CRWR Online Report 10-02

DRAINAGE HYDRAULICS OF POROUS PAVEMENT: COUPLING SURFACE AND SUBSURFACE FLOW

by

Bradley J. Eck, B.S.; M.S.E.

Randall J. Charbeneau, Ph.D.

Michael E. Barrett, Ph.D.

May 2010

Center for Research in Water Resources

The University of Texas at Austin

J.J. Pickle Research Campus

Austin, TX 78712-4497

This document is available online via the World Wide Web at:

http://www.crwr.utexas.edu/online.shtml

Drainage Hydraulics of Porous Pavement:

Coupling Surface and Subsurface Flow

Bradley Joseph Eck, Ph.D.

The University of Texas at Austin, 2010

Supervisor: Randall Charbeneau

Permeable friction course (PFC) is a porous asphalt pavement placed on top of a regular impermeable roadway. Under small rainfall intensities, drainage is contained within the PFC layer; but, under higher rainfall intensities drainage occurs both within and on top of the porous pavement. This dissertation develops a computer model—the permeable friction course drainage code (PERFCODE)—to study this two-dimensional unsteady drainage process. Given a hyetograph, geometric information, and hydraulic properties, the model predicts the variation of water depth within and on top of the PFC layer through time. The porous layer is treated as an unconfined aquifer of variable saturated thickness using Darcy's law and the Dupuit-Forchheimer assumptions. Surface flow is modeled using the diffusion wave approximation to the Saint-Venant equations. A mass balance approach is used to couple the surface and subsurface phases. Straight and curved roadway geometries are accommodated via a curvilinear grid. The model is validated using steady state solutions that were obtained independently. PERFCODE was applied to a field monitoring site near Austin, Texas and hydrographs predicted by the model were consistent with field measurements. For a sample storm studied in detail, PFC reduced the duration of sheet flow conditions by 80%. The model may be used to improve the drainage design of PFC roadways.

Acknowledgements

This research was supported by the Texas Department of Transportation under project 0-5220.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Symbol	Definition
A	Area
b	PFC thickness
C	Conveyance coefficient
d	Mean grain size of a porous medium OR characteristic length scale
g	Gravitational acceleration
H	Total head above datum
h	Flow depth
h_p	Saturated thickness in the PFC layer
h_s	Sheet flow thickness
I	Hydraulic gradient
i	Longitudinal index of grid cells
j	Transverse index of grid cells
K	Saturated hydraulic conductivity
K_{us}	Unsaturated hydraulic conductivity
k	Intrinsic Permeability
L	Depth to water table, Flow length
ℓ	Length of grid cell in the longitudinal direction
n	Porosity of a porous medium OR Manning's roughness coefficient
n_e	Effective porosity of a porous medium
P	Pressure
pf	Porosity function
Q	Volumetric Flow Rate
q	Darcy velocity
q_F	Specific discharge predicted by Forchheimer equation
R	Hydraulic radius or radius of curvature at roadway centerline
r	Rainfall rate
S	Slope
S_0	Bed slope
S_f	Friction slope
t	Time
U	Overland flow velocity
V	Volume
v	Velocity
W	Width of roadway
w	Width of a grid cell
x	Cartesian coordinate direction

y	Cartesian coordinate direction
Z	Elevation above datum
z	Cartesian coordinate or elevation above datum

Greek Symbol	**Definition**
α	First Forchheimer coefficient
$\hat{\alpha}$	Inverse of hydraulic conductivity
β	Non-linear coefficient in Forchheimer's equation
$\hat{\beta}$	Forchheimer coefficient or non-Darcy coefficient for pressure gradient formulation of Forchheimer equation
β^*	Darcy's law modification factor that depends on Reynold's number
∂	Partial differentiation operator
η	Transverse roadway coordinate in computational space
ξ	Longitudinal roadway coordinate in computational space
μ	Dynamic viscosity
ρ	Fluid density
ν	Kinematic viscosity
θ	Water content of porous medium or angular position of roadway centerline point
θ_r	Irreducible water content
Ψ	Cappilary pressure head (m)
λ	Pore size distribution index
Φ	Ratio comparing discharge predicted by Darcy's law and Forchheimer's equation

Dimensionless Number	**Definition**
Re	Reynold's number
F	Froude number
F_o	Forchhiemer number
N_k	Kinematic wave number

Abbreviation	**Definition**
MOC	Method of characteristics
PERFCODE	PERmeable Friction COurse Drainage codE, the name of the numerical model developed in this dissertation
PFC	Permeable friction course
RHS	Right hand side of an equation

CHAPTER 1: INTRODUCTION

1.1 Background and Motivation

New roadway materials are changing the wet weather driving experience. One exciting and innovative material is a porous pavement that allows water to drain through the roadway rather than across it. The porous pavement—also called permeable friction course (PFC)—is placed in a 50mm layer on top of conventional, impermeable, pavement. During rain events, water seeps into the porous layer and flows to the side of the road by gravity. By removing water from the road surface, PFC improves safety by reducing splashing and hydroplaning (Berbee et al., 1999). In addition to safety benefits, PFC has also been shown to reduce pollutants commonly observed in highway runoff (Barrett, 2006).

Although usually placed in a 50mm layer, the PFC thickness may be selected so that all of the rainfall for a design event drains within the pavement. However, structural and cost concerns prevent the use of an arbitrarily thick porous layer. Additionally, PFC has been shown to clog over time, resulting in lower subsurface drainage capacity (NCHRP, 2009). Therefore, some storms will exceed the installed capacity, forcing drainage to occur both on the pavement surface and within the porous matrix. Understanding this coupled flow process is the goal of this research.

A precise description of PFC's response to rainfall events is needed for several reasons including driver safety, water quality, and basic science. From a safety perspective, flow over traffic lanes can cause vehicles to hydroplane. Hydroplaning is especially hazardous when right and left tires encounter different water depths—the difference in resistance imposes a torque on the vehicle, potentially causing the driver to lose control. A detailed runoff model for PFC could identify areas of excessive sheet flow depth so that additional drainage can be provided. Such a model also has implications for water quality. Field studies of runoff from PFC have shown that runoff concentrations of pollutants are lower for PFC than conventional pavement, but the mechanisms responsible for lower concentrations have not been identified (Stanard, 2008). Possible mechanisms include reduced wash-off from vehicles, filtration and absorption within the pavement, and even biological activity. Studying these mechanisms in detail requires an accurate hydraulic model. Finally, the proposed model is of general scientific interest because the problem of flow over porous media appears in numerous applications. Civil engineering applications include surface irrigation, watershed modeling, and sediment transport. The concept of flow over porous media has also been applied to biological systems such as blood flow within the arterial wall (Dabaghmeshin, 2008). A better technical understanding of flow in PFC will contribute to a diverse scientific field and promote wider use of the material, thereby improving driver safety and the environment.

Figure 1 shows a photograph of a PFC layer. The PFC overlay is very thin compared to the length and width of the roadway section. A cross section of typical PFC roadway is shown in Figure 2 and a more detailed schematic of the PFC layer is shown in Figure 3.

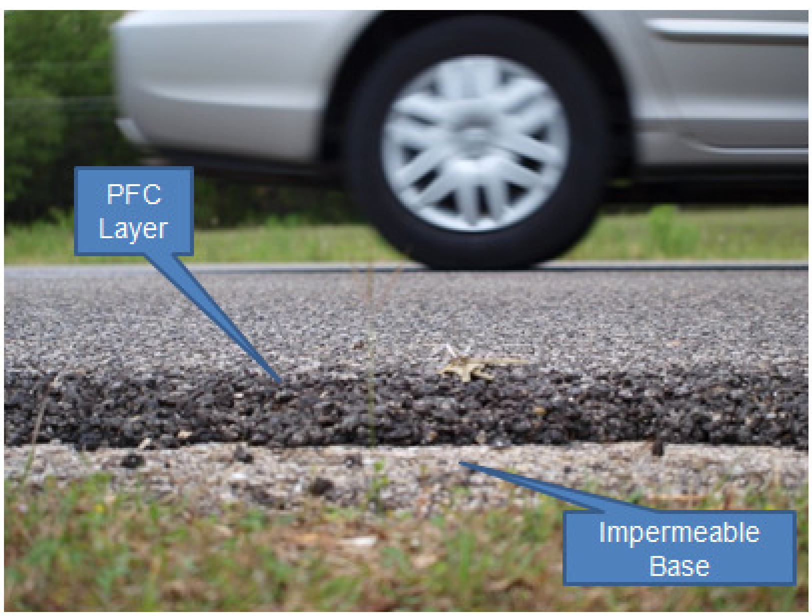

Figure 1: Photograph of PFC layer on Loop 360, Austin, Texas

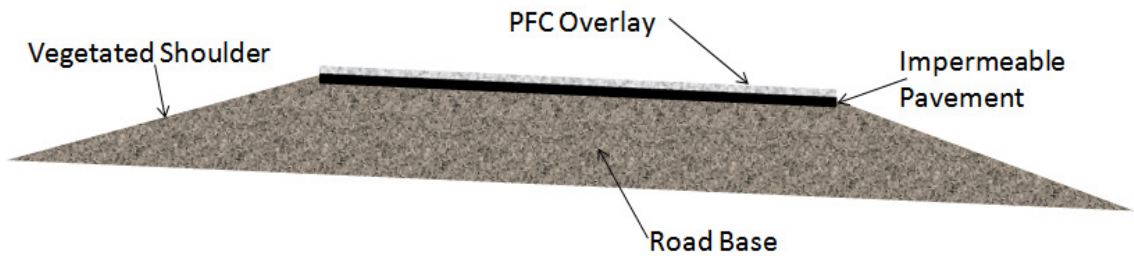

Figure 2: Cross section of a typical PFC roadway

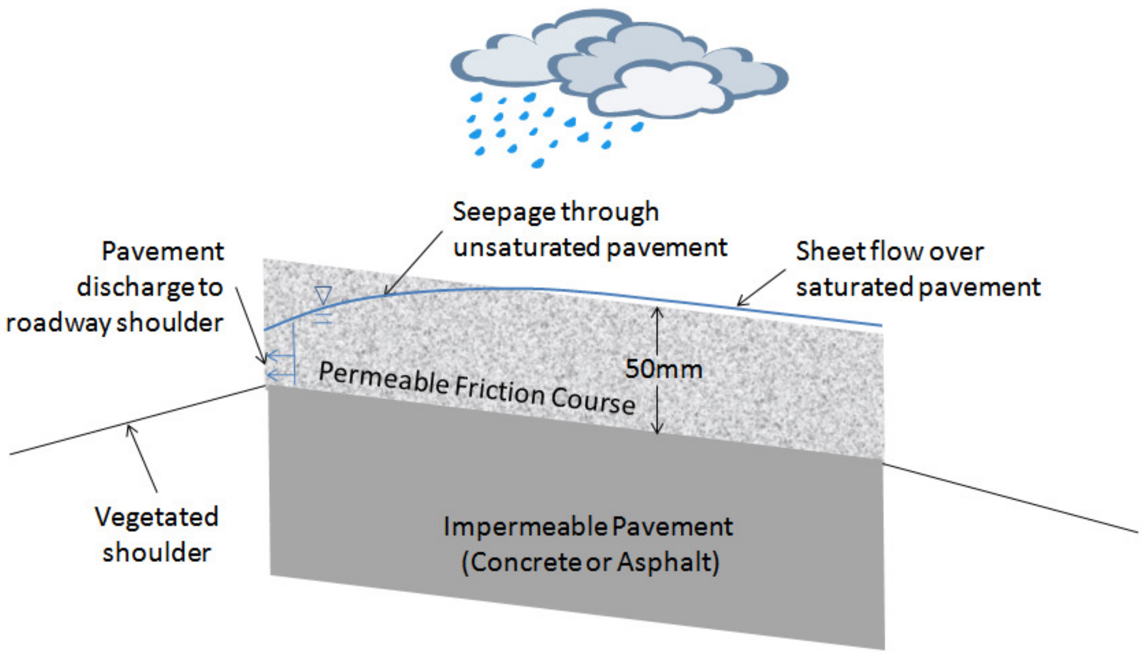

Figure 3: Schematic cross section of a roadway with a PFC overlay

1.2 Research Objectives

The goal of this research is to understand the coupling between overland flow and porous media flow in roadway applications. In this context, understanding the coupling means predicting water depths at a fine enough scale to assess the risk of hydroplaning. To accomplish this goal, a numerical model that predicts water surface elevations on roads overlain with PFC has been developed and validated. The model has as inputs the roadway geometry, rainfall intensity, and porous media properties. The model has been formulated to accommodate roadway geometries where the horizontal alignment may be straight or curved and to accommodate variable rainfall intensity.

Based on these inputs, the goal of understanding coupled flow between the surface and subsurface will be pursued through the following research objectives:

1. Identify governing equations for surface and subsurface flow for the geometry of interest
2. Develop a scheme to couple flow between the surface and subsurface
3. Implement the coupling scheme and numerical methods in a computer model that represents roadway geometry using a coordinate transformation
4. Validate the model using analytical solutions
5. Compare model predictions of runoff rates with values measured at an existing monitoring site

During the preparation of this dissertation, the National Cooperative Highway Research Program (NCHRP) issued Report 640 entitled "Construction and Maintenance Practices for Permeable Friction Courses" (NCHRP, 2009). The report signifies the growing popularity and importance PFC layers for highways in the USA. Several of the future research needs listed in the report are addressed in part by this dissertation:

- Field work to document how water flows within a PFC layer
- Methods for selecting the minimum PFC thickness
- Consideration for water sheets on the PFC surface

Field work included constructing a monitoring site to measure runoff hydrographs from a PFC roadway. The dynamic simulation model developed in this dissertation accounts for sheet flow on the PFC surface and seepage through the porous layer; it can be used to evaluate methods for selecting the thickness of a PFC layer. Another important and related research need identified in the report is a method to determine the permeability of PFC layers. The work of Klenzendorf (2010) addresses the hydraulic conductivity of PFC and this dissertation uses his results to simulate PFC flow on highways.

1.3 Organization of the Dissertation

This document is organized into six chapters. Chapter 1 has introduced the work and defined the research objectives. Chapter 2 reviews selected literature that bears on the work. A method for developing a predictive model for PFC drainage is given in Chapter 3. The proposed model is essentially a specialized hydrologic model so Chapters 2 and 3 are organized around hydrologic processes. The methods of Chapter 3 have been implemented in a Fortran computer model called PERFCODE, the structure of which is described in Appendix A. Chapter 4 validates the model's numerics by comparing model results with independently obtained solutions for simplified cases. Chapter 4 also discusses the model's stability and convergence properties. Chapter 5 applies the model to a field monitoring site, facilitating a comparison of modeled results with field measurements. Chapter 6 concludes the dissertation with a summary of the findings and possible avenues for future work.

CHAPTER 2: LITERATURE REVIEW

This review summarizes the literature that provides the theoretical foundation for this research. Developments related specifically to permeable friction course (PFC) are given first. A general discussion of subsurface flow is given next and readers who are unfamiliar with flow in porous media may prefer to review it prior to the section on PFC. A section on overland flow is given next, followed by a discussion of coupling schemes and models of coupled surface/subsurface systems. The final section identifies gaps in the literature that are addressed by this research.

2.1 Permeable Friction Course

2.1.1 Water Depth Predictions

Three authors have published predictions of water depth in PFC for straight roadway sections under constant rainfall. Ranieri (2002) gives a numerical solution to the governing equation. Tan et al. (2004) use a commercially available finite element program to model flow through PFC. Both Ranieri (2002) and Tan et al. (2004) provide charts to find the required thickness of PFC from slope information and rainfall intensity. Charbeneau and Barrett (2008) provide an analytical solution for the saturated thickness along the flow path.

These three papers consider the same roadway geometry: a straight road with a longitudinal slope and a cross slope. The drainage slope is the Pythagorean sum of the longitudinal slope and the cross slope. In these papers, the drainage slope is a constant, making the problem one dimensional—that is the saturated thickness only varies along the drainage path. Under the assumption of constant rainfall intensity the system reaches a steady state. It is this one-dimensional steady state solution that these authors present.

A comparison of their predictions for a single point reveals that Charbeneau and Barrett (2008) and Ranieri (2002) have essentially identical results. Tan et al. obtain a different result, predicting a thinner porous layer than the other workers. The reasons for this discrepancy are difficult to uncover because Tan et al. used a commercial finite element program for analysis.

The problem of drainage within a PFC layer of constant slope and under steady rainfall is analogous the problem of hillslope seepage under constant recharge. Most solutions make the Dupuit-Forchheimer assumptions of horizontal flow with the local discharge proportional to the slope of the water table. Equivalent results to those of Charbeneau and Barrett (2008) and Ranieri (2002) have been presented by Yates, Warrick and Lomen (1985) and also by Loaiciga (2005).

Very little has been mentioned in the literature regarding the coupling between surface and subsurface flow in PFCs. Charbeneau and Barrett (2008) address the issue briefly and provide an estimate of sheet flow thickness based on the Darcy-Weisbach equation. Eck et al. (2010) refined the coupling between PFC and sheet flow by using a different boundary condition for the PFC equation. The idea was to compute the location that sheet flow begins based on the principle of continuity and use that location and the pavement thickness as the initial point to integrate the first order ODE that governs the PFC part of the problem.

2.1.2 Hydraulic Properties of PFC

Hydraulic properties of PFC have been investigated by several authors, which have been summarized by Standard et al. (2008). Reported values for hydraulic conductivity range from $5*10^{-4}$ cm/s to 3 cm/s. Ongoing research by Klenzendorf (2010) investigates the porosity and the hydraulic conductivity of PFC. Porosity was measured from core samples and found to range from 0.12 to 0.23. Hydraulic conductivity was also measured from core samples and ranged from 0.1 to 3 cm/s. A new field method for measuring the in-situ hydraulic conductivity of PFC was developed and compared to the laboratory measurements.

2.2 Saturated Porous Media Flow

Saturated porous media flow refers to the movement of fluid through a porous medium when the pore space is filled with fluid. The boundary between saturated and unsaturated zones of a porous medium is the water table. The water table is at atmospheric pressure. Below the water table the media is saturated. Above the water table the media is considered unsaturated, though a small area of saturated pores may exist above the water table due to capillary effects. Quantitative predictions of saturated porous media flow apply Darcy's law or the Forchheimer equation to relate the hydraulic gradient and the specific discharge.

2.2.1 Darcy's Law

The usual way of characterizing flow through porous media is Darcy's law. Darcy's law states that the relationship between the hydraulic gradient and seepage velocity is linear when velocities are low enough to neglect inertia (Charbeneau, 2000). A simple statement of Darcy's law is:

$$Q = KIA \tag{2.1}$$

where Q is the volumetric flow rate, I is the hydraulic gradient, A is the cross sectional area of the flow, and K is a parameter called the hydraulic conductivity that depends on the properties of the porous medium and the fluid. Darcy's law is frequently presented in terms of the velocity obtained by dividing the flow rate by the area:

$$q = KI \tag{2.2}$$

where q is the fictitious velocity known as the Darcy velocity, or the specific discharge. The relative contributions of the porous medium and the fluid to the hydraulic conductivity can be seen by expressing the hydraulic conductivity as:

$$K = \frac{\rho g k}{\mu} \tag{2.3}$$

where ρ is the fluid density, g is the constant of gravitational acceleration, μ is the dynamic viscosity of the fluid , and k is a property of the medium called the intrinsic permeability which is related to the grain size distribution of the medium. From an analysis of the Fanning friction factor, one relationship between permeability and grain size is (Charbeneau, 2000):

$$k = \frac{d^2}{2000} \tag{2.4}$$

Bear (1972) gives several correlations between the mean or effective grain size and the intrinsic permeability. The hydraulic conductivity is typically preferred in groundwater hydrology because water is the only fluid of interest. In contrast, the petroleum industry uses the intrinsic permeability because several fluids are often of interest.

2.2.2 Reynolds Number and Porous Media Flow Regimes

Although Darcy's law neglects inertial effects, the inertial terms are physically real and do not disappear from the equations. In fluid mechanics the relative importance of inertial and viscous effects is quantified using the Reynolds number (Re), which expresses the ratio of these effects (White, 1999):

$$Re = \frac{\rho v d}{\mu} \tag{2.5}$$

In the expression for Reynolds number, d is a length scale of the problem, v is the fluid velocity, and other terms are defined previously. At low values of Reynolds number, the numerator (inertial effects) is small compared to the denominator (viscous effects). As Re increases, inertial effects become more important. In porous media applications Reynolds number is formulated using the seepage velocity and a representative length scale. Several length scales have been used including the median grain size (d_{50}) and $k^{\frac{1}{2}}$ (Ward, 1964).

As the value of Re increases, inertial effects become important and Darcy's law ceases to apply. This behavior suggests the identification of flow regimes in a porous media based on the Reynolds number. Bear (1972) identifies three such regimes:

(1) A linear regime where the Reynolds number is lower than a limit somewhere between 1 and 10 and Darcy's law applies.

(2) A non-linear regime where inertial effects are important, but the flow remains laminar. An upper limit of Re=100 has been suggested for this regime.

(3) A turbulent regime where Reynolds number is high.

Darcy's law applies in the first regime only.

2.2.3 Relations for Non-Darcy Flow

PFC drainage under highway drainage conditions is expected to fall in the Darcy regime of flow. However, experimental efforts to estimate the hydraulic conductivity of PFC have observed non-Darcy flow regimes (Ranieri 2002; Barrett et al. 2009). In this section, relations for non-Darcy flow are reviewed to provide a basis for estimating the error of the Darcy approximation and to identify methods of including a non-Darcy effect in future versions of the model.

Forchheimer's Equation

One approach for describing non-Darcy flow is Forchheimer's equation, which is written either in terms of the hydraulic gradient:

$$I = \alpha q + \beta q^2 \tag{2.6}$$

or equivalently in terms of the pressure gradient:

$$-\frac{dP}{dx} = \hat{\alpha} q + \hat{\beta} \rho q^2 \tag{2.7}$$

where the hat symbol distinguishes the coefficients between the equations. If $\beta = \hat{\beta} = 0$ and $\alpha = \frac{1}{K}$ then Forchheimer's equation reduces to Darcy's law. The coefficient $\hat{\beta}$ is often called the Forchheimer coefficient (Ruth and Ma, 1992) or the non-Darcy coefficient (Li and Engler, 2001). It is related to β of the hydraulic gradient formulation by the constant of gravitational acceleration:

$$\beta = \frac{\hat{\beta}}{g} \tag{2.8}$$

Many correlations for the Forchheimer coefficient have been developed. Ergun (1952) measured the pressure drop of gases through columns packed with granular material. He gives an empirical correlation for the energy loss based on a least squares treatment of the experimental data. Ergun partitioned the total energy loss between viscous and kinetic energy losses. Ergun's work was presented in the form of Forchheimer's equation by Bird et al. (1960):

$$I = \frac{150(1-n)^2\mu}{n^3 d^2 \rho g} q + \frac{1.75(1-n)}{g n^3 d} q^2 \tag{2.9}$$

where d is the mean grain diameter, n is the porosity of the medium, the values of $a = 1.75$ and $b = 150$ were obtained by Ergun, and other terms are defined previously. More recently Thauvin and Mohanty (1998) presented, but did not derive, an expression for the Forchheimer coefficient by dimensional analysis of Forchheimer's equation based on Ergun's work:

$$\hat{\beta} = a b^{-1/2} (10^{-8} k)^{-1/2} n^{-3/2} \tag{2.10}$$

where $\hat{\beta}$ is the non-Darcy coefficient in 1/cm and k is the permeability in units of darcy. Equation (2.10) is a different result than Equation (2.9). Ward (1964) also gives a correlation for the coefficients of Forchheimer's equation:

$$I = \frac{\mu}{k\rho g} q + \frac{0.55}{g\sqrt{k}} q^2 \tag{2.11}$$

Whereas Ergun's experimental work used gases, Ward's experiments were performed with water. In the Ward formula, the linear term is consistent with Darcy's law, and no estimate of the porosity is required. Many other correlations for the Forchheimer coefficient are reviewed by Li and Engler (2001).

