations which models the steady state, two-phase, two-dimensional turbulent flow field. The system of equations is found in Figure 2.

Application of this model is restricted to particulate volume fractions $\theta = \rho_p/\rho_s$ of order 10^{-3} or less. Since the solid material density of an ice grain $\rho_s = 910$ kg/m^3, the particulate phase density cannot be greater than order 1. With a fluid phase density $\rho = 1.29$ kg/m^3, this corresponds to the restriction that the particulate concentration $w = \rho_p/\rho$ must be of order ≤ 1. This criterion is easily met in both blowing snow and precipitating snow storms. For example, consider a snow storm with a moderate to high snow accumulation rate of 2 cm/hr, a snow settling velocity of 0.75 m/s, and a snow accumulation density of 200 kg/m^3. Here $w = \rho_p/\rho = 0.001$, which is still three orders of magnitude less than required by the above restriction. In the case of blowing and drifting snow, Budd et al. (1966) showed the criterion to be satisfied above 1 cm in the saltation layer. Also, extrapolation of data presented by Schmidt (1982b) suggests similar results.

$$\frac{\partial U}{\partial x} + \frac{\partial V}{\partial z} = 0 \tag{38}$$

$$U\frac{\partial U}{\partial x} + V\frac{\partial U}{\partial z} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \frac{\partial}{\partial x}\left(\left(C_\mu \frac{k^2}{\epsilon}\right)\frac{\partial U}{\partial x}\right) + \frac{\partial}{\partial z}\left(\left(C_\mu \frac{k^2}{\epsilon}\right)\frac{\partial U}{\partial z}\right) \tag{39}$$

$$U\frac{\partial V}{\partial x} + V\frac{\partial V}{\partial z} = -\frac{1}{\rho}\frac{\partial P}{\partial z} + \frac{\partial}{\partial x}\left(\left(C_\mu \frac{k^2}{\epsilon}\right)\frac{\partial V}{\partial x}\right) + \frac{\partial}{\partial z}\left(\left(C_\mu \frac{k^2}{\epsilon}\right)\frac{\partial V}{\partial z}\right) \tag{40}$$

$$\begin{aligned} U\frac{\partial k}{\partial x} + V\frac{\partial k}{\partial z} &= \frac{\partial}{\partial x}\left(\frac{C_\mu}{\sigma_k}\frac{k^2}{\epsilon}\frac{\partial k}{\partial x}\right) + \frac{\partial}{\partial z}\left(\frac{C_\mu}{\sigma_k}\frac{k^2}{\epsilon}\frac{\partial k}{\partial z}\right) \\ &+ C_\mu\frac{k^2}{\epsilon}\left(\frac{\partial U}{\partial z} - \frac{\partial V}{\partial x}\right)^2 - \epsilon - \frac{2k}{t^*}\left(1 - \exp\left(-0.5\,t^*\frac{\epsilon}{k}\right)\right)w \end{aligned} \tag{41}$$

$$\begin{aligned} U\frac{\partial \epsilon}{\partial x} + V\frac{\partial \epsilon}{\partial z} &= \frac{\partial}{\partial x}\left(\frac{C_\mu}{\sigma_\epsilon}\frac{k^2}{\epsilon}\frac{\partial \epsilon}{\partial x}\right) + \frac{\partial}{\partial z}\left(\frac{C_\mu}{\sigma_\epsilon}\frac{k^2}{\epsilon}\frac{\partial \epsilon}{\partial z}\right) \\ &+ C_{1\epsilon}C_\mu k\left(\frac{\partial U}{\partial z} - \frac{\partial V}{\partial x}\right)^2 - C_{2\epsilon}\frac{\epsilon^2}{k} - \frac{2\epsilon}{t^*}w \end{aligned} \tag{42}$$

$$U\frac{\partial w}{\partial x} + (V - V_s)\frac{\partial w}{\partial z} = \frac{\partial}{\partial x}\left(\frac{C_\mu}{\sigma_t}\frac{k^2}{\epsilon}\frac{\partial w}{\partial x}\right) + \frac{\partial}{\partial z}\left(\frac{C_\mu}{\sigma_t}\frac{k^2}{\epsilon}\frac{\partial w}{\partial z}\right) \tag{43}$$

Figure 2. The system of partial differential equations which models the steady state, two-phase, two-dimensional turbulent flow field.

CHAPTER 5

COMPUTATIONAL APPROACH TO MODEL SOLUTION

The six equation model presented in Figure 2 represents an imposing system of coupled partial differential equations. Any solution of the full steady or unsteady Navier-Stokes equations which includes a turbulence model is a formidable task, as illustrated by the following quote taken from ASCE (1988):

> The development of such a code is typically the result of teamwork by research groups at universities, specialized laboratories or institutes. A typical team working on the development of a turbulence model-based code may consist of two or more senior researchers, two or more junior researchers, and often graduate students.... The duration of a code development project may be from two to four years.

In light of this, an existing algorithm was implemented and modified to fit the requirements of this particular problem. The SIMPLER finite control volume algorithm described by Patankar (1980) was used to solve the continuity, x and z momentum, k and ϵ turbulence, and particle concentration equations. Since the details of

Patankar's computational scheme is well documented in his 1980 book, only a broad outline of the method will be presented here. In addition, problem specific modifications and implementations will be discussed.

The SIMPLER acronym stands for Semi-Implicit Method for Pressure-Linked Equations, Revised. The term semi-implicit refers to the method by which a pressure correction is related to a correction of the velocity as the solution progresses. In the Navier-Stokes equations, if the correct pressure field is given, then solving the momentum equations will lead to velocities which satisfy the continuity equation. Since the continuity equation is directly related to the pressure, it can be reformulated into a Poisson equation for pressure. The goal of the numerical scheme is to iteratively correct a guessed velocity field in order to produce a pressure field which is compatible with continuity. SIMPLER can be thought of as an iterative, pseudo time dependent formulation in which the discretized equations are marched from an initial condition, to the steady state solution. To ensure that the discretized equations are linear during the current iteration, some of the coefficients are evaluated at the previous iteration level. For this specific application the algorithm consists of the following sequence of steps:

1) Provide initial guesses for the U, V, k, ϵ, and w fields.
2) Compute a psuedo velocity from the momentum equations in the absence of the pressure terms.
3) Solve a pressure equation for the pressure field.
4) Using this pressure field, solve the momentum equations for the velocity.
5) Solve a pressure-correction equation.
6) Use this pressure-correction information to update the velocities.
7) Solve the k and ϵ equations using these new velocities.
8) Solve the concentration w equation using the new values for k and ϵ.
9) Return to step 2.

SIMPLER and related schemes have been widely tested and are a commonly used approach for solving incompressible viscous flow problems (Fletcher, 1988).

SIMPLER is a second order implicit formulation which is based on the concept of the control volume. The computational domain is divided into a series of control volumes, each of which encloses a grid point.

The differential equation is then integrated over each control volume. This produces the desirable characteristic of coarse grids yielding solutions which represent exact integral balances. The grid spacing can be nonuniform in both the x and z directions. This allows efficient use of computing energy by enabling the user to define course grids in regions where the gradients of the dependent variables are small.

Patankar (1978) introduced a harmonic mean approach to describe the value of the diffusion coefficient (viscosity) at control volume interfaces, based on diffusion coefficient values at the main grid points (control volume centers). This approach, as opposed to using the arithmetic mean, correctly deals with large step changes in the diffusion coefficient from one control volume to another. As an example application, a very large viscous coefficient can be used to force the velocity within a control volume to zero. The use of the harmonic mean ensures that neighboring control volumes consider the velocity throughout that control volume be zero. Effectively the zero velocity has been transferred from the main grid point to the control volume wall. The use of the harmonic mean has been implemented in SIMPLER, and the details are included in Patankar (1980).

To assist in the accurate numerical solution of convection dominated flows, an additional feature of SIMPLER is the use of a weighted upwinding scheme where the degree of upwinding imposed at each grid point depends on the ratio of the strengths of convection to diffusion. Applying central differences to convection dominated convection-diffusion problems can lead to physically unrealistic results. This can generally be attributed to the formation of negative coefficients (Patankar, 1980). Although this can be remedied by refining the grid spacing, it is much more desirable to develop a scheme which produces reasonable results using course grids. General upwinding at all grid points would be one solution to the problem, but in applications containing some regions not dominated by convection this would not be an acceptable practice. Patankar's approach is to compute the relative influences of convection and diffusion, and then weight the degree of upwinding accordingly. An additional concern which must be accounted for is that the scheme must be able to identify which direction is upwind, since in complex flows with flow reversals this may vary throughout the domain.

Application to the Turbulent Flow Model

To bring the model and associated boundary conditions to solution, several problem specific factors must be addressed. In this section such concerns as source term linearization, underrelaxation of the solution evolution, and declaring convergence will be discussed.

Implementation of the SIMPLER algorithm leads to a fully implicit system of linear algebraic equations for each discretized partial differential equation solved. Since two-dimensional problems lead to sparse systems containing five diagonal sequences of coefficients, considerable computer storage can be consumed by nonessential zeros in the systems of equations if they are stored in standard matrix form. To alleviate this problem a sparse storage scheme has been implemented which stores only the necessary coefficients and an index value describing their original positions, for each system of equations. Each of these systems is solved during every global iteration cycle. The Thomas algorithm (Anderson et al., 1984) is used to solve the systems of equations. This is a highly efficient direct solver for tridiagonal systems of equations found in one-dimensional problems. To apply the Thomas algorithm to the two-dimensional problems of interest in this study, a grid line is chosen in the x or z direction and the dependent variable is assumed known, from their latest values, at the grid lines on either side. The algorithm can then be used to solve for the dependent variables along that center line. In the solution implementation this iterative, line-by-line method is first used to sweep through the domain in the x and then the z direction, completing one iteration. The iterations are repeated until the maximum absolute difference between successive iterations is less than some set tolerance. If the x direction sweep is made in the direction of the main flow, a tolerance of 10^{-6} is typically achieved in fewer than seven iterations for all equations except the one for pressure.

Ferziger (1990) noted that it is typical for incompressible viscous flow solution schemes to spend the majority of their computing time solving the pressure equation. Since it is only the pressure differences which are meaningful in these applications, not the absolute value of pressure, it is common to apply a normal pressure derivative equal to zero condition to all flow domain boundaries. This implies that the pressure field and the pressure field plus any arbitrary constant are satisfactory solutions to the pressure equation. In this case a direct solution method would indicate a singular system of equations, while an iterative method like the Thomas line-by-line method will actually converge to a pressure field which marches by a constant at each subsequent iteration.

The nonlinear source terms in the partial differential equations for k and ϵ must be linearized, and this linearization must ensure that the dependent variables k and ϵ are always positive. This positive status is critical for both physical realism and numerical stability. To implement the linearization, the source terms S are cast in the form

$$S = S_c + S_\phi \phi \tag{44}$$

where S_c is the constant part of S, and S_ϕ is the coefficient of the dependent variable ϕ. For the case where the dependent variable is not required to be positive, the S_ϕ term must be negative or zero. Again this is required for physical reality and computational stability. For always positive variables, a positive or zero S_c is an additional requirement (Patankar, 1980).

To place S in this form, one desirable approach is to expand nonlinear terms in a Taylor series about the value of the dependent variable at the previous iteration, and keep the first two terms. This leads to the equation

$$S(\phi) \approx S(\phi^o) + (\phi - \phi^o)\left(\frac{dS}{d\phi}\right)^o \qquad (45)$$

where the superscript o indicates the value at the previous iteration. Another option which can be used separately or in conjunction with the Taylor expansion is to collect all positive source terms and set them equal to S_c. Then all negative source terms are divided by the dependent variable ϕ at the previous iteration and the result is set equal to S_ϕ, noting that S_ϕ was multiplied by ϕ in the formulation for S (Equation 44).

Implementing these two schemes leads to the following formulations for the source terms. For the k equation,

$$\begin{aligned} S_{ck} &= C_\mu \frac{k^2}{\epsilon}\left(\frac{\partial U}{\partial z} - \frac{\partial V}{\partial x}\right)^2 \\ S_{\phi k} &= -\frac{\epsilon}{k} - \frac{2}{t^*}\left(1 - \exp\left(-0.5\, t^* \frac{\epsilon}{k}\right)\right) w \end{aligned} \qquad (46)$$

and for the ϵ equation,

$$\begin{aligned} S_{c\epsilon} &= C_{1\epsilon} C_\mu k\left(\frac{\partial U}{\partial z} - \frac{\partial V}{\partial x}\right)^2 + C_{2\epsilon}\frac{\epsilon^2}{k} \\ S_{\phi\epsilon} &= -2\, C_{2\epsilon}\frac{\epsilon}{k} - \frac{2}{t^*} w \end{aligned} \qquad (47)$$

where k and ϵ in these equations are provided from the previous iteration. Here the terms in the k equation have been linearized using the second approach, and the terms in the ϵ equation have been linearized by first performing the Taylor expansion and then grouping the positive and negative terms.

Obtaining converged solutions to highly nonlinear systems of equations such as the one in this study is frequently difficult. To help ensure convergence of the global iterations, an underrelaxation scheme is implemented and applied to the velocities, k, ϵ, and the turbulent viscosity ν_t. Underrelaxation is a way to slow the iterative evolution of the solution. For the case of the turbulent viscosity the underrelaxation formula is

$$\nu_t = \alpha \nu_{t(new)} + (1 - \alpha) \nu_{t(old)} \qquad (48)$$

where $0 < \alpha < 1$ is a relaxation factor. Here the old values of the turbulent viscosity are advanced only a fraction of the amount suggested by the latest computed values. Although more complex, the underrelaxation scheme for the dependent variables is similar in form (Patankar, 1980). Relaxation factors $\alpha = 0.5$ for U and V, and $\alpha = 0.75$ for k, ϵ, and ν_t, have been found to ensure converged solutions in the applications of this study.

The iterative nature of the solution method requires that some kind of convergence criterion be satisfied. Computing the residual of the discretized equation, and requiring its magnitude to be some small number, can be used to avoid the illusion of a converged solution caused by slowly evolving, underrelaxed, dependent variables. In the SIMPLER algorithm the residual of the continuity equation is computed as part of the iterative solution and is readily available to serve as one indicator of convergence. A further requirement of a converged solution is that it must be invariant with further refinements of the grid. The solutions presented in the following chapters meet this criterion.

For the model application addressing flow over a solid wall, the use of very large viscous coefficients to block out the appropriate control volumes will be used to form the wall. As discussed earlier, this procedure places a zero boundary condition at the wall of the control volume. While this is a reasonable boundary condition for laminar flows, it was shown in Chapter 4 that a logarithmic condition is more appropriate for turbulent flows. Since the control volume blocking scheme will eventually be used to build snowdrifts which may cover large portions of the lower boundary, it is desirable to develop a system which can apply the logarithmic condition to horizontal block surfaces lying parallel to the dominant fluid flow direction.

In the control volume formulation the coefficients in a system of equations represent the convective and diffusive processes occurring at control volume faces. To illustrate the transfer of boundary conditions to interior points in the domain, consider the control volume adjacent to the top of a wall which has been formed using a large viscosity. The equation for that control volume includes itself and four neighbors with their associated coefficients. If this was a control volume lying on a boundary, the neighbor corresponding to the boundary would be, depending on the solution scheme, transferred to the right-hand side of the system of equations and the appropriate boundary condition implemented. The same idea can be applied to the block on top of the wall, but in this case, when the wall neighbor term has been transferred to the right-hand side, a zero has replaced the original coefficient in the main matrix. This has uncoupled the top control volume in the wall, and consequently the rest of the control volumes between there and the main rectangular domain boundary, from the control volume on top of the wall. Since the uncoupled control volumes are part of the solid wall, no change has been produced in them by removing their influence on the wall top control volume. The technique can also be used to apply nonzero boundary conditions to the vertical sides of blocked control volumes.

This method of applying boundary conditions to blocked surfaces adjacent to the fluid provides a way to realistically model many complex flows. It can only be applied to blocked control volume groups which extend continuously to the main domain boundary.

The SIMPLER algorithm and problem specific details outlined above have been implemented to solve the two-phase turbulent flow and snowdrift model. The computer program uses an efficient combination of the MATLAB computing package and the Fortran programming language; where MATLAB provides a desirable user environment and graphics capability, and Fortran programs are called to do the major numerical computing. The collection of programs used to solve the model is found in the Appendix.

CHAPTER 6

COMPUTATIONAL RESULTS AND DISCUSSION

In this chapter the two-dimensional computational model will be verified by comparison with the results of two experimental, single-phase, turbulent flow studies. The model will then be used to describe the flow field surrounding a solid vertical wall, with and without the presence of particles. The distribution of the particulate field will also be presented. Finally, the flow field computation will be used to compute the evolution and equilibrium configuration of the subsequent snowdrifts.

Comparison of Model and Experiment

Two experimental studies of atmospheric flow over a long, thin solid fence will be used to verify the model output. The first study was a full-scale field experiment (Jacobs, 1984), while the second study was conducted in an atmospheric wind tunnel (Perera, 1981).

In Jacobs' experiment, wind flow profiles were measured at nine locations aligned perpendicular to a thin solid fence of height h = 2 m, length 64 m, and thickness 0.02 m. The surface cover consisted of mowed heather with roughness length 0.035 m. Wind speeds were measured using cup anemometers.

Duplicating this experiment with the computational model, by providing the appropriate solid boundary configuration and incoming velocity profile, produced the flow field depicted in Figure 3. All major features of the flow have been simulated. Jacobs found a recirculation eddy at the front of the fence, starting near the surface at about -0.5 h, where h is the fence height, and reattaching on the fence at a height between 0.5 h and 1.0 h. The model places the eddy at approximately -0.7 h and 0.6 h for the surface and fence, respectively. Wind tunnel experiments have shown similar results with values of -0.5 h and 0.6 h (Good and Joubert, 1968). Behind the fence Jacobs found a recirculation eddy which started at the top of the fence and extended a distance of between 5 h and 10 h to the ground surface. The model has produced a reattachment point a distance 4.75 h from the fence. Other field visualization studies have shown reattachment points of 6 h for closed barriers (Ogawa and Diosy, 1980).

Figure 4 show a comparison of the velocity fields measured by Jacobs and computed by the model. Displayed

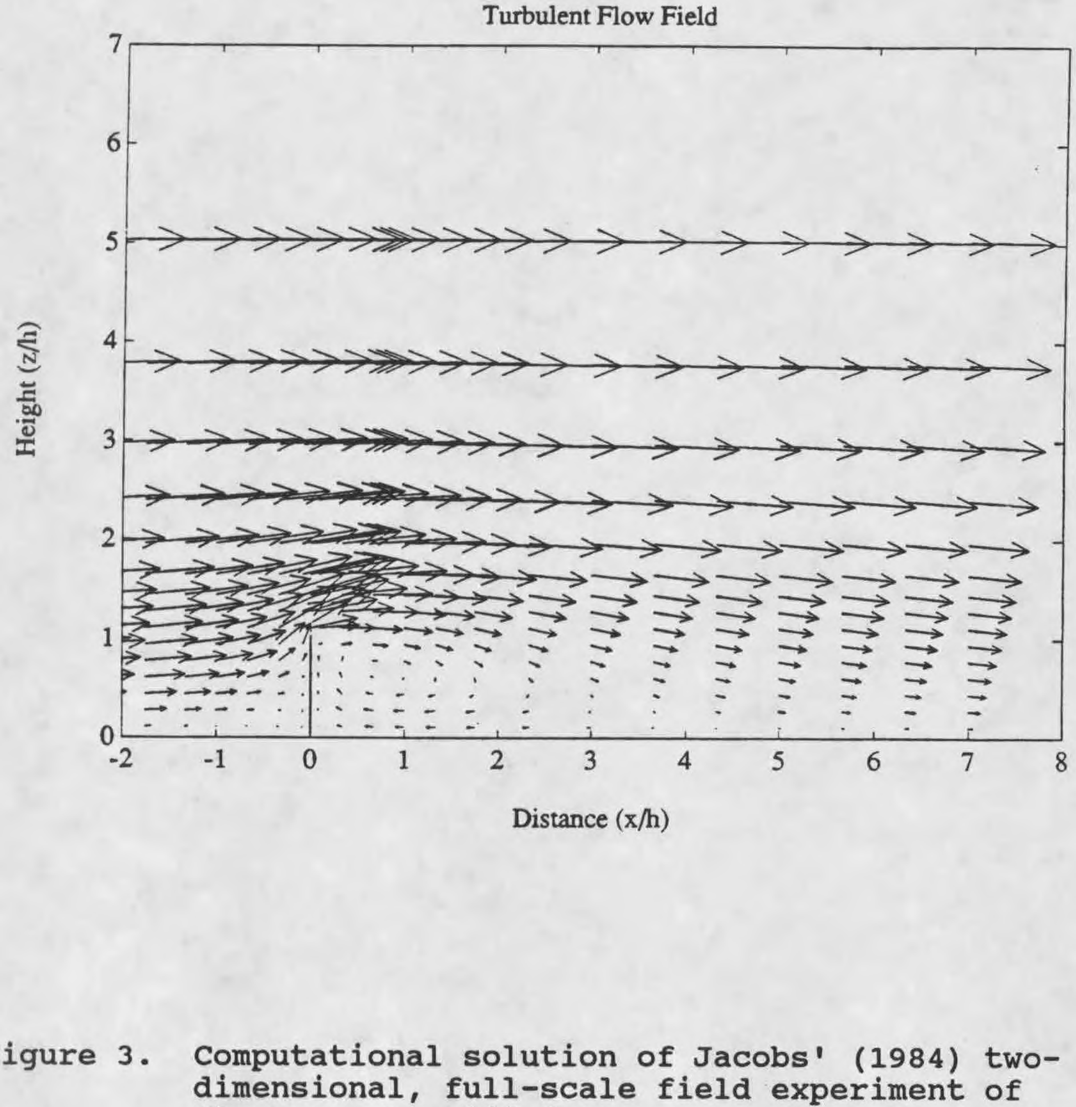

Figure 3. Computational solution of Jacobs' (1984) two-dimensional, full-scale field experiment of flow over a solid fence.