So far this review has used the Reynolds number to distinguish between linear and non-linear flow regimes in porous media. This usage is not entirely consistent because Darcy's law and Forchheimer's equation pertain to the macroscopic flow parameters of hydraulic or pressure gradient and seepage velocity, but the Reynolds number applies to the microscopic velocity. In order to avoid confusion, a dimensionless group similar to the Reynolds number, but called the Forchheimer number has been proposed by Zeng and Grieg (2006):

$$F_o = \frac{\rho q k \hat{\beta}}{\mu} \tag{2.12}$$

This proposal amounts to suggesting another representative length scale ($k\hat{\beta}$) for a porous medium. Ruth and Ma (1992) also define a Forchheimer number. Their formulation holds that the permeability depends on the velocity. Because this principle is not widely held, the Zeng and Grieg formulation is used in this work. A Forchheimer number of 0.11 corresponds to a 10% non-Darcy effect, and is recommended as a critical value for the transition to non-Darcy flow (Zeng and Grieg 2006).

Kovac's Hyperbola

Another approach to characterizing non-Darcy flow is given by Kovacs (1981). Kovacs reviews many correlations for porous media flow in the transition and turbulent regimes. He proposes a hyperbola to describe all of the flow regimes through porous media. Relations for the different regimes may be developed by approximating the hyperbola in that regime. The approximation proposed for the transition regime is of the form:

$$q = KI\beta^* \tag{2.13}$$

where q is the specific discharge, K is the Darcy hydraulic conductivity, I is the hydraulic gradient, and β^* is a function of the Reynolds number. Ranieri (2002) determined values for β^* from experimental data.

2.2.4 Dupuit-Forchheimer Assumptions

So far, this review has discussed several ways to predict how the hydraulic gradient (or pressure gradient) in a porous medium varies in space, but has not directly addressed the pressure distribution through the medium. In the case of flow through a PFC, the porous medium flow is always bounded above by a free surface so the flow is said to be unconfined. If the velocities are essentially horizontal, then the hydraulic head will be the same on any vertical line and the pressure distribution will be hydrostatic (Bear, 1972). In this case, the discharge is proportional to the hydraulic gradient. The assumptions that the head is independent of depth, and that the discharge is proportional to the hydraulic gradient are the Dupuit-Forchheimer assumptions (Charbeneau, 2000).

Irmay (1967) studied the error in predicting the hydraulic head using the Dupuit-Forchheimer assumptions. He gives formulas for computing the relative error at different depths for flat and inclined aquifers. For a flat aquifer, the maximum error occurs at mid depth and depends mostly on the hydraulic gradient. A hydraulic gradient of 10% caused a maximum error of 0.25% in the hydraulic head. As most roadways have a drainage slope smaller than 10%, the Dupuit-Forchheimer assumptions provide a good approximation.

2.3 Unsaturated Porous Media Flow

Unsaturated porous media flow occurs when the pore space is not completely filled with a single fluid. Unsaturated flow is more difficult to describe than saturated flow because the hydraulic conductivity and capillary pressure change with the water content. Richard's equation governs unsaturated flow and considers the variation of hydraulic conductivity and capillary pressure with water content:

$$\frac{\partial \theta}{\partial t} = div\left(-K_{us}(\theta)\frac{d\Psi}{d\theta}\, grad(\theta)\right) + \frac{\partial K_{us}(\theta)}{\partial z} \qquad (2.14)$$

In Richard's equation θ is the water content, Ψ is the capillary pressure head, and K_{us} is the unsaturated hydraulic conductivity (Charbeneau, 2000).

For PFC drainage, unsaturated flow is essentially vertical and the primary effect of interest is the travel time through the unsaturated zone. For this purpose, Richard's equation may be simplified by considering only vertical flow and neglecting capillary pressure gradients. This leads to the kinematic form of Darcy's law:

$$q = K_{us}(\theta) \qquad (2.15)$$

where q is the specific discharge and K_{us} is the unsaturated hydraulic conductivity which depends on the water content, θ. This form of Darcy's law applies specifically to vertical flow so the hydraulic gradient is unity.

In order to apply the kinematic form of Darcy's law a relationship between the hydraulic conductivity and water content must be obtained. One such relationship is the power law model of Brooks and Corey (Charbeneau, 2000):

$$K_{us} = K\Theta^{3+2/\lambda} \tag{2.16}$$

where K_{us} is the unsaturated hydraulic conductivity, K is the saturated hydraulic conductivity, Θ is the water content assuming zero field capacity, and λ is the pore size distribution index.

Using Equations (2.15) and (2.16), Charbeneau (2000) estimates the average pore-water velocity using an average value of the water content:

$$v = \frac{G}{\theta_r + (n - \theta_r)\left(\frac{G}{K}\right)^{\frac{\lambda}{3\lambda+2}}} \tag{2.17}$$

where G is net recharge rate (assumed equal the rainfall rate for the PFC), θ_r is the irreducible water content, K is the saturated hydraulic conductivity, and λ is the pore size distribution index. With this average velocity, the travel time through the unsaturated zone can be estimated:

$$t = \frac{L}{v} \tag{2.18}$$

where L is the depth to the water table. The equations presented in this section are used in Section 3.2.3 to evaluate the effect of unsaturated flow in the model.

2.4 Overland Flow

Overland flow is governed by a simplification of the Navier-Stokes equations first presented by Saint-Venant in 1871 (Chow et al., 1988). The full Saint-Venant equations retain all of the terms of the Navier-Stokes equations including terms for inertial, viscous, and gravitational forces, along with convective accelerations. For the purpose of predicting flow at shallow depths, various levels of approximation to the Saint-Venant equations have been applied (Chow et al., 1988). The kinematic wave approximation retains only the gravitational and viscous terms. The diffusion wave approximation adds the pressure term. The full Saint-Venant equations, with no simplifications, are known as the dynamic wave model.

Three non-dimensional parameters are important in characterizing the overland flow problem: (1) Reynolds number, (2) Froude number, (3) Kinematic wave number. Reynolds number is defined in Equation (2.5). The Froude number is defined as:

$$F = \frac{v}{\sqrt{gh}} \tag{2.19}$$

where v is the velocity, g is the gravitational constant, and h is the flow depth. The Froude number compares the speed of the flow with the speed of a gravity wave (White, 1999).

The kinematic wave number is defined as:

$$N_k = \frac{SL}{hF^2} \tag{2.20}$$

where S is the slope, L is the length, h is the depth and F is the Froude number. The symbol N_k is used here instead of the usual symbol K to avoid confusion with the saturated hydraulic conductivity. The kinematic wave number reflects the length and slope of the plane as well as the normal flow variables (Woolhiser and Liggett, 1967).

The ranges of applicability for the levels of approximation to the Saint-Venant equations are studied in terms of the Froude number and kinematic wave number by Daluz Vieira (1983). The author produced a plot showing the range of applicability for the kinematic wave, diffusion wave, and full Saint-Venant equations (Figure 4).

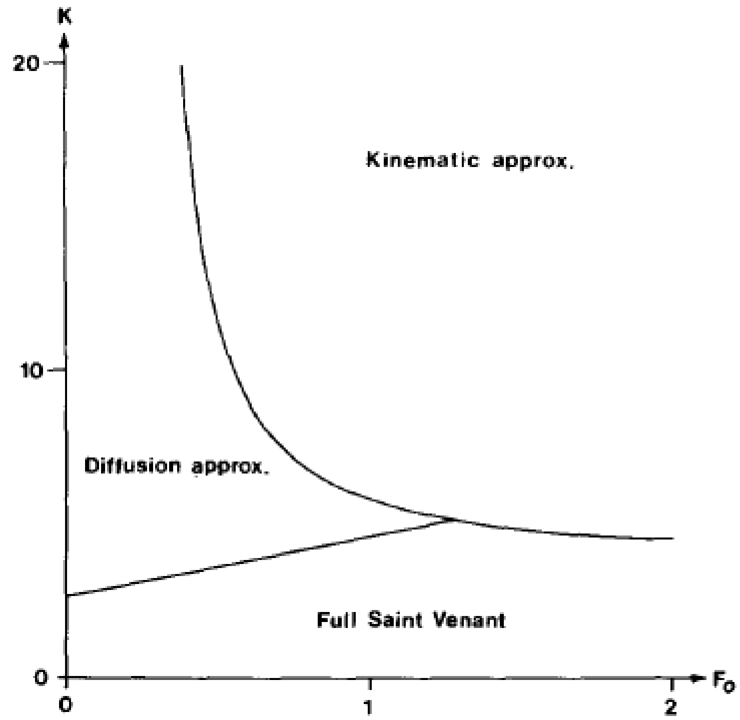

Figure 4: Range of applicability for sheet flow models (Daluz Vieira, 1983); used with permission

On smooth urban slopes the kinematic wave number lies between 5 and 20 (Daluz Vieira, 1983) so the diffusion wave approximation is appropriate for the full range of Froude numbers.

2.5 The CRWR Approach to Modeling Highway Drainage

The research presented in this dissertation is the latest advance in a long tradition of work in highway drainage hydraulics conducted at the Center for Research in Water Resources (CRWR) at The University of Texas at Austin. The present sub-section describes how different aspects of the previous research have been incorporated into the present work.

Previous highway drainage research at CRWR has included both experimental measurements and numerical modeling. Experimental work included measuring the sheet flow thickness on a laboratory roadway section under simulated rainfall. The roadway section is rectangular and situated so that the elevation of three corners can be adjusted to achieve a range of longitudinal and cross slopes. Sheet flow thicknesses and unit discharge were measured on three surfaces having different roughness under a range of slopes and rainfall conditions. Charbeneau et al. (2009) analyzed this data and evaluated depth-discharge relationships. They concluded that Manning's equation had equivalent accuracy to logarithmic boundary layer theory, and that the hydraulic effects of rainfall on sheet flow were negligible.

Previous research at CRWR in the area of numerical modeling developed a hydrodynamic diffusion wave model for sheet flow in superelevation transitions (Jeong, 2008). Beyond implementing the diffusion wave model for sheet flow, this work developed a curvilinear grid generation scheme that is well suited for highway drainage hydraulics. The idea of the grid generation scheme is that each point along a roadway centerline lies on the circumference of a circle. The coordinates of the center of the circle may be given explicitly, or estimated from neighboring points. The radius of curvature is assumed to vary linearly along the centerline between known points. The radius of curvature is very large for straight sections and smaller for curved sections. This approach to grid generation accommodates a wide range of roadway geometry, and gives models developed from it a consistent basis.

The superelevation transition study also formulated kinematic boundary conditions for a 2D diffusion wave model using the method of characteristics. Boundary conditions for highway drainage can be quite complicated, especially in unsteady conditions. Making the kinematic approximation is often reasonable and provides at least some dynamic behavior at drainage boundaries. Applying the method of characteristics along the drainage path allows the boundary condition to be physically reasonable, and to vary in time.

2.6 Coupling Schemes

The need to couple fluid behavior on the surface with that in the subsurface comes from the hydrologic cycle. Rain falls on the earth's surface as precipitation and infiltrates the soil to become groundwater. Various approaches to coupling surface and subsurface flow have been proposed. An early study by Beavers and Joseph (1967) investigated the interface region and detected a slip velocity at the interface. In hydrologic models the conductance method (Anderson and Woessner, 1992) is widely used. In this method, the flux between the phases is the gradient times the conductance. This approach is acceptable for a distinct boundary between phases, but the high surface roughness of PFC blurs this boundary. Recently, Kollet and Maxwell (2006) proposed coupling the surface and subsurface by requiring the pressure to be constant right at the land surface.

2.7 Coupled Surface-Subsurface Models

There many examples of hydrologic models that couple surface and subsurface flow processes. Most models focus on flow in only one phase, and use the other phase as a boundary condition. For example, in an irrigation system, the detailed solution of the groundwater system is not terribly important; the objective is a good representation of surface flow and infiltration. In the same way, subsurface flow models such as MODFLOW focus on the solution to the groundwater system, which is usually unaffected by the sheet flow dynamics. In contrast, models of entire watersheds do attempt to represent surface flow, infiltration, and subsurface flow. However, a detailed solution for overland flow is rarely found along with a detailed groundwater solution. Two notable exceptions are discussed below.

Researchers at the University of Mississippi recently published a paper entitled "Coupled Finite-Volume Model for 2D Surface and 3D Subsurface Flows" (He et al., 2008). This model couples a diffusion wave model on the surface with Richard's equation in the subsurface. The coupling is accomplished by requiring the pressure to be continuous right at the land surface. This formulation treats overland flow as a boundary to subsurface flow. The model predicts the variation of surface water depth through time over the watershed.

The MIKE-SHE model—maintained by the Danish Hydrologic Institute, Inc (DHI)—is a commercial software package for watershed simulation. The model simulates the major hydrological processes that occur in the land phase of the hydrologic cycle, including surface flow and groundwater flow (Refsgaard and Storm, 1995). For coupling between surface and subsurface phases, the program calculates the exchange flux from Darcy's law. The MIKE-SHE model has been used widely to model many watersheds and is often used to evaluate new models (e.g. He et al., 2008).

Numerous models that couple surface and subsurface processes have been reviewed by Furman (2008). In his review, Furman categorizes models according to the type of surface flow and subsurface flow that the model uses. In his summary of 26 models, there are seven models that deal with surface flow in two dimensions—of these only one deals with the subsurface as a groundwater problem instead of only infiltration or partial saturation. The one model that does both is a unique application by Liang et al. (2007) where buildings in the floodplain are modeled as a porous medium. In their formulation, Liang et al. (2007) restrict the solution at any point in the system to either surface flow or subsurface flow. The coupling is horizontal; water from the flood wave flows laterally into the buildings.

2.8 Uniqueness of this Dissertation

This research shares many attributes with previous studies—predicting water depth and runoff from rainfall is essentially a hydrologic model. The original contribution of this work comes from several areas:

- The model predicts the transient response of PFC, which has yet to be addressed in the literature.
- The work examines a surface/subsurface flow system at the fine spatial scale of a roadway, in contrast to the watershed scale studies identified above.
- In the PFC system, subsurface flow drives overland flow. This forcing contrasts with the natural process of ponding from overland flow causing infiltration.

CHAPTER 3: MODEL DEVELOPMENT

This chapter describes the development of the permeable friction course drainage code (PERFCODE). A statement of the research problem is given first along with a discussion of the physical processes involved. With this basis, a mathematical formulation is developed for each physical process. A discussion of major assumptions is provided next. The mathematical models are applied on a control volume to formulate the numerical model that will provide the predictions of interest. The chapter concludes with a discussion of model tolerances and the technique used for the transition between sheet flow and PFC flow.

3.1 Problem Statement

The research problem is predicting the elevation of the water surface throughout a PFC roadway during a rainstorm. PFC is a permeable pavement placed in a 50mm layer on top of regular, impermeable pavement. During rain events, water seeps into the porous layer and flows to the side of the road by gravity. When the rainfall intensity is small, all of the drainage is contained within the pavement. Under higher rainfall intensities drainage occurs both within and on top of the pavement. The model predicts depths in both cases.

For the straight roadway shown in Figure 5, the road has a longitudinal slope and a cross slope. The resultant of these slopes is the drainage slope, along which water particles move to the edge of the pavement. For straight roadway sections without shoulders the problem is one dimensional along the drainage slope. However, the drainage problem becomes two-dimensional when shoulders have a different slope than the traffic lanes or when the roadway is curved. PFC is frequently used to improve driving conditions in these cases. Some specific configurations of interest are:

- Roadways with shoulders
- Curved sections
- Superelevation transitions
- Sag vertical curves

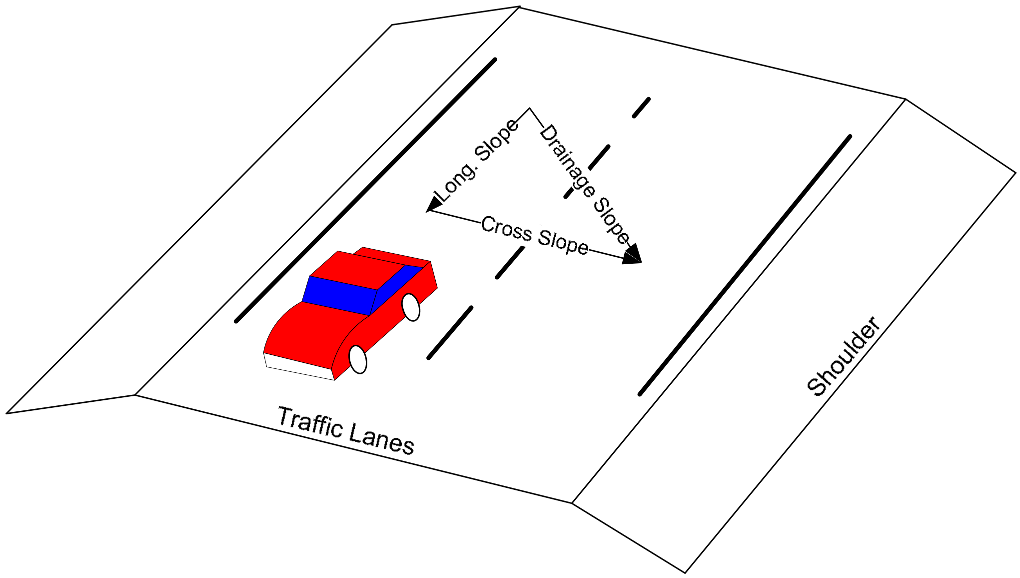

Figure 5: Straight roadway section

3.2 Physical Processes

In order to achieve the model aims, several physical processes must be considered. Modeling drainage from a PFC roadway can be considered as a specialized watershed model. As such, the physical processes may be categorized in terms of the hydrologic cycle. The hydrologic processes that occur in this system are: precipitation, evaporation, infiltration, unsaturated porous media flow, saturated porous media flow, and overland flow. One of these processes is important for the present work if it has a meaningful effect on the mass of water in the system or affects the travel time of a water particle moving through the system. The significance of each hydrologic process with respect to the model is evaluated in the following sub-sections.

3.2.1 Precipitation and Evaporation

Precipitation is the process by which water that has condensed in the atmosphere falls to earth. Precipitation can take the form of rain, sleet, snow or hail depending on atmospheric conditions. For the purposes of this research, rain is the only form of precipitation considered. The rainfall rate is a model input, assumed to be a known function of time.

Evaporation is the process of water changing from the liquid phase to the vapor phase. Key factors in determining the evaporation potential are the solar radiation and relative humidity (Charbeneau, 2000). In this work evaporation is neglected because most drainage occurs during or immediately following rainfall events while the relative humidity is high.

3.2.2 Infiltration

Infiltration is the process of rainfall entering the porous medium. Infiltration is governed by hydraulic conductivity, porosity and moisture content of the medium. For infiltration to be an important process with respect to PFC drainage, the process of water entering the pavement would have to cause a meaningful delay in the travel time of a water particle. Such a delay would cause water to pond on the pavement surface before the pore space was filled. According to the Green-Ampt method for calculating infiltration, ponding will not occur unless the rainfall intensity exceeds the hydraulic conductivity (Charbeneau, 2000). As an example, consider a five minute rainfall of one inch (2.54cm), which exceeds the 100-year 5-minute rainfall event for the entire eastern United States (Chow et al. 1988, pg 447). Such an event corresponds to a rainfall rate of 0.0085 cm/s--far below the 1 cm/s order of PFC hydraulic conductivity. Since the hydraulic conductivity of PFC is much higher than rainfall rates, infiltration is not expected to play an important role in this problem and is neglected in the model formulation.

3.2.3 Unsaturated Porous Media Flow

Although infiltration occurs very quickly for a PFC, unsaturated porous media flow from the pavement surface to the water table may play an important role. To quantify the effect of this process an estimate of the travel time for a range of rainfall intensities was made using Equations (2.17) and (2.18) and the results plotted in Figure 6.

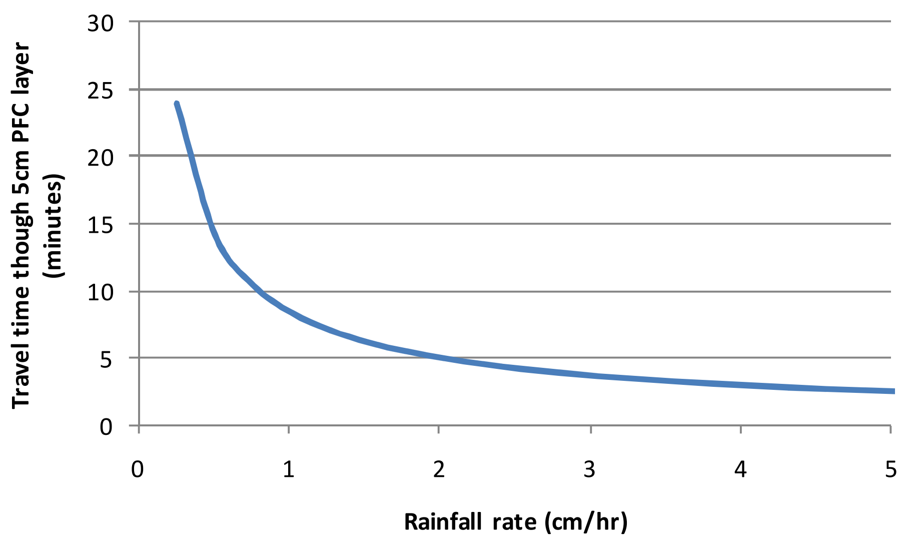

Figure 6: Travel time though an unsaturated PFC layer having a thickness of 5cm, irreducible water content of zero, pore size distribution index of 1.7, and a saturated hydraulic conductivity of 1 cm/s

Figure 6 shows that travel times are longer at lower rainfall intensities, but that the travel time is on the order of minutes. The significance of this delay depends on the model time step. Model time steps for this work are on the order of seconds, suggesting that the delay may be important. However, rainfall measurements necessarily report rainfall accumulation over a time period, frequently five or fifteen minutes. Considering the reporting period for rainfall data compared to the expected travel time, flow through the unsaturated PFC is neglected in this model.

3.2.4 Saturated Porous Media Flow

Saturated porous media flow refers to the movement of fluid through a porous medium when the pore space is filled with fluid. The boundary between saturated and unsaturated zones of a porous medium is the water table. At the water table, the pressure is atmospheric. Below the water table the media is saturated. Above the water table the media is considered unsaturated, though a small area of saturated pores may exist above the water table due to capillary effects. Saturated porous media flow is an essential process for the model because drainage to the edge of pavement occurs horizontally. This model treats all of the drainage through the PFC as saturated porous media flow.

Quantitative predictions of saturated porous media flow apply Darcy's law or Forchheimer's equation to relate the hydraulic gradient and the specific discharge. This model assumes that Darcy's law characterizes PFC drainage. The validity of this assumption is investigated in Section 3.4.2.

3.2.5 Overland Flow

Overland flow is the process of water flowing on the land surface, usually in a thin layer. Hydrologists categorize overland flow as either Hortonian overland flow or saturation overland flow (Chow et al., 1988). The distinction is the source of the flow. Hortonian overland flow occurs when the rainfall rate exceeds the infiltration capacity of the surface. Saturation overland flow occurs when the subsurface becomes saturated and discharges flow onto the land surface, usually at the bottom of a hill. In PFC drainage, overland flow occurs through the latter mechanism.

Overland flow velocities are generally much higher than subsurface flow velocities because viscous forces are smaller due to differences in surface area. Because of the higher velocities, overland flow drains water more quickly from the roadway than subsurface flow. The high drainage capacity of overland flow makes it an important process for modeling drainage from PFC roadways.

3.2.6 Summary of Physical Processes

The physical processes that occur during drainage from a PFC roadway have been identified and evaluated. The processes of precipitation, saturated porous media flow, and overland flow were found to be important for the current work. The interaction between these processes is shown in Figure 7.

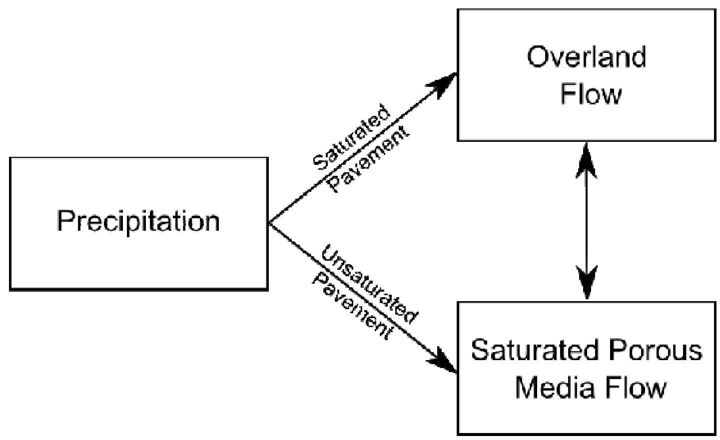

Figure 7: Interaction between physical processes in PERFCODE

3.3 Mathematical Model Development

Now that the important physical processes for PFC drainage have been identified, a mathematical description of each process is needed. For the precipitation process, the variation of rainfall over time is assumed to be known so no further description is required. Models for saturated porous media flow and overland flow are developed in the following sections. A sketch of the dimensional variables used to represent different physical quantities is shown in Figure 8.