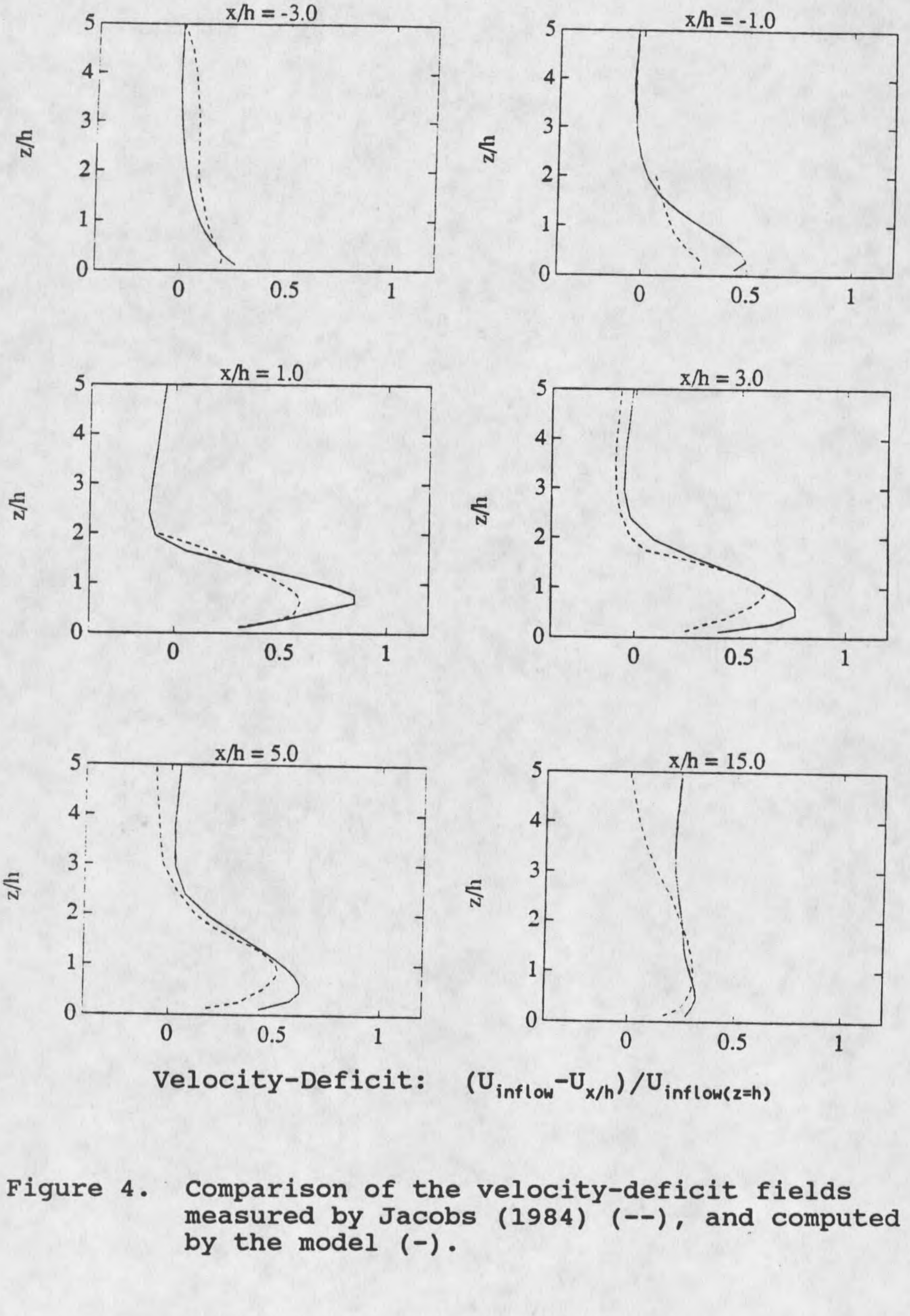

Figure 4. Comparison of the velocity-deficit fields measured by Jacobs (1984) (--), and computed by the model (-).

is the velocity deficit measured at locations x/h = -3.0, -1.0, 1.0, 3.0, 5.0, and 15.0, where the velocity deficit is defined as the incoming velocity profile minus the velocity profile at that location, divided by the incoming velocity at fence height h. Thus, the velocity deficit is a nondimensional measure of the reduction of velocity due to the presence of the barrier. The model can be seen to give a representative depiction of the measured velocity field. Inaccuracies in field measurement techniques and variations in such features as surface roughness and wind approach direction do not warrant a more detailed analysis of the depicted velocity differences.

Perera's experiment was also a two-dimensional, thin fence study, but here the roughness length was significantly smaller at 0.00036 m. Wind speeds were measured in a wind tunnel using pulsed-wire anemometers. These anemometers, unlike cup and hot-wire anemometers, have the ability to distinguish between positive and negative directions of wind flow.

A computational reproduction of this study is found in Figure 5. Again all major features of the flow have been reproduced. Note that the smaller roughness length has served to extend the windward recirculation eddy to approximately -1.0 h at the surface, and to extend the

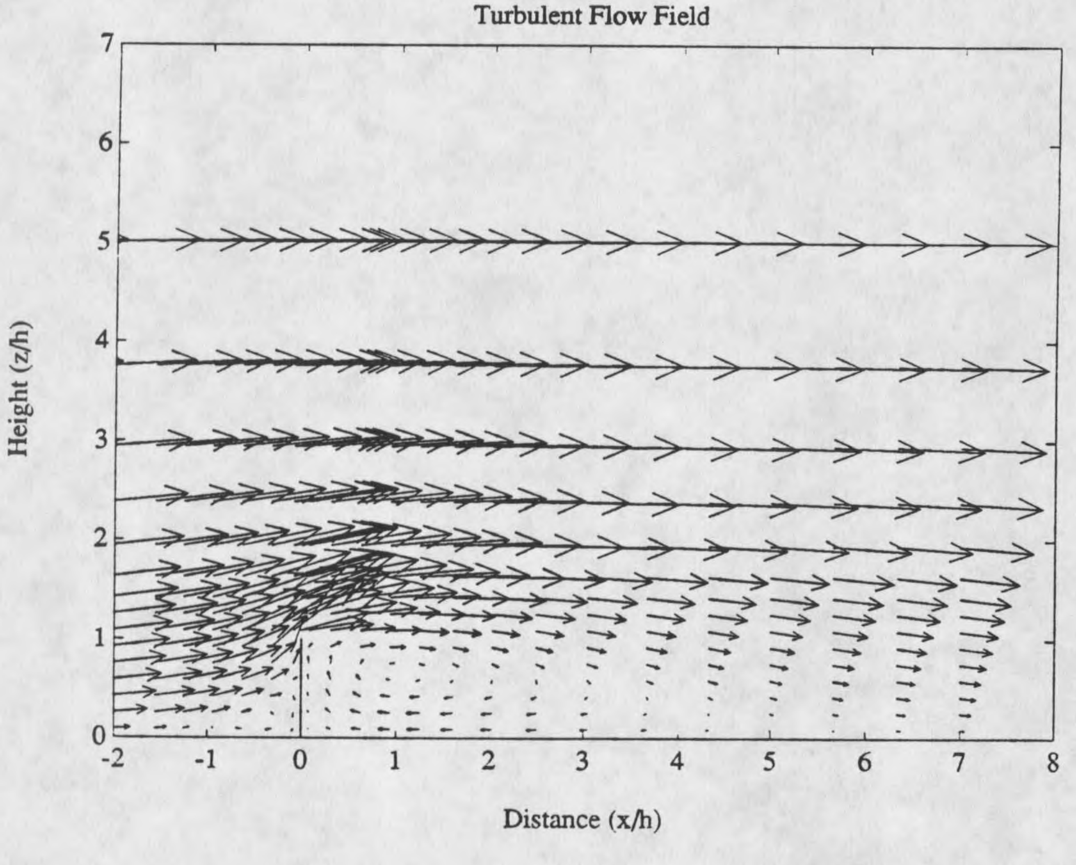

Figure 5. Computational solution of Perera's (1981) two-dimensional wind tunnel experiment of flow over a solid fence.

lee eddy to approximately 5.5 h. Figure 6 compares the experimental and model velocity deficit at locations x/h = -1.25, 0.625, 1.25, 2.5, 5.0, and 15.0 for Perera's study. Model output is seen to be satisfactory.

Flow Over a Solid Wall

The two-dimensional flow over a vertical wall will be used as a test case for computations of flow fields, particle concentration, and snow accumulation pattern evolution. For this test case, the wall is 2 m high and 0.5 m thick. The incoming velocity profile is defined by Equation 34 with a 10 m velocity of 10 m/s and a roughness length of 0.001 m. The computational domain is set 0.01 m above the ground surface. The computational grid, Figure 7, used in the computations contains 26 control volumes in the x direction and 14 in the z direction for a total of 364 grid points. When viewing this plot note that the aspect ratio is far from unity. The flow field for this configuration is given in Figure 8 for the case where the particle concentration w is zero.

In a precipitating snow storm, particle concentration w will be greater than zero. As noted in Chapter 4, a moderate to high snow accumulation rate of

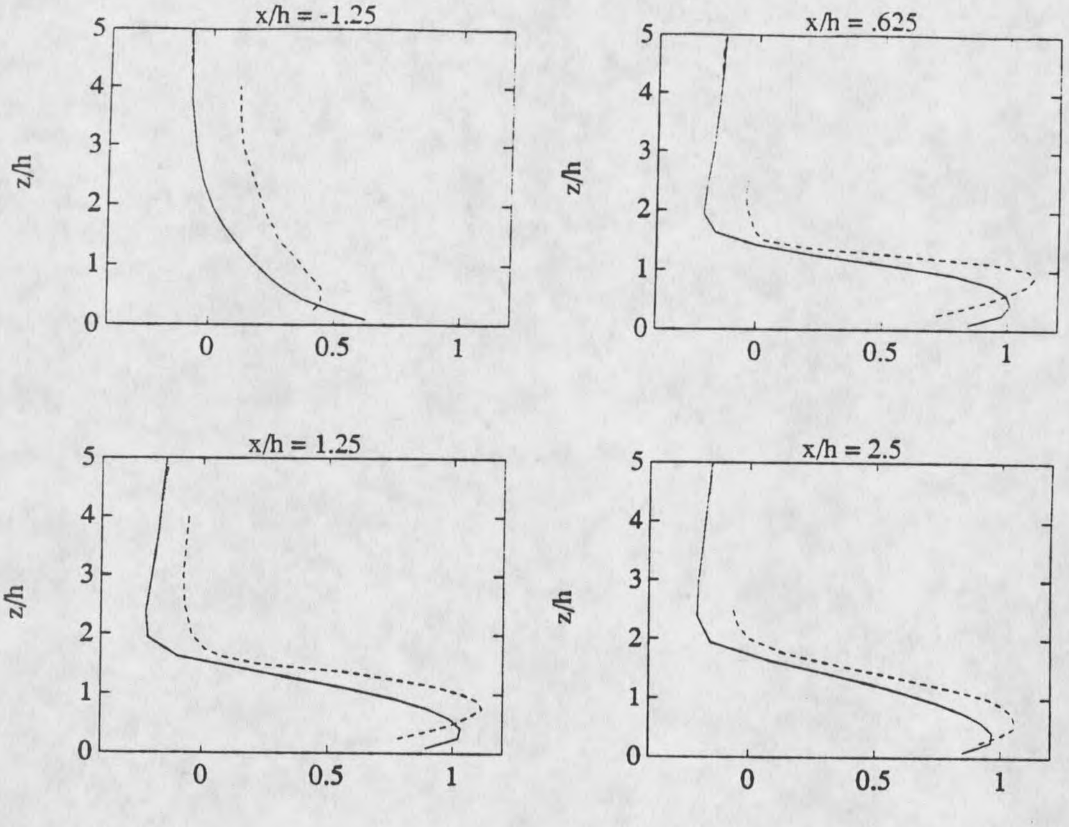

Figure 6. Comparison of the velocity-deficit fields measured by Perera (1981) (--), and computed by the model (-).

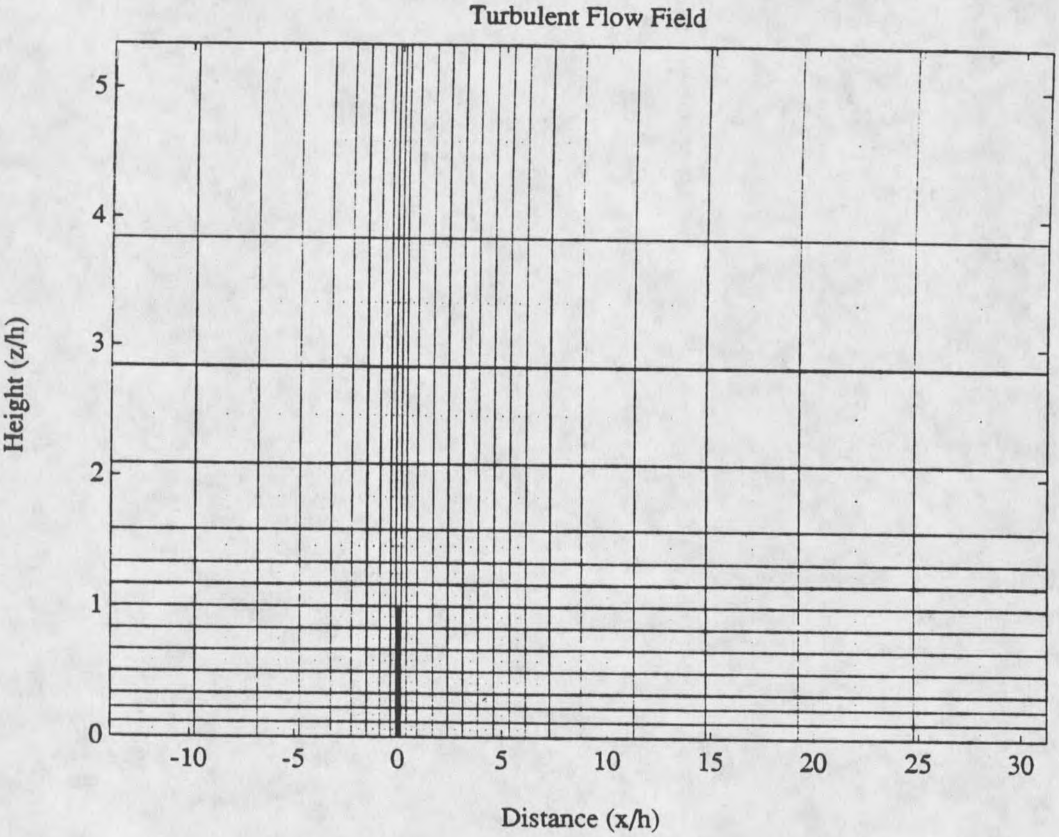

Figure 7. Computational grid used in test case problem considering flow over a solid wall 2 m high and 0.5 m thick.

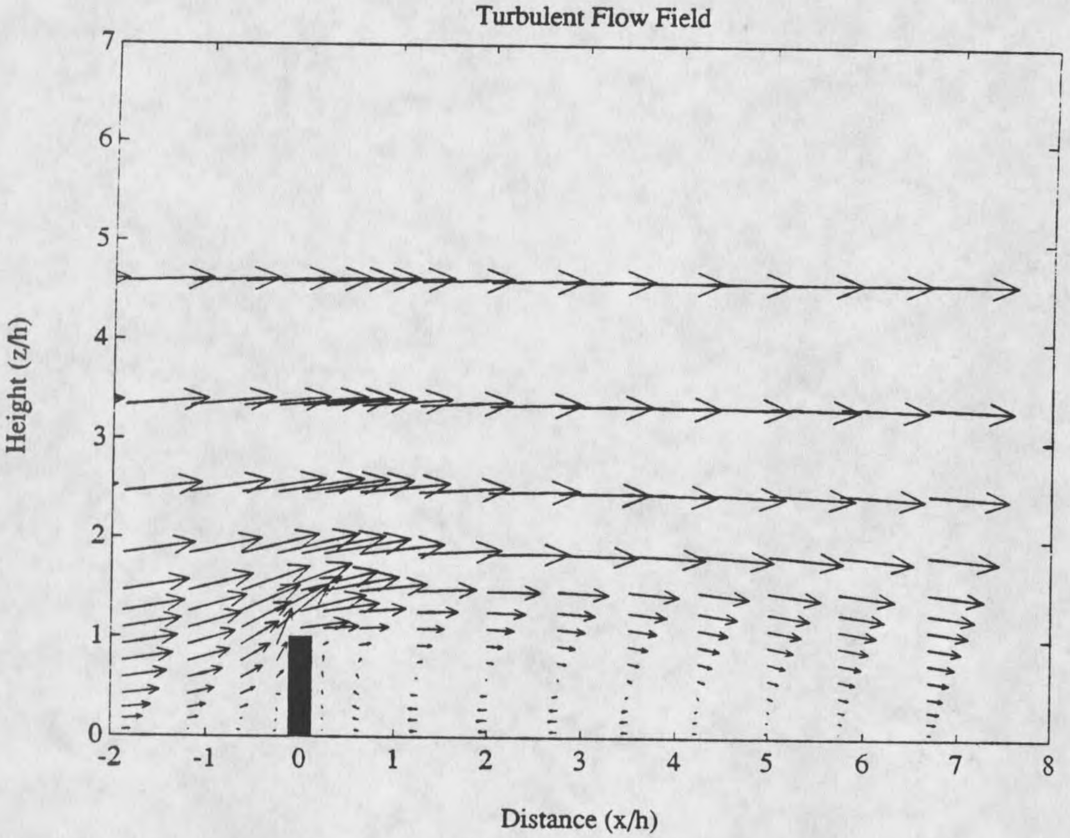

Figure 8. Computed flow field for the test case problem of single-phase, turbulent flow over a solid wall.

2 cm/hr corresponds to a concentration of w = 0.001. When this representative concentration is included in the model no change in the flow field is computed. At the inflow boundary, the two negative source terms in each of the k and ϵ equations are of the same magnitude and follow similar profiles if w at that boundary is set equal to unity. In this case the dissipation has been roughly doubled due to the presence of particles. If w is decreased by a factor of one-tenth, to 0.1, the increase in dissipation is proportionately less. This analysis suggests that particle concentrations of w = 0.01 will modify the values of k and ϵ less than a few percent. Consequently, the model indicates that snow particle concentrations found in the natural system can be considered passive and do not modify the structure of the flow field.

While the model shows the presence of particles is of negligible importance in the prediction of wind transported snows, other disciplines contain flows with high enough concentrations to warrant further investigation into the influence of these additional dissipation terms. Examples of two-phase flows include sediment transport in irrigation and navigation canals, rivers, estuaries, coastal waters, reservoirs, and sedimentation basins; heavier-than-air gas and particulate dispersion such as pollutant dispersion from a smoke stack and pollutant flow into a river or lake; coal slurry transport in pipes; blood flow; and a wide variety of combustion processes ranging from pulverized coal combustion to rocket exhaust plumes.

In addition to the computation for the single-phase flow field presented above, the computational model was run for two cases of non-zero concentration fields. For these simulations the concentration w was set equal to a constant on the incoming and top boundaries, simulating a precipitating snow storm event having unusually high particle concentrations. For these two cases, concentrations of $w = 0.2$ and 0.4 were used. The flow fields are depicted in Figures 9 and 10. Figure 11 displays the computed concentration at the horizontal surfaces of the bottom boundary for the case $w = 0.2$. Concentration is at a minimum immediately windward and to the lee of the wall. There is a concentration shadow corresponding to the lee recirculation eddy, and the peak at the position $x/h = 0$ represents the concentration at the top of the wall.

When these two flow fields are compared visually with the case $w = 0$ (Figure 8), it is obvious that the size of both windward and lee recirculation eddies has been reduced with increasing particle concentration.

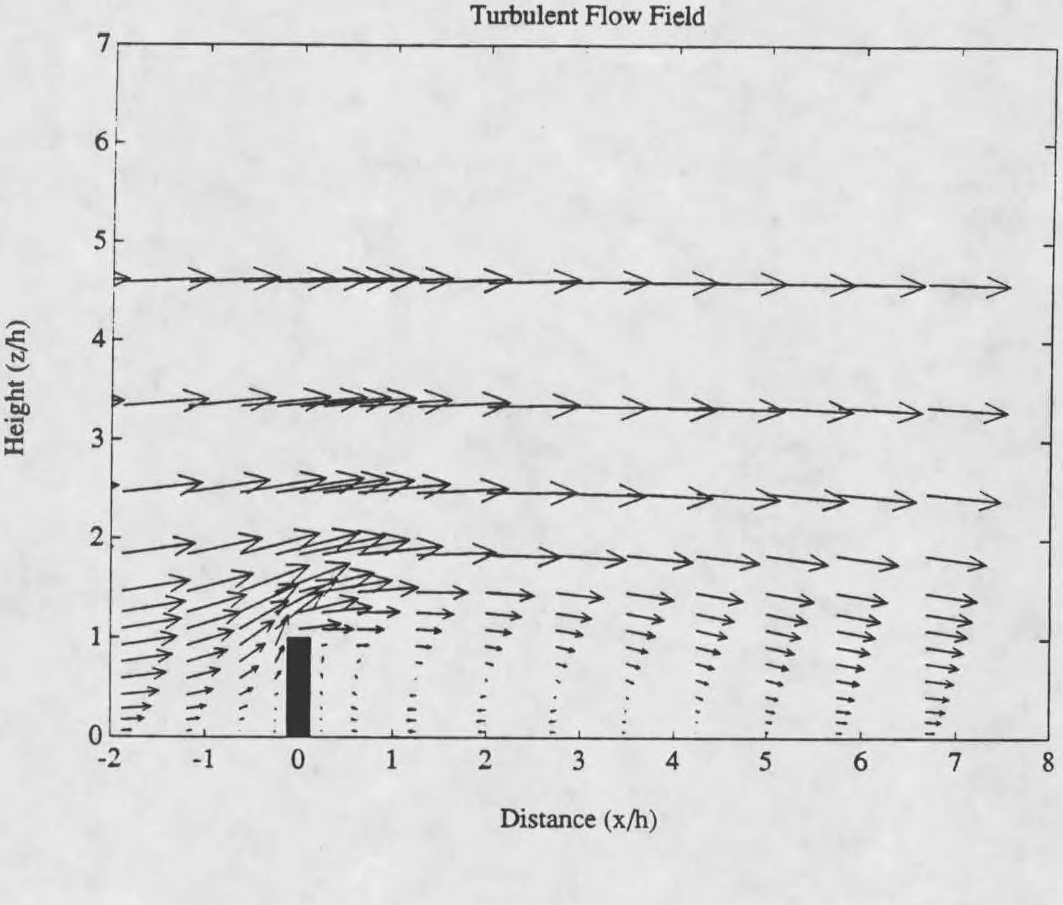

Figure 9. Computed flow field for the test case problem of two-phase, turbulent flow over a solid wall, with concentration w = 0.2 at the inflow and upper boundaries.

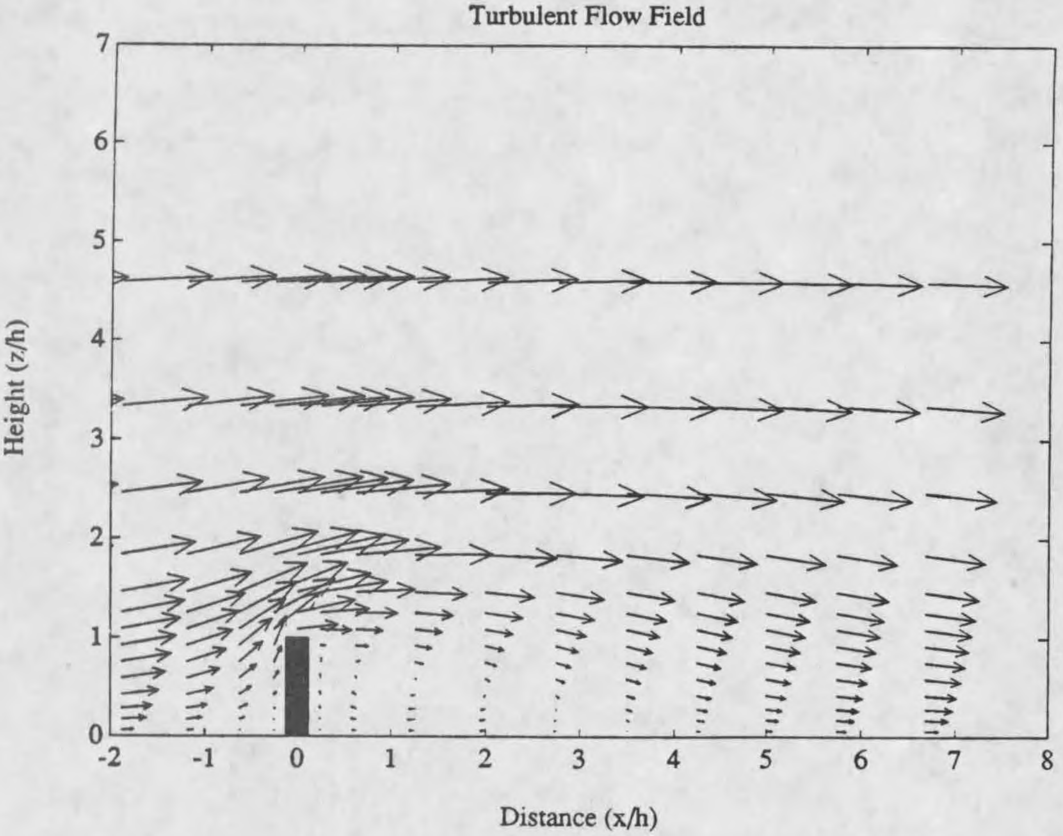

Figure 10. Computed flow field for the test case problem of two-phase, turbulent flow over a solid wall, with concentration w = 0.4 at the inflow and upper boundaries.

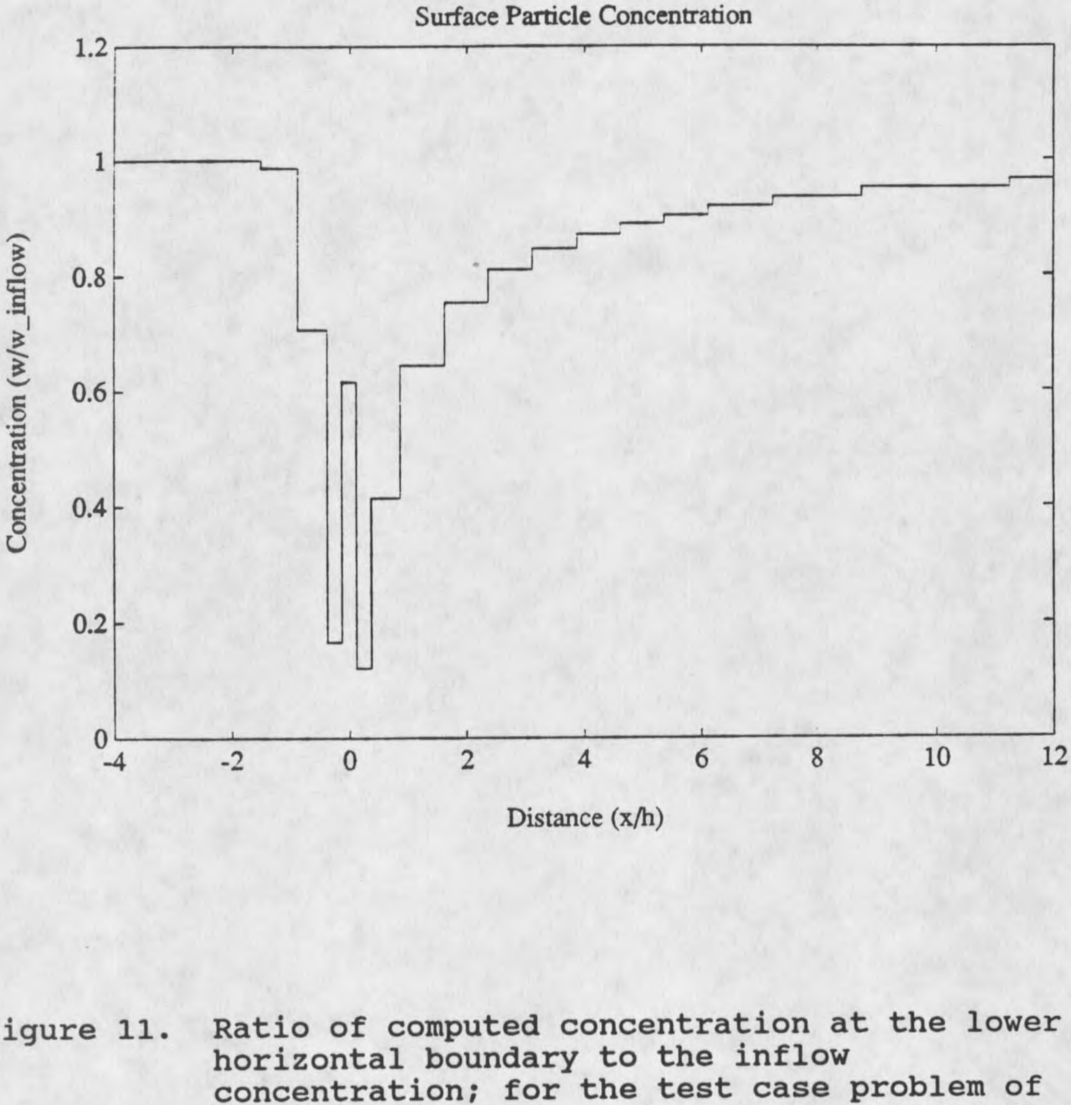

Figure 11. Ratio of computed concentration at the lower horizontal boundary to the inflow concentration; for the test case problem of two-phase, turbulent flow over a solid wall, with concentration w = 0.2 at inflow and upper boundaries.

The reattachment point for the lee eddy is $x/h = 5.75$, 4.5, and 3.25 for concentrations $w = 0$, 0.2, and 0.4, respectively. This behavior is indicative of an increase in turbulent viscosity ν_t, or what may be envisioned as a thickening of the fluid.

The addition of a negative source term on the right hand side of a transport equation such as the k and ϵ equations, assuming that all other factors remain the same, will serve to reduce the value of the dependent variable in that partial differential equation. Thus, this might lead one to expect the computed value of k and ϵ to be reduced for the case $w > 0$. Also note that the new value for the turbulent viscosity ν_t given by Equation 9 may or may not decrease in response to this decrease in k and ϵ. Chen and Wood (1985) conducted a study applying the k-ϵ turbulence model with particulate source terms to an axisymmetric jet. They show reductions of k, ϵ, and ν_t due to the presence of concentrations in the range $w = 0.22$ to 0.85.

Profiles of k, ϵ, ν_t, and U corresponding to the flow fields computed for concentrations $w = 0$, 0.2, and 0.4 (Figures 8, 9, and 10) are plotted in Figures 12 through 15, for the cross sections $x/h = -2.75$, 2.75, 5.0, and 10.0. These quantities have been nondimensionalized by dividing the parameter of interest

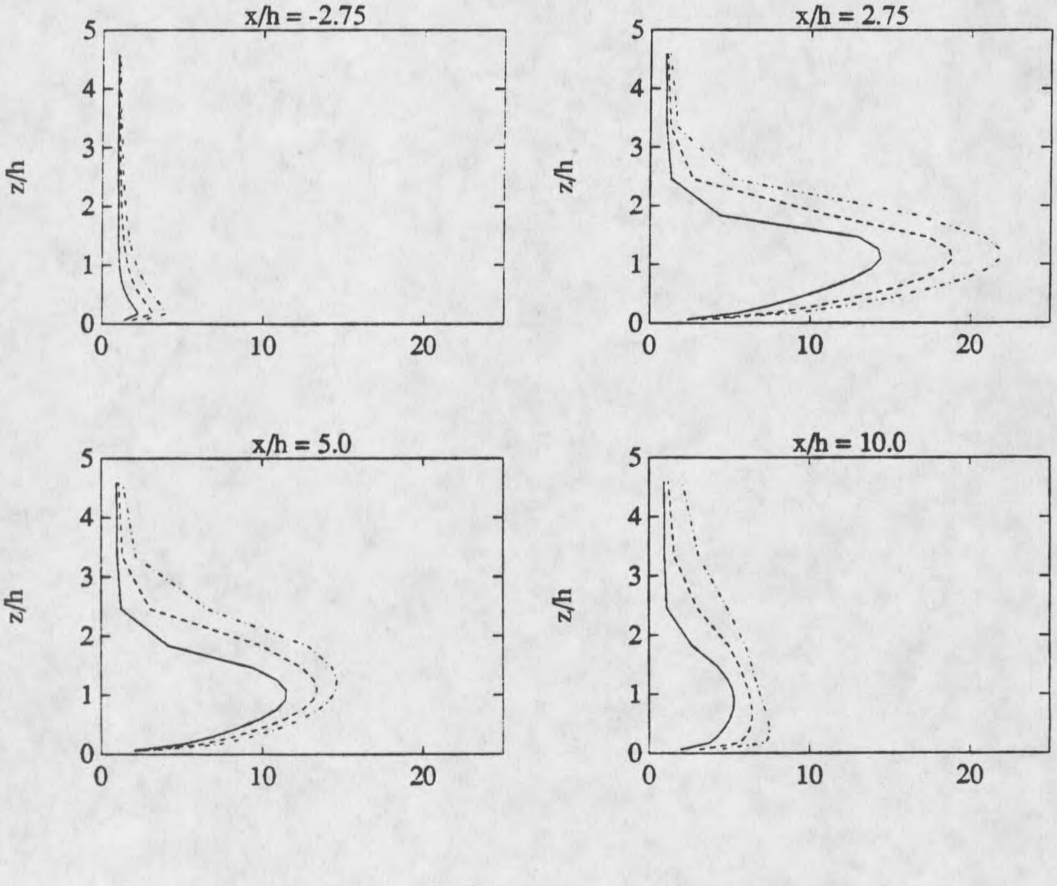

Figure 12. Profiles of turbulent kinetic energy k for the cases concentration w = 0 (-), 0.2 (--), and 0.4 (-.-) at inflow and upper boundaries.

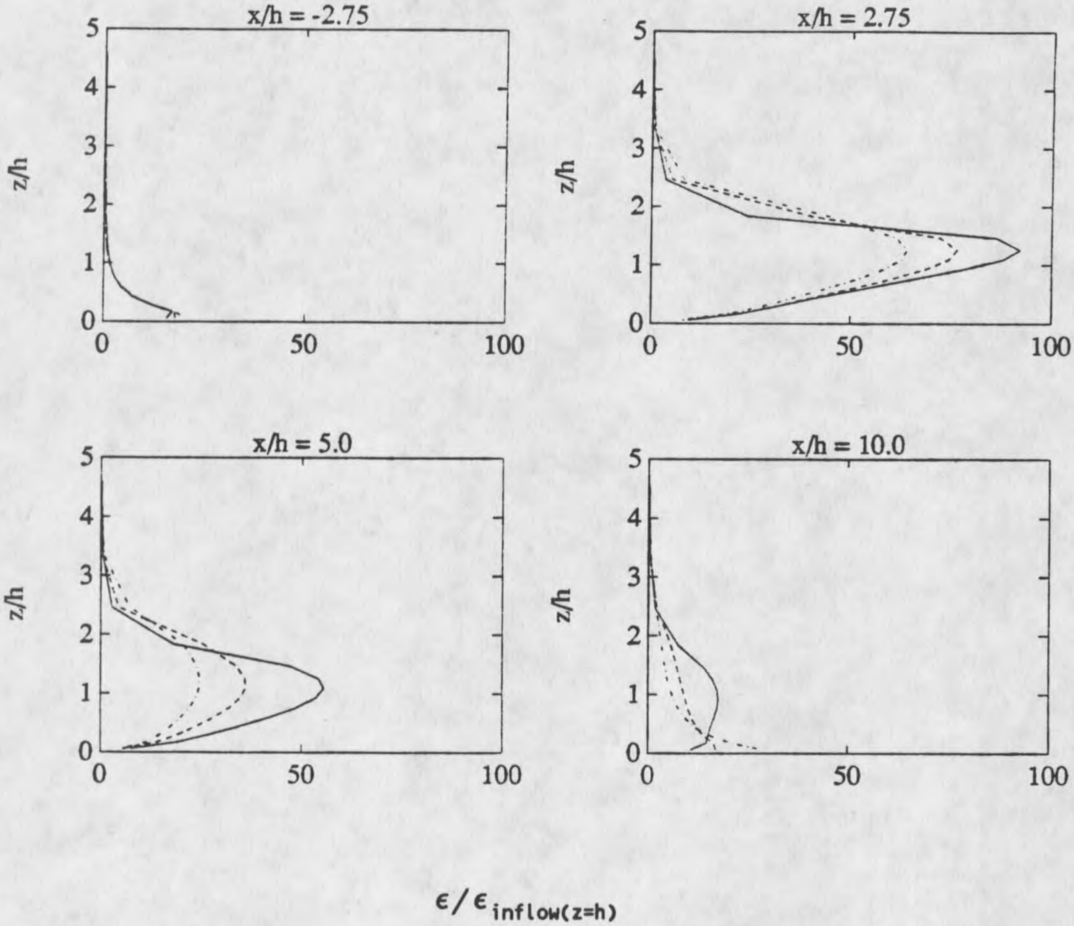

Figure 13. Profiles of the dissipation rate of turbulent kinetic energy ϵ for the cases concentration $w = 0$ (-), 0.2 (--), and 0.4 (-.-) at inflow and upper boundaries.

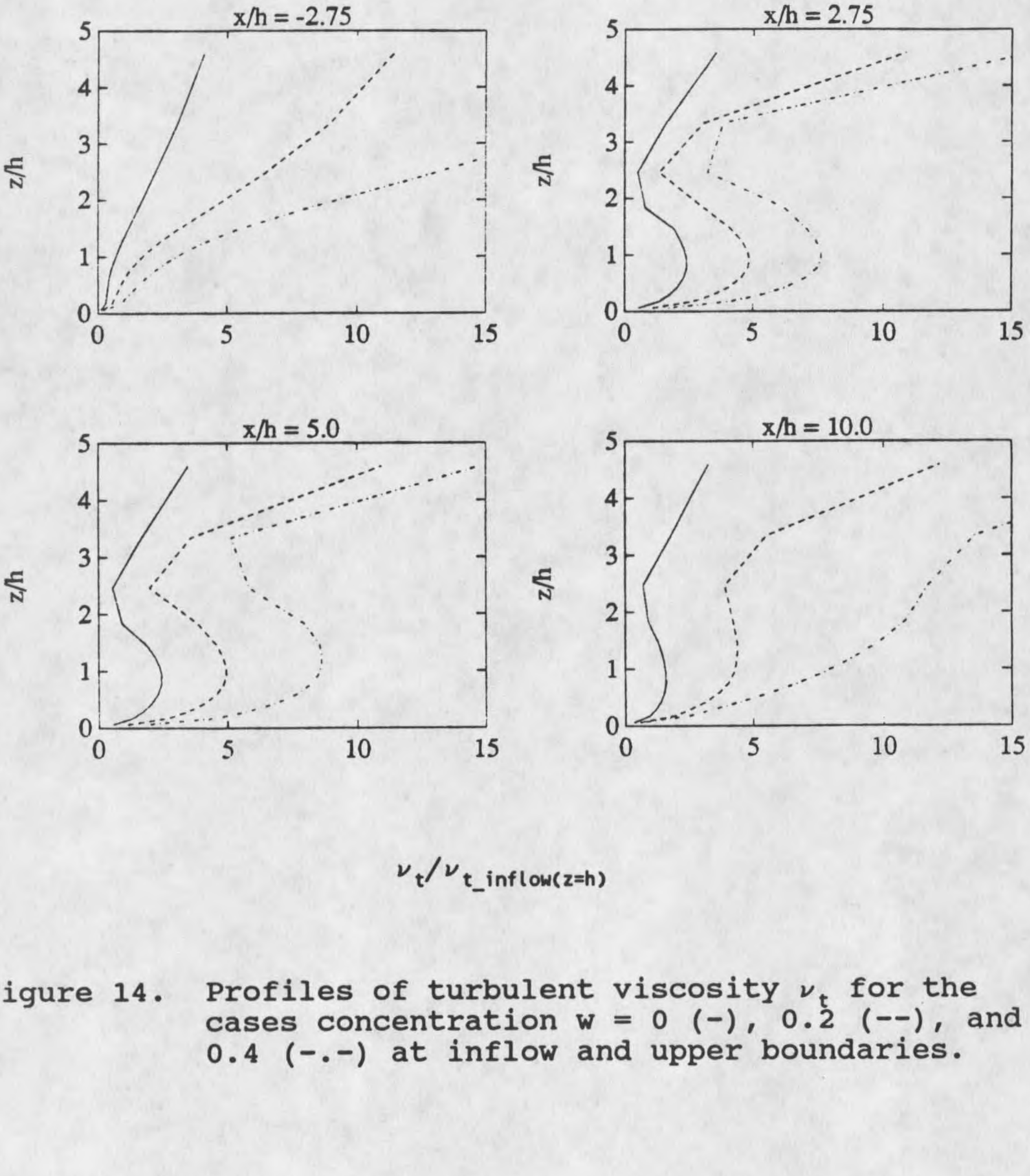

Figure 14. Profiles of turbulent viscosity ν_t for the cases concentration w = 0 (-), 0.2 (--), and 0.4 (-.-) at inflow and upper boundaries.

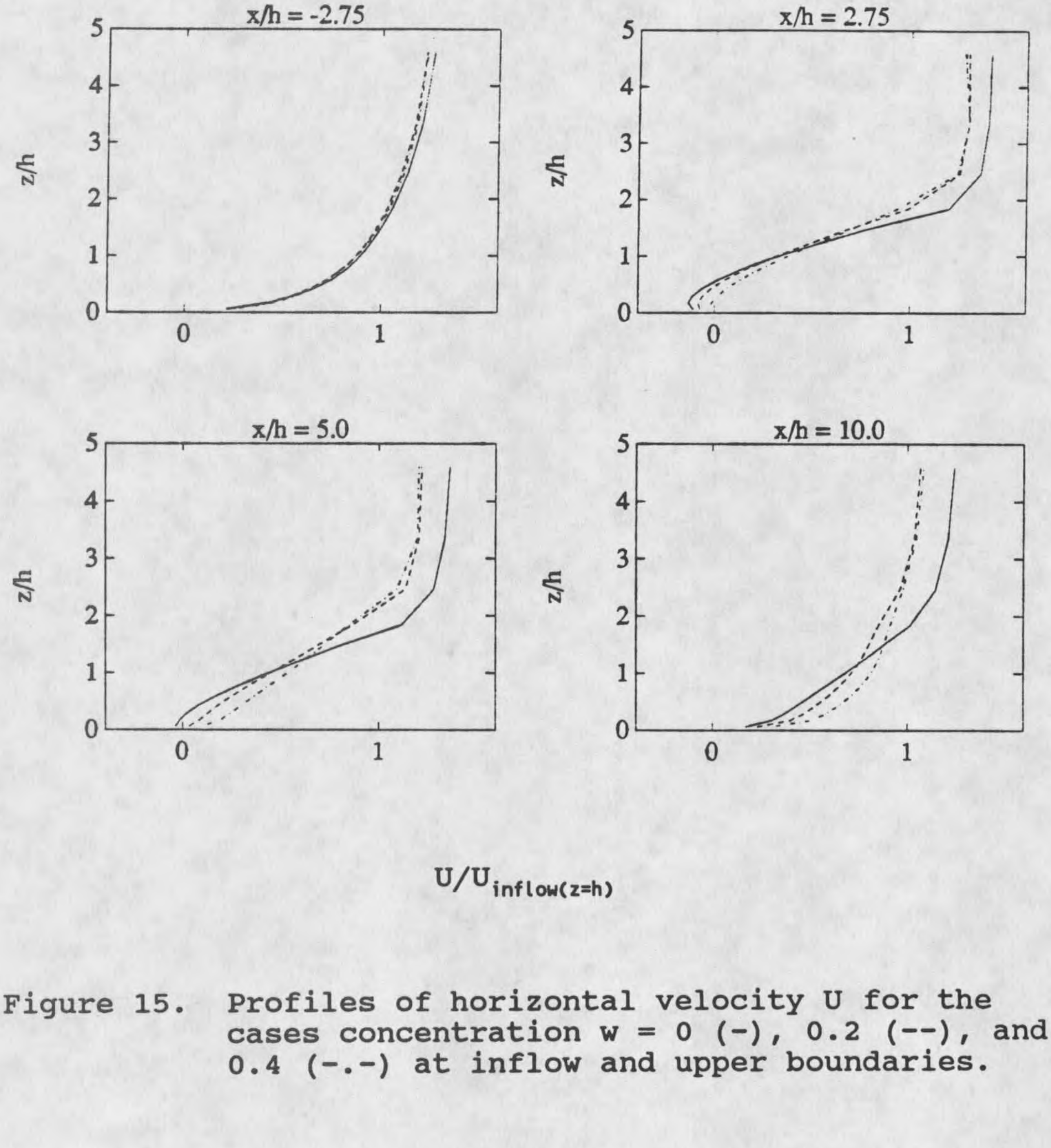

Figure 15. Profiles of horizontal velocity U for the cases concentration w = 0 (-), 0.2 (--), and 0.4 (-.-) at inflow and upper boundaries.

by its value at the inflow boundary at the wall height, $z = h$. These plots show, with increasing concentration, an increase in k, a decrease in ϵ, an increase in ν_t, and a decrease in velocity above the barrier and an increase in velocity behind the barrier.

In the computational model, the influence of the negative source terms does decrease the value of k and ϵ, if all else is held constant; this was verified by numerical experiment. But, in the true solution of the problem, the equations for k and ϵ are coupled to the continuity and x and z momentum equations, and the velocities contained within the k and ϵ equations change as k and ϵ change. A related concern lies with the production due to shear source terms, which are highly velocity dependent, present in the turbulence equations. In this application, during the computation of the solution, k and ϵ can be thought of as being initially reduced due to the presence of particles. This led to a reduction in ν_t which served to increase the fluid velocity. This increase in velocity, in particular in the region at the top and just to the lee of the wall, produced an increase in the velocity shear and a subsequent increase in the shear production source term. This in turn led to a further modification of the values of k and ϵ.

The relative importance of the shear production terms between the k and ϵ equations can be seen by looking at the coefficients which multiply the squared velocity gradient term in each equation. At the inflow boundary, the multiplication factors are of the same magnitude at a position just above the surface. At the wall top height $z = h$, the k equation multiplication factor is an order of magnitude greater than that of the ϵ equation; and at the top of the flow domain, the k equation factor is two orders of magnitude greater than the ϵ equation multiplication factor. As a consequence, the shear production term at the wall top is roughly a factor of 10 larger for the k equation than for the ϵ equation. The balance of these changes has led to an increase in k and a decrease in ϵ due to the addition of the particulate source terms. The velocity gradients encountered by Chen and Wood (1985) were much less severe, and it is expected that the change in the shear production term did not have the dominance found in the present study.

Snow Accumulation Patterns

The surface shear velocity U_* is computed as part of the turbulent flow field computation. Thus, the turbulent flow field can be readily used to run the saltation model (Equation 27) when a threshold shear velocity U_{*t} has been provided. The saltation model implies that if the shear velocity is greater than the threshold velocity, mass will be transported, and if the shear velocity drops below this value, mass transport will cease.

To develop snow accumulation patterns it is assumed that snow will be deposited within the control volume where the shear velocity drops below the threshold. Using the computational model's ability to block out control volumes, it will then be assumed that enough snow accumulates to fill that control volume. When the control volume has been blocked out a new lower boundary configuration presents itself. The flow field is then recomputed and the process is repeated until an equilibrium drift has been formed which contains no regions of shear velocity below the threshold. Since the saltation model has been developed to describe the mass flux over a uniform horizontal surface, any more sophisticated application of the saltation model to this more complex flow problem was not attempted. Also, since it can be shown that the saltating mass flux is at least an order of magnitude greater than the precipitating mass flux during a typical snow storm event, it will be assumed that the drift is formed through saltation mass transport only. As a consequence of this assumption, the vertically integrated rate of snow transport within the turbulent flow layer $Q_t(x)$, described by Equation 28, will be set equal to zero in Equation 29.

Since the drift is formed by blocking out rectangular control volumes, step changes in the bottom boundary result. This leads to unrealistic (lower) velocities in the control volumes immediately followed by a step. Therefore, for the purpose of computing surface shear velocities to locate snow accumulation regions, shear velocities are calculated using the velocity at the second control volume above the surface instead of the first. This effectively produces a smoothing of the shear velocity over the drift surface.