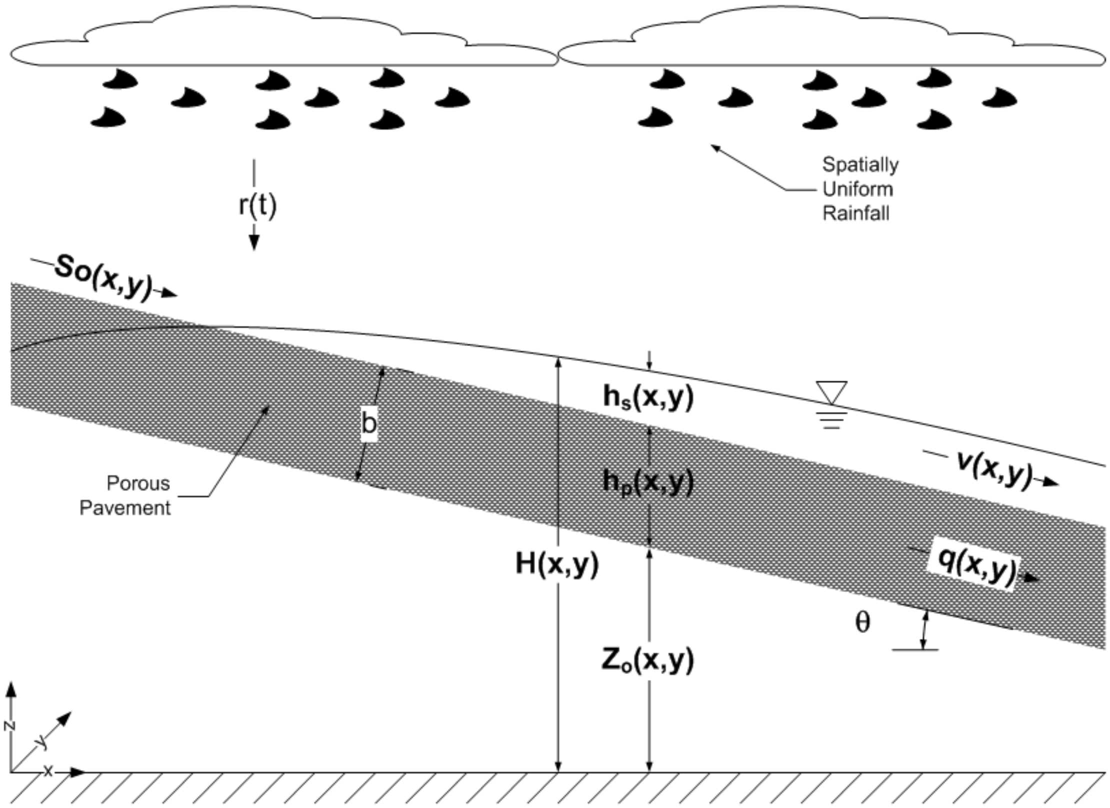

Figure 8: Cross section along drainage path

The rainfall rate $r(t)$ is assumed to be spatially uniform, but variable in time. The elevation of the bottom of the PFC layer with respect to a datum is $Z(x, y)$. The PFC layer has a thickness b, which is taken as constant throughout the domain. The saturated thickness of water in the PFC layer is $h_p(x, y)$ where the subscript refers to the pavement. The specific discharge through the PFC is $q(x, y)$. On the pavement surface, the thickness of sheet flow is h_s and the average velocity is $v(x, y)$. The total head of water at any point in the domain is $H(x, y)$.

3.3.1 Mathematical Model of Saturated Porous Media Flow

The equations of motion for saturated flow in a porous media consist of the continuity equation and the momentum equation. This development follows Halek and Svec (1979). Consider first the equation of continuity:

$$\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} = 0 \tag{3.1}$$

where q is the Darcy velocity in each of the coordinate directions. If the drainage slope is small enough, the only vertical fluxes are from rainfall or movement of the free surface. In the present problem, rainfall is prescribed and the free surface position is of interest. Integrating the continuity equation over the saturated thickness gives:

$$\int_0^{h_p} \left(\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} \right) dz = \frac{\partial}{\partial x}\left(q_x h_p\right) + \frac{\partial}{\partial y}\left(q_y h_p\right) + q_{h_p} - q_0 \tag{3.2}$$

This integration makes use of Leibnitz's rule to interchange the order of differentiation and integration. By assuming that the PFC has no resistance to flow in the vertical direction, the effects free surface movement and rainfall may be separated into q_{hp} and q_0, respectively. The movement of the free surface (within the PFC) in time is given by $q_{h_p} = n_e \frac{\partial h_p}{\partial t}$ and the rainfall may be expressed as $q_0 = r(t)$. Making these substitutions and rearranging:

$$n_e \frac{\partial h_p}{\partial t} = -\frac{\partial}{\partial x}\left(q_x h_p\right) - \frac{\partial}{\partial y}\left(q_y h_p\right) + r(t) \tag{3.3}$$

For the case of non-inertial flow, the momentum equation reduces to Darcy's law for each coordinate direction.

$$q_x = -K_x \frac{\partial H}{\partial x}, \qquad q_y = -K_y \frac{\partial H}{\partial y} \tag{3.4}$$

where q and K are the Darcy velocity and hydraulic conductivity in the coordinate directions. For the present case, horizontal anisotropy will be neglected so that $K_x = K_y = K$. Substituting Darcy's law into the vertically integrated continuity equation gives:

$$n_e \frac{\partial h_p}{\partial t} = K\left[\frac{\partial}{\partial x}\left(\frac{\partial H}{\partial x} h_p\right) + \frac{\partial}{\partial y}\left(\frac{\partial H}{\partial x} h_p\right)\right] + r(t) \tag{3.5}$$

Equation (3.5) is known as the Boussinesq equation. It describes unsteady two-dimensional flow in an unconfined porous medium with spatially uniform recharge.

3.3.2 Mathematical Model of Overland Flow

The following development of the mathematical model for overland flow follows that of Jeong (2008), except that the velocity is used as the primary variable rather than unit discharge. The dynamics of shallow water flow over the pavement surface are described by the Saint-Venant equations, which comprise a continuity equation and a momentum equation for each component direction. The continuity equation is expressed as:

$$\frac{\partial h_s}{\partial t} + \frac{\partial (v_x h_s)}{\partial x} + \frac{\partial (v_y h_s)}{\partial y} = r(t) \tag{3.6}$$

where h_s is the thickness of water on the surface, v is the average velocity in each coordinate direction, and $r(t)$ is the rainfall rate. The two full momentum equations are:

$$\begin{aligned} &\frac{\partial (v_x h_s)}{\partial t} + \frac{\partial ({v_x}^2 h_s)}{\partial x} + \frac{\partial (v_x v_y h_s)}{\partial y} + g h_s \left(Sf_x + \frac{\partial H}{\partial x}\right) = 0 \\ &\frac{\partial (v_y h_s)}{\partial t} + \frac{\partial ({v_y}^2 h_s)}{\partial y} + \frac{\partial (v_x v_y h_s)}{\partial x} + g h_s \left(Sf_y + \frac{\partial H}{\partial y}\right) = 0 \end{aligned} \tag{3.7}$$

This system of three partial differential equations may be reduced to a single equation by applying the diffusion wave approximation—neglecting local and convective accelerations. Neglecting inertial terms and dividing by $g\ h_s$ gives the simplified momentum equations:

$$S_{f_x} = -\frac{\partial H}{\partial x} \qquad S_{f_y} = -\frac{\partial H}{\partial y} \tag{3.8}$$

To combine continuity and momentum into a single equation, the velocity components (v_x and v_y) must be expressed in terms of the friction slope. Manning's equation relates the velocity and friction slope as follows:

$$v = \frac{1}{n} R^{2/3} S_f^{1/2} \tag{3.9}$$

Where v is the velocity, n is the Manning roughness coefficient, R is the hydraulic radius, and S_f is the friction slope. Manning's equation is a scalar equation that applies in the direction of flow. In order to apply the Manning's equation to this problem it needs to be formulated using the vector components of Equation (3.7). Inserting these components and approximating the hydraulic radius as the depth as is common for shallow flows yields:

$$\left(v_x^2 + v_y^2\right)^{1/2} = \frac{1}{n} h_s^{2/3} \left({S_f}_x^2 + {S_f}_y^2\right)^{1/2} \tag{3.10}$$

The friction slope term may also be expressed in terms of both vector components and the magnitude:

$$\left(v_x^2 + v_y^2\right)^{1/2} = \frac{1}{n} \frac{h_s^{2/3}}{\sqrt{Sf}} \left({S_f}_x^2 + {S_f}_y^2\right) \tag{3.11}$$

This formulation shows that Manning's equation can be written as the vector sum of the velocity components. Using the momentum result of Equation (3.8), the friction slope may also be written in terms of the hydraulic gradient.

$$\begin{aligned} v_x &= \frac{1}{n} \frac{h_s^{2/3}}{\sqrt{S_f}} {S_f}_x = -\frac{1}{n} \frac{h_s^{2/3}}{\sqrt{S_f}} \frac{\partial H}{\partial x} \\ v_y &= \frac{1}{n} \frac{h_s^{2/3}}{\sqrt{S_f}} {S_f}_y = -\frac{1}{n} \frac{h_s^{2/3}}{\sqrt{S_f}} \frac{\partial H}{\partial y} \end{aligned} \tag{3.12}$$

Substituting these velocity components into the continuity equation yields a single partial differential equation that contains the essential physics of the overland flow problem.

$$\frac{\partial h_s}{\partial t}+\frac{\partial}{\partial x}\left(-\frac{1}{n}\frac{h_s^{2/3}}{\sqrt{S_f}}\frac{\partial H}{\partial x}h_s\right)+\frac{\partial}{\partial y}\left(-\frac{1}{n}\frac{h_s^{2/3}}{\sqrt{S_f}}\frac{\partial H}{\partial y}h_s\right)=r(t) \qquad (3.13)$$

This equation may be simplified by lumping the non-differential terms within the spatial derivatives into a single coefficient, $D(h_s)$. Additionally, the time derivative must be converted from depth to elevation above datum. From Figure 8 the variables are related by $H = Z + h_p + h_s$. Taking the time derivative, dz/dt is zero and $\frac{\partial}{\partial t}h_p$ is zero when there is flow on the surface. That is, during surface flow, the saturated depth of the PFC will be equal to the pavement thickness. Making these substitutions gives the desired PDE:

$$\frac{\partial H}{\partial t}+\frac{\partial}{\partial x}\left(-D(h_s)\frac{\partial H}{\partial x}\right)+\frac{\partial}{\partial y}\left(-D(h_s)\frac{\partial H}{\partial y}\right)= r(t) \qquad (3.14)$$

where $D(h_s)=\frac{1}{n}\frac{h_s^{5/3}}{\sqrt{S_f}}$ and other terms are defined previously. This approach to describing surface flow is a two-dimensional diffusion wave model.

3.4 Mathematical Model Assumptions

The forgoing development made simplifying assumptions about the physical system. In particular it was assumed that the saturated subsurface varies hydrostatically, that porous media flow is slow enough to neglect inertial effects, and that inertial effects can also be neglected for overland flow. Each of these assumptions is discussed in the following sections.

3.4.1 Dupuit-Forchheimer Assumptions

In developing the mathematical model for saturated porous media flow, it was assumed that pressure varied hydrostatically and that the subsurface discharge was proportional to the hydraulic gradient. These are the Dupuit-Forchheimer assumptions.

Irmay (1967) studied the error made in predicting the hydraulic head using the Dupuit-Forchheimer assumptions. He gives formulas for computing the relative error at different depths for flat and inclined aquifers. For a flat aquifer, the maximum error occurs at mid depth and depends mostly on the hydraulic gradient. A hydraulic gradient of 10% caused a maximum error of 0.25% in the hydraulic head. As most roadways have a drainage slope smaller than 10%, the Dupuit-Forchheimer assumptions provide a good approximation.

3.4.2 Darcy's Law

Along with the Dupuit-Forchheimer assumptions, the model development assumed that Darcy's law applies for flow through PFC. However, experimental efforts to estimate the hydraulic conductivity of PFC have shown that Darcy's law does not apply once hydraulic gradients become sufficiently large (Klenzendorf, 2010). Forchheimer's equation is frequently used to describe flow in this case:

$$I = \alpha q_F + \beta q_F^2 \tag{3.15}$$

In Equation (3.15), I is the hydraulic gradient taking a downward slope as positive, q_F is the specific discharge of the fluid as predicted by the Forchheimer equation, and α and β are coefficients. In the case that β is zero, Forchheimer's equation reduces to Darcy's law with the coefficient α equal to the inverse of the hydraulic conductivity K. To facilitate a comparison with Darcy's law, the Forchheimer specific discharge q_F is obtained using the quadratic formula. The positive radical is taken since a negative discharge is not meaningful in this case.

$$q_F = \frac{-\alpha + \sqrt{\alpha^2 + 4\beta I}}{2\beta} = \frac{\alpha}{2\beta}\left[\sqrt{1 + \frac{4\beta I}{\alpha^2}} - 1\right] \tag{3.16}$$

Using this form of Forchheimer's equation, a vector form comparable to Darcy's law may be obtained:

$$\bar{q}_F = \bar{I}\frac{\alpha}{2\beta I}\left[\sqrt{1 + \frac{4\beta I}{\alpha^2}} - 1\right] \tag{3.17}$$

Since Darcy's law is $\bar{q} = K\bar{I}$, the specific discharge predicted by the two equations can be compared using a ratio, termed the Discharge Ratio (Φ).

$$\Phi = \frac{q_F}{q_D} = \frac{\alpha^2}{2\beta I}\left[\sqrt{1+\frac{4\beta I}{\alpha^2}} - 1\right] \tag{3.18}$$

The value of Φ ranges from 0 to 1. At a value of 1 the Forchheimer specific discharge matches the Darcy specific discharge. At values less than 1, the Forchheimer specific discharge is less than the Darcy specific discharge. The value of Φ depends upon the hydraulic gradient I and the coefficients α and β. A change in one of these variables that results in a higher velocity pushes the flow away from the Darcy regime toward Forchheimer flow.

For the present purposes, the region of applicability of Darcy's law is of interest. To determine this region, the value of Φ over a range of values for I, α & β is investigated. The hydraulic gradient can be estimated as the roadway slope. A reasonable slope range might be 0% to 10%. Values of α can be approximated by taking the inverse of the hydraulic conductivity. The hydraulic conductivity of PFC is an area of ongoing research. Preliminary results indicate that values range from 0 to 5 cm/s. Values of β are estimated using equations from the literature and compared to recent experimental results.

Li and Engler (2001) give a literature review of correlations for the Non-Darcy coefficient. Of the correlations they give, an extension of the work of Ergun (1952) given by Thauvin and Mohanty (1998) appeared relevant to this research:

$$\hat{\beta} = ab^{-1/2}(10^{-8}k)^{-1/2}\phi^{-3/2} \tag{3.19}$$

where $\hat{\beta}$ is the non-Darcy coefficient in 1/cm, k is the permeability in units of Darcy, ϕ is the porosity. The values of $a = 1.75$ and $b = 150$ were obtained by Ergun (1952) using a least squares fit to experimental data. This correlation was chosen because the experimental data come from columns packed with porous materials (e.g. sand, pulverized coke) rather than geologic formations. The non-Darcy coefficient is related to β by the constant of gravitational acceleration:

$$\beta = \frac{\hat{\beta}}{g} \tag{3.20}$$

Another correlation for the coefficients of the Forchheimer equation is given by Ward (1964):

$$I = \frac{\mu}{k\rho g}q + \frac{0.55}{g\sqrt{k}}q^2 \tag{3.21}$$

In Ward's equation, the linear term is consistent with Darcy's law and no estimate of the porosity is required.

Recent work by Klenzendorf (2010) has used a combination of numerical modeling and laboratory experiments to determine the Forchheimer coefficients for PFC. Comparing the coefficients obtained by Klenzendorf to the relationships proposed by Ward and Thauvin and Mohanty suggests that Ward's equation provides better estimates for PFC flow (Figure 9). This result applies especially at higher values of hydraulic conductivity, where non-linear effects are more pronounced.

A comparison of the value of β with the hydraulic conductivity shows that the variables are inversely related (Figure 9). Conceptually, this relationship says that smaller values of hydraulic conductivity have higher values of β. The meaning of this trend is that inertial effects reduce the drainage capacity of PFC. Darcy's law will under-predict the water depth.

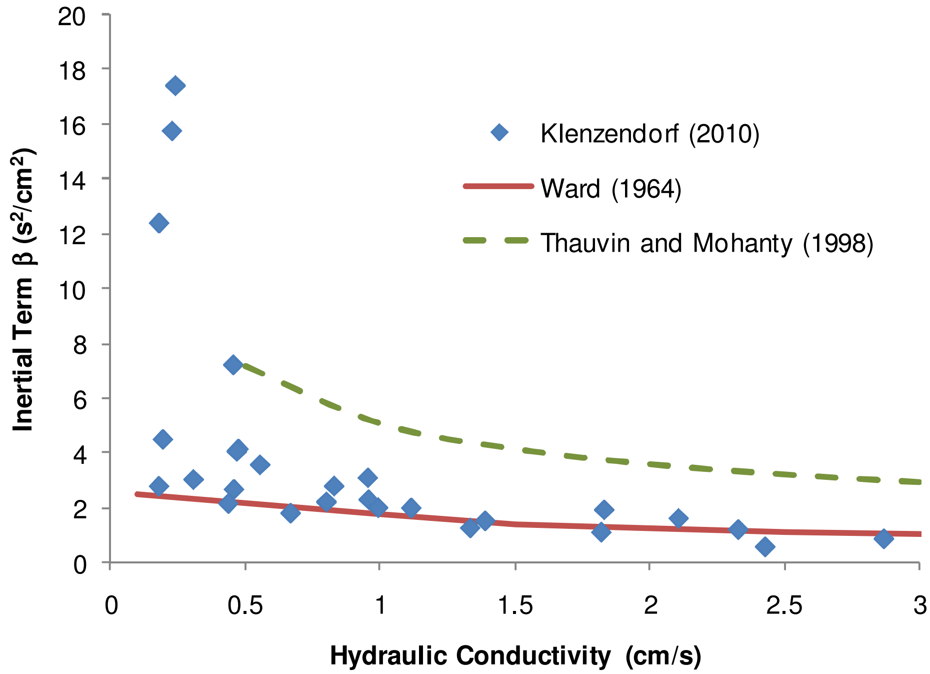

Figure 9: Comparison of Forchheimer coefficients for PFC obtained by Klenzendorf (2010) with the relationships proposed by Ward (1964) and Thauvin and Mohanty (1998). Three of Klenzendorf's data points [(0.047,167); (0.056,64.3); (0.10,29.1)] are excluded for clarity.

Invoking either relationship for the Forchhiemer coefficients reduces the discharge ratio to a function of two variables. By establishing a threshold value for Φ, we can get a sense of which PFC roadways can be reasonably represented by Darcy's law. A 10% non-Darcy effect—corresponding to $\Phi = 0.9$ —has been suggested as reasonable (Zeng and Grigg, 2006) and is adopted here. Using this criterion, a surface plot of the discharge ratio shows that Darcy's law provides acceptable predictions at low hydraulic gradients (small slopes) and small hydraulic conductivities (Figure 10). Furthermore, this figure shows that even modest roadway slopes can lead to non-Darcy flow.

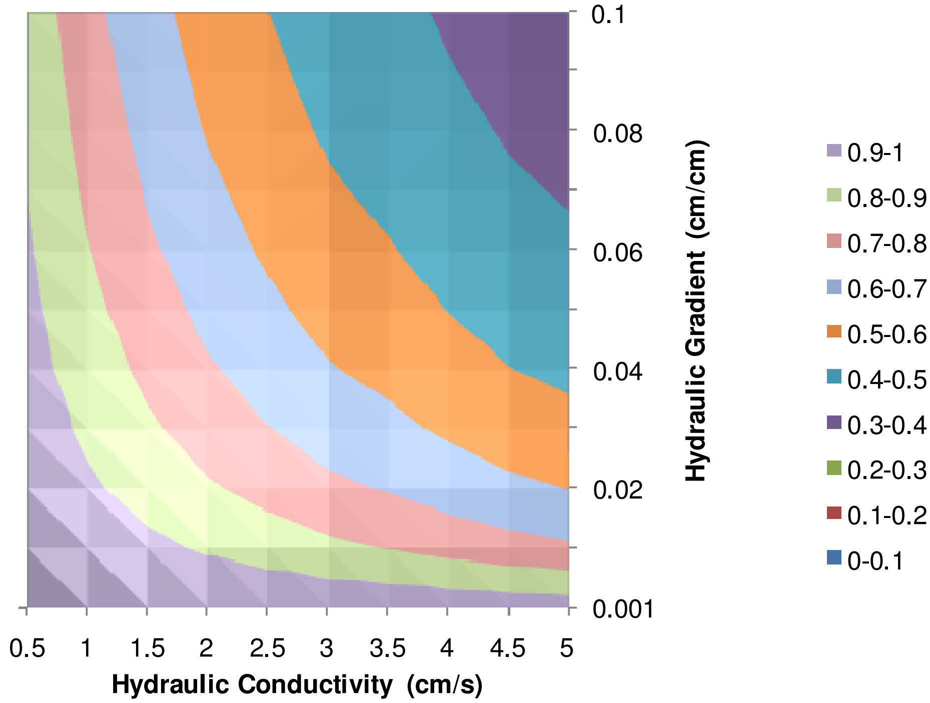

Figure 10: Contour plot of discharge ratio using Thauvin and Mohanty (1998) with porosity of 0.2.

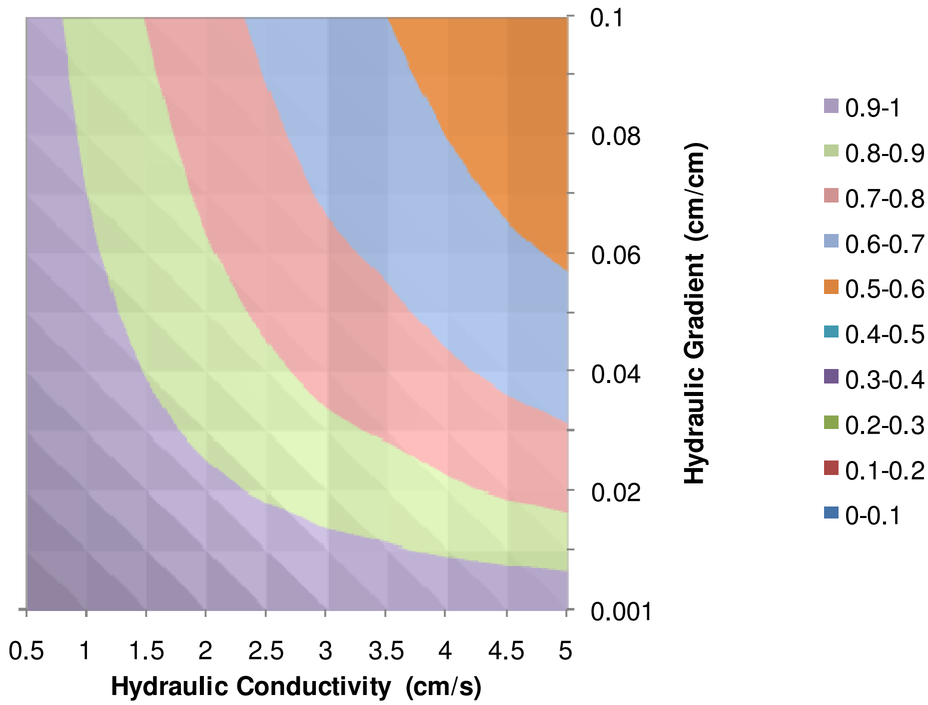

Figure 11: Contour plot of discharge ratio using the relationship of Ward (1964)

The contour plot of the discharge ratio using Ward's formula (Figure 11) shows the same general trends as Figure 10, but Ward's formula—which agrees more closely with experimental data for PFC—gives a larger region where Darcy's law is acceptable.

3.4.3 Diffusion Wave Approximation

The reasons for selecting the diffusion wave approximation are discussed more thoroughly in the literature review. Briefly, the diffusion wave model provides a balance between accuracy and computational efficiency. The kinematic wave approximation is too simplified because it cannot deal with adverse slopes or backwater effects. The dynamic wave model would be ideal, but comes at a high computational cost and is not expected to give substantially different results than the diffusion wave model.