The equilibrium drift configuration computed for a threshold shear velocity $U_{*t} = 0.2$ is given in Figure 16. The evolution of the profile is illustrated in Figure 17. Here the numbers indicate the sequence in which control volumes were blocked out as the drift evolved.

Figure 17 shows the drift first forming windward of the wall, in the region of the recirculation eddy, where a shear velocity below the threshold is first encountered. This drift builds at a position removed

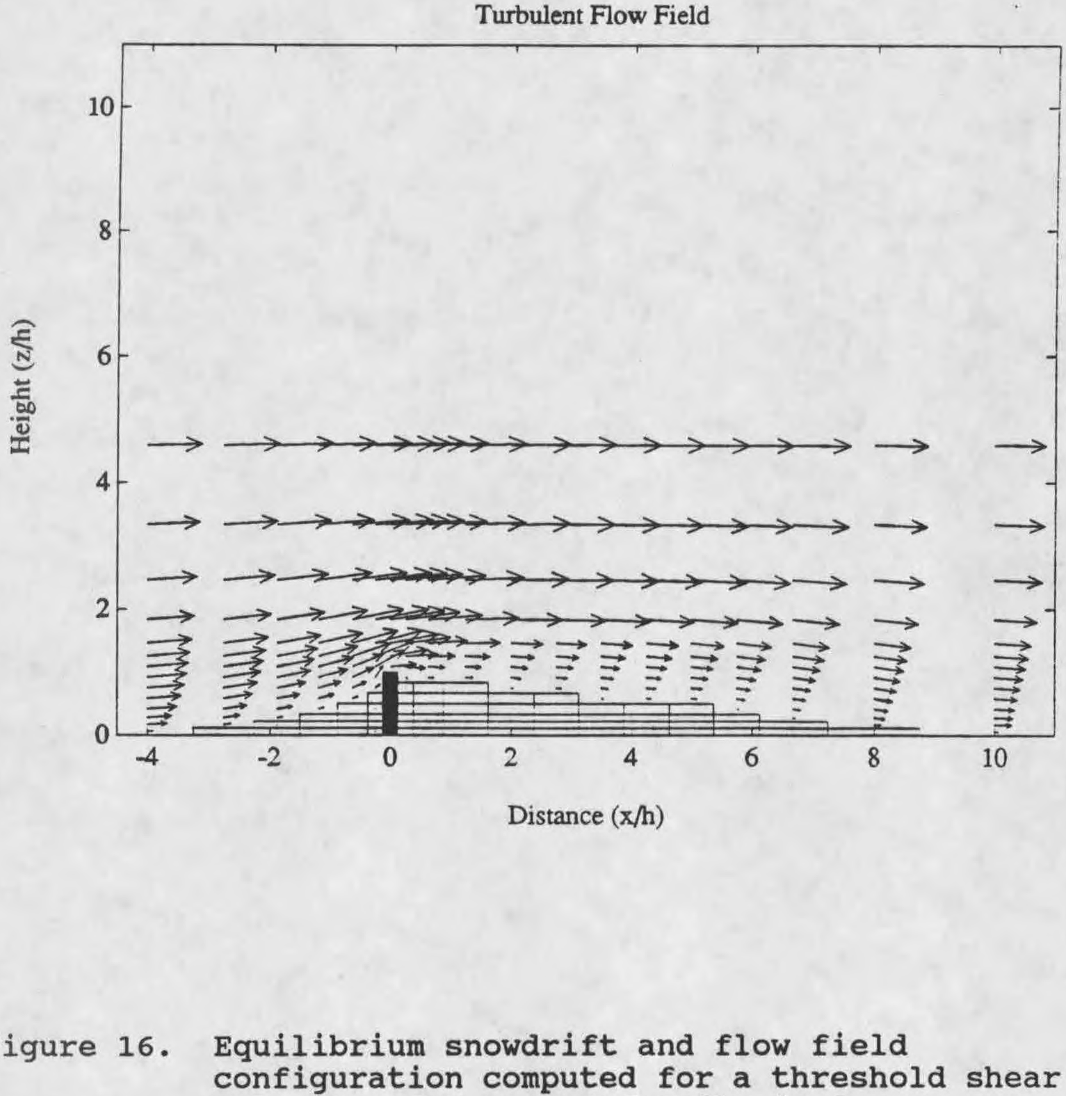

Figure 16. Equilibrium snowdrift and flow field configuration computed for a threshold shear velocity of $U_{*_t} = 0.2$ for the test case single-phase, turbulent flow problem.

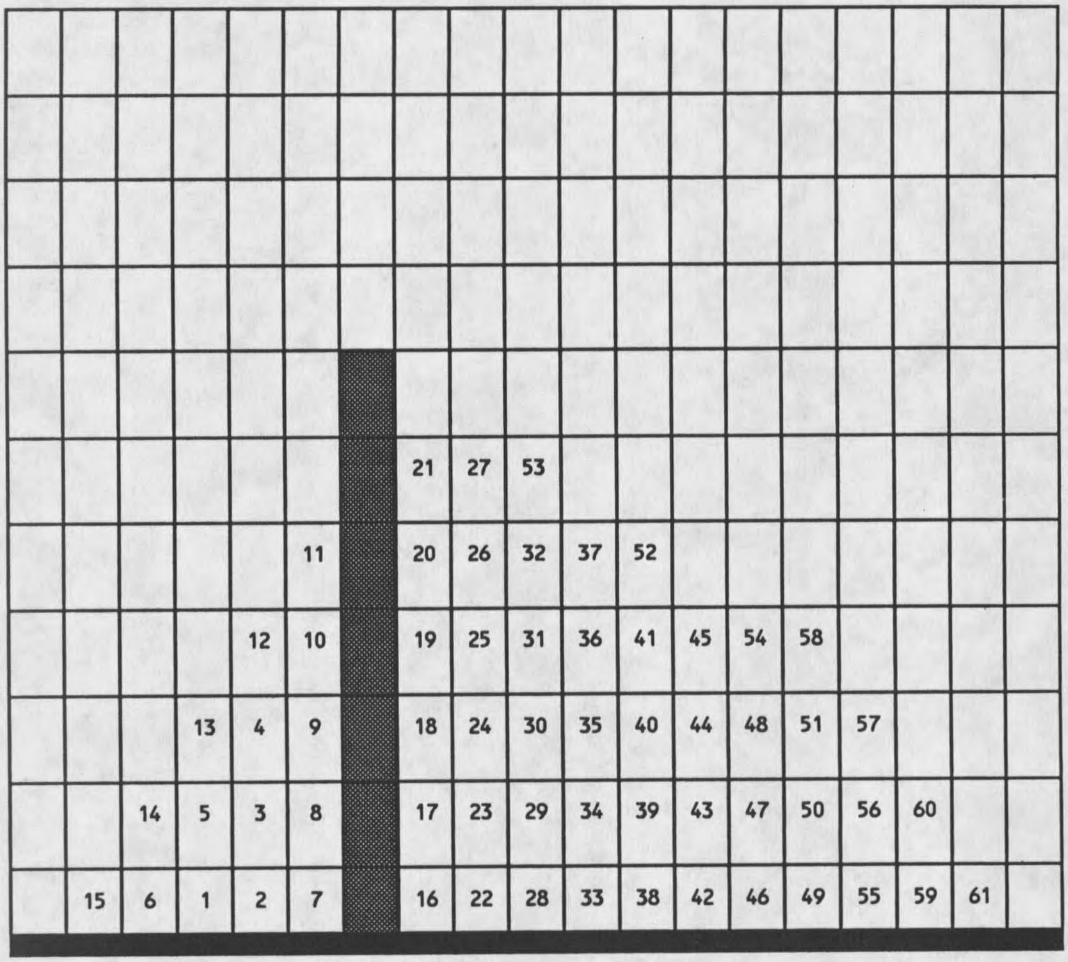

Figure 17. Progression of snowdrift development for the case $U_{*t} = 0.2$. The numbers indicate the sequence with which the model blocked control volumes as the drift evolved to the equilibrium configuration (Figure 16).

from the wall and then fills in between the wall and the drift. After the windward drift reaches equilibrium, snow begins to accumulate in the lee of the wall. The shoulder or scarp formation found in the lee drift at intermediate stages, for example after the 41st blocked control volume has been blocked, is a commonly observed feature of snowdrift formation (Mellor, 1965; Tabler, 1975b). As the lee drift extends downwind of the wall the flow field is modified such that accumulation occurs at locations windward of the main drift scarp. This intuitively appealing result also occurs in a similar fashion with the drift windward of the wall.

The lee equilibrium drift surface has a slope of 10 to 13 percent, depending on how it is measured. Studies considering embankment slopes descending from a level plain are available for comparison. For this case Finney (1939) found that no snow accumulates on embankment slopes of less than 17 percent. Berg and Caine (1975) and Tabler (1975b) both found only small accumulations on slopes of 17 percent for this same slope configuration. For an approach slope of approximately 10 percent, Tabler's study found snow accumulated on a lee slope of approximately 10 percent. It is expected that the relatively steep windward snowdrift, and associated wind flow pattern, should induce a longer drift than those produced in the examples having an approach slope of zero. Since the snow accumulation scheme is strongly dependent upon the threshold shear velocity, increasing or decreasing its value will govern the size of the equilibrium drift.

CHAPTER 7

SUMMARY AND CONCLUSIONS

A physically based computational model describing two-phase, turbulent flow has been developed and applied to the problem of blowing and drifting snow. In developing the model it has been assumed that the flow of interest is composed of two distinct layers, one containing an upper turbulent mixture of air and snow, and a second containing the snow transported by saltation process very close to the ground. This lowest layer is considered to provide the majority of mass transport which is deposited as snowdrifts and produces the snow accumulation patterns found in the natural system.

The two layers are described by two different models and they are coupled through their common boundary. The turbulent flow model consists of a general solution of the time averaged, steady state, two-dimensional Navier-Stokes equations, where the k-ϵ turbulence model has been used to close the system of equations. The effect of particulates on the turbulent flow field is accounted for by computing the particle concentration field using a convection-diffusion equation and a subsequent modification of the k-ϵ turbulence model. In this modification, the particulate phase is assumed to follow the mean flow, but on the high frequency turbulent fluctuation level, the particles do not exactly follow the fluid. The resulting slip between the phases is considered to be a hydrodynamic drag force.

For the snow particle concentrations found during heavily precipitating snow storms, the model has shown that snow particles can be considered a passive additive to the air flow; the snow is transported with the velocity of the fluid, and the particles have not modified the turbulent or flow structure of the fluid. For the case of much higher concentrations found within other two-phase flows, the model shows the turbulence and flow field to have been significantly modified by the presence of particles.

A basic premise of the snow accumulation part of the model is that the saltating mass flux is governed by the fluid velocity just above the ground surface, and that when the velocity just above the surface drops below some threshold value, snow accumulates in that region. Implementation of these ideas into the model has lead to the computation of snow accumulation profiles which are very similar to published field and experimental data. Clearly the ability of the model to duplicate these measurements of snowdrift profiles rests on an accurate modeled representation of the turbulent flow field.

Benefits of this study have included an increased understanding of the physical processes which govern multi-phase, turbulent atmospheric boundary layer flows, and the computational capability of addressing scientific and economic questions related to blowing and drifting snow.

REFERENCES CITED

REFERENCES CITED

ASCE (1988). Turbulence modeling of surface water flow and transport: Part IV. J. Hyd. Engr., 114, 1034-1051.

Anderson, D.A., Tannehill, J.C. and R.H. Pletcher (1984). Computational Fluid Mechanics and Heat Transfer. Hemisphere Publishing Company, New York.

Anno, Y. (1984a). Requirements for modeling of a snowdrift. Cold Regions Sci. Technol., 8, 241-252.

Anno, Y. (1984b). Applications of Anno's modeling conditions to outdoor modeling of snowdrifts. Cold Regions Sci. Technol., 9, 179-181.

Bagnold, R.A. (1941). The physics of blown sand and desert dunes. Methuen, London, 265 pp.

Bagnold, R.A. (1966). An approach to the sediment transport problem from general physics. U. S. Geol. Surv. Prof. Pap., 37 pp.

Benson, C.S. (1982). Reassessment of winter precipitation on Alaska's Arctic Slope and measurements on the flux of wind blown snow. Geophysical Institute, University of Alaska Report UAG R-288, September 1982, 26 pp.

Berg, N.H. (1986). A deterministic model for snowdrift accumulation. Proceedings of the International Snow Science Workshop, Lake Tahoe, California, 22-25 Oct., 29-36.

Berg, N.H. and N. Caine (1975). Prediction of natural snowdrift accumulation in alpine areas. Final Report to Rocky Mountain Forest and Range Expt. Station (USFS 19-388-CA), Boulder, Dept. of Geography, University of Colorado, 69 pp.

Budd, W.F. (1966). The drifting of non-uniform snow particles. In: M. Rubin (ed.), Studies in Antarctic Meteorol., American Geophys. Union, Antarctic Res. Ser., 9, 59-70.

Budd, W.F., Dingle, R. and U. Radok (1966). The Byrd snow drift project: Outline and basic results. In: M. Rubin (ed.), Studies in Antarctic Meteorol., American Geophys. Union, Antarctic Res. Ser., 9, 71-134.

Chen, C.P. (1983). Studies in two-phase turbulence closure modeling. Ph.D. Thesis, Michigan State University.

Chen, C.P. and P.E. Wood (1984). Turbulence closure modeling of two-phase flows. Chem. Eng. Comm., 29, 291-310.

Chen, C.P. and P.E. Wood (1985). A turbulence closure model for dilute gas-particle flows. The Can. J. ChE., 63, 349-360.

Crowe, C.T. (1982). Review-numerical models for dilute gas-particle flows. J. Fluids Eng., 104, 297-303.

Danon, H., M. Wolfshtein and G. Hestroni (1977). Numerical calculations of two-phase turbulent round jet. Int. J. Multiphase Flow, 3, 223.

Decker, A.R. and R.L. Brown (1983). A turbulent mixture theory for the atmospheric mixture of snow and air. Annals of Glaciol., 4, 37-41.

Decker, A.R. and R.L. Brown (1985). Two dimensional solutions for a turbulent continuum theory for the atmospheric mixture of snow and air. Annals of Glaciol., 6, 53-58.

Dyunin, A.K. (1963). The mechanical conditions of snow erosion. National Research Council of Canada, Technical Translation 1101, 14 pp.

Dyunin, A.K. (1967). Fundamentals of the mechanics of snow storms. In: H. Oura (ed.), Physics of Snow and Ice, Vol I, Part 2, Institute of Low Temperature Science, Sapporo, Japan, 1065-1073.

Elghobashi, S.E. and T.W. Abou-Arab (1983). A two-equation turbulence model for two-phase flows.

Phys. Fluids, 26, 931-938.

Ferziger, J.H. (1990). Approaches to turbulent flow computation: Applications to flow over obstacles. J. Wind Eng. and Indust. Aero., 35, 1-19.

Finney, E.A. (1934). Snow control on the highways. Bulletin, Michigan Engineering Experiment Station (East Lansing), No. 57.

Finney, E.A. (1939). Snow drift control by highway design. Bulletin, Michigan State College Engineering Experiment Station, No. 86.

Fletcher, C.A.J. (1988). Computational Techniques for Fluid Dynamics, Vol. 2. Springer-Verlag, New York.

Fohn, P. (1980). Snow transport over mountain crests. J. Glaciol., 26(94), 469-480.

Frost, W., Harper, W.L. and G.H. Fichtl (1975). Analysis of atmospheric flow over a surface protrusion using the turbulence kinetic energy equation. Boundary-Layer Meteorol., 8, 401-417.

Genchev, Z.D. and D.S. Karpunov (1980). Effects of the motion of dust particles on turbulence transport equations. J. Fluid Mech., 101, 833.

Gibson, M.M. and B.E. Launder (1978). Ground effects on pressure fluctuations in the atmospheric boundary layer. J. Fluid Mech., 86, 491-511.

Good, M.C. and P.N. Joubert (1968). The form drag of two-dimensional bluff-plates immersed in turbulent boundary layers. J. Fluid Mech., 31, 547-582.

Hanjalic, K. and B.E. Launder (1972). A Reynolds-stress model of turbulence and its application to thin shear flows. J. Fluid Mech., 52, 609-638.

Hanjalic, K. and B.E. Launder (1980). Sensitizing the dissipation equation to irrotational strains. J. Fluids Eng., 102, 34-40.

Hestroni, G. and M. Sokolov (1971). Distribution of mass velocity and intensity of turbulence in a two-phase turbulent jet. J. Appl. Mech., 93, 315-327.

Hinze, J.O. (1975). Turbulence. McGraw-Hill, New York.

Holton, J.R. (1979). An Introduction to Dynamic Meteorology. Academic Press, New York.

Iversen, J.D. (1979). Drifting snow similitude. J. Hyd. Div. ASCE, 105 (HY6), 737-753.

Iversen, J.D. (1980). Drifting snow similitude - transport rate and roughness modeling. J. Glaciol., 26, 393-403.

Iversen, J.D., Greeley, R., White, B.R. and J.B. Pollack (1975). Eolian erosion of the Martian surface, Part 1: Erosion rate similitude. Icarus, 26, 321-331.

Jacobs, A. (1984). The flow around a thin closed fence. Boundary-Layer Meteorol., 28, 317-328.

Jenssen, D. (1963). Snow saltation (Part IV). Ph.D. Thesis, University of Melbourne, 274-316.

Jones, W.P. and B.E. Launder (1972). The prediction of laminarization with a two-equation model of turbulence. Int. J. Heat Mass Transfer, 15, 301-314.

Kind, R.J. (1976). A critical examination of the requirements for model simulation of wind-induced erosion/deposition phenomena such as snow drifting. Atmos. Environment, 10, 219-227.

Kind, R.J. (1981). Snow drifting. In: D.M. Gray and D.H. Male (eds.), Handbook of Snow - Principles, Processes, Management and Use, Pergamon Press Canada Ltd., Toronto.

Kind R.J. (1986). Snowdrifting: A review of modeling methods. Cold Regions Sci. Technol., 12, 217-228.

Kind, R.J. and S.B. Murray (1982). Saltation flow measurements relating to modeling of snowdrifting. J. Wind Eng. and Indust. Aero., 10, 89-102.

Kobayashi, D. (1972). Studies on snow transport in low-level drifting snow. Contrib. Inst. Low Temp. Sci., Hokkaido Univ., Series A, No. 24, 1-58.

Kobayashi, S. (1979). Studies on interaction between wind and dry snow surface. Contrib. Inst. Low Temp. Sci., Hokkaido Univ., Series A, No. 29, 1-64.

Lakshminarayana, B. (1986). Turbulence modeling for complex shear flows. AIAA J., 24, 1900-1917.

Launder, B.E. and D.B. Spalding (1974). The numerical computation of turbulent flow. Comp. Meth. in Appl. Mech. and Eng., 3, 269.

Lee, L.W. (1975). Sublimation of snow in a turbulent atmosphere. Ph.D. Dissertation, Univ. of Wyo., Laramie.

Liston, G.E. (1986). Seasonal snowcover of the foothills region of Alaska's arctic slope: A survey of properties and processes. M.S. Thesis, University of Alaska, Fairbanks.

Lumley, J.L. and H.A. Panofsky (1964). The Structure of Atmospheric Turbulence. John Wiley and Sons, New York.

Maeno, N., K. Araoka, K. Nishimura, and Y. Kaneda (1979). Physical aspects of the wind-snow interactions in blowing snow. J. Fac. Sci., Hokkaido Univ., ser. 8, 6(1), 127-141.

Male, D.H. (1980). The seasonal snowcover. In: S. Colbeck (ed.), Dynamics of Snow and Ice Masses. Academic Press, New York, 305-395.

Marble, F.E. (1970). Dynamics of dusty gases. Annual Rev. Fluid Mech., 2, 397.

Martinelli, M. and A.D. Ozment (1985). Some strength features of natural snow surfaces that affect snow drifting. Cold Regions Sci. Technol., 11, 267-283.

McFadden, J.D. and R.A. Ragtzkie (1967). Climatological significance of albedo in central Canada. J. Geophys. Res., 72, 1135-1143.

Mellor, M. (1965). Blowing snow. U.S.A. CRREL Monogr., part 3, sec. A3c, U.S. Army Cold Regions Research and Engineering Laboratory, Hanover, N.H., 79 pp.

Mellor, M. (1970). A brief review of snowdrifting research. Snow Removal and Ice Control Research, Spec. Rep. 115, U.S. Army Cold Reg. Res. and Eng. Lab, Hanover, N.H., 196-209.

Mellor, M. and U. Radok (1960). Some properties of drifting snow. In: Antarctic Meteorology, Pergamon Press, Oxford, 333-346.

Melville, W.K. and K.N.C. Bray (1979). A model of the two-phase turbulent jet. Int. J. Heat Mass Transfer, 22, 647-656.

Moderrass, D., H. Tan and S. Elghobashi (1984). Two-component LDA measurement in a two-phase turbulent jet. AIAA J., 22, 624-630.

Narita, H. (1978). Controlling factors of drifting snow. Memoirs of National Institute of Polar Research Special Issue, 7, 81-92.

Ogawa, Y. and P.G. Diosy (1980). Surface roughness and thermal stratification effects on the flow behind a two-dimensional fence. Atmos. Envir., 14, 1301-1320.

Oura, H. (1967). Studies of blowing snow I. In: H. Oura (ed.), Physics of Snow and Ice, Proc. Int. Conf. on Low Temp. Sci., Hokkaido University, Sapporo, Vol. 1, part 2, 1085-1097.

Oura, H., T. Ishida, D. Kobayashi, S. Kobayashi, and T. Yamada (1967). Studies on blowing snow II. In: H. Oura (ed.),Physics of Snow and Ice, Vol I, part 2, Institute of Low Temperature Science, Sapporo, Japan, 1099-1117.

Owen, P.R. (1964). Saltation of uniform grains in air. J. Fluid Mech., 20, 225-242.

Panofsky, H.A. and J.A. Dutton (1984). Atmospheric Turbulence. John Wiley and Sons, New York.

Patankar, S.V. (1978). A numerical method for conduction in composite materials, flow in irregular geometries and conjugate heat transfer. Proc. 6th Int. Heat Transfer Conf., Toronto, Vol. 3, 297.

Patankar, S.V. (1980). Numerical heat transfer and fluid flow. Hemisphere Publishing, New York.

Perera, M.D.A.E.S. (1981). Shelter behind two-dimensional solid and porous fences. J. Wind Eng. and Indust. Aero., 8, 93-104.

Peyret, R. and T.D. Taylor (1983). Computational methods for fluid flow. Springer-Verlag, New York.

Popper, J., Abnaf, N. and G. Hestroni (1974). Velocity measurements in a two-phase turbulent jet. Int. J. Multiphase Flow, 1, 715.

Radok, U. (1977). Snow drift. J. Glaciol., 19(81), 123-129.

Reynolds, A.J. (1976). The variation of turbulent Prandtl and Schmidt numbers in wakes and jets. Int. J. Heat and Mass Transfer, 19, 757.

Rodi, W. (1980). Turbulence models for environmental problems. In: Prediction Methods for Turbulent Flow, von Karman Inst., McGraw-Hill, New York, 276-281.

Rodi, W. (1982). Examples of turbulence models for incompressible flows. AIAA J., 20, 872-879.

Rodi, W. (1984). Turbulence models and their application in hydraulics. International Association for Hydraulic Research, Delft, The Netherlands.