3.5 Computational Grid

In order to implement the mathematical models of the physical processes for real roadways, a computational grid for the roadway must be developed. This research uses the same grid generation employed by Jeong (2008), which is summarized below.

The idea of the grid generation scheme is that each point along a roadway centerline lies on the circumference of a circle. The coordinates of the center of the circle may be given explicitly, or estimated from neighboring points. The radius of curvature is assumed to vary linearly along the centerline between known points. The radius of curvature is very large for straight sections and smaller for curved sections.

The center and radius of curvature can be obtained by specifying them directly as was done in this work, or by analyzing a digital elevation model as was done by Jeong (2008). In either approach, a point along the roadway centerline has the following attributes:

- Cartesian X,Y coordinates (input)
- Coordinates of center of curvature, (x_{cc}, y_{cc}) (output)
- Radius of curvature, R (output)

- Angle (from positive horizontal axis) of ray from center of curvature to centerline point, Θ (output)

Considering adjacent DEM points, the difference in radius of curvature and angular position are ΔR and $\Delta\Theta$, respectively. Using these quantities the curvilinear roadway can be mapped to a rectangular representation through the coordinate transformation functions (Jeong 2008):

$$\begin{aligned} x(\xi,\eta) &= \left(\mathrm{x}_{cc1} + \xi(\mathrm{x}_{cc2} - \mathrm{x}_{cc1})\right) + (\mathrm{R}_1 + \xi\Delta\mathrm{R} + (\eta - 0.5)\mathrm{W})\cos(\Theta_1 + \xi\Delta\Theta) \\ y(\xi,\eta) &= (\mathrm{y}_{cc1} + \xi(\mathrm{y}_{cc2} - \mathrm{y}_{cc1})) + (\mathrm{R}_1 + \xi\Delta\mathrm{R} + (\eta - 0.5)\mathrm{W})\sin(\Theta_1 + \xi\Delta\Theta) \end{aligned} \tag{3.22}$$

In Equation (3.22), ξ and η are parameters that range from 0 to 1; W is the width of the roadway. This equation only applies between adjacent DEM points.

The length ℓ, and width w of a line segment centered at the point (ξ, η) are computed using the partial derivatives of the coordinate transformation functions:

$$\begin{aligned} \ell(\xi,\eta) &= \Delta\xi\sqrt{\left(\frac{\partial x}{\partial \xi}\right)^2 + \left(\frac{\partial y}{\partial \xi}\right)^2} \\ w(\xi,\eta) &= W\Delta\eta \end{aligned} \tag{3.23}$$

with $\Delta\xi = 1/N_\xi$ and $\Delta\eta = 1/N_\eta$, N being the number of elements between DEM points in each direction.

The area of a grid cell is computed from the Jacobian of the transformation functions:

$$\Delta A = J(\xi,\eta) = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{vmatrix} \tag{3.24}$$

Equations (3.23) and (3.24) provide the information needed to develop a numerical formulation in the computational space. The coordinate transformation process is depicted visually in Figure 12.

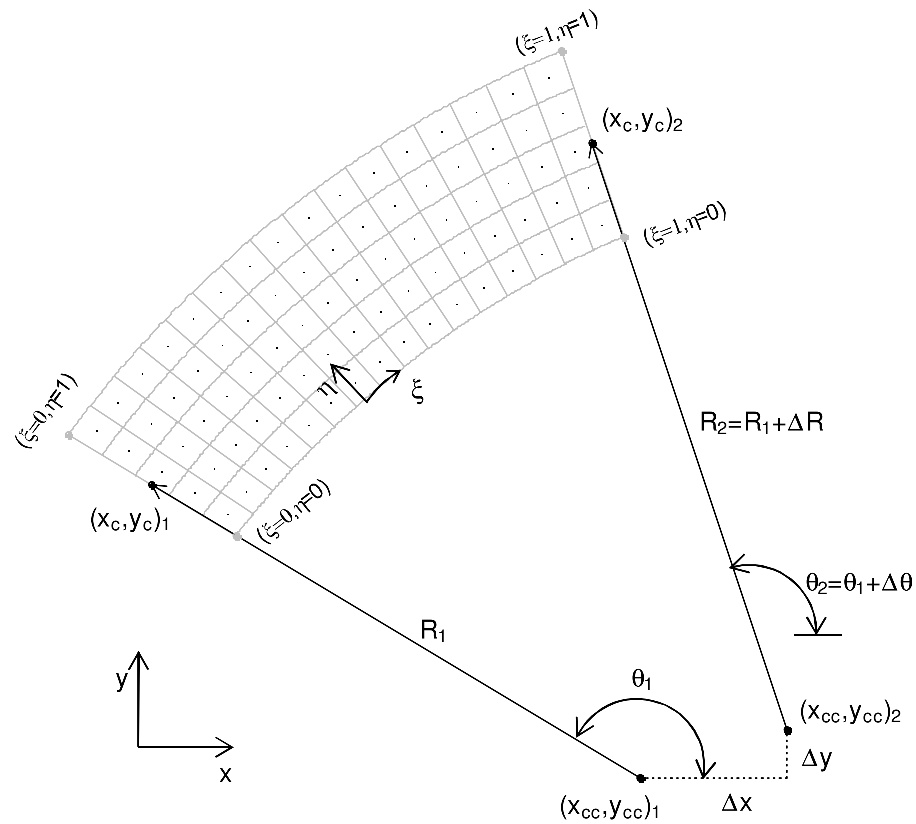

Figure 12: Development of computational grid from roadway geometry

3.6 Numerical Formulation

The major goal of this research is the development of a numerical model for the drainage of water from a PFC. The Boussinesq equation and the diffusion wave model developed above provide the theoretical basis for the system of interest. However, predicting flow behavior in a real system requires that the surface and subsurface behaviors interact.

The numerical formulation uses the finite volume method with central differencing in space and the Crank-Nicolson method in time. A mass balance is developed for an interior grid cell with flux components for rainfall, subsurface flow, and surface flow. The flux across each face of the grid cell is estimated using Darcy's law and the diffusion wave model. The mass balance is initially expressed in terms of the total head at adjacent cells and then re-expressed in terms of the depth at adjacent cells.

3.6.1 Mass Balance on a Grid Cell

An interior grid cell is shown in Figure 14 and Figure 14 with horizontal dimensions in computational space. The total head for the center of the grid cell is:

$$H = z + h_p + h_s \tag{3.25}$$

where z is the elevation above the datum, h_p is the saturated thickness in the pavement and h_s is the thickness on the pavement surface. The volume of the grid cell is:

$$V = Area * Depth = \Delta A(H - z) = \Delta A(h_p + h_s) \tag{3.26}$$

The volume of *water* in the grid cell must account for the porosity, and is given by:

$$V_{H_2O} = \Delta A h_p n_e + \Delta A h_s \tag{3.27}$$

where n_e is the effective porosity of the pavement.

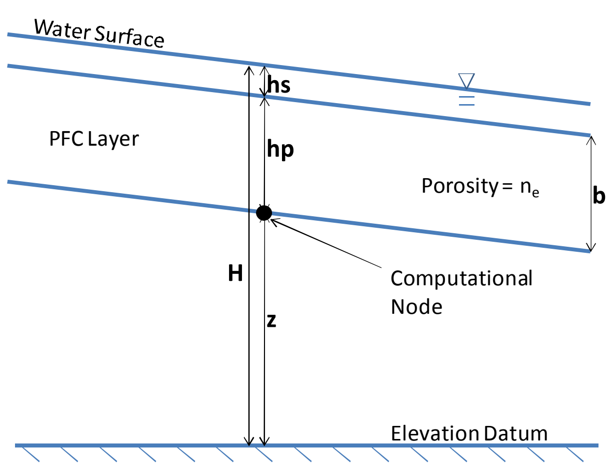

Figure 13: Profile view of interior grid cell

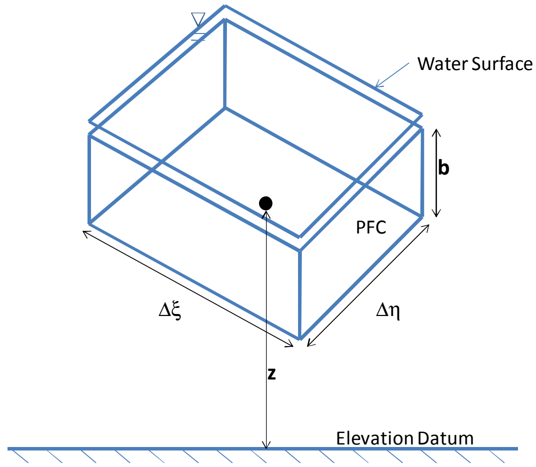

Figure 14: Isometric View of Interior Grid Cell

The change in volume of water in the cell over time is found from the partial derivative of Equation (3.27). This derivative must consider the physical constraint that either $\frac{\partial h_p}{\partial t}$ or $\frac{\partial h_s}{\partial t}$ will be zero at all times according to the location of the free surface with respect to the pavement surface.

$$\frac{\partial V_{H_2O}}{\partial t} = \begin{cases} \Delta A\, n_e \frac{\partial h_p}{\partial t} & for\ h_p < b \\ \Delta A \frac{\partial h_s}{\partial t} & for\ h_p \geq b \end{cases} \tag{3.28}$$

The principle of continuity states that the time rate of change of volume is equal to the net flow rate, which can be expressed mathematically as:

$$\frac{\partial V_{H_2O}}{\partial t} = Q_{in} - Q_{out} \tag{3.29}$$

The volume of water in the cell changes by rainfall, subsurface flow, and surface flow. Flow into the grid cell is considered positive. To estimate the flow rate due to each component, consider an interior control volume and its adjacent cells as in Figure 15. The central cell in the figure has node i, j at the center. The faces of the center cell are identified with the compass directions.

Note that the grid in computational space is uniform—each cell has the same value of $\Delta\eta$ and $\Delta\xi$ and the grid is situated so that the cell faces lie halfway between the cell centers. The grid in physical space is not uniform because cells have different lengths in the longitudinal direction according their radial position. In the figure, the subscripts of $\Delta\eta$ and $\Delta\xi$ refer to the metric coefficients, which do vary in space.

In the indexing scheme for the model, the *i* index changes longitudinally through the domain and the *j* index changes transversely. These indices are related to the compass directions within a grid cell for convenience. In terms of coordinate directions, the local north and south compass directions correspond to the positive and negative η directions.

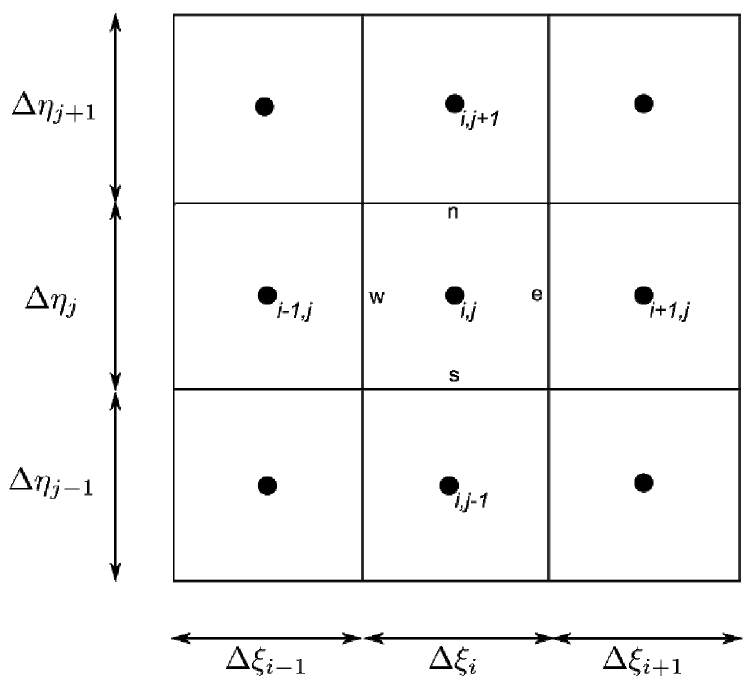

Figure 15: Top View of Grid in Computational Space

For cell i, j the flow rate due to rainfall is given by the rainfall intensity and the cell area:

$$Q_{rain} = r(t) * \Delta\mathrm{A} \tag{3.30}$$

The flow rate due to subsurface flow can be estimated using Darcy's law, ($Q = KIA$), where K is the hydraulic conductivity, I is the hydraulic gradient, and A is the cross sectional area. The hydraulic gradient and cross sectional area must be estimated using the physical lengths of the cells. Considering Figure 15, the head gradient with respect to ξ at location w can be approximated as:

$$\left.\frac{\partial H}{\partial \xi}\right|_{w} = \frac{H_{i-1,j} - H_{i,j}}{1/2\,(\Delta\xi_{\mathrm{i-1}} + \Delta\xi_{\mathrm{i}})} \tag{3.31}$$

Since ξ is dimensionless, this equation does not have the dimensions of hydraulic gradient. In order to estimate the hydraulic gradient at cell face w, cell size computed in Equation (3.23) must be used. Applying the transformation gives an estimate for the hydraulic gradient:

$$\left.\frac{\partial H}{\partial \ell}\right|_{w} = \frac{H_{i-1,j} - H_{i,j}}{1/2\,(\ell_{\mathrm{i-1,j}} + \ell_{\mathrm{i,j}})} \tag{3.32}$$

Using this formulation for the hydraulic gradient, the subsurface flow into the each face of cell i, j is expressed:

$$\begin{aligned}
Q_{p,w} &= K\frac{H_{i-1,j} - H_{i,j}}{1/2\,(\ell_{\mathrm{i-1,j}} + \ell_{\mathrm{i,j}})} h_{p,w} w_{i,j} \\
Q_{p,e} &= K\frac{H_{i+1,j} - H_{i,j}}{1/2\,(\ell_{\mathrm{i+1,j}} + \ell_{\mathrm{i,j}})}\; h_{p,e} w_{\mathrm{i,j}} \\
Q_{p,s} &= K\frac{H_{i,j-1} - H_{i,j}}{1/2\,(w_{\mathrm{i,j-1}} + w_{\mathrm{i,j}})}\; h_{p,s} \ell_{\mathrm{i,j,s}} \\
Q_{p,n} &= K\frac{H_{i,j+1} - H_{i,j}}{1/2\,(w_{\mathrm{i,j+1}} + w_{\mathrm{i,j}})}\; h_{p,n}\, \ell_{\mathrm{i,j,n}}
\end{aligned} \tag{3.33}$$

Here the hydraulic gradient at the cell boundary is estimated as the difference in head divided by the distance between nodes. The cross sectional area is the saturated thickness times the length of the cell boundary. The length of the cell boundary has the same value for the east and west faces ($\mathcal{w}_{i,j}$), but differs for the north and south faces $\left(\ell_{i,j,s}\ or\ \ell_{i,j,n}\right)$ because the radius of curvature is different.

The flow rates due to surface flow can be estimated using the diffusion wave model according to the equation:

$$Q = V * A = \frac{1}{n}\frac{h_s^{\frac{2}{3}}}{\sqrt{S_f}}\frac{\partial H}{\partial x} * h_s \Delta \mathrm{y} \tag{3.34}$$

Here, h_s is the thickness on the pavement surface and S_f is the magnitude of the slope of the water surface. Using the same estimate of the hydraulic gradient as for subsurface flow gives the following estimates for the flow rate into cell i, j at each of the cell boundaries.

$$\begin{aligned}
Q_{s,w} &= \frac{1}{n}\frac{h_{s,w}^{\frac{2}{3}}}{\sqrt{S_{f,w}}}\left(\frac{H_{i-1,j} - H_{i,j}}{1/2\left(\ell_{\mathrm{i-1,j}} + \ell_{\mathrm{i,j}}\right)}\right) * h_{s,w}\mathcal{w}_{i,j} \\
Q_{s,e} &= \frac{1}{n}\frac{h_{s,e}^{\frac{2}{3}}}{\sqrt{S_{f,e}}}\left(\frac{H_{i-1,j} - H_{i,j}}{1/2\left(\ell_{\mathrm{i+1,j}} + \ell_{\mathrm{i,j}}\right)}\right) * h_{s,e}\mathcal{w}_{i,j} \\
Q_{s,s} &= \frac{1}{n}\frac{h_{s,s}^{\frac{2}{3}}}{\sqrt{S_{f,s}}}\left(\frac{H_{i-1,j} - H_{i,j}}{1/2\left(\mathcal{w}_{\mathrm{i,j-1}} + \mathcal{w}_{\mathrm{i,j}}\right)}\right) * h_{s,s}\ell_{\mathrm{i,j,s}} \\
Q_{s,n} &= \frac{1}{n}\frac{h_{s,n}^{\frac{2}{3}}}{\sqrt{S_{f,n}}}\left(\frac{H_{i-1,j} - H_{i,j}}{1/2\left(\mathcal{w}_{\mathrm{i,j+1}} + \mathcal{w}_{\mathrm{i,j}}\right)}\right) * h_{s,n}\ell_{\mathrm{i,j,n}}
\end{aligned} \tag{3.35}$$

Now that flow rates for each cell boundary have been developed, the water balance on a grid cell can be expressed in terms of the flow rates. All of the flow rates are formulated as being positive because of the arrangement of the $H_{i,j}$ term. If the head in cell i, j is lower than the cell it is subtracted from, water will flow into cell i, j. The flow rates were formulated this way to make it easier to check the equations. For the 2D case, the mass balance has nine flow components:

$$\frac{\partial V_{H_2O}}{\partial t} = Q_{p,w} + Q_{s,w} + Q_{p,e} + Q_{s,e} + Q_{p,s} + Q_{s,s} + Q_{p,n} + Q_{s,n} + Q_{rain}$$

or (3.36)

$$\frac{\partial V_{H_2O}}{\partial t} = Q_{p,w} + Q_{s,w} + Q_{p,e} + Q_{s,e} + Q_{rain}$$

Substituting the flow rates for rainfall, subsurface, and surface flow into the continuity equation gives a mass balance for an interior grid cell:

$$\begin{aligned}
\frac{\partial V_{H_2O}}{\partial t} = {} & \\
& K \frac{H_{i-1,j} - H_{i,j}}{1/2\left(\ell_{\mathrm{i-1,j}} + \ell_{\mathrm{i,j}}\right)} h_{p,w} w_{i,j} + \frac{1}{n} \frac{h_{s,w}^{\frac{2}{3}}}{\sqrt{S_{f,w}}} \left(\frac{H_{i-1,j} - H_{i,j}}{1/2\left(\ell_{\mathrm{i-1,j}} + \ell_{\mathrm{i,j}}\right)} \right) * h_{s,w} w_{i,j} \\
& + K \frac{H_{i+1,j} - H_{i,j}}{1/2\left(\ell_{\mathrm{i+1,j}} + \ell_{\mathrm{i,j}}\right)} h_{p,e} w_{\mathrm{i,j}} + \frac{1}{n} \frac{h_{s,e}^{\frac{2}{3}}}{\sqrt{S_{f,e}}} \left(\frac{H_{i-1,j} - H_{i,j}}{1/2\left(\ell_{\mathrm{i+1,j}} + \ell_{\mathrm{i,j}}\right)} \right) * h_{s,e} w_{i,j} \\
& + K \frac{H_{i,j-1} - H_{i,j}}{1/2\left(w_{\mathrm{i,j-1}} + w_{\mathrm{i,j}}\right)} h_{p,s} \ell_{\mathrm{i,j,s}} + \frac{1}{n} \frac{h_{s,s}^{\frac{2}{3}}}{\sqrt{S_{f,s}}} \left(\frac{H_{i-1,j} - H_{i,j}}{1/2\left(w_{\mathrm{i,j-1}} + w_{\mathrm{i,j}}\right)} \right) \\
& \quad * h_{s,s} \ell_{\mathrm{i,j,s}} \\
& + K \frac{H_{i,j+1} - H_{i,j}}{1/2\left(w_{\mathrm{i,j+1}} + w_{\mathrm{i,j}}\right)} h_{p,n} \ell_{\mathrm{i,j,n}} + \frac{1}{n} \frac{h_{s,n}^{\frac{2}{3}}}{\sqrt{S_{f,n}}} \left(\frac{H_{i-1,j} - H_{i,j}}{1/2\left(w_{\mathrm{i,j+1}} + w_{\mathrm{i,j}}\right)} \right) \\
& \quad * h_{s,n} \ell_{\mathrm{i,j,n}} \\
& + r(t) * \Delta \mathrm{A}
\end{aligned} \tag{3.37}$$

Equation (3.37) contains four dependent variables: $V_{H_2O}, H, h_p, and\ h_s$. A fifth variable, the total thickness h, may be formed as the sum of the thickness in the pavement and the thickness on the surface.

$$h = h_p + h_s \tag{3.38}$$

So the total head is:

$$H = z + h \tag{3.39}$$

In order to solve the problem, Equation (3.37) must be expressed in terms of the total head or total thickness. Choosing the total head is perhaps more intuitive, and makes the equations simpler, but the total thickness is a better choice numerically because it avoids subtracting two large numbers (the elevation being much larger than the total thickness). The equation will be expressed first in terms of the head, and then expressed again in terms of the thickness.

3.6.2 Formulation using Total Head

To express the equations in terms of the head, h_s , h_p and S_f must be expressed at the cell center and the boundaries in terms of H. Each of these terms will be examined in turn, starting with those on right hand side of Equation (3.37). In the development, it will also be convenient to define conveyance coefficients and a porosity function.

Saturated Thickness and Sheet Flow Depth

The saturated thickness at the grid cell boundaries—$h_{p,*}$—can be estimated from the total head at the cell centers by linear interpolation. Since the computational grid is evenly spaced, the interpolation is just the average of the head values. To find the saturated thickness at the boundary, the total head at the cell boundary is estimated from the adjacent nodes, and the elevation at the boundary is subtracted to give the saturated thickness:

$$\begin{aligned}
h_{p,w} &= \frac{\mathrm{H_{i,j}}\ell_{i-1,j} + H_{i-1,j}\ell_{i,j}}{\ell_{i,j} + \ell_{i-1,j}} - z_w \\
h_{p,e} &= \frac{\mathrm{H_{i,j}}\ell_{i+1,j} + H_{i+1,j}\ell_{i,j}}{\ell_{i,j} + \ell_{i+1,j}} - z_e \\
h_{p,s} &= \frac{\mathrm{H_{i,j}}w_{i,j-1} + H_{i,j-1}w_{i,j}}{w_{i,j} + w_{i,j-1}} - z_s \\
h_{p,n} &= \frac{\mathrm{H_{i,j}}w_{i,j+1} + H_{i,j+1}w}{w_{i,j} + w_{i,j+1}} - z_n
\end{aligned} \tag{3.40}$$

The surface flow thickness at the grid cell boundaries—$h_{s,*}$—is estimated in the same way as the saturated thickness. The elevation at the cell boundary and the PFC thickness are subtracted from the interpolated total head at the boundary to give an estimate of the thickness of sheet flow:

$$
\begin{aligned}
h_{s,w} &= \frac{\mathrm{H_{i,j}}\ell_{i-1,j} + H_{i-1,j}\ell_{i,j}}{\ell_{i,j} + \ell_{i-1,j}} - z_w - b \\
h_{s,e} &= \frac{\mathrm{H_{i,j}}\ell_{i+1,j} + H_{i+1,j}\ell_{i,j}}{\ell_{i,j} + \ell_{i+1,j}} - z_e - b \\
h_{s,s} &= \frac{\mathrm{H_{i,j}}w_{i,j-1} + H_{i,j-1}w_{i,j}}{w_{i,j} + w_{i,j-1}} - z_s - b \\
h_{s,n} &= \frac{\mathrm{H_{i,j}}w_{i,j+1} + H_{i,j+1}w}{w_{i,j} + w_{i,j+1}} - z_n - b
\end{aligned}
\tag{3.41}
$$

The approximations given in Equations (3.40) and (3.41) must consider the physical constraints on and interdependence of the saturated thickness and surface thickness. The saturated thickness must be greater than or equal to zero and less than or equal to the thickness of the PFC layer. The surface thickness must be positive, and must be zero when the saturated thickness is less than the thickness of the PFC layer. These constraints are expressed mathematically as:

$$
\begin{gathered}
0 \leq h_p \leq b \\
h_s = 0 \; for \; h_p < b
\end{gathered}
\tag{3.42}
$$

These constraints are imposed on the estimates of thickness at the cell boundaries using minimum and maximum functions. Examples of how these functions are used are given for the western boundary. The other boundaries are calculated in a similar way.

$$
\begin{aligned}
h_{p,w} &= \min\left(b\,;\, \frac{\mathrm{H_{i,j}}\ell_{i-1,j} + H_{i-1,j}\ell_{i,j}}{\ell_{i,j} + \ell_{i-1,j}} - z_w\right) \\
h_{s,w} &= \max\left(0\,;\, \frac{\mathrm{H_{i,j}}\ell_{i-1,j} + H_{i-1,j}\ell_{i,j}}{\ell_{i,j} + \ell_{i-1,j}} - z_w - b\right)
\end{aligned}
\tag{3.43}
$$

Use of these functions means that the overall mass balance equation is no longer smooth in the mathematical sense; however the physical system under consideration is not smooth either. There is a shift in the behavior of the system when the PFC layer becomes saturated and sheet flow begins, or when sheet flow disappears into the pavement because the rainfall intensity decreased. The minimum and maximum functions have the advantages of ease implementation in a numerical scheme and of facilitating the use of a single equation to describe subsurface flow and combined surface/subsurface flow.

Friction Slope

By the Dupuit-Forchheimer assumptions, the friction slope is the same as the hydraulic gradient. This is a vector quantity, so the component in each coordinate direction will be estimated. Estimates of the component in the proper direction and the overall magnitude are needed for the sheet flow part of the problem.

The ξ-component of the friction slope at the middle of the west and east faces are computed from the node values of neighboring cells.

$$S_{f\xi,w} = \frac{H_{i-1,j} - H_{i,j}}{1/2\,(\ell_{\mathrm{i-1,j}} + \ell_{\mathrm{i,j}})}$$
$$S_{f\xi,e} = \frac{H_{i+1,j} - H_{i,j}}{1/2\,(\ell_{\mathrm{i+1,j}} + \ell_{\mathrm{i,j}})} \tag{3.44}$$

Similarly, the η-component of the friction slope at the middle of the south and north faces are computed from the node values of neighboring cells.

$$S_{f\eta,s} = \frac{H_{i,j-1} - H_{i,j}}{1/2\,(w_{\mathrm{i,j-1}} + w_{\mathrm{i,j}})}$$
$$S_{f\eta,n} = \frac{H_{i,j+1} - H_{i,j}}{1/2\,(w_{\mathrm{i,j+1}} + w_{\mathrm{i,j}})} \tag{3.45}$$

The other friction slope component for each face is found from a weighted average of the component in that direction from the nearest four faces where it was computed. This means the η-component at the western face is estimated as the weighted average of the η-component at the north and south faces of the central cell and its western neighbor.

$$S_{f\eta,w} = \frac{(S_{f\eta,n} + S_{f\eta,s})\ell_{i-1,j} + \left(S_{f\eta,n} + S_{f\eta,s}\right)_{i-1}\ell_{i,j}}{2(\ell_{i,j} + \ell_{i-1,j})}$$
$$S_{f\eta,e} = \frac{(S_{f\eta,n} + S_{f\eta,s})\ell_{i+1,j} + \left(S_{f\eta,n} + S_{f\eta,s}\right)_{i+1}\ell_{i,j}}{2(\ell_{i,j} + \ell_{i+1,j})} \tag{3.46}$$

The ξ-component of the friction slope at the southern and northern faces is estimated in a similar way:

$$S_{f\xi,n} = \frac{\left(S_{f\xi,e} + S_{f\xi,w}\right)w_{i,j+1} + \left(S_{f\xi,e} + S_{f\xi,w}\right)_{i,j+1}w_{i,j}}{2\left(w_{i,j} + w_{i,j+1}\right)} \tag{3.47}$$

Note that Equations (3.46) and (3.47) could equivalently use the metric coefficients corresponding to each cell face rather than the actual lengths and widths. The magnitude of the total friction slope at any location is the Pythagorean sum of the components.

$$S_{f,w} = \sqrt{{S_{f\xi,w}}^2 + {S_{f\eta,w}}^2} \tag{3.48}$$

Conveyance Coefficients

Now that all of the terms on the right hand side of the mass balance given in (3.37) are expressed in terms of the total head, we return to the overall equation. Collecting collecting like terms and dividing by the cell area gives the model equation where terms in square brackets are defined to be conveyance coefficients:

$$\begin{aligned}\frac{1}{\Delta A}\frac{\partial \forall_{H_2O}}{\partial t} = &\left[\left(K * h_{p,w} + \frac{1}{n}\frac{h_{s,w}^{\frac{5}{3}}}{\sqrt{S_{f,w}}}\right)\left(\frac{2w_{i,j}}{\ell_{i-1,j}+\ell_{i,j}}\right)\left(\frac{1}{\Delta A}\right)\right] \\ &* \left(H_{i-1,j} - H_{i,j}\right) \\ &+ \left[\left(K * h_{p,e} + \frac{1}{n}\frac{h_{s,e}^{\frac{5}{3}}}{\sqrt{S_{f,e}}}\right)\left(\frac{2w_{i,j}}{\ell_{i+1,j}+\ell_{i,j}}\right)\left(\frac{1}{\Delta A}\right)\right] \\ &* \left(H_{i+1,j} - H_{i,j}\right) \\ &+ \left[\left(K * h_{p,s} + \frac{1}{n}\frac{h_{s,s}^{\frac{5}{3}}}{\sqrt{S_{f,s}}}\right)\left(\frac{2l_{i,j}}{w_{\mathrm{i,j-1}}+w_{\mathrm{i,j}}}\right)\left(\frac{1}{\Delta A}\right)\right] \\ &* \left(H_{i,j-1} - H_{i,j}\right) \\ &+ \left[\left(K * h_{p,n} + \frac{1}{n}\frac{h_{s,n}^{\frac{5}{3}}}{\sqrt{S_{f,n}}}\right)\left(\frac{2l_{i,j}}{w_{\mathrm{i,j+1}}+w_{\mathrm{i,j}}}\right)\left(\frac{1}{\Delta A}\right)\right] \\ &* \left(H_{i,j+1} - H_{i,j}\right)\end{aligned} \tag{3.49}$$

In Equation (3.49) the terms in square brackets are conveyance coefficients. There is a conveyance coefficient for each face of the grid cell. The thickness estimates at the cell boundary appear only in the conveyance coefficient. Substituting the thickness estimates of Equation (3.43) yields the final conveyance coefficients for the faces. The conveyance coefficient for the western boundary is:

$$C_w = \left(\begin{array}{c} K * \min\left(b\ ;\ \frac{\mathrm{H_{i,j}}\ell_{i-1,j} + H_{i-1,j}\ell_{i1,j}}{\ell_{i,j} + \ell_{i-1,j}} - z_w\right) \\ + \frac{1}{n}\frac{\max\left(0\ ;\ \frac{\mathrm{H_{i,j}}\ell_{i-1,j} + H_{i-1,j}\ell_{i1,j}}{\ell_{i,j} + \ell_{i-1,j}} - z_w - b\right)^{\frac{5}{3}}}{\sqrt{S_{f,w}}} \end{array} \right) \left(\frac{2w_{i,j}}{\ell_{i-1,j} + \ell_{i,j}}\right)\left(\frac{1}{\Delta A}\right) \quad (3.50)$$

Conveyance coefficients allow the mass balance equation to be expressed more concisely:

$$\begin{aligned} \frac{1}{\Delta A}\frac{\partial \forall_{H_2O}}{\partial t} = {} & C_w * \left(H_{i-1,j} - H_{i,j}\right) + C_e * \left(H_{i+1,j} - H_{i,j}\right) + C_s \\ & * \left(H_{i,j-1} - H_{i,j}\right) + C_n * \left(H_{i,j+1} - H_{i,j}\right) + r(t) \end{aligned} \quad (3.51)$$

Porosity Function

With the right hand side of the mass balance expressed in terms of the total head we turn to the left hand side of Equation (3.51) and recall that the volume of water in a grid cell must consider the porosity of the PFC. Considering Equation (3.28), the left hand side of Equation (3.51) can be expressed as:

$$\frac{1}{\Delta A}\frac{\partial \forall_{H_2O}}{\partial t} = \begin{cases} n_e \frac{\partial h_p}{\partial t} & for\ h_p < b \\ \frac{\partial h_s}{\partial t} & for\ h_p \geq b \end{cases} \quad (3.52)$$

The constraints on h_p and h_s are imposed by the physical system are that either $\frac{\partial h_p}{\partial t}$ or $\frac{\partial h_s}{\partial t}$ will be zero at all times. In other words the time derivative of the total head, $\frac{\partial H}{\partial t}$, will be completely given by $\frac{\partial h_p}{\partial t}$ when the flow is contained within the pavement. For the case of combined surface/subsurface flow, the pavement is saturated, therefore the saturated thickness is constant and $\frac{\partial h_p}{\partial t}$ is zero, leaving changes in the total head to the surface component. Table 1 summarizes these cases.

Table 1: Flow Cases

	Flow Condition	Time Derivative of Total Head	Left Hand Side of Mass Balance
Case 1	Flow completely within pavement	$\frac{\partial H}{\partial t} = \frac{\partial h_p}{\partial t}$	$n_e \frac{\partial h_p}{\partial t}$
Case 2	Combined surface/subsurface flow	$\frac{\partial H}{\partial t} = \frac{\partial h_s}{\partial t}$	$\frac{\partial h_s}{\partial t}$

The difference between these flow conditions is reflected in the mass balance equation through the porosity. When the water is contained in the pavement, changes in the volume of water in the grid cell are reflected in the head through the porosity. Consider for example, a cell having an area of 1 square meter that receives 1 mm of rainfall and has no other fluxes. In either case 1 or case 2 the volume of water in the cell increases by 1 liter. In case 1 the total head increases by 1mm/n_e, while in case 2 the head increases by only 1mm.

To combine the time derivatives into a single term, we must apply the porosity to the right hand side based on the flow condition. For this purpose a "porosity function" is defined to accomplish switching between the phases. This function says to divide by the porosity if the flow is contained within the pavement, but not change anything if the pavement is saturated.

$$pf(H,z,b,n_e) = \begin{cases} 1 & for\ H - z \geq b \\ 1/n_e & for\ H - z < b \end{cases} \tag{3.53}$$

Model Equation in terms of Total Head

With the use of the porosity function, we can combine the time derivatives of thickness into the time derivative of total head, and express the mass balance for a grid cell in terms of the total head and problem parameters. The equation is arranged in order of the bands that appear in the coefficient matrix.

$$\begin{aligned} \frac{\partial H}{\partial t} = \ & pf * [C_w H_{i-1,j} + C_s H_{i,j-1} - (C_w + C_s + C_n + C_e) H_{i,j} + C_n H_{i,j+1} \\ & + C_e H_{i+1,j} + r(t)] \end{aligned} \tag{3.54}$$

Equation (3.54) accomplishes the goals set out for this numerical formulation. The mass balance is expressed in terms of the total head at the center of a grid cell and a single equation applies for both subsurface flow and combined surface/subsurface flow. When the saturated thickness (h_p) is less than the thickness of the PFC layer, the porosity function is active, the max function removes the surface flow part of the conveyance coefficient, and Equation (3.54) reduces to the Boussinesq equation. When the saturated thickness is equal to or greater than the thickness of the PFC layer, the porosity function turns off, the minimum function forces the saturated thickness to the PFC layer thickness, and the surface flow part of the conveyance coefficient is non-zero.

3.6.3 Depth Formulation, Time Discretization, Linearization

As mentioned earlier, the discretized equations will now be re-expressed in terms of the thickness rather than the total head. This is accomplished by making the substitution $H = h + z$. The time derivative converts directly because the elevation does not change in time.

$$\begin{aligned}\frac{\partial h_{i,j}}{\partial t} = \ pf * [C_w(h+z)_{i-1,j} + C_sH(h+z)_{i,j-1} \\ - (C_w + C_s + C_n + C_e)(h+z)_{i,j} + C_n(h+z)_{i,j+1} \\ + C_e(h+z)_{i+1,j} + r(t)]\end{aligned} \tag{3.55}$$

To solve Equation (3.55) the time dimension is discretized using the Crank-Nicolson method. The resulting non-linear system is linearized by lagging the conveyance coefficients using an inner iteration loop. The Crank-Nicolson method is summarized as follows, using the superscript n as the time level (Ferziger and Peric, 2002).

$$\frac{h_{i,j}^{n+1} - h_{i,j}^{n}}{\Delta t} = \frac{1}{2}[\ RHS]^{n+1} + \frac{1}{2}[\ RHS]^{n} \tag{3.56}$$

Now the system is arranged for solving as a linear system by moving the unknowns—the depths at time level $n+1$—to the left side of the equation and moving the known quantities to the right.

$$h_{i,j}^{n+1} - \frac{\Delta t}{2}[\ RHS\ ]^{n+1} = \frac{\Delta t}{2}[\ RHS\ ]^{n} + h_{i,j}^{n} \tag{3.57}$$

Let A, B, C, D, E be the bands of the penta-diagonal coefficient matrix and F be the right side of the linear system, or force vector. A linear index is needed to relate grid points using i, j indices to a single index for the matrix system. The linear index is formed by numbering the grid cells consecutively along the columns starting in the southwest corner of the domain. Taking the largest value of the domain column index as j_{max} the linear index k for any grid cell is computed from:

$$k(i,j) = (i-1) * j_{max} + j \tag{3.58}$$

Using the linear index, the system can be written as:

$$A_k\ h_{k-j_{max}}^{n+1} + B_k\ h_{k-1}^{n+1} + C_k\ h_{k}^{n+1} + D_k\ h_{k+1}^{n+1} + E_k\ h_{k+j_{max}}^{n+1} = F_k \tag{3.59}$$

where the expressions for the matrix coefficients are (with the conveyance coefficients at the n+1 level):

$$\begin{aligned}
A_k &= -\frac{\Delta t}{2} * pf * C_w{}^{n+1} \\
B_k &= -\frac{\Delta t}{2} * pf * C_s{}^{n+1} \\
C_k &= \frac{\Delta t}{2} * pf * \left(C_w{}^{n+1} + C_s{}^{n+1} + C_n{}^{n+1} + C_e{}^{n+1}\right) + 1 \\
D_k &= -\frac{\Delta t}{2} * pf * C_n{}^{n+1} \\
E_k &= -\frac{\Delta t}{2} * pf * C_e{}^{n+1}
\end{aligned} \tag{3.60}$$

The right hand side of the system is:

$$F_k = pf^n \frac{\Delta t}{2} \begin{Bmatrix} C_w h_{i-1,j} + C_s h_{i,j-1} - \\ (C_w + C_s + C_n + C_e) h_{i,j} + \\ C_n h_{i,j+1} + C_e h_{i+1,j} + \\ C_w z_{i-1,j} + C_s z_{i,j-1} - \\ (C_w + C_s + C_n + C_e) z_{i,j} + \\ C_n z_{i,j+1} + C_e z_{i+1,j} + r(t) \end{Bmatrix}^n + h_{i,j}^n \tag{3.61}$$

$$+ pf^{n+1} \frac{\Delta t}{2} \begin{Bmatrix} C_w z_{i-1,j} + C_s z_{i,j-1} - \\ (C_w + C_s + C_n + C_e) z_{i,j} + \\ C_n z_{i,j+1} + C_e z_{i+1,j} + r(t) \end{Bmatrix}^{n+1}$$

Note that the value of in each band for an interior grid cell depends upon the four cells on its borders and on itself so the computational molecule is comprised of five cells and the coefficient matrix is penta-diagonal.

The values of the coefficient matrix (A, B, C, D, E) depend on the conveyance coefficients, which in turn depend on the unknown thicknesses so the system of equations is non-linear. Linearization is accomplished using the fixed point method—conveyance coefficients are computed using old values of the depths and these coefficients are then used to compute new depths (Ferziger and Peric, 2002). The new depths are used to update the conveyance coefficients and this process is repeated until values of the depths stop changing within the iteration. At each iteration, the linearized system of equations is solved using the Gauss-Seidel method for solving linear systems of equations.

3.7 Initial Conditions and Boundary Conditions

Solution of the governing equations requires suitable initial conditions and boundary conditions. In the following sections initial conditions are discussed first, followed by the no-flow boundary condition. The subsequent section proposes a new boundary condition for PFC flow—the kinematic condition. A formulation for kinematic boundary conditions in the case of sheet flow is also given, followed by an algorithm combining the kinematic condition for PFC and sheet flow.

3.7.1 Initial Conditions

The initial condition for the entire system is that of zero depth, corresponding to a PFC roadway that is completely dry at the onset of rainfall. Any known depth could theoretically be used as an initial condition, but the zero depth condition arises frequently in practice.

3.7.2 No Flow Boundaries

A no flow boundary is a Neumann type condition because the derivative is specified at the boundary. For a no-flow boundary, the conveyance coefficient for the cell face corresponding to the boundary is set to zero, effectively enforcing the condition of a zero head gradient.

$$\frac{dH}{d\eta} = 0 \tag{3.62}$$

Considering Equation (3.49), which shows the conveyance coefficients in brackets, setting the conveyance coefficient equal to zero is equivalent to the zero gradient condition. Note that this approach works for PFC flow and sheet flow.

3.7.3 Kinematic Boundary Conditions for PFC Flow

Boundary conditions other than no-flow boundaries are difficult to formulate for PFC roadways. Boundary conditions are classified as Dirichlet type when the solution is prescribed at the boundary, Neumann type when the first derivative is specified at the boundary and as Robin type when some combination of the solution and its derivative are specified at the boundary (Kreyszig, 1999). Formulating boundary conditions for PFC flow—especially under unsteady conditions—is difficult because the solution at the boundary varies according to the external forcing (rainfall), the solution within the domain, and the geometry of the domain itself. In addition, the boundary condition should be able to transition back and forth between sheet flow conditions.

Strictly speaking, the edge of a PFC is a seepage face because the pressure at any point along the edge is atmospheric. Treating the edge of pavement as a seepage surface is problematic for at least two reasons: (1) the velocity field near a seepage face has a strong vertical component (see the experiments of Simpson et al. 2003) but the model equation excludes vertical velocities; and (2) the Dupuit-Forchheimer assumptions on which the model is based do not allow for a seepage surface since they require the pressure to vary along a vertical line.

As a way to overcome these challenges it is desirable to specify the saturated thickness at the center of a boundary grid cell based on the forcing, geometry, and solution from the previous time step. The center of a boundary cell is a nodal unknown, the value of which is referred to by the adjacent cells. Specifying the value at such a location is a Dirichlet condition because the value of the solution is prescribed.

The following formulation develops a new method for specifying boundary conditions to a Dupuit-Forchheimer flow model. The principle assumption is that of kinematic flow. In the following three subsections, the algorithm is developed for a linear roadway; the effect of the algorithm on the steady state solution is investigated; and the applicability to curved roads is assessed.

Linear Roadways

The saturated thickness at the center of a boundary cell may be estimated by applying the method of characteristics (MOC) to the PDE for one-dimensional flow under kinematic conditions. The MOC is a mathematical solution technique for PDEs of first-order and for hyperbolic PDEs of second-order (Street, 1973). The concept of kinematic flow refers to the case where pressure and acceleration are neglected in the momentum equation.

The continuity equation for flow in a porous medium under unsteady conditions and with a free surface is given by Equation (3.3); considering only the x direction the equation becomes

$$n_e \frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(q * h) = r \tag{3.63}$$

where n_e is the effective porosity, h is the saturated thickness, r is the rainfall rate and the Darcy velocity is

$$q = -K\frac{\partial H}{\partial x} = -K\frac{\partial h}{\partial x} - KS_0 \tag{3.64}$$

Making this substitution and expanding the terms gives

$$n_e \frac{\partial h}{\partial t} - Kh\frac{\partial^2 h}{\partial x^2} - K\left(\frac{\partial h}{\partial x}\right)^2 - KS_0\frac{\partial h}{\partial x} = r \tag{3.65}$$

The assumption of kinematic conditions means that the depth gradient is neglected in the Darcy velocity, which removes the higher order terms in Equation (3.65) and gives

$$n_e \frac{\partial h}{\partial t} - KS_0\frac{\partial h}{\partial x} = r \tag{3.66}$$

Removing the higher order terms destroys the parabolic nature of the PDE. This is not a typical approximation for porous media flow and does introduce some error in the solution. However, neglecting these terms allows the formulation of a boundary algorithm that considers the problem parameters and can transition smoothly to sheet flow conditions.

The MOC procedure given by Street (1973) is followed here. The solution of Equation (3.66) can be considered as a surface in $x, t, h(x,t)$ space. The tangent plane to the surface is given by the total differential

$$dh = \frac{\partial h}{\partial t}dt + \frac{\partial h}{\partial x}dx \tag{3.67}$$

and the normal vector to this tangent plane is $(\frac{\partial h}{\partial t}, \frac{\partial h}{\partial x}, -1)$. This normal vector is tangent to the vector $(n_e, -KS_0, r)$because their dot product is zero by Equation (3.66).

$$\left(\frac{\partial h}{\partial t}, \frac{\partial h}{\partial x}, -1\right) \cdot (n_e, -KS_0, r) = n_e \frac{\partial h}{\partial t} - KS_0 \frac{\partial h}{\partial x} - r = 0 \tag{3.68}$$

The vector $(n_e, -KS_0, r)$ must be tangent to the solution surface because it is orthogonal to the surface normal. A position vector for a point on the solution surface can also be represented parametrically as $(\ x(s),\ t(s),\ h(s)\ )$. Its tangent vector is $\left(\frac{dx}{ds}, \frac{dt}{ds}, \frac{dh}{ds}\right)$. The fact that components of the tangent vectors must be proportional leads to the MOC formulation of the problem:

$$\frac{(d\,x/ds)}{n_e} = \frac{(d\,t/ds)}{-KS_0} = \frac{(d\,h/ds)}{r} \tag{3.69}$$

This formulation is usually presented after ds has been eliminated from the equations:

$$\frac{dt}{n_e} = \frac{dx}{-KS_0} = \frac{dh}{r} \tag{3.70}$$

To obtain a Dirichlet type boundary condition for the domain, we need to estimate the saturated thickness in the boundary cell at the new time level based on the solution from the previous time-step. Since the solution travels along characteristic curves, the idea is to figure out how far the solution will move along a characteristic during a time-step. In this way the solution at time level n+1 is estimated by going up the characteristic by the proper distance. In other words, if A and B are points along the characteristic curve, the solution at point A and time level n can be used to find the solution at point B for time level n+1. The problem now is to find the distance from point B to point A. This estimate comes from integrating Equation (3.70).

Integrating the second and third terms of (3.70) gives an estimate of the boundary value in terms of the distance up the characteristic curve

$$\frac{x_2 - x_1}{-KS_0} = \frac{h_2 - h_1}{r} \quad \rightarrow \quad h_2 = h_1 - \frac{r}{KS_0}(x_2 - x_1) \tag{3.71}$$

Integrating the first and second terms of (3.70) yields an estimate of the distance in terms of the time-step:

$$\frac{t_2 - t_1}{n_e} = \frac{x_2 - x_1}{-KS_0} \quad \rightarrow \quad \Delta x = -\frac{KS_0 \Delta t}{n_e} \tag{3.72}$$

Substituting (3.72) into (3.71) gives the desired estimate:

$$h_2 = h_1 + \frac{\mathrm{r}\,\Delta t}{n_e} \tag{3.73}$$

The value of h_1 is estimated as the solution at time level n a distance Δx up the drainage slope from point h_2.

The kinematic approximation implies a maximum value for the saturated thickness that is not reflected in the algorithm of Equations (3.72) and (3.73). At steady state there is no change with time so $\Delta t = 0$, which makes $\Delta x = 0$ and puts h_1 and h_2 at the same location. Since the hydraulic gradient was approximated as the pavement slope, the Darcy velocity is constant (see Equation (3.64)) and the saturated thickness is determined by the flow rate per unit width. For the one dimensional case, the steady state flow rate per unit width is given by the rainfall rate, r, and length of the drainage path, L.

$$h_{ss} = \frac{rL}{KS_0} \tag{3.74}$$

When the kinematic condition is applied to a 1D problem, the boundary is the edge of pavement and the approximation gives a maximum depth as just described. A 2D problem has boundaries at both the edge of pavement and the ends of the domain, where the road continues beyond the modeled area. The kinematic boundary condition can also be applied at the end of the domain, but the boundary values—having neglected the depth gradient in Darcy's law—will be inconsistent with the domain interior. This inconsistency results in a boundary effect. The model domain should be expanded so that this effect does not influence the area of interest. One approach is to ensure the drainage path for a water particle starting at the boundary exits the model domain rather than entering the area of interest, thereby "washing out" the error. The required distance is found from the longitudinal and cross slopes and the width.