Rodi, W. (1985). Calculation of stably stratified shear-layer flows with a buoyancy-extended k-e turbulence model. In: Turbulence and Diffusion in Stable Environments, (ed.) J.C.R. Hunt, Clarendon Press, Oxford.

Schlichting, H. (1979). Boundary-layer theory. McGraw-Hill, New York.

Schmidt, R.A. (1972). Sublimation of wind-transported snow-A model. Res. Pap. RM-90, Rocky Mtn. For. and Range Expr. Sta., For. Serv., U.S. Dept. of Agric., Fort Collins, Colo..

Schmidt, R.A. (1980). Threshold wind-speeds and elastic impact in snow transport. J. Glaciol., 26, 453-

467.

Schmidt, R.A. (1981). Estimates of threshold windspeed from particle sizes in blowing snow. Cold Reg. Sci. Technol., 4, 187-193.

Schmidt, R.A. (1982a). Properties of blowing snow. Rev. Geophys. Space Phys., 20, 39-44.

Schmidt, R.A. (1982b). Vertical profiles of wind speed, snow concentration, and humidity in blowing snow. Boundary-Layer Meteorol., 23, 223-246.

Schmidt, R.A. (1984). Measuring particle size and snowfall intensity in drifting snow. Cold Reg. Sci. Technol., 9, 121-129.

Schmidt, R.A. (1986a). Transport rate of drifting snow and the mean wind speed profile. Boundary-Layer Meteorol., 34, 213-241.

Schmidt, R.A. (1986b). Snow surface strength and the efficiency of relocation by wind. Cold Regions Hydrology Symp., Fairbanks, Alaska, July 1986, Am. Water Resources Assoc., 355-358.

Schmidt, R.A., R.D. Tabler, and R.L. Jairell (1982). A new device for sampling mass flux of blowing snow. Proc. Western Snow Conference, 50, 102-111.

Schmidt, R.A., R. Meister, and H. Gubler (1984). Comparison of snow drifting measurements at an alpine ridge crest. Cold Reg. Sci. Technol., 9, 131-141.

Slaughter, C.W., Mellor, M., Sellmann, P.V., Brown, J. and L. Brown (1975). Accumulating snow to augment the fresh water supply at Barrow, Alaska. Cold Regions Res. and Eng. Lab. Spec. Rept. 217, 21p.

Sorbjan, Z. (1989). Structure of the Atmospheric Boundary Layer. Prentice Hall, New Jersey.

Sproull, W.T. (1961). Viscosity of dusty gases. Nature, 190, 976-978.

Sutton, S.B., Brandt, H. and B.R. White (1986). Atmospheric dispersion of a heavier-than-air gas near a two-dimensional obstacle. Boundary-Layer

Meteorol., 35, 125-153.

Tabler, R.D. (1975a). Estimating the transport and evaporation of blowing snow. Proc. Snow Management on the Great Plains, Great Plains Agr. Council, Publ. No. 73, Agr. Exp. Sta., U. of Nebraska, Lincoln, 85-104.

Tabler, R.D. (1975b). Predicting profiles of snowdrifts in topographic catchments. Proc. Western Snow Conference, 43, 87-97.

Tabler, R.D. (1980). Geometry and density of drifts formed by snow fences. J. Glaciol., 26(94), 405-419.

Tabler, R.D. and L. Jairell (1981). Studying snowdrifting problems with small-scale models outdoors. Proc. 1980 Western Snow Conference, 48, 1-13.

Tabler, R.D. and R.A. Schmidt (1972). Weather conditions that determine snow transport distances at a site in Wyoming. Proc. UNESCO/WMO/IAHS Int. Symp. on Role of Snow and Ice in Hydrology (Banff, Alberta, Canada), 118-127.

Takeuchi, M. (1980). Vertical profiles and horizontal increase of drift snow transport. J. Glaciol., 26(94), 481-492.

Uematsu, T., Kaneda, Y., Takeuchi, K., Nakata, T. and M. Yukumi (1989). Numerical simulation of snowdrift development. Annals of Glaciol., 13, 265-268.

White, B. and J. Schulz (1977). Magnus effect in saltation. J. Fluid Mech., 81(3), 497-512.

Wyngaard, J.C. and O.R. Cote (1971). The budgets of turbulent kinetic energy and temperature variance in the atmospheric surface layer. J. Atmos. Sci., 28, 190-201.

Yuu, S., Yasukonchis, N., Hirosawa, Y. and T. Jotaki (1978). Particle turbulent diffusion in a dust laden round jet. AIChE J., 24, 509.

APPENDIX

COMPUTER PROGRAMS

Figure 18. MATLAB Programs.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOLVE.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%------------------------------------------------------------------
% This is the main driver program for SOLVing the momentum and
%   k - epsilon equations describing the turbulent flow field.
%------------------------------------------------------------------

clear

%------------------------------------------------------------------
% Provide maximum number of global iterations.
%------------------------------------------------------------------

max_iteration = 40;

%------------------------------------------------------------------
% Provide relaxation values for velocity and viscosity
%   computations.
%------------------------------------------------------------------

relax_gamma = .75;
relax_k = .75;
relax_eps = .75;
relax = [.5*ones(1,max_iteration)];

%------------------------------------------------------------------
% Provide values of Pressure Control Volume size in x and y
%  directions.
%------------------------------------------------------------------

dx_p = [3 2 1.5 1 .5 1/3*ones(1,6) 2/9 2/9 2/9];
dy_p = [8 6 4 3 2 1.5 1.25 1 .5 .5 .5 1 1.5*ones(1,7) 2.25 3 5 ...
        7 9 11 13];

%------------------------------------------------------------------
% Supply the constants used in applying log-wall boundary
%  conditions.
%------------------------------------------------------------------

rough_length = .001; % (roughness length = z0 = .1 cm)
comp_bndry = .01;    % (dist from ground sfc to computational bndry
                     %     = 1 cm)

%------------------------------------------------------------------
% Set initial velocity fields.
%------------------------------------------------------------------

vel_at_10m = 10;   % Incomming v(z = 10 meters), (m/s)

%------------------------------------------------------------------
% Provide positions of the top blocked off pressure control volumes
%   for permanent walls or structures.
%------------------------------------------------------------------
```

```
i_pos = [8];
j_pos = [10];
j_pos_wall = j_pos;

%-----------------------------------------------------------------

conc_const = 0.2;

%-----------------------------------------------------------------

     INFO
     BLOCKED

for iteration = 1:max_iteration
     SOLV_VEL
     SOLV_K
     SOLV_EPS
     SOLV_CON
end

%-----------------------------------------------------------------
% POST PROCESS THE DATA
%-----------------------------------------------------------------

% COMPUTE VELOCITY VALUES AT PRESSURE GRID POINTS.

     u = .5 * (ubc(2:I+1,2:J+1) + ubc(1:I,2:J+1));
     v = .5 * (vbc(2:I+1,2:J+1) + vbc(2:I+1,1:J));

% Set gamma values to 'nan' in blocked off regions.

     for k = 1:j_pos_len
       gamma(i_pos(k):I,j_pos(k)) = nan + 0 * ...
                                 gamma(i_pos(k):I,j_pos(k));
     end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INFO.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-----------------------------------------------------------------
% THIS PROGRAM PROVIDES PROBLEM DEPENDENT FLOW INFORMATION
%-----------------------------------------------------------------

% Compute the number of interior pressure nodes in the x and y
%   directions.

     I = length(dx_p);
     J = length(dy_p);

% COMPUTE PRESSURE CONTROL VOLUME SIZE AND POSITION INFORMATION

     [delx_p,dely_p,f_e,f_n,x_coords,y_coords,x_wall,y_wall] = ...
              CV_INFO(dx_p,dy_p,comp_bndry);

% Define constants used in turbulent flow computations.

     ro = 1.3;          %  Density of Air, (kg/m^3).
     k_visc = 13e-6;    %  Kinematic Viscosity of Air, (m^2/s).
```

```
      ro_ice = 910;
      d_particle = 5e-4;
      Ufall = .75;          %  Settling velocity of particulate field
                            %    (m/s).

      C_mu = .03;
      C1_eps = 1.16;
      C2_eps = 1.92;
      sigma_k = 1.0;
      sigma_eps = 1.3;
      tstar = d_particle^2 * ro_ice / (18*k_visc*ro);
      kappa = .4;
      Sc_num = .5;          %  Turbulent Schmidt Number.

% Compute the constants used in applying log-wall boundary
%   conditions.

      log_const = log(comp_bndry/rough_length) / ...
                  log((comp_bndry + .5 * dx_p(I))/rough_length);
      tau_const = kappa / ...
                  log((comp_bndry + .5 * dx_p(I))/rough_length);

% Develop k, epsilon, and velocity inflow boundary conditions.

       Z = x_coords';

       Utau_inflow = kappa * vel_at_10m / ...
                           log(10/rough_length); % (10=ht. z)
       eps_sbc = Utau_inflow^3 ./ (kappa * Z(2:I+1));
       k_sbc = Utau_inflow^2 / sqrt(C_mu) * ones(I,1);
       vbcs = Utau_inflow / kappa * log(Z/rough_length);

% SET INITIAL VELOCITY FIELDS.

       ubc = zeros(I+1,J+2);
       for j = 1:J+1
         vbc(:,j) = vbcs;
       end
       ustar = ubc;
       vstar = vbc;

% SET INITIAL TURBULENT VISCOSITY FIELDS.

       gamma_inflow = ro .* C_mu * k_sbc.^2 ./ eps_sbc;

       for j = 1:J
       km(:,j) = k_sbc;
          epsm(:,j) = eps_sbc;
          gamma(:,j) = gamma_inflow;
       end

       km_old = km;
       epsm_old = epsm;
       conc = conc_const * ones(I,J);

% SET gamma = 5e30 IN PERMANENTLY BLOCKED OFF REGIONS.

       for k = 1:length(j_pos)
         gamma(i_pos(k):I,j_pos(k)) = 5e30 + ...
```

```
        gamma(i_pos(k):I,j_pos(k));
      end

      gamma_old = gamma;
      gamma_orig = gamma;

% DEVELOP SPARSE STORAGE INDEXING SCHEME.

      COLINDX_P = INDEX(I,J);
      COLINDX_U = INDEX(I+1,J+2);
      COLINDX_V = INDEX(I+2,J+1);

% SET FLAGS AND TOLERANCES.

      flag_p = 1;
      flag_vel = 0;

      max_iter_p = I*J;
      max_iter_vel = I*J;

      TOL_p = 1e-2;
      TOL_vel = 1e-5;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INDEX.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function COLINDEX = INDEX(I,J)

% Set up of an indexing, sparse storage scheme for the SIMPLER
%   program.

     col_count = (1:I*J)';

     COLINDEX = zeros(I*J,5);

     COLINDEX(I+1:I*J,1) = col_count(1:I*(J-1));
     COLINDEX(2:I*J,2) = col_count(1:I*J-1);
     COLINDEX(:,3) = col_count;
     COLINDEX(1:I*J-1,4) = col_count(2:I*J);
     COLINDEX(1:I*(J-1),5) = col_count(I+1:I*J);

     for k = 1:J-1
       COLINDEX(I*k+1,2) = 0;
       COLINDEX(I*k,4) = 0;
     end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BLOCKED.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Whenever a new c.v. is blocked off, BLOCKED.M must be run.

% Establish the coordinates of the top blocked off control volume.

i_pos = [];
j_pos = [];

for i = 1:I
```

```
  for j = 1:J
    if gamma(i,j) >= 1e3
    if gamma(i-1,j) < 1e3
      i_pos = [i_pos i];
      j_pos = [j_pos j];
    end
    end
  end
end

j_pos_len = length(j_pos);
gamma_old = gamma;

% Set ubc and vbc velocities to zero in blocked off regions.

for q = 1:j_pos_len
  ubc(i_pos(q):I+1,j_pos(q)+1) = 0 * ubc(i_pos(q):I+1,j_pos(q)+1);
  vbc(i_pos(q)+1:I+2,j_pos(q)) = 0 * vbc(i_pos(q)+1:I+2,j_pos(q));
  vbc(i_pos(q)+1:I+2,j_pos(q)+1) = 0 *
vbc(i_pos(q)+1:I+2,j_pos(q)+1);
end

ustar = ubc;
vstar = vbc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOLV_VEL.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

     disp('*************************************')
%-------------------------------------------------------------------
% COMPUTE COEFFICIENTS AND BUILD U and V SYSTEMS OF EQUATIONS.
%-------------------------------------------------------------------

     [gamma_bndry_ew,gamma_bndry_ns] = GAMMA1(gamma,f_e,f_n);
     gamma_corners = GAMMA2(gamma_bndry_ns,f_e);
     [A_u,A_v] = UV_COEF(ubc,vbc,dx_p,dy_p,delx_p,dely_p, ...
                         f_e,f_n,gamma,gamma_corners,ro);

%-------------------------------------------------------------------
% MODIFY THE COEFFICIENTS TO ACCOUNT FOR BOUNDARY CONDITIONS AND
%   COMPUTE RIGHT HAND SIDE BOUNDARY CONDITION (FORCE) VECTORS.
%-------------------------------------------------------------------

     SET_BC

%-------------------------------------------------------------------
% COMPUTE UHAT AND VHAT.
%-------------------------------------------------------------------

     [uhat,vhat,A_u_main,A_v_main,sum_au,sum_av] = ...
          UV_HAT(ubc,vbc,A_u,A_v,COLINDX_U,COLINDX_V,bu_bc,bv_bc);

%-------------------------------------------------------------------
% COMPUTE THE PRESSURE COEFFICIENTS.
%-------------------------------------------------------------------

     [A_p,b_p] = P_COEF(A_u,A_v,uhat,vhat,dx_p,dy_p,ro);
```

```
%-------------------------------------------------------------------
% SOLVE THE PRESSURE EQUATION (p.131), ASSUME dp/dn=0 on
%   BOUNDARIES (p.130).
%-------------------------------------------------------------------

     disp('      solve pressure equation')
     p = THOMAS(A_p,COLINDX_P,b_p,zeros(I,J),TOL_p, ...
                                             max_iter_p,flag_p);

%-------------------------------------------------------------------
% CREATE Pp-Pe AND Pp-Pn MATRICES FOR U AND V MOMENTUM EQNS(p.122).
%-------------------------------------------------------------------

     [dup,dvp] = P_FORCE(p,dx_p,dy_p);

%-------------------------------------------------------------------
% SOLVE FOR Ustar and Vstar (p.122).
%-------------------------------------------------------------------

     disp('   solve U  -  velocity equation')
     A_u(:,3) = A_u(:,3)/relax(iteration);
     b_u = dup + bu_bc + (1 - relax(iteration)) / ...
                          relax(iteration) * A_u_main .* ustar;
     ustar = THOMAS(A_u,COLINDX_U,b_u,ubc,TOL_vel, ...
                                          max_iter_vel,flag_vel);

     disp('   solve V  -  velocity equation')
     A_v(:,3) = A_v(:,3)/relax(iteration);
     b_v = dvp + bv_bc + (1 - relax(iteration)) / ...
                          relax(iteration) * A_v_main .* vstar;
     vstar = THOMAS(A_v,COLINDX_V,b_v,vbc,TOL_vel, ...
                                          max_iter_vel,flag_vel);

%-------------------------------------------------------------------
% CALCULATE MASS SOURCE, b_pp (p.125, 126). THIS IS THE AMOUNT BY
%   WHICH THE CONTINUITY EQUATION IS NOT SATISFIED FOR EACH CONTROL
%   VOLUME.
%-------------------------------------------------------------------

     b_pp = CONT_EQ(ustar,vstar,dx_p,dy_p,ro);

%-------------------------------------------------------------------
% SOLVE THE PPRIME EQUATION (p.125).  ASSUME dp/dn=0 on
%   BOUNDARIES (p.130).
%-------------------------------------------------------------------

     disp('  solve pressure correction equation')
     pprime = THOMAS(A_p,COLINDX_P,b_pp,zeros(I,J),TOL_p, ...
                                                max_iter_p,flag_p);

%-------------------------------------------------------------------
% CORRECT THE VELOCITY FIELD USING PPRIME (p.123).
%-------------------------------------------------------------------

     [ubc,vbc] = COR_VEL(A_u_main,A_v_main,dx_p,dy_p, ...
                                              ustar,vstar,pprime);

%-------------------------------------------------------------------
% COMPUTE AND DISPLAY VELOCITY FIELD CONVERGENCE INFORMATION.
```

```
%-----------------------------------------------------------------

     resid_cont = b_pp;
     resid_cont(I,:) = 0.0 * resid_cont(I,:);
     for k = 1:j_pos_len
       i = i_pos(k);
       j = j_pos(k);
       resid_cont(i-1,j) = 0.0;
       resid_cont(i:I,j-1:j+1) = 0.0 * resid_cont(i:I,j-1:j+1);
     end

     cont = [cont;iteration max(max(abs(resid_cont))) ...
                  sum(sum(abs(resid_cont)))/(I*J)];

     disp('  iteration  maximum    average')
     disp('             continuity    cont')
     disp(cont)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SET_BC.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-----------------------------------------------------------------
% Apply U boundary conditions to the system.
%-----------------------------------------------------------------

bu_bc = zeros(I+1,J+2);

% Since du/dn = 0 bndry require bu_bc = 0, nothing needs to be done

%  for that b.c.

% For Essential b.c.'s set main diag = 1, zero the other coeffs in
%  that row and set bu_bc = b.c.

j = 1;
for i = 1:I+1
  A_u(i,:) = [0 0 1 0 0];
% bu_bc(i,j) = 0.0; % not required
end

i = I+1;
for j = 1:J+2
  k = i + (j-1)*(I+1);
  A_u(k,:) = [0 0 1 0 0];
% bu_bc(i,j) = 0.0;
end

%-----------------------------------------------------------------
% Apply V boundary conditions to the system.
%-----------------------------------------------------------------

bv_bc = zeros(I+2,J+1);

% w and n bndry's require nothing.

% s boundary.

j = 1;
```

```
for i = 1:I+2
  A_v(i,:) = [0 0 1 0 0];
  bv_bc(i,j) = vbcs(i);
end

% e bndry: must change a term in A_v. b is not changed.
% The b.c. is v0 = log_const * v1.

i = I+2;
for j = 2:J+1
  k = i + (j-1)*(I+2);
  A_v(k,2) = log_const * A_v(k,2);
end

% Modify A_v to account for log_wall b.c.'s on top of blocked
control volumes.

for q = 1:j_pos_len
  i = i_pos(q);
  j = j_pos(q);
  k = i+(j-1)*(I+2);
    A_v(k,3) = A_v(k,3) - log_const * (- A_v(k,4));
    A_v(k,4) = 0.0;
    A_v(k+(I+2),3) = A_v(k+(I+2),3) - log_const * ...
                                            (- A_v(k+(I+2),4));
    A_v(k+(I+2),4) = 0.0;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOLV_K.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-------------------------------------------------------------------
% Note that since sigma_k = 1.0, gamma_k = gamma, where gamma is
%   the turbulent viscosity.  Therefore, I will use gamma in place
%   of gamma_k in the k computation.
%-------------------------------------------------------------------

% Compute G1, the kinetic energy production due to shear.

     G1 = SHEAR(ubc,vbc,dx_p,dy_p,i_pos,j_pos,f_e,f_n);

% Compute the surface shear stress.  Also consider the tops of
%   blocked c.v.'s.  Since u is small, Utau_s will be approximated
%   by zero in the presence of a blocked control volume.

     Utau = tau_const * .5 * abs(vbc(I+1,1:J) + vbc(I+1,2:J+1));
     for k = 1:j_pos_len
       Utau(j_pos(k)) = tau_const * .5 * ...
          abs(vbc(i_pos(k),j_pos(k)) + vbc(i_pos(k),j_pos(k)+1));
     end

% Compute source terms for k.

     Sc_k1 = ro * C_mu * G1 .* km.^2 ./ epsm;
     Sp_k1 = - ro * epsm ./ km - ...
          2 * ro * conc / tstar .* (1 - exp(-.5*tstar*epsm ./ km));
     [Sc_k,Sp_k] = SOURCE(Sc_k1,Sp_k1,dx_p,dy_p);
```

```
% CALCULATE COEFFICIENTS.

     [aE,aW,aN,aS] = GE_COEF(ubc,vbc,dx_p,dy_p,delx_p,dely_p, ...
                                gamma_bndry_ew,gamma_bndry_ns,ro);

% Modify the coefficient matrices to account for flux boundary
%   conditions on west and north boundaries, d∂/dn = 0.

     aW(1,:) = 0 * aW(1,:);
     aN(:,J) = 0 * aN(:,J);

% BUILD SOUTH AND EAST BOUNDARY CONDITION MATRICES.

     sbc_k = zeros(I,J);
     sbc_k(:,1) = aS(:,1) .* k_sbc;
     ebc_k = zeros(I,J);
     ebc_k(I,:) = aE(I,:) .* (Utau.^2 / sqrt(C_mu));

% Transfer b.c.'s to account for blocked control volumes.

     aE_blocked = zeros(I,J);
     for k = 1:j_pos_len
       aE_bocked(i_pos(k)-1,j_pos(k)) = aE(i_pos(k)-1,j_pos(k));
       aE(i_pos(k)-1,j_pos(k)) = 0;
       ebc_k(i_pos(k)-1,j_pos(k)) = ...
                           aE_blocked(i_pos(k)-1,j_pos(k)) * ...
                           (Utau(j_pos(k))^2/sqrt(C_mu));
     end

% Set d/dn(k) = 0 bc's on blocked vertical walls.

     for i = 1:I
       for j = J-1:-1:1
         if gamma(i,j) >= 1e3
         if gamma(i,j+1) < 1e3
           aS(i,j+1) = 0;   % d∂/dn = 0.
         end
         end
       end
     end

     for i = 1:I
       for j = 2:J
         if gamma(i,j) >= 1e3
         if gamma(i,j-1) < 1e3
           aN(i,j-1) = 0;   % d∂/dn = 0.
         end
         end
       end
     end

%------------------------------------------------------------------
% BUILD A_k MATRIX
%------------------------------------------------------------------

     aP = ((aE + aE_blocked) + aW + aN + aS - Sp_k) / relax_k;
     A_k = [-aS(:) -aW(:) aP(:) -aE(:) -aN(:)];

%------------------------------------------------------------------
```

```
%    SOLVE THE SYSTEM OF EQUATIONS FOR k
%-------------------------------------------------------------------

     b_k = sbc_k + ebc_k + Sc_k + (1 - relax_k) * aP .* km_old;
     disp('          solve k equation')
     km = THOMAS(A_k,COLINDX_P,b_k,km_old,TOL_vel, ...
                                    max_iter_vel,flag_vel);

     if min(min(km)) < 0
       disp(' Negative Values of k Found ')
     end

     km_old = km;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOLV_EPS.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%-------------------------------------------------------------------
% SOLVE FOR EPSILON.  Note that G1 has already been computed in
%   SHEAR.  Also, since we no longer need gamma, I will set
%   gamma_eps = gamma.
%-------------------------------------------------------------------

% Compute source terms for epsilon.

     Sc_eps1 = ro * (C_mu * C1_eps * G1 .* km + ...
                             C2_eps * epsm.^2 ./ km);
     Sp_eps1 = - 2 * ro * C2_eps * epsm ./ km - ...
                              2 * ro * conc / tstar;
     [Sc_eps,Sp_eps] = SOURCE(Sc_eps1,Sp_eps1,dx_p,dy_p);

% Compute gamma_eps and compute gamma_bndry's.

     gamma = gamma / sigma_eps;
     [gamma_bndry_ew,gamma_bndry_ns] = GAMMA1(gamma,f_e,f_n);

% CALCULATE COEFFICIENTS

     [aE,aW,aN,aS] = GE_COEF(ubc,vbc,dx_p,dy_p,delx_p,dely_p, ...
                        gamma_bndry_ew,gamma_bndry_ns,ro);

% Modify the coefficient matrices to account for flux boundary
%   conditions on west and north boundaries, d@/dn = 0.

     aW(1,:) = 0 * aW(1,:);
     aN(:,J) = 0 * aN(:,J);

% BUILD SOUTH AND EAST BOUNDARY CONDITION MATRICES

     sbc_eps = zeros(I,J);
     sbc_eps(:,1) = aS(:,1) .*  eps_sbc;
     ebc_eps = zeros(I,J);
     ebc_eps(I,:) = aE(I,:) .* (Utau.^3 / (kappa * comp_bndry));

% Transfer b.c.'s to account for blocked control volumes.

     aE_blocked = zeros(I,J);
     for k = 1:j_pos_len
```

```
       aE_bocked(i_pos(k)-1,j_pos(k)) = aE(i_pos(k)-1,j_pos(k));
       aE(i_pos(k)-1,j_pos(k)) = 0;
       ebc_eps(i_pos(k)-1,j_pos(k)) = ...
                         aE_blocked(i_pos(k)-1,j_pos(k)) * ...
                         (Utau(j_pos(k))^3 / (kappa * comp_bndry));
     end

% Set d/dn(eps) = 0 bc's on blocked vertical walls.

     for i = 1:I
       for j = J-1:-1:1
         if gamma(i,j) >= 1e3
         if gamma(i,j+1) < 1e3
           aS(i,j+1) = 0;   % d∂/dn = 0.
         end
         end
       end
     end

     for i = 1:I
       for j = 2:J
         if gamma(i,j) >= 1e3
         if gamma(i,j-1) < 1e3
           aN(i,j-1) = 0;   % d∂/dn = 0.
         end
         end
       end
     end

%-----------------------------------------------------------------------
% BUILD A_eps MATRIX
%-----------------------------------------------------------------------

     aP = ((aE + aE_blocked) + aW + aN + aS - Sp_eps) / relax_eps;
     A_eps = [-aS(:) -aW(:) aP(:) -aE(:) -aN(:)];

% SOLVE THE SYSTEM OF EQUATIONS FOR espilon.

     b_eps = sbc_eps + ebc_eps + Sc_eps + (1 - relax_eps) * ...
                                                  aP .* epsm_old;
     disp('        solve eps equation')
     epsm = THOMAS(A_eps,COLINDX_P,b_eps,epsm_old,TOL_vel, ...
                                          max_iter_vel,flag_vel);

     if min(min(epsm)) < 0
       disp(' Negative Values of epsilon Found ')
     end

     epsm_old = epsm;

% Compute the residual for the k and epsilon systems of equations.

     RESIDUAL

% Compute gamma = turbulent viscosity using new k and epsilon.

     gamma = ro * C_mu * km.^2 ./ epsm + ro * k_visc;
     gamma = relax_gamma * gamma + (1 - relax_gamma) * gamma_old;
```

```
% RESET gamma = 5e30 IN BLOCKED OFF REGIONS.
     for k = 1:j_pos_len
       gamma(i_pos(k):I,j_pos(k)) = 5e30 + 0 * ...
                                          gamma(i_pos(k):I,j_pos(k));
     end
     gamma_old = gamma;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% RESIDUAL.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%------------------------------------------------------------------
% COMPUTE THE RESIDUAL FOR THE SYSTEMS k AND epsilon.
%------------------------------------------------------------------

     AtimesK = ATIMESX(km,A_k,COLINDX_P,relax_k);
     R_k = AtimesK - sbc_k - ebc_k - Sc_k;

     AtimesEps = ATIMESX(epsm,A_eps,COLINDX_P,relax_eps);
     R_eps = AtimesEps - sbc_eps - ebc_eps - Sc_eps;

% Elliminate blocked regions from consideration.

     for q = 1:j_pos_len
       R_k(i_pos(q):I,j_pos(q)) = 0 * R_k(i_pos(q):I,j_pos(q));
       R_eps(i_pos(q):I,j_pos(q)) = 0 * R_eps(i_pos(q):I,j_pos(q));
     end

     R_keps = [R_keps [max(max(abs(R_k)));max(max(abs(R_eps)))]];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SOLV_CON.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

     ubc_conc = Ufall + ubc;

% CALCULATE COEFFICIENTS

     [gamma_bndry_ew,gamma_bndry_ns] = ...
                               GAMMA1(gamma/Sc_num,f_e,f_n);
     [aEc,aWc,aNc,aSc] = GE_COEF(ubc_conc,vbc,dx_p,dy_p, ...
                    delx_p,dely_p,gamma_bndry_ew,gamma_bndry_ns,ro);

% Set B.C.'s for w.

     conc_sbc = conc_const * ones(I,1);
     conc_wbc = conc_const * ones(1,J);

% BUILD SOUTH AND WEST BOUNDARY CONDITION MATRICES

     sbc_conc = zeros(I,J);
     sbc_conc(:,1) = aSc(:,1) .* conc_sbc;
     wbc_conc = zeros(I,J);
     wbc_conc(1,:) = aWc(1,:) .* conc_wbc;

% Modify the coefficient matrices to account for flux boundary
%   conditions on N and E boundaries, d@/dn = 0.

     aNc(:,J) = 0 * aNc(:,J);
     % The following will help impose a zero conc bc at this point
     %   on the bndry.  It also preserves the blocked out status on
     %   the boundary.

     for j = 1:J
       if aEc(I,j) < 1e5
         aEc(I,j) = 0.0;
       end
     end

% Impose w = 0 bc's on blocked vertical walls.

     %  Set d@/dn = 0 on west (top) of blocks.
     for k = 1:j_pos_len
       aEc(i_pos(k)-1,j_pos(k)) = 0;
     end

%-----------------------------------------------------------------
% BUILD A_conc MATRIX

     aPc = aEc + aWc + aNc + aSc;
     A_conc = [-aSc(:) -aWc(:) aPc(:) -aEc(:) -aNc(:)];

%-----------------------------------------------------------------
%    SOLVE THE SYSTEM OF EQUATIONS FOR W

     b_conc = sbc_conc + wbc_conc;
     disp('   solve concentration equation')
     conc = THOMAS(A_conc,COLINDX_P,b_conc,conc,TOL_vel, ...
                                           max_iter_vel,flag_vel);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DRIFT.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This program is used to compute the snowdrift evolution.
% Import the initial flowfield solution.

     load testcase

% Start developing snow drifts.

     Utau_threshold = .2;

% Compute the surface shear stress to be used in snow drift
%   development computation.  The smoothing constant is used to
%   compensate for the step changes in surface produced by blocking
%   control volumes.  It assumes a logarithmic profile from the
%   velocity computed at the second control volume above the
%   surface, down to the surface.  The factor 2 is added to
%   optimize the relationship between the representative values of
%   the surface shear stress, the threshold shear stress, and the
%   size of the equilibrum snow drift.

     smoothing_const = 2 * log(x_coords(I+1)/rough_length)/ ...
                       log(x_coords(I)/rough_length);

     Utau_drift = tau_const * (smoothing_const * abs(v(I-1,:)));
```

```
% Compute the shear stress at the top of the blocks.

     for k = 1:j_pos_len
       Utau_drift(1,j_pos(k)) = tau_const * smoothing_const * ...
                           abs(v(i_pos(k)-2,j_pos(k)));
     end

% Locate the first cell where Utau is below Utau_threshold.

     for j = 1:J
       if Utau_drift(j) < Utau_threshold
         jj = j;
         break
       end
     end

% SET INITIAL VELOCITY FIELDS.

     ubc = zeros(I+1,J+2);
     for j = 1:J+1
       vbc(:,j) = vbcs;
     end
     ustar = ubc;
     vstar = vbc;

% SET INITIAL TURBULENT VISCOSITY FIELDS.

     for j = 1:J
       km(:,j) = k_sbc;
       epsm(:,j) = eps_sbc;
       gamma(:,j) = gamma_inflow;
     end

     km_old = km;
     epsm_old = epsm;

% Set gamma = large for newly blocked off control volume and reset
%   existing blocked off regions to high gamma values.

     for k = 1:j_pos_len
       gamma(i_pos(k):I,j_pos(k)) = 5e30 + 0 * ...
                                    gamma(i_pos(k):I,j_pos(k));
         if j_pos(k) == jj
           gamma(i_pos(k)-1,jj) = 5e30;
         end
     end
     gamma(I,jj) = 5e30;

% Recompute the flow field.

     BLOCKED
     max_iteration = 30;
     cont = [];

for iteration = 1:max_iteration
     SOLV_VEL
     SOLV_K
     SOLV_EPS
end
```

```
%---------------------------------------------------------------------
% POST PROCESS THE DATA
%---------------------------------------------------------------------

% COMPUTE VELOCITY VALUES AT PRESSURE GRID POINTS.

     u = .5 * (ubc(2:I+1,2:J+1) + ubc(1:I,2:J+1));
     v = .5 * (vbc(2:I+1,2:J+1) + vbc(2:I+1,1:J));

% Set grid point velocity values to zero in blocked off regions.

     for k = 1:j_pos_len
       gamma(i_pos(k):I,j_pos(k)) = nan + 0 * ...
                                    gamma(i_pos(k):I,j_pos(k));
     end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 19. Fortran Programs.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c CV_INFO.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
        subroutine CV_INFO(delx_p,dely_p,f_e,f_n,x_crds,y_crds,
     +             x_wall,y_wall,dx_p,dy_p,cmp_bn,II,JJ)
c
        integer*4 II,JJ
        real*8 dx_pbc(62),dy_pbc(62),delx_p(II+1),dely_p(JJ+1)
        real*8 f_e(II+1),f_n(JJ+1),dx_p(II),dy_p(JJ),cmp_bn,temp
        real*8 x_crds(II+2),y_crds(JJ+2),x_wal2(61)
        real*8 x_wall(II+1),y_wall(JJ+1)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c PRESSURE CONTROL VOLUME SIZE AND POSITION INFORMATION
c
c   Include exterior boundary pressure grid points.
c
        dx_pbc(1) = 0.0
        do 10 i = 2,II+1
          dx_pbc(i) = dx_p(i-1)
  10    continue
        dx_pbc(II+2) = 0.0
c
        dy_pbc(1) = 0.0
        do 15 j = 2,JJ+1
          dy_pbc(j) = dy_p(j-1)
  15    continue
        dy_pbc(JJ+2) = 0.0
c
c Compute the distance between pressure grid points.
c
        do 20 i = 1,II+1
          delx_p(i) = .5 * (dx_pbc(i) + dx_pbc(i+1))
  20    continue
        do 25 j = 1,JJ+1
          dely_p(j) = .5 * (dy_pbc(j) + dy_pbc(j+1))
  25    continue
c
c Compute the distance between the pressure grid points and the
c   control volume wall.  (The following is true because the grid
c   points do pressure are defined to be in the center of the
c   control volume.)  And then compute f_e and f_n.  These two
c   steps are combined below.
c
        do 30 i = 1,II+1
          f_e(i) = .5 * dx_pbc(i+1) / delx_p(i)
  30    continue
        do 35 j = 1,JJ+1
          f_n(j) = .5 * dy_pbc(j+1) / dely_p(j)
  35    continue
c
c Compute the x and y coordinates of the pressure c.v. grid points,
c   including boundaries.
c
```

```
        temp = cmp_bn
        do 40 i = II+2,1,-1
          x_crds(i) = temp + .5 * dx_pbc(i)
          temp = temp + dx_pbc(i)
  40    continue
c
        temp = 0.0
        do 45 j = 1,JJ+2
          y_crds(j) = temp + .5 * dy_pbc(j)
          temp = temp + dy_pbc(j)
  45    continue
c
c Compute the x and y coordinates of the pressure c.v. walls.
c
        x_wal2(1) = cmp_bn
        x_wall(II+1) = x_wal2(1)
        do 50 i = 2,II+1
          x_wal2(i) = x_wal2(i-1) + dx_p(II-i+2)
          x_wall(II-i+2) = x_wal2(i)
  50    continue
c
        y_wall(1) = 0.0
        do 55 j = 2,JJ+1
          y_wall(j) = y_wall(j-1) + dy_p(j-1)
  55    continue
c
       return
       end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c CV_INFOG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 delxP,delyP,f_eP,f_nP,x_crdP,y_crdP
      INTEGER*4 x_walP,y_walP,dx_pP,dy_pP,cmpbnP
c
c  Declare local variables.
c
      INTEGER*4 I, J, dummy
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(1),dummy,I)
      CALL GETSIZ(PRHS(2),dummy,J)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(1,I+1,0)
      PLHS(2) = CRTMAT(1,J+1,0)
      PLHS(3) = CRTMAT(1,I+1,0)
      PLHS(4) = CRTMAT(1,J+1,0)
      PLHS(5) = CRTMAT(1,I+2,0)
```

```
         PLHS(6) = CRTMAT(1,J+2,0)
         PLHS(7) = CRTMAT(1,I+1,0)
         PLHS(8) = CRTMAT(1,J+1,0)
c
c  Assign pointers to the various parameters.
c
        delxP = REALP(PLHS(1))
        delyP = REALP(PLHS(2))
        f_eP = REALP(PLHS(3))
        f_nP = REALP(PLHS(4))
        x_crdP = REALP(PLHS(5))
        y_crdP = REALP(PLHS(6))
        x_walP = REALP(PLHS(7))
        y_walP = REALP(PLHS(8))
c
        dx_pP = REALP(PRHS(1))
        dy_pP = REALP(PRHS(2))
        cmpbnP = REALP(PRHS(3))
c
c  Do the actual computation in a subroutine.
c
        CALL CV_INFO(%VAL(delxP),%VAL(delyP),%VAL(f_eP),%VAL(f_nP),
     +        %VAL(x_crdP),%VAL(y_crdP),%VAL(x_walP),%VAL(y_walP),
     +        %VAL(dx_pP),%VAL(dy_pP),%VAL(cmpbnP),I,J)
        RETURN
        END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c GAMMA1.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
        subroutine GAMMA1(g_b_ew,g_b_ns,gamma,f_e,f_n,II,JJ)
c
        integer*4 II,JJ
        real*8 g_b_ew(II+1,JJ),g_b_ns(II,JJ+1),gamma(II,JJ)
        real*8 g_ew(60+2,60),g_ns(60,60+2),f_e(II+1),f_n(JJ+1)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c  GAMMA INTERPOLATIONS FOR u, v, k, and epsilon COMPUTATIONS.
c    This provides gamma information on c.v. walls.
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c    Gamma must be input
c
c    Example:   gamma = [30 30 30
c                        30 30 30
c                        10 30 30]
c
c    Include gamma just outside of n, s boundaries.
c    Include gamma just outside of e, w boundaries.
c
      do 10 i = 1,II
        g_ns(i,1) = gamma(i,1)
          do 20 j = 2,JJ+1
            g_ns(i,j) = gamma(i,j-1)
 20          continue
        g_ns(i,JJ+2) = gamma(i,JJ)
 10       continue
c
```

```
      do 30 j = 1,JJ
        g_ew(1,j) = gamma(1,j)
          do 40 i = 2,II+1
            g_ew(i,j) = gamma(i-1,j)
  40          continue
        g_ew(II+2,j) = gamma(II,j)
  30      continue
c
c     Compute gamma (diffusion coefficient) at the n, s control
c     volume boundaries using equation 4.9, p. 45.
c
        do 50 i = 1,II
            do 60 j = 1,JJ+1
                 g_b_ns(i,j) = 1 / ((1 - f_n(j)) /
     +           g_ns(i,j) + f_n(j) / g_ns(i,j+1))
  60        continue
  50    continue
c
c     Compute gamma (diffusion coefficient) at the e, w control
c     volume boundaries using equation 4.9, p. 45.
c
        do 70 j = 1,JJ
          do 80 i = 1,II+1
                 g_b_ew(i,j) = 1 / ((1 - f_e(i)) /
     +           g_ew(i,j) + f_e(i) / g_ew(i+1,j))
  80        continue
  70    continue
c
        return
        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c GAMMA1G.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 gb_ewP,gb_nsP,gammaP,f_eP,f_nP
c
c  Declare local variables.
c
      INTEGER*4 I, J
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(1),I,J)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(I+1,J,0)
      PLHS(2) = CRTMAT(I,J+1,0)
c
c  Assign pointers to the various parameters.
c
```

```
      gb_ewP = REALP(PLHS(1))
      gb_nsP = REALP(PLHS(2))
c
      gammaP = REALP(PRHS(1))
      f_eP = REALP(PRHS(2))
      f_nP = REALP(PRHS(3))
c
c   Do the actual computation in a subroutine.
c
      CALL GAMMA1(%VAL(gb_ewP),%VAL(gb_nsP),%VAL(gammaP),
     +             %VAL(f_eP),%VAL(f_nP),I,J)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c GAMMA2.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
        subroutine GAMMA2(g_cnrs,g_b_ns,f_e,II,JJ)
c
        integer*4 II,JJ
        real*8 g_cnrs(II+1,JJ+1),g_b_ns(II,JJ+1)
        real*8 f_e(II+1),g_nsew(60+2,60+1)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c         GAMMA INTERPOLATIONS FOR u AND v COMPUTATIONS
c    This provides gamma at c.v. corners to be used in u, v
c    computations.
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c    Include e, w boundary control for gamma computed on the n, s
c    control volume boundaries.
c
     do 10 j = 1,JJ+1
       g_nsew(1,j) = g_b_ns(1,j)
         do 20 i = 2,II+1
           g_nsew(i,j) = g_b_ns(i-1,j)
  20        continue
       g_nsew(II+2,j) = g_b_ns(II,j)
  10    continue
c
c    Compute the value of gamma at the corners of the pressure
c    control volumes.
c
       do 30 j = 1,JJ+1
          do 40 i = 1,II+1
                g_cnrs(i,j) = 1 / ((1 - f_e(i)) / g_nsew(i,j) +
     +                                    f_e(i) / g_nsew(i+1,j))
  40      continue
  30   continue
c
       return
       end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c GAMMA2G.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 g_cnrsP,gb_nsP,f_eP
c
c  Declare local variables.
c
      INTEGER*4 I, J
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(1),I,J)
c
      J = J-1
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(I+1,J+1,0)
c
c  Assign pointers to the various parameters.
c
      g_cnrsP = REALP(PLHS(1))
c
      gb_nsP = REALP(PRHS(1))
      f_eP = REALP(PRHS(2))
c
c  Do the actual computation in a subroutine.
c
      CALL GAMMA2(%VAL(g_cnrsP),%VAL(gb_nsP),%VAL(f_eP),I,J)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c UV_COEF.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine UV_COEF(A_u,A_v,ubc,vbc,dx_p,dy_p,delx_p,dely_p,
     +                   f_e,f_n,gamma,g_cnrs,ro,II,JJ)
c
      integer*4 II, JJ
      real*8 A_u((II+1)*(JJ+2),5),A_v((II+2)*(JJ+1),5)
      real*8 ubc(II+1,JJ+2),vbc(II+2,JJ+1),dx_p(II),dy_p(JJ)
      real*8 delx_p(II+1),dely_p(JJ+1),f_e(II+1),f_n(JJ+1)
      real*8 gamma(II,JJ),g_cnrs(II+1,JJ+1),ro
      real*8 ue,uw,vn,vs,De,Dw,Dn,Ds,pe,pw,pn,ps,fe,fw,fn,fs,zero
c
      zero = 0.0
c
c  CALCULATE THE COEFFICIENTS AU (p.99, 121)
c
          j = 1
          do 21 i = 1,II+1
c
          k = i + (j-1) * (II+1)
c
               vn = f_e(i) * vbc(i,j) + (1-f_e(i)) * vbc(i+1,j)
```

```
         Dn = g_cnrs(i,j)    * delx_p(i) / dely_p(j)
         fn = ro * delx_p(i) * vn
         pn = fn / Dn
         pn = (1. - .1 * dabs(pn))**5
c
    A_u(k,4) = 0.0
    A_u(k,2) = 0.0
    A_u(k,5) = -(Dn * dmax1(zero,pn) + dmax1(-fn,zero))
    A_u(k,1) = 0.0
    A_u(k,3) = - A_u(k,5)
c
 21        continue
c
c
      j = JJ+2
      do 22 i = 1,II+1
c
      k = i + (j-1) * (II+1)
c
         vs = f_e(i) * vbc(i,j-1) + (1-f_e(i)) * vbc(i+1,j-1)
         Ds = g_cnrs(i,j-1)  * delx_p(i) / dely_p(j-1)
         fs = ro * delx_p(i) * vs
         ps = fs / Ds
         ps = (1. - .1 * dabs(ps))**5
c
    A_u(k,4) = 0.0
    A_u(k,2) = 0.0
    A_u(k,5) = 0.0
    A_u(k,1) = -(Ds * dmax1(zero,ps) + dmax1(fs,zero))
    A_u(k,3) = - A_u(k,1)
c
 22        continue
c
c
      i = 1
      do 23 j = 2,JJ+1
c
      k = i + (j-1) * (II+1)
c
         ue = .5 * (ubc(i,j)+ubc(i+1,j))
         vn = f_e(i) * vbc(i,j) + (1-f_e(i)) * vbc(i+1,j)
         vs = f_e(i) * vbc(i,j-1) + (1-f_e(i)) * vbc(i+1,j-1)
c
         De = gamma(i,j-1)   * dy_p(j-1) / dx_p(i)
         Dn = g_cnrs(i,j)    * delx_p(i) / dely_p(j)
         Ds = g_cnrs(i,j-1)  * delx_p(i) / dely_p(j-1)
c
         fe = ro * dy_p(j-1) * ue
         fn = ro * delx_p(i) * vn
         fs = ro * delx_p(i) * vs
c
         pe = fe / De
         pn = fn / Dn
         ps = fs / Ds
c
         pe = (1. - .1 * dabs(pe))**5
         pn = (1. - .1 * dabs(pn))**5
         ps = (1. - .1 * dabs(ps))**5
c
```