Effect on Steady State Solution

The steady state solution for 1D drainage in PFC is given by an ODE and an initial point along the solution curve is needed to integrate the equation (Charbeneau and Barrett, 2008). The kinematic approximation described above is one approach to specifying such an initial point based on the problem parameters. Figure 16 shows that the shape of the solution curve, especially near the boundary, depends upon the value that was specified at the boundary (hL). The solution curves show that the kinematic approximation does not allow the solution to 'draw down' near the boundary as is usual near a seepage face (Simpson et al., 2003). This draw down is required because the phreatic surface must be tangent to the seepage face (Bear, 1972). This draw-down decreases the saturated thickness but increases the hydraulic gradient. In contrast, the approximation over-estimates the saturated thickness and reduces the hydraulic gradient. Which one of the curves is closest to the true physical solution is unknown, but a range of possible solutions has now been established.

In Figure 16, the solutions collapse to a single curve away from the downstream boundary, but this behavior depends on the problem parameters. Doubling the rainfall rate for example pushes the point at which the curves collapse to the left, provided that the thickness of the PFC layer is sufficient to contain the additional flow (Figure 17). If the PFC thickness is 5cm, then doubling the rainfall rate to 1cm/hr causes sheet flow and the boundary condition for the region of PFC flow is given by the pavement thickness (Eck et al., 2010). In general, a finite pavement thickness means that the uncertainty in the boundary value matters most for low rainfall rates. Together, these examples illustrate that:

- the predicted value of the saturated thickness depends on the boundary value;
- the boundary value is unknown only for low rainfall rates; and
- the solution is less sensitive to the boundary value in this case.

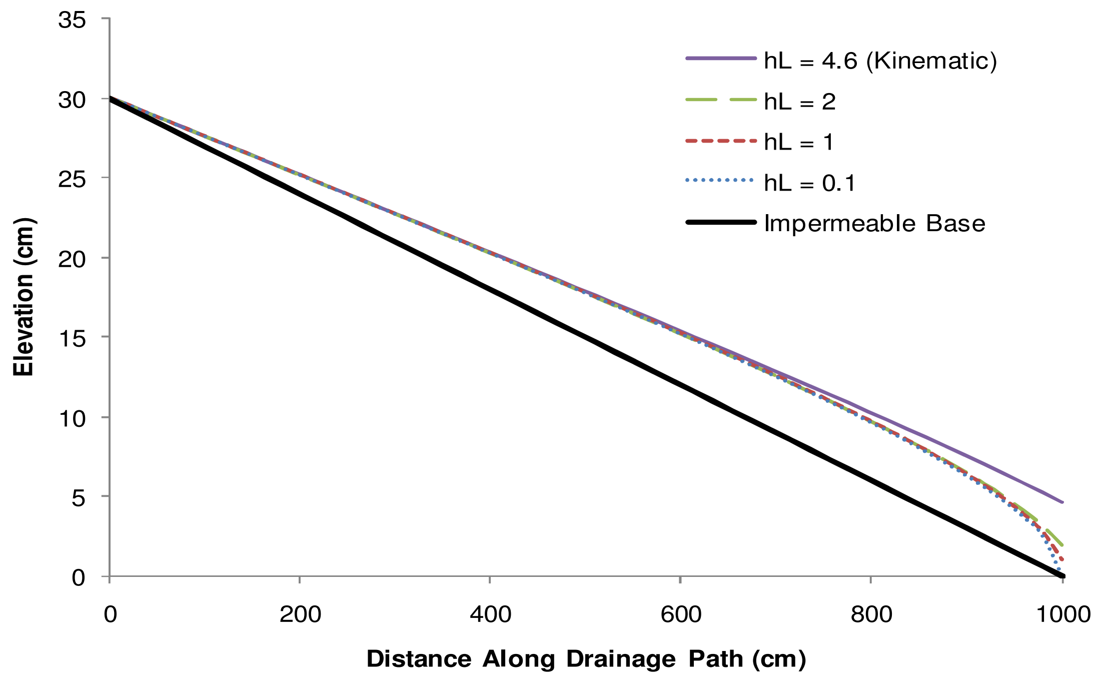

Figure 16: Steady state drainage profile for different boundary values; all cases used K=1cm/s, S_0=3%; r=0.5cm/hr

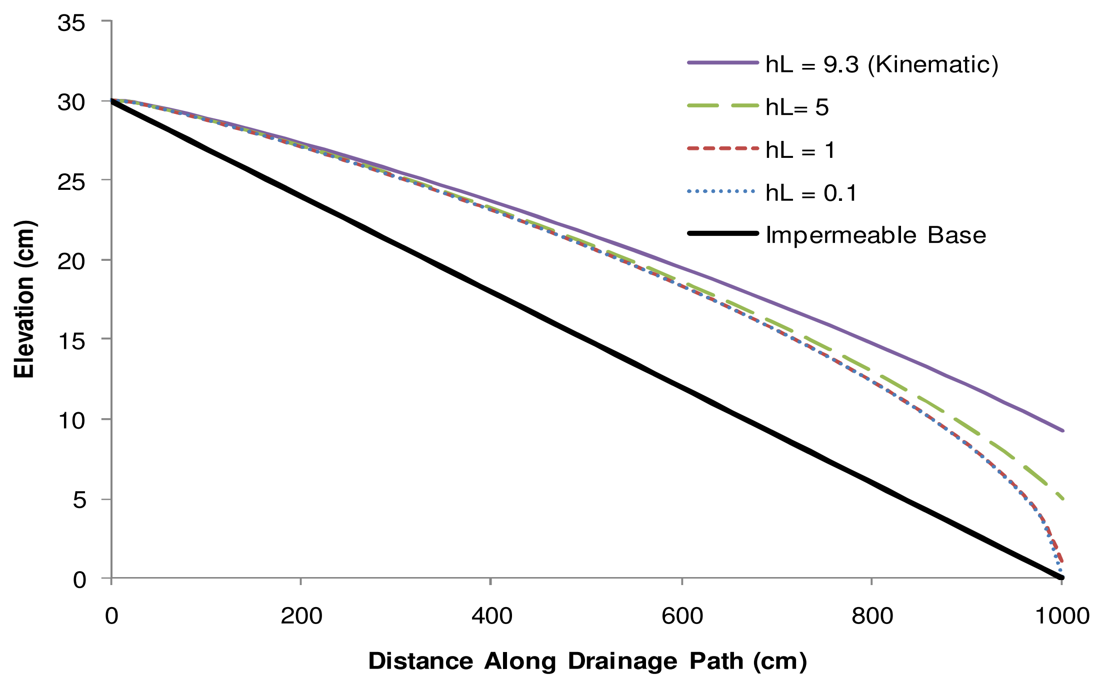

Figure 17: Steady state drainage profile for different boundary values; all cases used K=1cm/s, S_0=3%; r=1cm/hr

Kinematic Boundary for Curved Roadways

The algorithm outlined in Equations (3.72) and (3.73) was developed under the assumption of a straight roadway section and not a curved one. An order of magnitude approach is used to assess the applicability of the linear algorithm for curved sections.

The continuity equation for radial flow is

$$n_e \frac{\partial h}{\partial t} + \frac{1}{R}\frac{\partial}{\partial R}(Rq_R) = r \tag{3.75}$$

where R is the radial coordinate and r is the rainfall rate. Darcy's law for radial flow is

$$q_R = -Kh\frac{\partial h}{\partial R} + KhS_o \tag{3.76}$$

Neglecting depth gradients in Darcy's law and using the continuity equation for one-dimensional radial flow gives a PDE in $h(R, t)$.

$$n_e \frac{\partial h}{\partial t} + \frac{KhS_o}{R} + KS_o\frac{\partial h}{\partial R} = r \tag{3.77}$$

Using the method of characteristics approach described above gives the formulation:

$$\frac{dt}{n_e} = \frac{dR}{KS_o} = \frac{dh}{r - \frac{KhS_o}{R}} \tag{3.78}$$

The order of magnitude for the quantities in Equation (3.78) can be estimated as r = 5cm/hr ~ 10^{-3}cm/s; h~1cm; So~0.03; R=10^4cm. Using these values, $KhS_0/R = 3(10)^6$ cm/s, which is much less than the rainfall rate of 10^{-3}cm/s. This result suggests that the linear domain kinematic approximation should be adequate for calculating boundary conditions to curved domains of interest.

3.7.4 Kinematic Boundary Conditions for Sheet Flow

Kinematic boundary conditions for sheet flow were derived by Jeong (2008). The resulting algorithm is repeated here for completeness. The distance up the drainage path is estimated in terms of the time-step and the boundary depth, h_2, at time level n.

$$\Delta s = \frac{\sqrt{S_0}}{n\, r}\left((h_2^n + r\Delta t)^{\frac{5}{3}} - (h_2^n)^{\frac{5}{3}}\right) \tag{3.79}$$

The solution at the upstream point is obtained using bi-linear interpolation, and the value of the boundary depth at time level $n+1$ is

$$h_2^{n+1} = \left((h_1^n)^{\frac{5}{3}} + (h_2^n + r\Delta t)^{\frac{5}{3}} - (h_2^n)^{\frac{5}{3}}\right)^{0.6} \tag{3.80}$$

3.7.5 Combined Kinematic Boundary Condition for PFC and Sheet flow

The algorithms for kinematic boundary conditions for sheet flow and PFC flow have been developed separately, but need to be combined so that the appropriate condition is used within the model. The combined algorithm must select between the PFC and sheet flow equations, handle the case of zero rainfall, and provide for a transition between PFC and sheet flow. This is accomplished through nested if-then statements as depicted in Figure 18.

When the flow depth is less than the pavement thickness, the PFC algorithm is used. The distance up the drainage slope is computed from Equation (3.72) and the solution at this location is estimated using bi-linear interpolation. Then the boundary value for the next time-step is computed from Equation (3.73). No modification to the algorithm is required for zero rainfall. The computed boundary value is compared to the maximum depth of Equation (3.74).

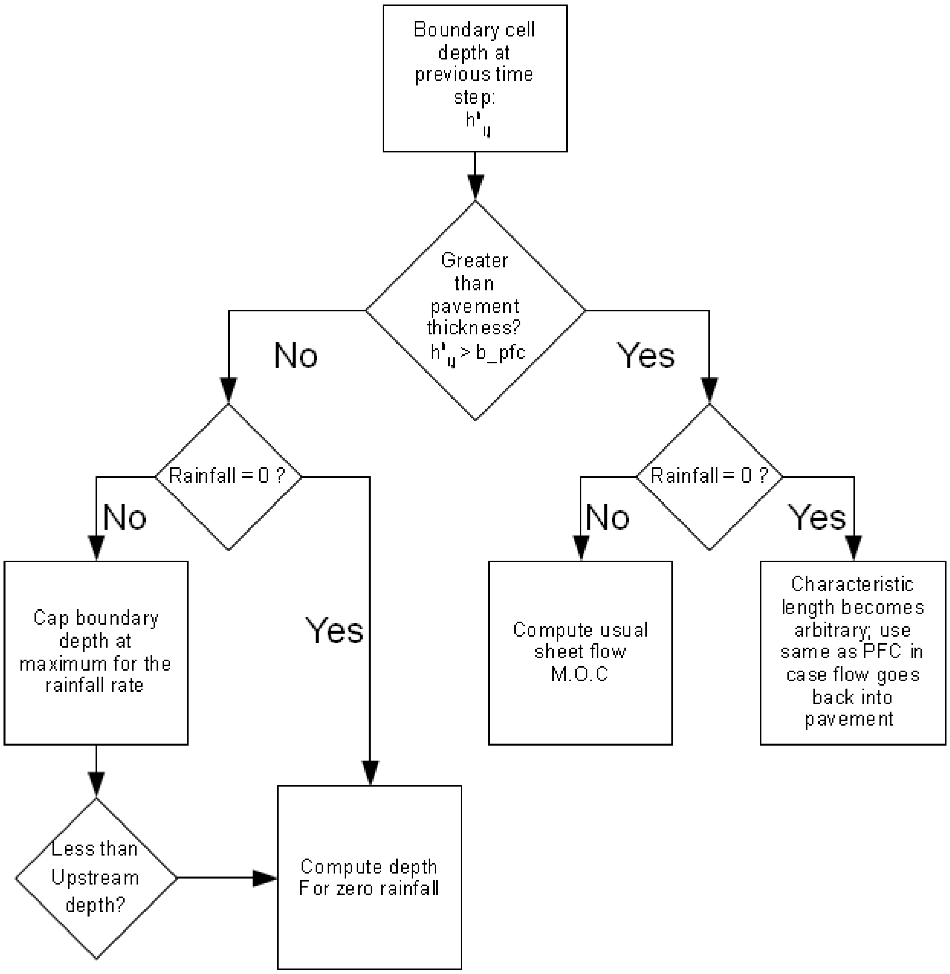

Figure 18: Combined algorithm for kinematic boundary condition

Implementation of the sheet flow algorithm is more complex due to the possibilities of zero rainfall and transition back to PFC flow. If the rainfall rate is zero, the distance to interpolate up the drainage path becomes arbitrary; the PFC distance is used in case a transition back to PFC flow is indicated. If the rainfall rate is greater than zero the interpolation distance is computed according to Equation (3.79) and the solution is estimated using bi-linear interpolation. If the interpolated value suggests PFC flow then the boundary value is estimated using the PFC equations, otherwise the sheet flow equation is used.

3.8 Solution Procedure and Tolerances

The numerical formulation and boundary conditions described in this chapter have been implemented in a Fortran computer code. The general solution procedure can is outlined as follows and depicted in flow chart form (Figure 19):

- Read model parameters, geometry information and rainfall from input files
- Create a curvilinear grid for the domain. The grid includes the coordinates, length, width and area of each grid cell.
- Assign elevations to the center of each grid cell.
- Loop through the time steps, recording details of the solution at each step
- Within a time-step, iteratively compute the depths using the fixed point method.
- Within each iteration, solve the linearized system of equations using the Gauss-Seidel method.

A vector of errors or residuals is calculated at each iteration in order to determine when the non-linear iteration loop has converged. Absolute errors are computed when the solution is near zero and relative errors are computed when the solution is away from zero. Two norms of the error vector are checked; the L_∞ norm is simply the largest value in the error vector, and the L_2 norm is the square root of the sum of the squared errors (Kreyzig, 1999). Both the L_2 norm and the L_∞ norm must be less than the tolerance for the loop to converge. A typical tolerance value of 10^{-3} was used for simulations.

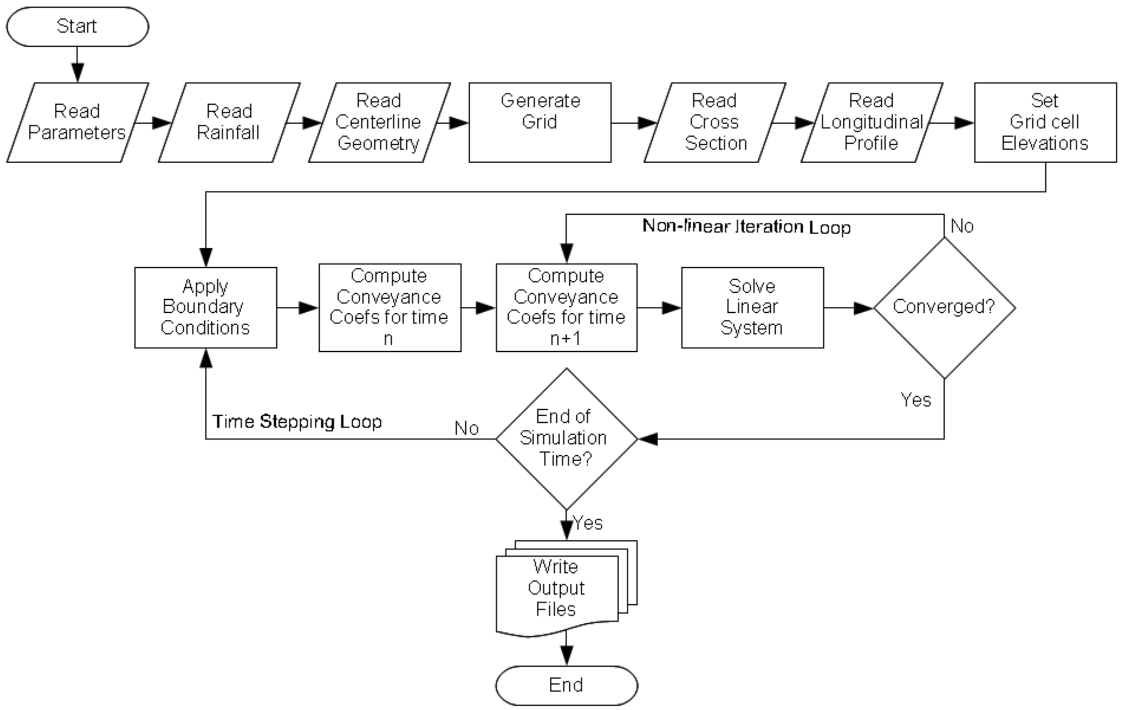

Figure 19: Flow chart of solution process

3.9 Convergence and the Transition to Sheet Flow

Trial runs during the model development process revealed numerical difficulties regarding the transition from PFC flow to sheet flow. During the time step that a grid cell transitioned from PFC flow to sheet flow the solution frequently oscillated between the PFC and sheet flow states, never reaching a solution. Physically, this transition represents a change in the character of the flow. Mathematically, there is a change in the governing equations. Given these changes, some oscillatory behavior was not wholly unexpected.

Several schemes were tried in order to overcome the numerical difficulties but the most successful approach was using an under-relaxation factor. This approach is based on the method of successive over relaxation for solving linear systems (Ferziger and Peric, 2002). The idea in successive over relaxation is to reduce the number of iterations by amplifying the change at each step using an over-relaxation factor. The under-relaxation approach aims to increase the number of iterations by making smaller changes at each step. In this way, only part of a large oscillation is taken, thus reducing the overshoot of the actual solution.

Under relaxation was found to reduce the errors by an order of magnitude, but even still a looser iterative tolerance was needed for convergence. During a simulation, the model detects a transition time-step, loosens the tolerance by a factor of 10 (changes the tolerance from 10^{-3} to 10^{-2}) and applies under-relaxation. When no grid cells are switching between PFC and sheet flow no relaxation factor is applied and the usual tolerance is imposed. An example of the relaxation factor's effect is given at the end of Section 5.3.

CHAPTER 4: MODEL VALIDATION

This chapter presents modeling results from PERFCODE for two simplified geometries: a linear section or straight road and a converging section or curved road. The purpose of the chapter is to demonstrate that solutions obtained by simulating the domain through time agree with steady state solutions, which were obtained independently of the model. Three simulations are presented for each geometric configuration: (1) PFC flow only, (2) sheet flow only, and (3) combined PFC and sheet flow. The unsteady simulations provide runoff hydrographs, which are also discussed.

4.1 Linear Section (Straight Roadway)

The linear section selected for testing is 10m wide and 20m long with a 3% cross slope. Other parameters common to all simulations were a hydraulic conductivity, porosity and rainfall rate (Table 2). Holding these parameters constant, the PFC thickness was set to 15cm, 0cm, and 5cm to simulate PFC flow only, sheet flow only, and combined PFC/sheet flow.

Table 2: Model parameters for simulating a linear section

Parameter	Unit	Value
Roadway width	m	10
Domain length	m	20
Cross Slope	%	3
Hydraulic Conductivity	cm/s	1
Porosity	--	0.2
Rainfall Rate	cm/hr	1

A plan view of the model domain for the linear section (Figure 20) shows elevation contours, locations of grid cell centers and boundary conditions imposed on the model. Because the objective of these simulations was a comparison with analytical solutions, the domain and boundary conditions were chosen to make the flow one-dimensional.

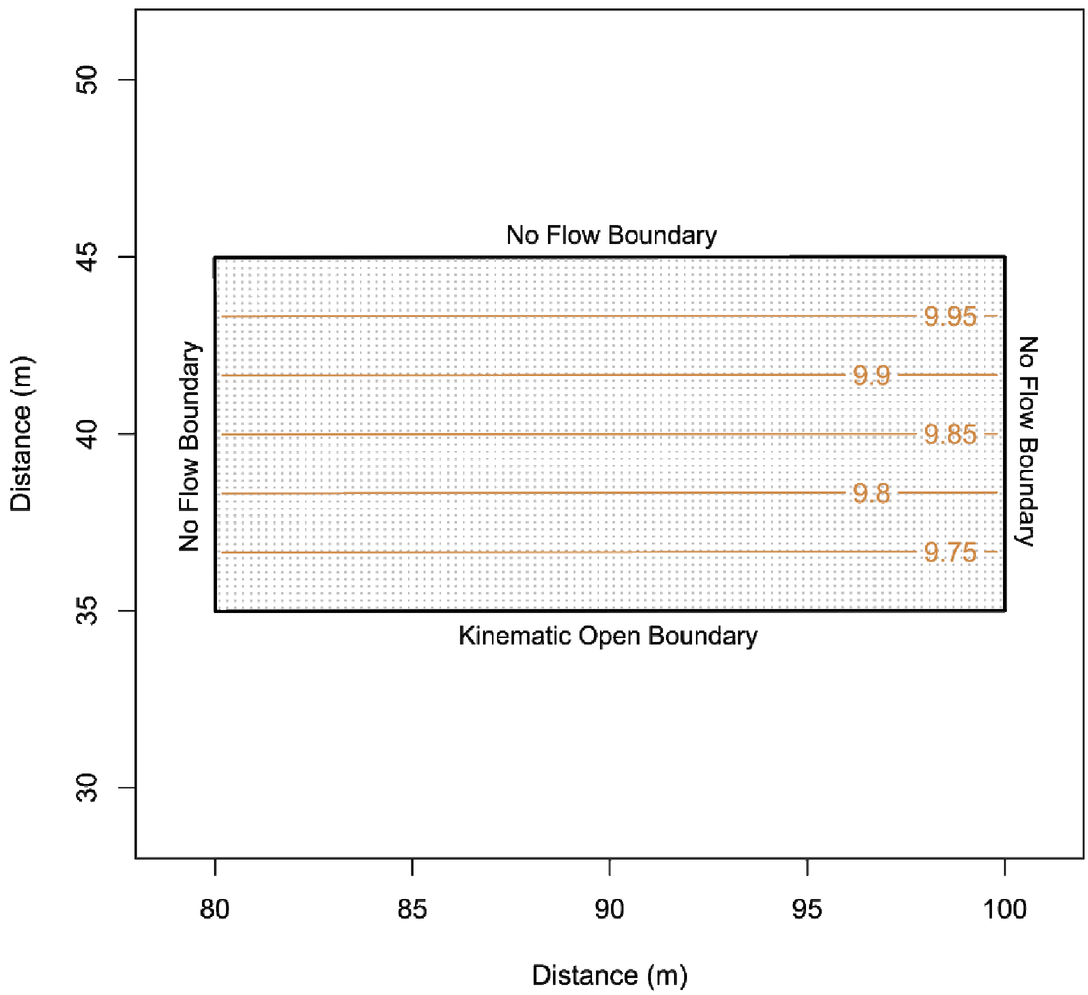

Figure 20: Linear domain showing elevation contours, grid cell centers, and boundary conditions

4.1.1 PFC Flow Only

This first simulation sets the PFC thickness at 15cm so that the steady state drainage profile will stay within the pavement. The model starts from an initial condition of zero depth and continues until steady state is reached. The model converged to a steady state solution after 20,480 seconds of rainfall. In computing the steady state solution, the initial point for integrating the ODE was found from

$$h_L = \frac{rx}{Ks} = \frac{1\frac{cm}{hr} * 1000cm}{1\frac{cm}{s} * 3\%} = 9.26cm \tag{4.1}$$

This value corresponds to the kinematic boundary condition used in the model—the hydraulic gradient is only due to the slope of the pavement.

Modeled values of the saturated thickness along the drainage path agreed closely with the analytical solution (Figure 21). In the figure, the normalized width variable η is plotted on the abscissa. For the linear section a value of $\eta = 1$ corresponds to the no flow boundary at the edge of pavement and a value of $\eta = 0$ corresponds to the kinematic drainage boundary at the edge of pavement. The scale on the figure has been plotted in reverse order so that drainage occurs from left to right.