```
       A_u(k,4) = -(De * dmax1(zero,pe) + dmax1(-fe,zero))
       A_u(k,2) = 0.0
       A_u(k,5) = -(Dn * dmax1(zero,pn) + dmax1(-fn,zero))
       A_u(k,1) = -(Ds * dmax1(zero,ps) + dmax1(fs,zero))
       A_u(k,3) = - A_u(k,4) - A_u(k,5) - A_u(k,1)
c
  23              continue
c
c
          i = II+1
          do 24 j = 2,JJ+1
c
          k = i + (j-1) * (II+1)
c
               uw = .5 * (ubc(i-1,j)+ubc(i,j))
               vn = f_e(i) * vbc(i,j) + (1-f_e(i)) * vbc(i+1,j)
               vs = f_e(i) * vbc(i,j-1) + (1-f_e(i)) * vbc(i+1,j-1)
c
               Dw = gamma(i-1,j-1) * dy_p(j-1) / dx_p(i-1)
               Dn = g_cnrs(i,j)    * delx_p(i) / dely_p(j)
               Ds = g_cnrs(i,j-1)  * delx_p(i) / dely_p(j-1)
c
               fw = ro * dy_p(j-1) * uw
               fn = ro * delx_p(i) * vn
               fs = ro * delx_p(i) * vs
c
               pw = fw / Dw
               pn = fn / Dn
               ps = fs / Ds
c
               pw = (1. - .1 * dabs(pw))**5
               pn = (1. - .1 * dabs(pn))**5
               ps = (1. - .1 * dabs(ps))**5
c
       A_u(k,4) = 0.0
       A_u(k,2) = -(Dw * dmax1(zero,pw) + dmax1(fw,zero))
       A_u(k,5) = -(Dn * dmax1(zero,pn) + dmax1(-fn,zero))
       A_u(k,1) = -(Ds * dmax1(zero,ps) + dmax1(fs,zero))
       A_u(k,3) = - A_u(k,2) - A_u(k,5) - A_u(k,1)
c
  24              continue
c
c
     do 10 j = 2,JJ+1
          do 20 i = 2,II
c
          k = i + (j-1) * (II+1)
c
               ue = .5 * (ubc(i,j)+ubc(i+1,j))
               uw = .5 * (ubc(i-1,j)+ubc(i,j))
               vn = f_e(i) * vbc(i,j) + (1-f_e(i)) * vbc(i+1,j)
               vs = f_e(i) * vbc(i,j-1) + (1-f_e(i)) * vbc(i+1,j-1)
c
               De = gamma(i,j-1)   * dy_p(j-1) / dx_p(i)
               Dw = gamma(i-1,j-1) * dy_p(j-1) / dx_p(i-1)
               Dn = g_cnrs(i,j)    * delx_p(i) / dely_p(j)
               Ds = g_cnrs(i,j-1)  * delx_p(i) / dely_p(j-1)
c
               fe = ro * dy_p(j-1) * ue
               fw = ro * dy_p(j-1) * uw
               fn = ro * delx_p(i) * vn
               fs = ro * delx_p(i) * vs
c
               pe = fe / De
               pw = fw / Dw
               pn = fn / Dn
               ps = fs / Ds
c
               pe = (1. - .1 * dabs(pe))**5
               pw = (1. - .1 * dabs(pw))**5
               pn = (1. - .1 * dabs(pn))**5
               ps = (1. - .1 * dabs(ps))**5
c
      A_u(k,4) = -(De * dmax1(zero,pe) + dmax1(-fe,zero))
      A_u(k,2) = -(Dw * dmax1(zero,pw) + dmax1(fw,zero))
      A_u(k,5) = -(Dn * dmax1(zero,pn) + dmax1(-fn,zero))
      A_u(k,1) = -(Ds * dmax1(zero,ps) + dmax1(fs,zero))
      A_u(k,3) = - A_u(k,4) - A_u(k,2) - A_u(k,5) - A_u(k,1)
c
  20          continue
  10     continue
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c  CALCULATE THE COEFFICIENTS OF AV (p.99, 121)
c
          i = 1
          do 51 j = 1,JJ+1
c
          k = i + (j-1) * (II+2)
c
               ue = f_n(j) * ubc(i,j) + (1-f_n(j)) * ubc(i,j+1)
               De = g_cnrs(i,j)     * dely_p(j) / delx_p(i)
               fe = ro * dely_p(j) * ue
               pe = fe / De
               pe = (1. - .1 * dabs(pe))**5
c
      A_v(k,4) = -(De * dmax1(zero,pe) + dmax1(-fe,zero))
      A_v(k,2) = 0.0
      A_v(k,5) = 0.0
      A_v(k,1) = 0.0
      A_v(k,3) = - A_v(k,4)
c
  51           continue
c
c
          i = II+2
          do 52 j = 1,JJ+1
c
          k = i + (j-1) * (II+2)
c
               uw = f_n(j) * ubc(i-1,j) + (1-f_n(j)) * ubc(i-1,j+1)
               Dw = g_cnrs(i-1,j)  * dely_p(j) / delx_p(i-1)
               fw = ro * dely_p(j) * uw
               pw = fw / Dw
               pw = (1. - .1 * dabs(pw))**5
c
      A_v(k,4) = 0.0
```

```
       A_v(k,2) = -(Dw * dmax1(zero,pw) + dmax1(fw,zero))
       A_v(k,5) = 0.0
       A_v(k,1) = 0.0
       A_v(k,3) = - A_v(k,2)
c
 52             continue
c
c
          j = 1
          do 53 i = 2,II+1
c
          k = i + (j-1) * (II+2)
c
               ue = f_n(j) * ubc(i,j) + (1-f_n(j)) * ubc(i,j+1)
               uw = f_n(j) * ubc(i-1,j) + (1-f_n(j)) * ubc(i-1,j+1)
               vn = .5 * (vbc(i,j) + vbc(i,j+1))
c
               De = g_cnrs(i,j)    * dely_p(j) / delx_p(i)
               Dw = g_cnrs(i-1,j)  * dely_p(j) / delx_p(i-1)
               Dn = gamma(i-1,j)   * dx_p(i-1) / dy_p(j)
c
               fe = ro * dely_p(j) * ue
               fw = ro * dely_p(j) * uw
               fn = ro * dx_p(i-1) * vn
c
               pe = fe / De
               pw = fw / Dw
               pn = fn / Dn
c
               pe = (1. - .1 * dabs(pe))**5
               pw = (1. - .1 * dabs(pw))**5
               pn = (1. - .1 * dabs(pn))**5
c
       A_v(k,4) = -(De * dmax1(zero,pe) + dmax1(-fe,zero))
       A_v(k,2) = -(Dw * dmax1(zero,pw) + dmax1(fw,zero))
       A_v(k,5) = -(Dn * dmax1(zero,pn) + dmax1(-fn,zero))
       A_v(k,1) = 0.0
       A_v(k,3) = - A_v(k,4) - A_v(k,2) - A_v(k,5)
c
 53             continue
c
c
          j = JJ + 1
          do 54 i = 2,II+1
c
          k = i + (j-1) * (II+2)
c
               ue = f_n(j) * ubc(i,j) + (1-f_n(j)) * ubc(i,j+1)
               uw = f_n(j) * ubc(i-1,j) + (1-f_n(j)) * ubc(i-1,j+1)
               vs = .5 * (vbc(i,j) + vbc(i,j-1))
c
               De = g_cnrs(i,j)    * dely_p(j) / delx_p(i)
               Dw = g_cnrs(i-1,j)  * dely_p(j) / delx_p(i-1)
               Ds = gamma(i-1,j-1) * dx_p(i-1) / dy_p(j-1)
c
               fe = ro * dely_p(j) * ue
               fw = ro * dely_p(j) * uw
               fs = ro * dx_p(i-1) * vs
c
```

```
         pe = fe / De
         pw = fw / Dw
         ps = fs / Ds
c
         pe = (1. - .1 * dabs(pe))**5
         pw = (1. - .1 * dabs(pw))**5
         ps = (1. - .1 * dabs(ps))**5
c
    A_v(k,4) = -(De * dmax1(zero,pe) + dmax1(-fe,zero))
    A_v(k,2) = -(Dw * dmax1(zero,pw) + dmax1(fw,zero))
    A_v(k,5) = 0.0
    A_v(k,1) = -(Ds * dmax1(zero,ps) + dmax1(fs,zero))
    A_v(k,3) = - A_v(k,4) - A_v(k,2) - A_v(k,1)
c
 54      continue
c
c
   do 40 j = 2,JJ
      do 50 i = 2,II+1
c
      k = i + (j-1) * (II+2)
c
         ue = f_n(j) * ubc(i,j) + (1-f_n(j)) * ubc(i,j+1)
         uw = f_n(j) * ubc(i-1,j) + (1-f_n(j)) * ubc(i-1,j+1)
         vn = .5 * (vbc(i,j) + vbc(i,j+1))
         vs = .5 * (vbc(i,j) + vbc(i,j-1))
c
         De = g_cnrs(i,j)     * dely_p(j) / delx_p(i)
         Dw = g_cnrs(i-1,j)   * dely_p(j) / delx_p(i-1)
         Dn = gamma(i-1,j)    * dx_p(i-1) / dy_p(j)
         Ds = gamma(i-1,j-1)  * dx_p(i-1) / dy_p(j-1)
c
         fe = ro * dely_p(j) * ue
         fw = ro * dely_p(j) * uw
         fn = ro * dx_p(i-1) * vn
         fs = ro * dx_p(i-1) * vs
c
         pe = fe / De
         pw = fw / Dw
         pn = fn / Dn
         ps = fs / Ds
c
         pe = (1. - .1 * dabs(pe))**5
         pw = (1. - .1 * dabs(pw))**5
         pn = (1. - .1 * dabs(pn))**5
         ps = (1. - .1 * dabs(ps))**5
c
    A_v(k,4) = -(De * dmax1(zero,pe) + dmax1(-fe,zero))
    A_v(k,2) = -(Dw * dmax1(zero,pw) + dmax1(fw,zero))
    A_v(k,5) = -(Dn * dmax1(zero,pn) + dmax1(-fn,zero))
    A_v(k,1) = -(Ds * dmax1(zero,ps) + dmax1(fs,zero))
    A_v(k,3) = - A_v(k,4) - A_v(k,2) - A_v(k,5) - A_v(k,1)
c
 50         continue
 40     continue
c
   return
   end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c UV_COEFG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 A_uP,A_vP,ubcP,vbcP,dx_pP,dy_pP,dlx_pP,dly_pP
      INTEGER*4 f_eP,f_nP,gammaP,g_cnrP,roP
c
c  Declare local variables.
c
      INTEGER*4 I, J
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(9),I,J)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT((I+1)*(J+2),5,0)
      PLHS(2) = CRTMAT((I+2)*(J+1),5,0)
c
c  Assign pointers to the various parameters.
c
      A_uP = REALP(PLHS(1))
      A_vP = REALP(PLHS(2))
c
      ubcP = REALP(PRHS(1))
      vbcP = REALP(PRHS(2))
      dx_pP = REALP(PRHS(3))
      dy_pP = REALP(PRHS(4))
      dlx_pP = REALP(PRHS(5))
      dly_pP = REALP(PRHS(6))
      f_eP = REALP(PRHS(7))
      f_nP = REALP(PRHS(8))
      gammaP = REALP(PRHS(9))
      g_cnrP = REALP(PRHS(10))
      roP = REALP(PRHS(11))
c
c  Do the actual computation in a subroutine.
c
      CALL UV_COEF(%VAL(A_uP),%VAL(A_vP),
     +             %VAL(ubcP),%VAL(vbcP),%VAL(dx_pP),%VAL(dy_pP),
     +             %VAL(dlx_pP),%VAL(dly_pP),%VAL(f_eP),%VAL(f_nP),
     +             %VAL(gammaP),%VAL(g_cnrP),%VAL(roP),I,J)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c UV_HAT.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine UV_HAT(uhat,vhat,A_u_mn,A_v_mn,sum_au,sum_av,
     +    ubc,vbc,A_u,A_v,IND_U,IND_V,bu_bc,bv_bc,Iu,Ju,Iv,Jv)
```

```
c
      integer*4 Iu,Ju,Iv,Jv,INDX_U(1200,5),INDX_V(1200,5)
      real*8 uhat(Iu,Ju),vhat(Iv,Jv),ubc(Iu,Ju),vbc(Iv,Jv)
      real*8 A_u(Iu*Ju,5),A_v(Iv*Jv,5),bu_bc(Iu,Ju),bv_bc(Iv,Jv)
      real*8 IND_U(Iu*Ju,5),IND_V(Iv*Jv,5),ubc2(1200),vbc2(1200)
      real*8 A_u_mn(Iu,Ju),A_v_mn(Iv,Jv),sum_au(Iu,Ju)
      real*8 sum_av(Iv,Jv)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      do 2 j = 1,5
        do 3 i = 1,Iu*Ju
          INDX_U(i,j) = IDNINT(IND_U(i,j))
   3    continue
   2  continue
c
      do 4 j = 1,5
        do 5 i = 1,Iv*Jv
          INDX_V(i,j) = IDNINT(IND_V(i,j))
   5    continue
   4  continue
c
c   Compute uhat.
c
      do 10 j = 1,Ju
        do 15 i = 1,Iu
         k = i+(j-1)*Iu
         ubc2(k) = ubc(i,j)
         A_u_mn(i,j) = A_u(k,3)
  15    continue
  10  continue
c
      do 20 jj = 1,Ju
      do 25 ii = 1,Iu
       sum_au(ii,jj) = 0.0
       k = ii+(jj-1)*Iu
        do 30 j = 1,5
          if (INDX_U(k,j).ne.0) then
          if (j.ne.3) then
          sum_au(ii,jj) = sum_au(ii,jj)-A_u(k,j)*ubc2(INDX_U(k,j))
         endif
         endif
  30        continue
      uhat(ii,jj) = (sum_au(ii,jj) + bu_bc(ii,jj)) / A_u_mn(ii,jj)
  25     continue
  20     continue
c
c   Compute vhat.
c
      do 35 j = 1,Jv
       do 40 i = 1,Iv
         k = i+(j-1)*Iv
         vbc2(k) = vbc(i,j)
         A_v_mn(i,j) = A_v(k,3)
  40   continue
  35  continue
c
      do 45 jj = 1,Jv
      do 50 ii = 1,Iv
      sum_av(ii,jj) = 0.0
      k = ii+(jj-1)*Iv
       do 55 j = 1,5
         if (INDX_V(k,j).ne.0) then
         if (j.ne.3) then
         sum_av(ii,jj) = sum_av(ii,jj)-A_v(k,j)*vbc2(INDX_V(k,j))
         endif
         endif
  55   continue
      vhat(ii,jj) = (sum_av(ii,jj) + bv_bc(ii,jj)) / A_v_mn(ii,jj)
  50  continue
  45  continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c UV_HATG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 uhatP,vhatP,A_umP,A_vmP,ubcP,vbcP,A_uP,A_vP
      INTEGER*4 sum_uP,sum_vP,IND_UP,IND_VP,bu_bcP,bv_bcP
c
c  Declare local variables.
c
      INTEGER*4 Iu, Ju, Iv, Jv
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(1),Iu,Ju)
      CALL GETSIZ(PRHS(2),Iv,Jv)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(Iu,Ju,0)
      PLHS(2) = CRTMAT(Iv,Jv,0)
      PLHS(3) = CRTMAT(Iu,Ju,0)
      PLHS(4) = CRTMAT(Iv,Jv,0)
      PLHS(5) = CRTMAT(Iu,Ju,0)
      PLHS(6) = CRTMAT(Iv,Jv,0)
c
c  Assign pointers to the various parameters.
c
      uhatP = REALP(PLHS(1))
      vhatP = REALP(PLHS(2))
      A_umP = REALP(PLHS(3))
      A_vmP = REALP(PLHS(4))
      sum_uP = REALP(PLHS(5))
      sum_vP = REALP(PLHS(6))
c
      ubcP = REALP(PRHS(1))
      vbcP = REALP(PRHS(2))
      A_uP = REALP(PRHS(3))
      A_vP = REALP(PRHS(4))
      IND_UP = REALP(PRHS(5))
      IND_VP = REALP(PRHS(6))
      bu_bcP = REALP(PRHS(7))
      bv_bcP = REALP(PRHS(8))
c
c  Do the actual computation in a subroutine.
c
      CALL UV_HAT(%VAL(uhatP),%VAL(vhatP),%VAL(A_umP),%VAL(A_vmP),
     +            %VAL(sum_uP),%VAL(sum_vP),%VAL(ubcP),%VAL(vbcP),
     +            %VAL(A_uP),%VAL(A_vP),%VAL(IND_UP),%VAL(IND_VP),
     +            %VAL(bu_bcP),%VAL(bv_bcP),Iu,Ju,Iv,Jv)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c P_COEF.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine P_COEF(A_p,b_p,A_u,A_v,uhat,vhat,dx_p,dy_p,
     +                  ro,II,JJ)
c
      integer*4 II,JJ
      real*8 A_p(II*JJ,5),b_p(II,JJ),A_u((II+1)*(JJ+2),5)
      real*8 A_v((II+2)*(JJ+1),5),dx_p(II),dy_p(JJ)
      real*8 ro,uhat(II+1,JJ+2),vhat(II+2,JJ+1)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c CALCULATE THE COEFFICIENTS OF THE PRESSURE EQUATION (p.125,
c   130, 133)
c
c   For ue and uw we have,
c
      do 10 j = 2,JJ+1
       do 20 i = 2,II
          k = i+(j-1)*(II+1)
          l = (i-1)+(j-2)*II
          A_p(l,4) = - ro * dy_p(j-1)**2 / A_u(k,3)
          A_p(l+1,2) = A_p(l,4)
 20    continue
          A_p((j-1)*II,4) = 0.0
          A_p((j-2)*II+1,2) = 0.0
 10   continue
c
c   For vn and vs we have,
c
      do 30 i = 2,II+1
       do 40 j = 2,JJ
          k = i+(j-1)*(II+2)
          l = (i-1)+(j-2)*II
          A_p(l,5) = - ro * dx_p(i-1)**2 / A_v(k,3)
          A_p(l+II,1) = A_p(l,5)
 40    continue
          A_p(i-1,1) = 0.0
          A_p((JJ-1)*II+(i-1),5) = 0.0
 30   continue
c
```

```
c    Build A_p(:,3).
c
       do 50 i = 1,II*JJ
        A_p(i,3) = - (A_p(i,1) + A_p(i,2) + A_p(i,4) + A_p(i,5))
   50  continue
c
c    Compute b_p from uhat and vhat.
c
       do 60 j = 1,JJ
        do 70 i = 1,II
          b_p(i,j) = ro * dy_p(j) * (uhat(i,j+1) - uhat(i+1,j+1)) +
     +           ro * dx_p(i) * (vhat(i+1,j) - vhat(i+1,j+1))
   70   continue
   60  continue
c
       return
       end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c P_COEFG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
       SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
       INTEGER NLHS,NRHS
       INTEGER*4 PLHS(*), PRHS(*)
       INTEGER*4 CRTMAT, REALP, GETSIZ
c
c   Declare integers to be used as pointers.
c
       INTEGER*4 A_pP,b_pP,A_uP,A_vP,uhatP,vhatP,dx_pP,dy_pP,roP
c
c   Declare local variables.
c
       INTEGER*4 I, J, dummy
c
c   Get size of input argument.
c
       CALL GETSIZ(PRHS(5),dummy,I)
       CALL GETSIZ(PRHS(6),dummy,J)
c
c   Create matrices for return arguments.
c
       PLHS(1) = CRTMAT(I*J,5,0)
       PLHS(2) = CRTMAT(I,J,0)
c
c   Assign pointers to the various parameters.
c
       A_pP = REALP(PLHS(1))
       b_pP = REALP(PLHS(2))
c
       A_uP = REALP(PRHS(1))
       A_vP = REALP(PRHS(2))
       uhatP = REALP(PRHS(3))
       vhatP = REALP(PRHS(4))
       dx_pP = REALP(PRHS(5))
       dy_pP = REALP(PRHS(6))
       roP = REALP(PRHS(7))
c
c   Do the actual computation in a subroutine.
```

```
c
      CALL P_COEF(%VAL(A_pP),%VAL(b_pP),%VAL(A_uP),%VAL(A_vP),
     +            %VAL(uhatP),%VAL(vhatP),%VAL(dx_pP),%VAL(dy_pP),
     +            %VAL(roP),I,J)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c THOMAS.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine THOMAS(xnew,a,CINDX,bmtrx,xguess,TOL,maxit1,
     +   II,JJ,M,flag)
c
      integer*4 II,JJ,M,INDX(900,5),maxit
      real*8 xnew(II,JJ),a(M,5),CINDX(M,5),bmtrx(II,JJ)
      real*8 b(900),x0_old(900),d(900),a_sub(60,3),bd_sub(60)
      real*8 x_sub(60),dif,maxdif,TOL,x0(900),P(60),Q(60),maxit1
      real*8 mdif_p,xguess(II,JJ),mdif_2,flag
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c    This solves a block tridiagonal matrix using the Thomas
c    Algorithm (in both directions).  For A in sparse storage mode.
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      maxit = IDNINT(maxit1)
c
      mdif_2 = 99999.9
c
      do 2 j = 1,5
        do 3 i = 1,M
         INDX(i,j) = IDNINT(CINDX(i,j))
   3    continue
   2  continue
c
      do 4 j = 1,JJ
        do 5 i = 1,II
          x0(i+(j-1)*II) = xguess(i,j)
          b(i+(j-1)*II) = bmtrx(i,j)
   5    continue
   4  continue
c
c    Implement the Algorithm.
c
      l = 1
c
  90  if (l.le.maxit) then
c
      do 6 i = 1,M
        x0_old(i) = x0(i)
  6   continue
c
      do 10 j = 1,JJ
c
      do 15 kk = 1,M
       d(kk) = 0.0
         do 20 ll = 1,5,4
           if (INDX(kk,ll).ne.0) then
           d(kk) = d(kk) + a(kk,ll) * x0(INDX(kk,ll))
           endif
  20     continue
  15  continue
c
      do 25 kk = 1,II
       do 30 ll = 1,3
         a_sub(kk,ll) = a((j-1)*II+kk,ll+1)
  30   continue
  25  continue
c
      do 35 kk = 1,II
        bd_sub(kk) = b((j-1)*II+kk) - d((j-1)*II+kk)
  35  continue
c
      P(1) = - a_sub(1,3) / a_sub(1,2)
      Q(1) = bd_sub(1) / a_sub(1,2)
      a_sub(II,3) = 0.0
c
      do 40 i = 2,II
        P(i) = - a_sub(i,3) / (a_sub(i,2) + a_sub(i,1) * P(i-1))
c
        Q(i) = (bd_sub(i) - a_sub(i,1) * Q(i-1)) /
     +                       (a_sub(i,2) + a_sub(i,1) * P(i-1))
  40  continue
c
      x_sub(II) = Q(II)
c
      do 45 i = II-1,1,-1
        x_sub(i) = P(i) * x_sub(i+1) + Q(i)
  45  continue
c
      do 50 kk = 1,II
        x0((j-1)*II+kk) = x_sub(kk)
  50  continue
c
  10  continue
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      do 11 j = 1,II
c
      do 16 kk = 1,M
       d(kk) = 0.0
         do 21 ll = 2,4,2
           if (INDX(kk,ll).ne.0) then
            d(kk) = d(kk) + a(kk,ll) * x0(INDX(kk,ll))
           endif
  21     continue
  16  continue
c
      do 26 kk = 1,JJ
       do 31 ll = 1,3
         a_sub(kk,ll) = a((kk-1)*II+j,2*ll-1)
  31   continue
  26  continue
c
      do 36 kk = 1,JJ
       bd_sub(kk) = b((kk-1)*II+j) - d((kk-1)*II+j)
  36  continue
```