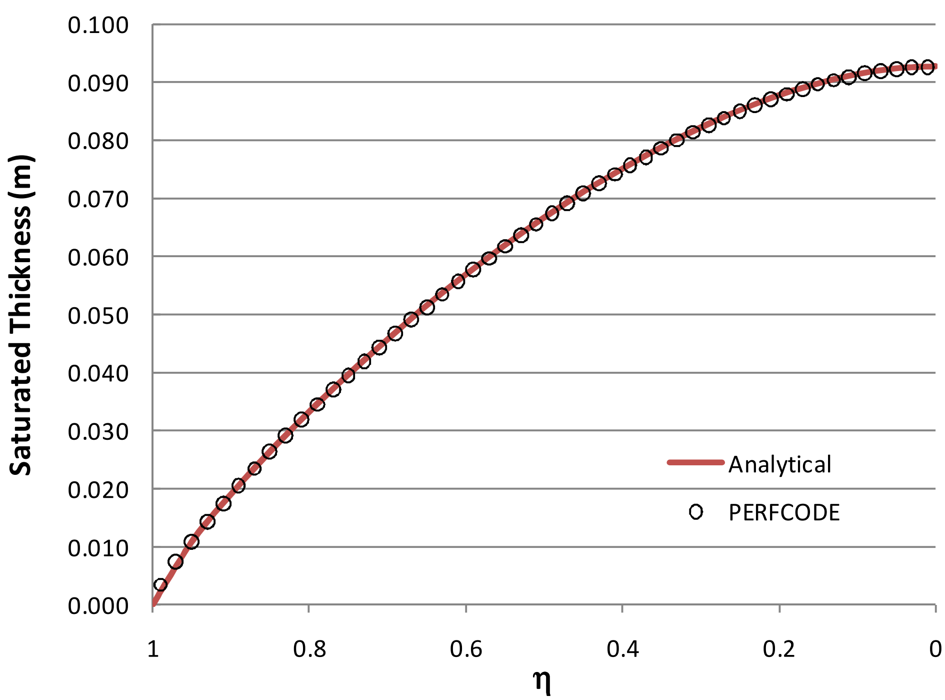

Figure 21: Depth profile for linear section with drainage by PFC flow only

4.1.2 Sheet Flow Only

The next simulation set the PFC thickness to zero so that all drainage occurs as sheet flow. The sheet flow simulation converged to a steady state solution after 252 seconds of rainfall. The flow thickness along the drainage path compares well with the analytical solution from the kinematic model (Figure 22). Sheet flow reaches steady state much faster PFC flow. The difference in time scales for transport via sheet flow versus PFC flow foreshadows some challenges of modeling the coupled flow process.

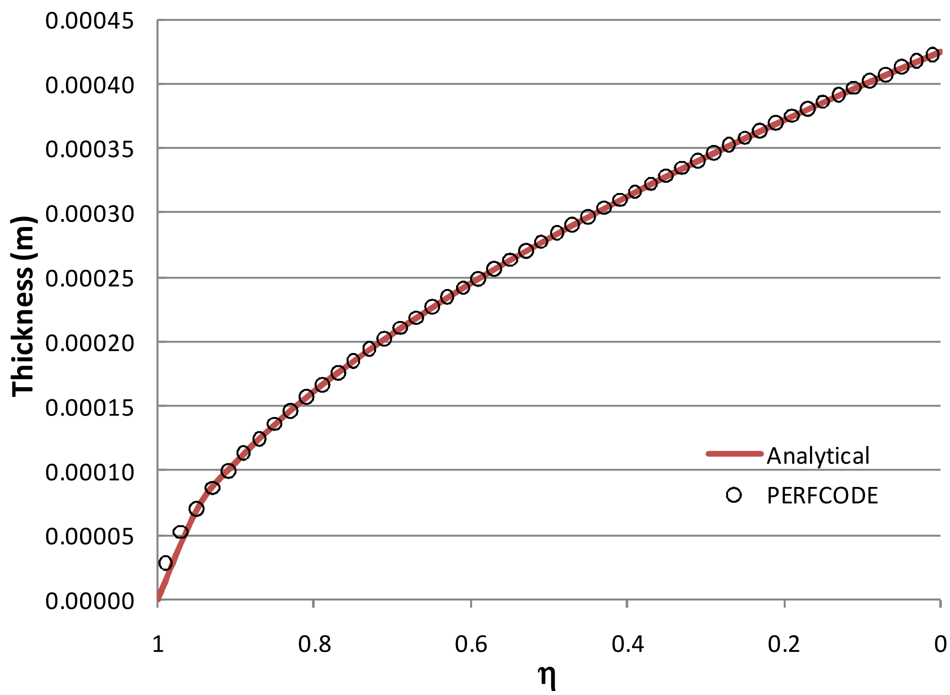

Figure 22: Depth profile for linear section with drainage by sheet flow only

4.1.3 Combined Flow

For the combined flow simulation, the PFC thickness was set to 5cm. Steady state was reached after 5,128 seconds of rainfall. Good agreement was again obtained between the numerical and analytical solutions.

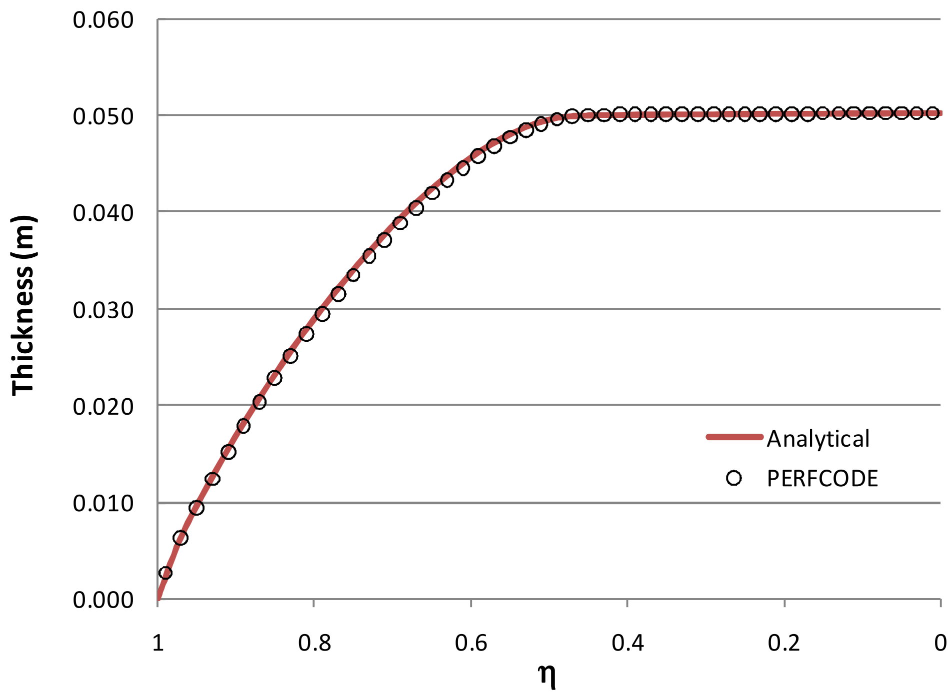

Figure 23: Depth profile for linear section with drainage by PFC and sheet flow

4.1.4 Runoff hydrographs

For each simulation the discharge from the outflow boundary was tracked through time. These rising hydrographs are plotted on a logarithmic scale on account of the wide range of times required to reach steady state (Figure 24). Several points of interest are noted on the hydrographs.

- The presence of a PFC layer delays the initial discharge from the roadway, in this case by about 1 minute from when rainfall begins.
- PFC delays the peak flow by nearly 10,000 seconds—much longer than most actual storms.

- For the combined case, the transition to sheet flow is evidenced as a sharp increase in the slope of the hydrograph.
- For the PFC flow only, the break in slope corresponds to the time when the outflow boundary reaches the maximum depth allowed by the kinematic condition.

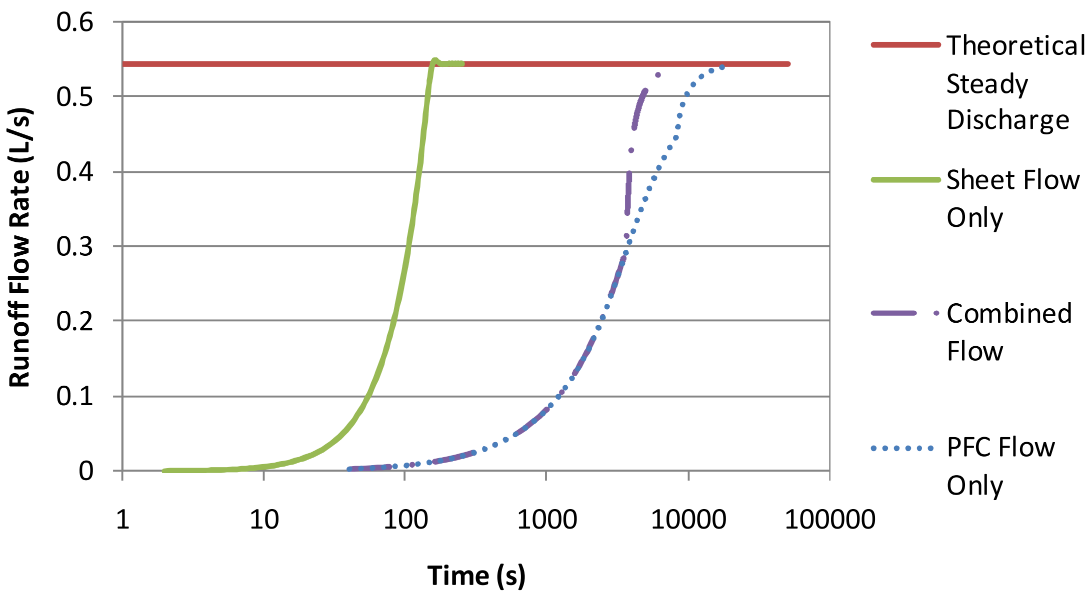

Figure 24: Runoff hydrographs from a linear section

4.2 Converging Section (Curved Roadway)

The next geometry investigated in the validation process was a fully super-elevated roadway section with a constant radius of curvature. For the purposes of this discussion such a geometry is called a *converging section*. This roadway geometry is of interest for evaluating the model's ability to simulate flow on a curved road. Keeping the cross-slope and radius of curvature constant makes the problem one-dimensional.

The converging section selected for testing is similar to the linear section, except that the radius of curvature at the roadway center is 60m. Simulation parameters are summarized in Table 3. A plan view of the model domain for the converging section (Figure 20) shows elevation contours, locations of grid cell centers and boundary conditions imposed on the model. Holding these parameters constant, the PFC thickness was set to 15cm, 0cm, and 5cm to simulate PFC flow only, sheet flow only, and combined PFC/sheet flow.

Table 3: Model parameters for simulating a converging section

Parameter	Unit	Value
Roadway width	m	10
Domain length	m	20
Cross Slope	%	3
Radius of curvature at roadway center	m	60
Hydraulic Conductivity	cm/s	1
Porosity	--	0.2
Rainfall Rate	cm/hr	1

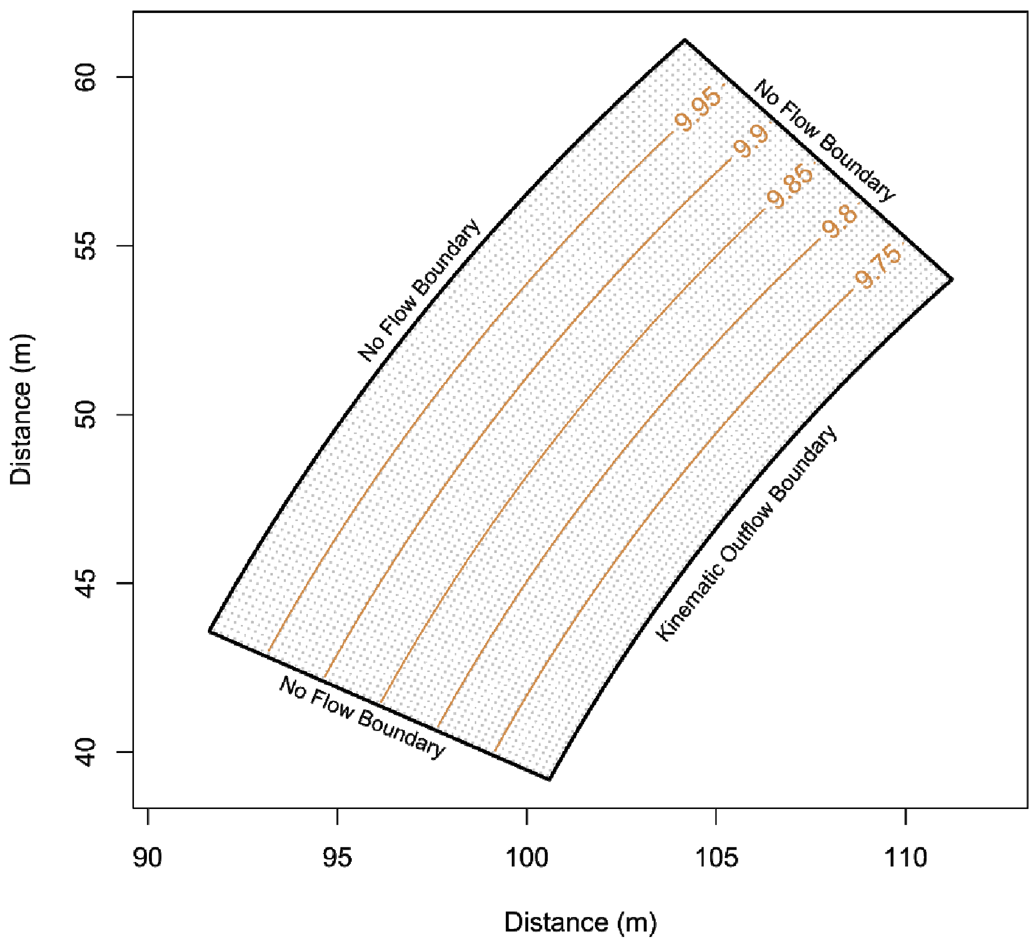

Figure 25: Converging domain showing elevation contours, grid cell centers, and boundary conditions

4.2.1 Derivation of ODE for PFC Flow on Converging Sections

The steady state solution for PFC flow on a linear domain is given by Charbeneau and Barrett (2008). Steady-state solutions for sheet flow on linear and converging sections are given by Eck et al. (2010), and also Jeong et al. (2010). What is missing is the solution for PFC flow on a converging section, which is the topic of the present subsection.

Consider a section of roadway having a constant radius of curvature and constant cross-slope as shown in Figure 26. Geometrically, this shape is equivalent to an inverted cone. A cross section view along the radius is shown in Figure 27. It is important to realize the coordinate system is arranged so that flow moves from a large radial position to a smaller radial position as it moves down the slope.

At steady state, the volumetric flow-rate into an area equals the flow-rate out of that area. For a converging section, the discharge is radial. The flow rate is the rainfall rate times the contributing area. The area is found by subtracting the area of the sector at radius R from the area of the sector at R_{max}.

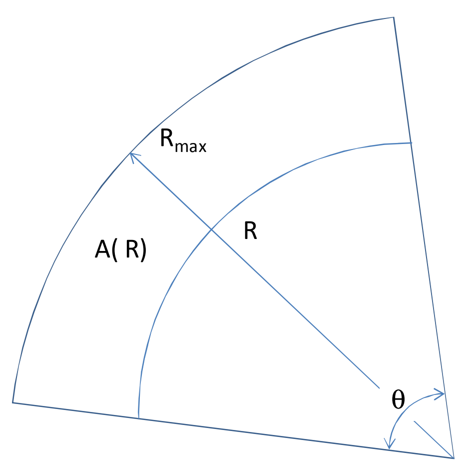

Figure 26: Schematic of converging section

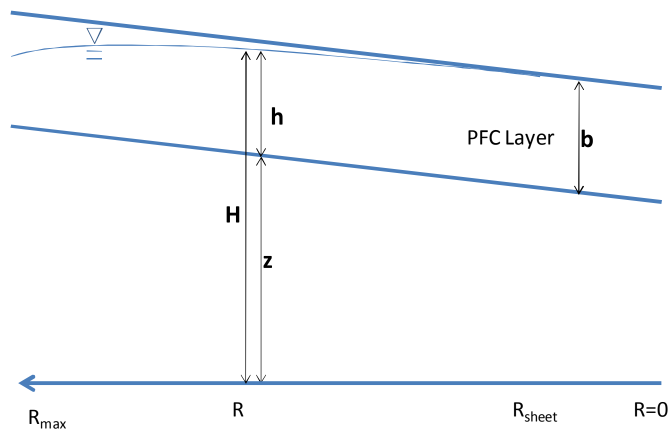

Figure 27: Cross section view

For the discharge through station R, the area is:

$$A(R) = \frac{\theta}{2\pi}\pi Rmax^2 - \frac{\theta}{2\pi}\pi R^2 = \frac{\theta}{2} * (R_{max}^2 - R^2) \tag{4.2}$$

where θ is the included angle. The flow rate is given by:

$$Q(R) = r * A(R) = \frac{r\theta}{2}(R_{max}^2 - R^2) \tag{4.3}$$

The unit flux past radius R is the flow rate divided by the arc length at R:

$$U(R) = \frac{Q(R)}{\theta R} = \frac{r}{2R}(R_{max}^2 - R^2) \tag{4.4}$$

Because flow through a PFC is the problem of interest, Darcy's law is the appropriate form of the momentum equation:

$$U = K * h * \frac{dH}{dR} \tag{4.5}$$

The hydraulic gradient decomposes as:

$$\frac{dH}{dR} = \frac{dh}{dR} + \frac{dz}{dR} = \frac{dh}{dR} + s \tag{4.6}$$

where s is the slope, which due to the choice of coordinate system is positive for a down-slope flux.

In order to agree with this convention, a positive hydraulic gradient in Darcy's law should cause a down-slope flux. This requirement is satisfied because the coordinate system for this problem is reversed from our usual system—the origin is at the down-hill end of the domain rather than the uphill end.

Combining Equations (4.4), (4.5) and (4.6) gives the ODE for PFC flow on a converging section:

$$Kh\left(\frac{dh}{dR}+s\right)=\frac{r}{2R}\left(R_{max}^2-R^2\right)$$

or (4.7)

$$\frac{dh}{dR}=-s+\frac{r}{2Kh}\left(\frac{R_{max}^2-R^2}{R}\right)$$

This ODE is first-order, but non-linear, and an analytical solution is not known at this time. The same general features of the ODE for the linear section (see Charbeneau and Barrett, 2008) also apply to the ODE for the converging section:

1. The location of maximum radius, R_{max}, is automatically a no-flow boundary because for $R = R_{\max}$, $\frac{dh}{dR} = -s$, and from (4.6) this implies $\frac{dH}{dR} = 0$.
2. The thickness initially increases as the radius decreases because $s > 0$.
3. At the location of maximum depth $\frac{dh}{dR} = 0$ and the variables are related by

$$h_{\max}=\frac{r}{Ks}\frac{R_{\max}^2-R^2}{2R} \tag{4.8}$$

The ODE of (4.7) applies on a domain where flow is completely contained within the PFC. To integrate the ODE, an initial point is needed somewhere on the solution curve. The appropriate initial point depends on problem conditions. When flow is completely contained in the PFC the saturated thickness at the edge of the domain can be specified; in the case of combined PFC and sheet flow the appropriate point is the PFC thickness taken at the location where sheet flow begins. This location is found by equating (4.4) and (4.5) and setting the hydraulic gradient to the pavement slope. Note that a hydraulic gradient equal to the pavement slope is a requirement for sheet flow to occur.

$$r * \left(\frac{R_{max}^2 - R^2}{2R}\right) = K * b * s \tag{4.9}$$

Applying the quadratic formula gives the location where sheet flow begins:

$$R_{sheet} = \frac{1}{2}\left(-\frac{2Kbs}{r}\right) + \frac{1}{2}\sqrt{\left(\frac{2Kbs}{r}\right)^2 + 4R_{max}^2}$$

or (4.10)

$$R_{sheet} = \left(-\frac{Kbs}{r}\right) + \sqrt{\left(\frac{Kbs}{r}\right)^2 + R_{max}^2}$$

As an analytical solution is not known at this time, a numerical solution was developed using a fourth order Runge-Kutta scheme (Figure 28). Comparisons between linear and converging sections are discussed in Section 4.3 of this dissertation.

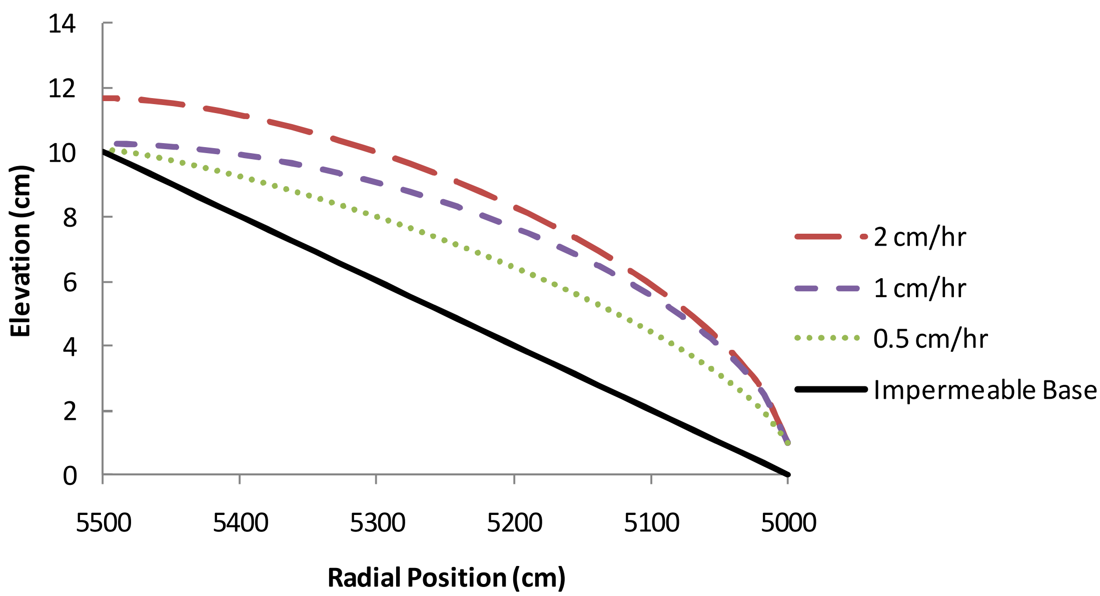

Figure 28: Drainage depth profiles for a converging section with maximum radius of 55m, hydraulic conductivity 1cm/s, slope of 2%, initial depth of 1cm at R=5000cm and range of rainfall rates.

4.2.2 PFC Flow Only

The first simulation of the converging section set the PFC thickness to 15cm so that all of the drainage would be contained in the pavement. The model reached a steady state solution after 21,760 seconds of rainfall and showed good agreement with the steady state ODE (Figure 29). The linear kinematic boundary condition of Equation (4.1) was applied to the converging section. An order of magnitude analysis suggests that this approximation is appropriate (see Section 3.7.3).

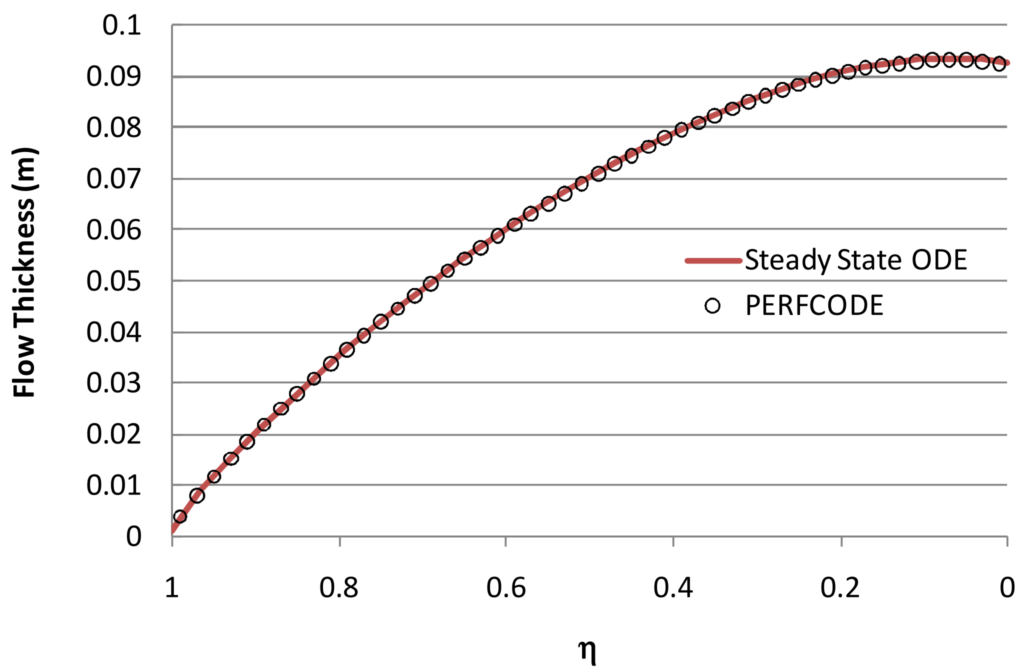

Figure 29: Depth profile for converging section with drainage by PFC flow only

4.2.3 Sheet Flow Only

The next simulation set the PFC thickness to zero so that all drainage occurred as sheet flow. Steady state was reached in 196 seconds and had good agreement with the analytical solution (Figure 30).