```
c
      P(1) = - a_sub(1,3) / a_sub(1,2)
      Q(1) = bd_sub(1) / a_sub(1,2)
      a_sub(JJ,3) = 0.0
c
      do 41 i = 2,JJ
        P(i) = - a_sub(i,3) / (a_sub(i,2) + a_sub(i,1) * P(i-1))
c
        Q(i) = (bd_sub(i) - a_sub(i,1) * Q(i-1)) /
     +                       (a_sub(i,2) + a_sub(i,1) * P(i-1))
  41  continue
c
      x_sub(JJ) = Q(JJ)
c
      do 46 i = JJ-1,1,-1
          x_sub(i) = P(i) * x_sub(i+1) + Q(i)
  46  continue
c
      do 51 kk = 1,JJ
       x0((kk-1)*II+j) = x_sub(kk)
  51  continue
c
  11  continue
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      maxdif = 0.0
      do 60 kk = 1,M
          dif = dabs(x0(kk) - x0_old(kk))
          maxdif = dmax1(dif,maxdif)
  60  continue
c
        if (flag.EQ.0.0) then
c
            if (maxdif.LT.TOL) then
               print*, '           ITERATION   RESIDUAL NORM'
               print*, l, maxdif
               l = maxit + 1
            endif
c
        else
c
            mdif_p = dabs(mdif_2 - maxdif)
            if (mdif_p.LE.TOL) then
               print*, '           ITERATION   RESIDUAL NORM'
               print*, l, mdif_p
               l = maxit + 1
            endif
            mdif_2 = maxdif
c
        endif
c
       l = l + 1
c
      go to 90
      endif
c
      if (l.EQ.maxit+1) then
        print*, 'MAXIMUM **** ITERATION **** EXCEEDED'
        print*, '     ITERATION   RESIDUAL NORM'
        print*, l-1, maxdif
      endif
c
      do 70 j = 1,JJ
       do 80 i = 1,II
           xnew(i,j) = x0(i+(j-1)*II)
  80   continue
  70  continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c THOMASG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
      INTEGER*4 xnewP,aP,CINDXP,bP,xguesP,TOLP,maxitP,flagP
c
      INTEGER*4 I, J, M
c
      CALL GETSIZ(PRHS(4),I,J)
c
      PLHS(1) = CRTMAT(I,J,0)
c
      M = I * J
c
      xnewP = REALP(PLHS(1))
c
      aP = REALP(PRHS(1))
      CINDXP = REALP(PRHS(2))
      bP = REALP(PRHS(3))
      xguesP = REALP(PRHS(4))
      TOLP = REALP(PRHS(5))
      maxitP = REALP(PRHS(6))
      flagP = REALP(PRHS(7))
c
      CALL THOMAS(%VAL(xnewP),%VAL(aP),%VAL(CINDXP),%VAL(bP),
     +            %VAL(xguesP),%VAL(TOLP),%VAL(maxitP),I,J,M,
     +            %VAL(flagP))
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c P_FORCE.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine P_FORCE(dup,dvp,p,dx_p,dy_p,II,JJ)
c
      integer*4 II,JJ
      real*8 dup(II+1,JJ+2),dvp(II+2,JJ+1),p(II,JJ),dx_p(II)
      real*8 dy_p(JJ)
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
c
      j = 1
      do 11 i = 1,II+1
        dup(i,j) = 0.0
   11 continue
c
      j = JJ+2
      do 12 i = 1,II+1
        dup(i,j) = 0.0
   12 continue
c
      i = 1
      do 13 j = 2,JJ+1
        dup(i,j) = 0.0
   13 continue
c
      i = II+1
      do 14 j = 2,JJ+1
        dup(i,j) = 0.0
   14 continue
c
      do 15 j = 2,JJ+1
        do 16 i = 2,II
          dup(i,j) = dy_p(j-1) * (p(i-1,j-1) - p(i,j-1))
   16   continue
   15 continue
c
cccccccccccccccccccccccccccc
c
      j = 1
      do 17 i = 1,II+2
        dvp(i,j) = 0.0
   17 continue
c
      j = JJ+1
      do 18 i = 1,II+2
        dvp(i,j) = 0.0
   18 continue
c
      i = 1
      do 19 j = 2,JJ
        dvp(i,j) = 0.0
   19 continue
c
      i = II+2
      do 20 j = 2,JJ
        dvp(i,j) = 0.0
   20 continue
c
      do 21 j = 2,JJ
        do 22 i = 2,II+1
          dvp(i,j) = dx_p(i-1) * (p(i-1,j-1) - p(i-1,j))
   22   continue
   21 continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
c P_FORCEG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 dupP,dvpP,pP,dx_pP,dy_pP
c
c  Declare local variables.
c
      INTEGER*4 I, J
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(1),I,J)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(I+1,J+2,0)
      PLHS(2) = CRTMAT(I+2,J+1,0)
c
c  Assign pointers to the various parameters.
c
      dupP = REALP(PLHS(1))
      dvpP = REALP(PLHS(2))
c
      pP = REALP(PRHS(1))
      dx_pP = REALP(PRHS(2))
      dy_pP = REALP(PRHS(3))
c
c  Do the actual computation in a subroutine.
c
      CALL P_FORCE(%VAL(dupP),%VAL(dvpP),%VAL(pP),
     +             %VAL(dx_pP),%VAL(dy_pP),I,J)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c CONT_EQ.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine CONT_EQ(b_pp,ustar,vstar,dx_p,dy_p,ro,II,JJ)
c
      integer*4 II,JJ
      real*8 b_pp(II,JJ),ro,dx_p(II),dy_p(JJ)
      real*8 ustar(II+1,JJ+2),vstar(II+2,JJ+1)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c  CALCULATE MASS SOURCE b_pp (p.125, 126)
c      THIS IS THE AMOUNT BY WHICH THE CONTINUITY EQUATION IS NOT
c     SATISFIED FOR EACH CONTROL VOLUME.
c
      do 50 j = 1,JJ
       do 60 i = 1,II
```

```
c
      b_pp(i,j) = ro * dy_p(j) * (ustar(i,j+1) - ustar(i+1,j+1)) +
     +                ro * dx_p(i) * (vstar(i+1,j) - vstar(i+1,j+1))
c
 60    continue
 50   continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c CONT_EQG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 b_ppP,ustarP,vstarP,dx_pP,dy_pP,roP
c
c  Declare local variables.
c
      INTEGER*4 I, J, dummy
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(3),dummy,I)
      CALL GETSIZ(PRHS(4),dummy,J)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(I,J,0)
c
c  Assign pointers to the various parameters.
c
      b_ppP = REALP(PLHS(1))
c
      ustarP = REALP(PRHS(1))
      vstarP = REALP(PRHS(2))
      dx_pP = REALP(PRHS(3))
      dy_pP = REALP(PRHS(4))
      roP = REALP(PRHS(5))
c
c  Do the actual computation in a subroutine.
c
      CALL CONT_EQ(%VAL(b_ppP),%VAL(ustarP),%VAL(vstarP),
     +             %VAL(dx_pP),%VAL(dy_pP),%VAL(roP),I,J)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c COR_VEL.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine COR_VEL(ubc,vbc,Aumain,Avmain,
     +             dx_p,dy_p,ustar,vstar,pprime,II,JJ)
```

```
c
      integer*4 II,JJ
      real*8 ubc(II+1,JJ+2),vbc(II+2,JJ+1),ustar(II+1,JJ+2)
      real*8 vstar(II+2,JJ+1),Aumain(II+1,JJ+2),Avmain(II+2,JJ+1)
      real*8 dx_p(II),dy_p(JJ),pprime(II,JJ)
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c  CORRECT THE VELOCITY FIELD USING PPRIME (p.123)
c
      j = 1
      do 11 i = 1,II+1
        ubc(i,j) = ustar(i,j)
   11 continue
c
      j = JJ+2
      do 12 i = 1,II+1
        ubc(i,j) = ustar(i,j)
   12 continue
c
      i = 1
      do 13 j = 2,JJ+1
        ubc(i,j) = ustar(i,j)
   13 continue
c
      i = II+1
      do 14 j = 2,JJ+1
        ubc(i,j) = ustar(i,j)
   14 continue
c
      do 15 j = 2,JJ+1
        do 16 i = 2,II
          ubc(i,j) = ustar(i,j) + dy_p(j-1) / Aumain(i,j) *
     +                           (pprime(i-1,j-1) - pprime(i,j-1))
   16   continue
   15 continue
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      j = 1
      do 17 i = 1,II+2
        vbc(i,j) = vstar(i,j)
   17 continue
c
      j = JJ+1
      do 18 i = 1,II+2
        vbc(i,j) = vstar(i,j)
   18 continue
c
      i = 1
      do 19 j = 2,JJ
        vbc(i,j) = vstar(i,j)
   19 continue
c
      i = II+2
      do 20 j = 2,JJ
        vbc(i,j) = vstar(i,j)
   20 continue
c
```

```
      do 21 j = 2,JJ
        do 22 i = 2,II+1
          vbc(i,j) = vstar(i,j) + dx_p(i-1) / Avmain(i,j) *
     +                           (pprime(i-1,j-1) - pprime(i-1,j))
  22    continue
  21  continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c COR_VELG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 ubcP,vbcP,AumaiP,AvmaiP,ustarP,vstarP
      INTEGER*4 dx_pP,dy_pP,pprimP
c
c  Declare local variables.
c
      INTEGER*4 I, J, dummy
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(3),dummy,I)
      CALL GETSIZ(PRHS(4),dummy,J)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(I+1,J+2,0)
      PLHS(2) = CRTMAT(I+2,J+1,0)
c
c  Assign pointers to the various parameters.
c
      ubcP = REALP(PLHS(1))
      vbcP = REALP(PLHS(2))
c
      AumaiP = REALP(PRHS(1))
      AvmaiP = REALP(PRHS(2))
      dx_pP = REALP(PRHS(3))
      dy_pP = REALP(PRHS(4))
      ustarP = REALP(PRHS(5))
      vstarP = REALP(PRHS(6))
      pprimP = REALP(PRHS(7))
c
c  Do the actual computation in a subroutine.
c
      CALL COR_VEL(%VAL(ubcP),%VAL(vbcP),%VAL(AumaiP),%VAL(AvmaiP),
     +            %VAL(dx_pP),%VAL(dy_pP),%VAL(ustarP),%VAL(vstarP),
     +            %VAL(pprimP),I,J)
      RETURN
      END
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c SHEAR.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine SHEAR(G1,ubc,vbc,dx_p,dy_p,i_pos,j_pos,f_e,f_n,
     +                 II,JJ,J_LEN)
c
      integer*4 II, JJ, J_LEN
      real*8 G1(II,JJ),ubc(II+1,JJ+2),vbc(II+2,JJ+1),dx_p(II)
      real*8 dy_p(JJ),f_e(II+1),f_n(JJ+1),u_cent(60,60+2)
      real*8 v_cent(60+2,60),u_wall(60,60+1),v_wall(60+1,60)
      real*8 j_pos(J_LEN),i_pos(J_LEN)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c    Compute the production of Kinetic Energy due to SHEAR, G1
c
c       G1 = (du/dy - dv/dx)^2
c
c    for each pressure control volume.
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c    Compute u at center of pressure c.v.
c
      do 10 j = 1,JJ+2
        do 20 i = 1,II
          u_cent(i,j) = .5 * (ubc(i,j) + ubc(i+1,j))
 20     continue
 10   continue
c
c    Compute u at walls of pressure c.v.
c
      do 30 j = 1,JJ+1
        do 40 i = 1,II
         u_wall(i,j) = f_n(j)*u_cent(i,j)+(1.-f_n(j))*u_cent(i,j+1)
 40     continue
 30   continue
c
c    Compute v at center of pressure c.v.
c
      do 50 j = 1,JJ
        do 60 i = 1,II+2
          v_cent(i,j) = .5 * (vbc(i,j) + vbc(i,j+1))
 60     continue
 50   continue
c
c    Compute v at pressure c.v. walls.
c
      do 90 j = 1,JJ
        do 95 i = 1,II+1
          v_wall(i,j) = f_e(i) * v_cent(i,j) +
     +               (1 - f_e(i)) * v_cent(i+1,j)
 95     continue
 90   continue
c
c    Set u and v velocities on walls = 0.0
c
      do 25 k = 1,J_LEN
        i = IDNINT(i_pos(k))
        j = IDNINT(j_pos(k))
```

```
          do 35 l = i,II
            u_wall(l,j) = 0.0
            u_wall(l,j+1) = 0.0
  35      continue
          do 45 l = i,II+1
            v_wall(l,j) = 0.0
  45      continue
  25  continue
c
c    Compute G1 for each pressure control volume.
c
      do 70 j = 1,JJ
        do 80 i = 1,II
          G1(i,j) = ((u_wall(i,j+1) - u_wall(i,j)) / dy_p(j) -
     +                (v_wall(i+1,j) - v_wall(i,j)) / dx_p(i))**2
  80    continue
  70  continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c SHEARG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 G1P,ubcP,vbcP,dx_pP,dy_pP,f_eP,f_nP,i_posP,j_posP
c
c  Declare local variables.
c
      INTEGER*4 I, J, J_LEN, dummy
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(3),dummy,I)
      CALL GETSIZ(PRHS(4),dummy,J)
      CALL GETSIZ(PRHS(6),dummy,J_LEN)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(I,J,0)
c
c  Assign pointers to the various parameters.
c
      G1P = REALP(PLHS(1))
c
      ubcP = REALP(PRHS(1))
      vbcP = REALP(PRHS(2))
      dx_pP = REALP(PRHS(3))
      dy_pP = REALP(PRHS(4))
      i_posP = REALP(PRHS(5))
      j_posP = REALP(PRHS(6))
      f_eP = REALP(PRHS(7))
      f_nP = REALP(PRHS(8))
c
c  Do the actual computation in a subroutine.
c
      CALL SHEAR(%VAL(G1P),%VAL(ubcP),%VAL(vbcP),%VAL(dx_pP),
     +             %VAL(dy_pP),%VAL(i_posP),%VAL(j_posP),
     +             %VAL(f_eP),%VAL(f_nP),I,J,J_LEN)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c SOURCE.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine SOURCE(b_Sc,a_Sp,Sc,Sp,dx_p,dy_p,II,JJ)
c
      integer*4 II,JJ
      real*8 b_Sc(II,JJ),a_Sp(II,JJ),Sc(II,JJ),Sp(II,JJ)
      real*8 dx_p(II),dy_p(JJ)
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c  Compute source terms.
c
      do 10 j = 1,JJ
          do 20 i = 1,II
                 b_Sc(i,j) = Sc(i,j) * dx_p(i) * dy_p(j)
                 a_Sp(i,j) = Sp(i,j) * dx_p(i) * dy_p(j)
  20      continue
  10  continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c SOURCEG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 b_ScP,a_SpP,ScP,SpP,dx_pP,dy_pP
c
c  Declare local variables.
c
      INTEGER*4 I, J, dummy
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(3),dummy,I)
      CALL GETSIZ(PRHS(4),dummy,J)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(I,J,0)
      PLHS(2) = CRTMAT(I,J,0)
c
c  Assign pointers to the various parameters.
c
      b_ScP = REALP(PLHS(1))
      a_SpP = REALP(PLHS(2))
c
      ScP = REALP(PRHS(1))
      SpP = REALP(PRHS(2))
      dx_pP = REALP(PRHS(3))
      dy_pP = REALP(PRHS(4))
c
c  Do the actual computation in a subroutine.
c
      CALL SOURCE(%VAL(b_ScP),%VAL(a_SpP),%VAL(ScP),%VAL(SpP),
     +            %VAL(dx_pP),%VAL(dy_pP),I,J)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c GE_COEF.FOR     (General phi Equation COEFficients)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine GE_COEF(aE,aW,aN,aS,
     +   ubc,vbc,dx_p,dy_p,delx_p,dely_p,g_b_ew,g_b_ns,ro,II,JJ)
c
      integer*4 II, JJ
      real*8 aE(II,JJ),aW(II,JJ),aN(II,JJ),aS(II,JJ)
      real*8 ubc(II+1,JJ+2),vbc(II+2,JJ+1),dx_p(II),dy_p(JJ)
      real*8 delx_p(II+1),dely_p(JJ+1)
      real*8 g_b_ew(II+1,JJ),g_b_ns(II,JJ+1),ro
      real*8 De,Dw,Dn,Ds,Pe,Pw,Pn,Ps,Fe,Fw,Fn,Fs,zero
c
      zero = 0.0
c
c  CALCULATE THE COEFFICIENTS aP, for the general phi equation.
c
      do 10 j = 2,JJ+1
        do 20 i = 2,II+1
c
          Fe = ro * ubc(i,j)   * dy_p(j-1)
          Fw = ro * ubc(i-1,j) * dy_p(j-1)
          Fn = ro * vbc(i,j)   * dx_p(i-1)
          Fs = ro * vbc(i,j-1) * dx_p(i-1)
c
          De = g_b_ew(i,j-1)   * dy_p(j-1) / delx_p(i)
          Dw = g_b_ew(i-1,j-1) * dy_p(j-1) / delx_p(i-1)
          Dn = g_b_ns(i-1,j)   * dx_p(i-1) / dely_p(j)
          Ds = g_b_ns(i-1,j-1) * dx_p(i-1) / dely_p(j-1)
c
          Pe = Fe / De
          Pw = Fw / Dw
          Pn = Fn / Dn
          Ps = Fs / Ds
c
          Pe = (1. - .1 * dabs(Pe))**5
          Pw = (1. - .1 * dabs(Pw))**5
          Pn = (1. - .1 * dabs(Pn))**5
          Ps = (1. - .1 * dabs(Ps))**5
```

```
c
       aE(i-1,j-1) = De * dmax1(zero,Pe) + dmax1(-Fe,zero)
       aW(i-1,j-1) = Dw * dmax1(zero,Pw) + dmax1(Fw,zero)
       aN(i-1,j-1) = Dn * dmax1(zero,Pn) + dmax1(-Fn,zero)
       aS(i-1,j-1) = Ds * dmax1(zero,Ps) + dmax1(Fs,zero)
c
 20     continue
 10   continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c GE_COEFG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 aeP,awP,anP,asP,g_ewP,g_nsP,roP
      INTEGER*4 ubcP,vbcP,dx_pP,dy_pP,dlx_pP,dly_pP
c
c  Declare local variables.
c
      INTEGER*4 I, J, dummy
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(3),dummy,I)
      CALL GETSIZ(PRHS(4),dummy,J)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(I,J,0)
      PLHS(2) = CRTMAT(I,J,0)
      PLHS(3) = CRTMAT(I,J,0)
      PLHS(4) = CRTMAT(I,J,0)
c
c  Assign pointers to the various parameters.
c
      aeP = REALP(PLHS(1))
      awP = REALP(PLHS(2))
      anP = REALP(PLHS(3))
      asP = REALP(PLHS(4))
c
      ubcP = REALP(PRHS(1))
      vbcP = REALP(PRHS(2))
      dx_pP = REALP(PRHS(3))
      dy_pP = REALP(PRHS(4))
      dlx_pP = REALP(PRHS(5))
      dly_pP = REALP(PRHS(6))
      g_ewP = REALP(PRHS(7))
      g_nsP = REALP(PRHS(8))
      roP = REALP(PRHS(9))
c
```

```
c  Do the actual computation in a subroutine.
c
      CALL GE_COEF(%VAL(aeP),%VAL(awP),%VAL(anP),%VAL(asP),
     +                   %VAL(ubcP),%VAL(vbcP),%VAL(dx_pP),%VAL(dy_pP),
     +                   %VAL(dlx_pP),%VAL(dly_pP),
     +                   %VAL(g_ewP),%VAL(g_nsP),%VAL(roP),I,J)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c ATIMESX.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      subroutine ATIMESX(atx,phi,A_phi,IND_P,relax,Ip,Jp)
c
      integer*4 Ip,Jp,INDX_P(1200,5)
      real*8 phi(Ip,Jp),atx(Ip,Jp)
      real*8 A_phi(Ip*Jp,5),relax,IND_P(Ip*Jp,5),phi2(1200)
c
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
      do 2 j = 1,5
        do 3 i = 1,Ip*Jp
          INDX_P(i,j) = IDNINT(IND_P(i,j))
   3    continue
   2  continue
c
c   Compute A times X.
c
      do 10 j = 1,Jp
        do 15 i = 1,Ip
          k = i+(j-1)*Ip
          phi2(k) = phi(i,j)
          A_phi(k,3) = relax * A_phi(k,3)
  15    continue
  10  continue
c
      do 20 jj = 1,Jp
      do 25 ii = 1,Ip
        atx(ii,jj) = 0.0
        k = ii+(jj-1)*Ip
          do 30 j = 1,5
           if (INDX_P(k,j).ne.0) then
           atx(ii,jj) = atx(ii,jj) + A_phi(k,j) * phi2(INDX_P(k,j))
           endif
  30      continue
  25  continue
  20  continue
c
      return
      end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c ATIMESXG.FOR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      SUBROUTINE USRFCN(NLHS,PLHS,NRHS,PRHS)
      INTEGER NLHS,NRHS
      INTEGER*4 PLHS(*), PRHS(*)
      INTEGER*4 CRTMAT, REALP, GETSIZ
c
c  Declare integers to be used as pointers.
c
      INTEGER*4 atxP,phiP,A_phiP,IND_PP,relaxP
c
c  Declare local variables.
c
      INTEGER*4 Ip, Jp
c
c  Get size of input argument.
c
      CALL GETSIZ(PRHS(1),Ip,Jp)
c
c  Create matrices for return arguments.
c
      PLHS(1) = CRTMAT(Ip,Jp,0)
c
c  Assign pointers to the various parameters.
c
      atxP = REALP(PLHS(1))
c
      phiP = REALP(PRHS(1))
      A_phiP = REALP(PRHS(2))
      IND_PP = REALP(PRHS(3))
      relaxP = REALP(PRHS(4))
c
c  Do the actual computation in a subroutine.
c
      CALL ATIMESX(%VAL(atxP),%VAL(phiP),%VAL(A_phiP),%VAL(IND_PP),
     +              %VAL(relaxP),Ip,Jp)
      RETURN
      END

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 20. MATLAB Plotting Programs.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ARROWS.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Plot the flow field using arrows to indicate direction and
%   magnitude.

xx = [0 1 .7 1 .7].';
yy = [0 0 .12 0 -.12].';
arrow = xx + yy.*sqrt(-1);

x_c = .5 * fliplr(x_coords(2:I+1));
y_c2 = .5 * y_coords(2:J+1);
y_c = y_c2 - y_c2(j_pos_wall);

[xx,yy] = meshdom(y_c,x_c);
GRID = xx + yy.*sqrt(-1);
GRID = GRID(:);
x = v(:);
y = -u(:);
z = (x + y.*sqrt(-1)).';
scale = 1 ./ max(max(abs(z)));
a = mag * scale * arrow * z + ones(5,1) * GRID.';
plot(real(a),imag(a),'-'), xlabel('Distance (x/h)'),
  ylabel('Height (z/h)'),
  title('Turbulent Flow Field'),
hold on,boxesw,hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BOXES.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Draw boxes around control volumes.

hold on

x_w = .5 * x_wall;
y_w2 = .5 * y_wall;
y_w = y_w2 - .5 * (y_w2(j_pos_wall) + y_w2(j_pos_wall+1));

for j = 1:J
 for i = 1:I

  if gamma(i,j) == nan
   xbox = [x_w(i) x_w(i+1) x_w(i+1) x_w(i) x_w(i)];
   ybox = [y_w(j) y_w(j) y_w(j+1) y_w(j+1) y_w(j)];
   box = ybox + xbox .* sqrt(-1);
   plot(box,'-')
  end

  if gamma_orig(i,j) >= 1e3
   xbox = [x_w(i) x_w(i+1) x_w(i+1) x_w(i) x_w(i)];
   ybox = [y_w(j) y_w(j) y_w(j+1) y_w(j+1) y_w(j)];
   box = ybox + xbox .* sqrt(-1);
   plot(box,'-')
   q_max = (x_w(i)-x_w(i+1))/.01;
   for q = 1:q_max
     t = q * .01;
     xline = [x_w(i)-t x_w(i)-t];
     yline = [y_w(j) y_w(j+1)];
     plot(yline,xline,'-')
   end
  end

 end
end

hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% STAIRS.M
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Draw stair steps.

function [xo,yo] = stairs(dy_p,dh,y_coords,j_pos_wall)

y_coords = .5 * y_coords;
y_coords = y_coords - y_coords(j_pos_wall);

J = length(dy_p);

nn = 2 * J;
yy = zeros(nn+2,1);
xx = yy;
tt = y_coords(:)' - .5 * .5 * dy_p;
xx(1:2:nn) = tt;
xx(2:2:nn) = tt;
xx(nn+1:nn+2) = tt(J) + .5 * [dy_p(J);dy_p(J)];
yy(2:2:nn) = dh;
yy(3:2:nn+1) = dh;

% Don't include the left and right sides.

xo = xx(2:nn+1);
yo = yy(2:nn+1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```