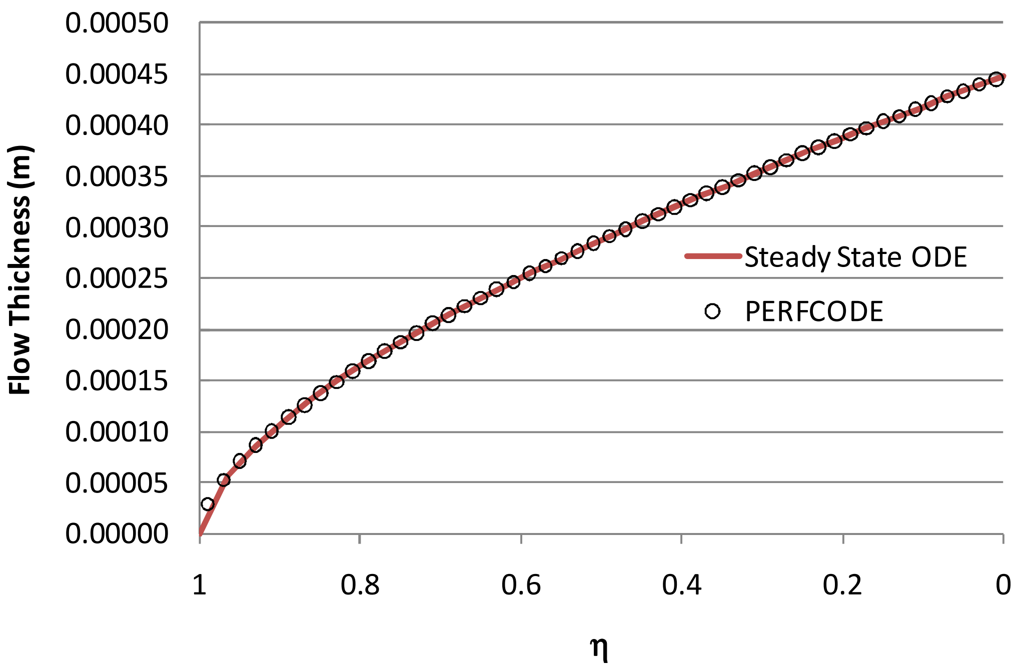

Figure 30: Depth profile a converging section with sheet flow only

4.2.4 Combined Flow

This simulation set the PFC thickness to 5cm so that drainage occurred both within the pavement and on the surface. The model reached a steady state solution in 5,398 seconds, and showed generally good agreement with the analytical solution (Figure 31).

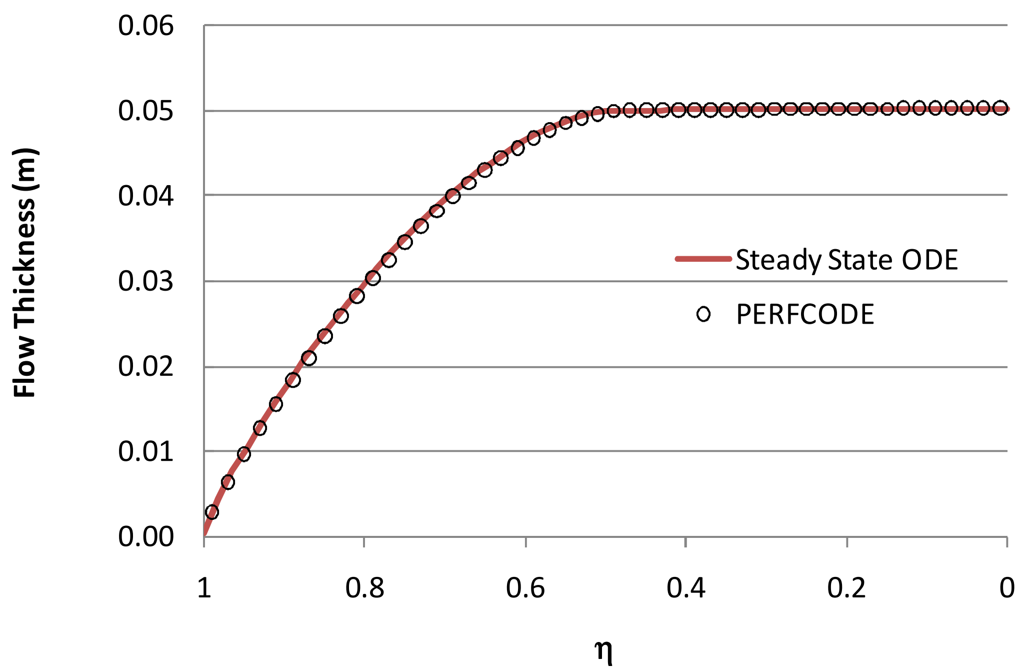

Figure 31: Depth profile for a converging section with combined PFC and sheet flow

4.2.5 Runoff Hydrographs

For each simulation the discharge from the outflow boundary was tracked through time. These rising hydrographs are plotted on a logarithmic scale on account of the wide range of times required to reach steady state (Figure 24). Hydrographs from the converging section show the same general trends as the linear section (see page 72). A comparison of the linear and converging cases is presented in the next section.

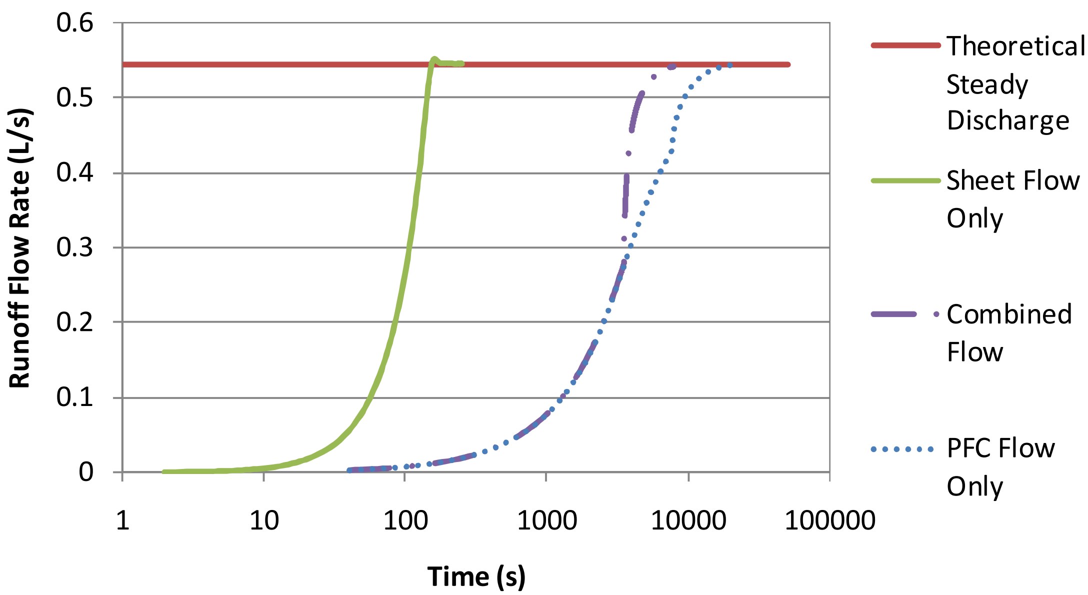

Figure 32: Runoff hydrographs for converging section

4.3 Comparison of Linear and Converging Sections

So far, this chapter has considered two extremes of roadway geometry: perfectly straight and perfectly curved. Most real roads fall into neither category, but these extreme cases are useful for bounding the range of problems likely to be encountered in practice.

A converging section has the effect of increasing the flow depth along the drainage path. This increase occurs because the width available for drainage decreases as the flow moves toward the center of a curve. How much the depth increases compared to a linear section depends on the radius of curvature and on the road width.

Depth profiles for the combined flow scenarios (10m width, 3% cross slope, 1 cm/hr rainfall, 5cm PFC thickness, 1 cm/s PFC hydraulic conductivity, 60m radius of curvature at center) are shown in Figure 33. As expected, the flow thickness for the converging section is slightly higher than the linear section and the difference increases as the effect of convergence becomes more pronounced moving down the slope. The difference drops sharply near the transition to sheet flow because the porosity no longer amplifies the depth. Sheet flow also begins slightly higher on the converging section.

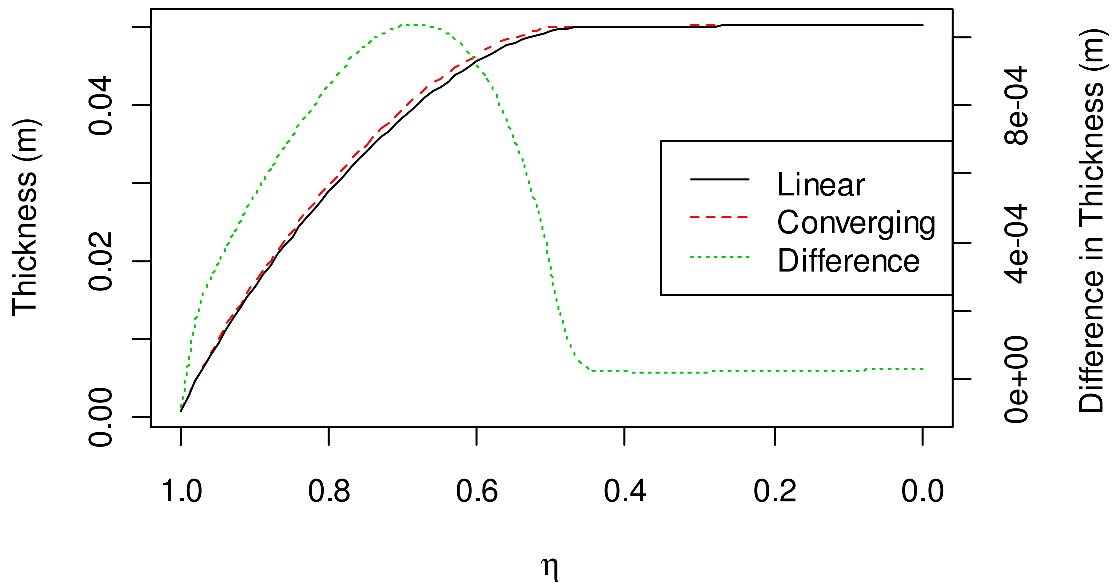

Figure 33: Comparison of exact solutions for steady state flow thickness on linear and converging sections, other parameters given in Table 2 and Table 3.

The effect of a converging section on flow depth can be determined from the steady state ODEs, but the influence on the outflow hydrograph requires numerical simulation. The hydrographs for the combined PFC/Sheet Flow scenarios from Figure 24 and Figure 32 are plotted together in Figure 34 to illustrate the effect of convergence on the outflow hydrograph. Unlike previous the figures, an arithmetic scale is used because the relevant time range is smaller. The converging section begins sheet flow earlier than the linear section by 110 seconds. The figure also shows the evolution of the depth at the domain boundary. Adding this line to the plot emphasizes that the sharp increase in the flow rate is associated with the transition to sheet flow.

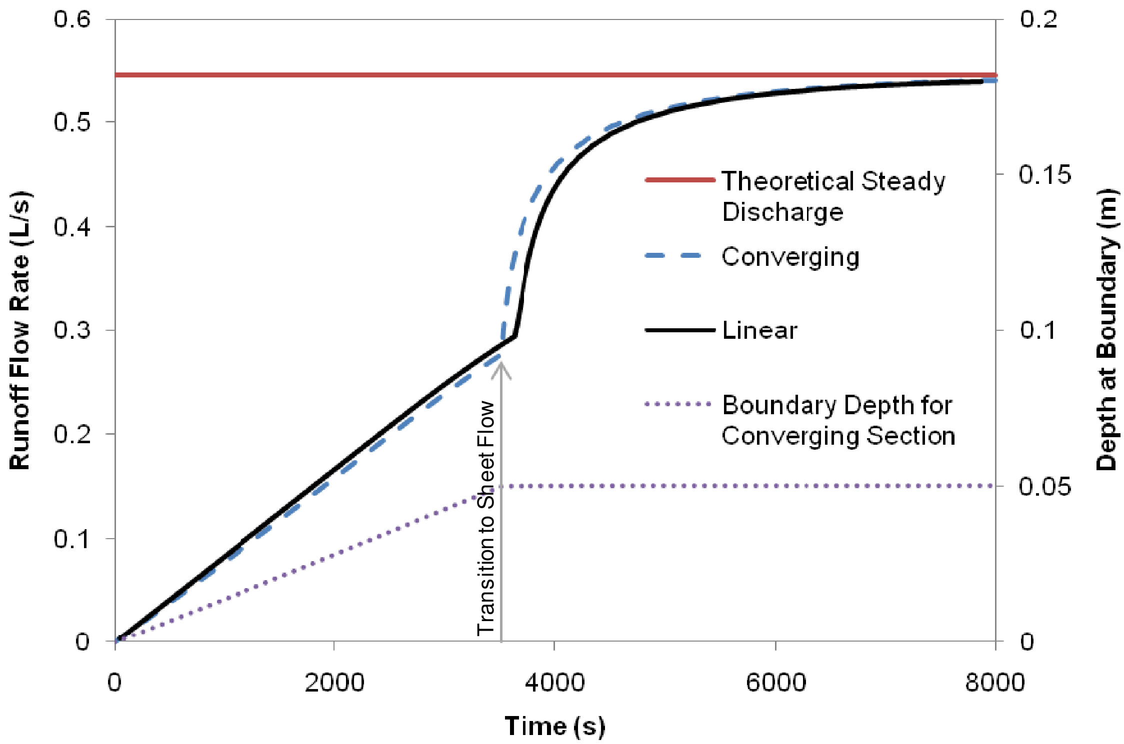

Figure 34: Hydrograph comparison for linear and converging sections, PFC thickness was 0.05m

4.4 Stability

A numerical method is considered to be stable if errors introduced into the solution are not amplified by the method (Ferziger and Peric, 2002). An amplification factor for a method may be computed by introducing a small error into the solution (as a Fourier component) at time level n and seeing how the error grows by time level $n + 1$. The amplification factor is the ratio of these errors. An amplification factor of less than unity is required for a method to be stable. This analysis of stability is called the von Neumann stability analysis. The von Neumann approach applies only to linear problems; there are no comprehensive methods for assessing stability of non-linear problems (Ferziger and Peric, 2002). The non-linear coefficients are frozen here so that the von Neumann approach may be used.

The model equation for stability this analysis is formulated in terms of the total head (see Equation (3.49)) rather than the depth for simplicity. With reference to Equation (3.49), the substitutions $\ell = \Delta x$; $w = \Delta y$; $\Delta A = \Delta x \Delta y$; $D = K * h + \frac{h^{\frac{5}{3}}}{n\sqrt{S_0}}$ give a simplified expression of the model equation

$$\frac{\partial H_{i,j}}{\partial t} = \frac{D}{\Delta x^2}\left(H_{i-1,j} - 2H_{i,j} + H_{i+1,j}\right) + \frac{D}{\Delta y^2}\left(H_{i,j-1} - 2H_{i,j} + H_{i,j+1}\right) + r \tag{4.11}$$

In this formulation the diffusion coefficient D is assumed to be a constant so the equation is linear. Applying Crank-Nicolson to the time dimension gives

$$\begin{aligned}\frac{H_{i,j}^{n+1} - H_{i,j}^{n}}{\Delta t} &= \frac{1}{2}\frac{D}{\Delta x^2}\left(H_{i-1,j}^{n} - 2H_{i,j}^{n} + H_{i+1,j}^{n}\right) \\ &+ \frac{1}{2}\frac{D}{\Delta y^2}\left(H_{i,j-1}^{n} - 2H_{i,j}^{n} + H_{i,j+1}^{n}\right) \\ &+ \frac{1}{2}\frac{D}{\Delta x^2}\left(H_{i-1,j}^{n+1} - 2H_{i,j}^{n+1} + H_{i+1,j}^{n+1}\right) \\ &+ \frac{1}{2}\frac{D}{\Delta y^2}\left(H_{i,j-1}^{n+1} - 2H_{i,j}^{n+1} + H_{i,j+1}^{n+1}\right) + r\end{aligned} \tag{4.12}$$

The value of the solution at $H_{i,j}^{n}$ can be expressed as a Fourier component

$$H_{i,j}^{n} = A^n e^{Ipi\Delta x} e^{Iqj\Delta y} \tag{4.13}$$

where A is the amplitude at time level n, $I = \sqrt{-1}$, and p and q are the wave numbers in the x and y directions and i, j are the indices of the grid cell. The details of the substitution of (4.13) into (4.12) are shown for the first term on the right side of (4.12).

$$\frac{1}{2}\frac{D}{\Delta x^2}\left(A^n e^{Ip(i-1)\Delta x} e^{Iqj\Delta y} - 2A^n e^{Ipi\Delta x} e^{Iqj\Delta y} + A^n e^{Ip(i+1)\Delta x} e^{Iqj\Delta y}\right) \tag{4.14}$$

Making similar substitutions for the remaining terms and dividing by $A^n e^{Ipi\Delta x} e^{Iqj\Delta y}$ gives

$$\frac{1}{\Delta t}\left(\frac{A^{n+1}}{A^n} - 1\right)$$

$$\begin{aligned} = \quad & \frac{1}{2}\frac{D}{\Delta x^2}\left(e^{-Ip\Delta x} - 2 + e^{Ip\Delta x}\right) \\ & + \frac{1}{2}\frac{D}{\Delta y^2}\left(e^{-Iq\Delta y} - 2 + e^{Iq\Delta y}\right) \\ & + \frac{1}{2}\frac{D}{\Delta x^2}\left(\frac{A^{n+1}}{A^n}e^{-Ip\Delta x} - \frac{2A^{n+1}}{A^n} + \frac{A^{n+1}}{A^n}e^{Ip\Delta x}\right) \\ & + + \frac{1}{2}\frac{D}{\Delta y^2}\left(\frac{A^{n+1}}{A^n}e^{-Iq\Delta y} - \frac{2A^{n+1}}{A^n} + \frac{A^{n+1}}{A^n}e^{Iq\Delta y}\right) \end{aligned} \tag{4.15}$$

Making use of the identity:

$$e^{-Ip\Delta x} + e^{Ip\Delta x} = 2\cos(p\Delta x) \tag{4.16}$$

and defining the amplification factor $G = \frac{A^{n+1}}{A^n}$ the linearized model equation can be written as an equation for the amplification factor

$$\begin{aligned} \frac{1}{\Delta t}(G-1) = \frac{D}{\Delta x^2}(\cos(p\Delta x) - 1) + \frac{D}{\Delta y^2}(\cos(q\Delta y) - 1) \\ + G\left(\frac{D}{\Delta x^2}\right)(\cos(p\Delta x) - 1) + G\left(\frac{D}{\Delta y^2}\right)(\cos(p\Delta y) - 1) \end{aligned} \tag{4.17}$$

Solving this expression for the amplification factor gives

$$G = \frac{1}{1 + 4\left(\frac{D}{\Delta x^2}\right)\sin^2\left(\frac{p\Delta x}{2}\right) + 4\left(\frac{D}{\Delta x^2}\right)\sin^2\left(\frac{p\Delta y}{2}\right)} \tag{4.18}$$

Equation (4.18) shows that the amplification factor will always be less than unity because the coefficient D is always positive and $\sin^2$ is also always positive. This stability analysis has shown that the Crank-Nicolson method is unconditionally stable for a linear diffusion problem. The actual model equations however are non-linear and so may exhibit some stability problems.

4.5 Model Convergence

A numerical solution is said to converge if the errors in the solution decrease as the grid is refined. This model was developed using central differencing scheme. Based on a Taylor series expansion, central differencing schemes can be shown to have a second-order truncation error (Ferziger & Peric, 2002). This means that the largest term in the neglected part of the Taylor series expansion contains the grid spacing term raised to the second power. The observed order of the truncation error for a model can be obtained by comparing model runs for different grid sizes.

The model domain selected for the convergence study is the same domain studied in Section 4.2.4—10m width, 3% cross slope, 1 cm/hr rainfall, 5cm PFC thickness, 1 cm/s PFC hydraulic conductivity, 60m radius of curvature at the roadway centerline. Double precision variables were used for the convergence study to assure that differences in the solution at the various grid sizes were due to truncating the Taylor series approximations for derivatives and not due to floating point errors. Even with double precision variables, the solutions using a 10cm grid was indistinguishable from the solution using a 5cm grid. A plot of the solution for various grid sizes shows that the model converges to the same solution independent of the grid size (Figure 35).

For the purposes of this convergence study, the model solution for a nominal grid spacing of 5cm was used as the exact solution. The difference between the model solution and the exact (5cm) solution, or the residual, was computed for each point. The portion of the domain in PFC flow had higher residuals than the sheet flow part of the domain (Figure 36). That the sheet flow and PFC flow parts of the domain would have different behaviors is not completely unexpected because the governing equations differ. What should be consistent though, is the rate at which the errors change with grid size.

The observed convergence rate of the model was investigated by computing the residual with respect to the 5cm grid at several locations along a cross section in the center of the domain (at different points along the cross-section for the longitudinal station in the middle of the domain). The grid refinement study (Figure 37) shows that the model gives second order behavior as the grid is refined.

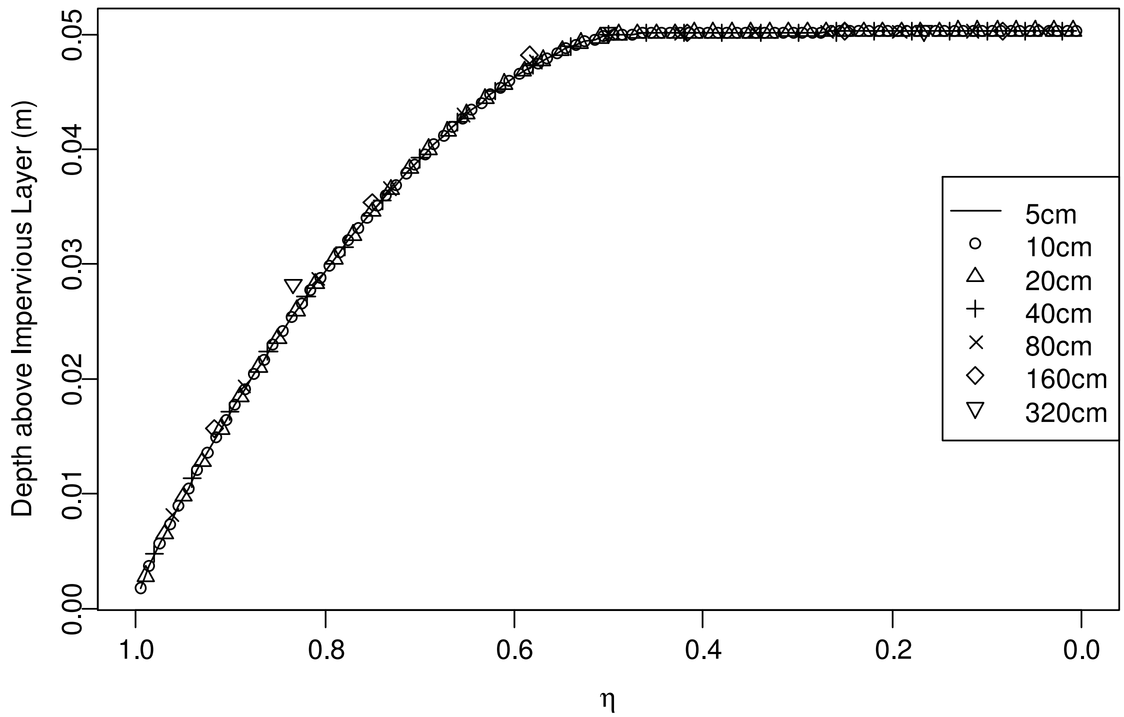

Figure 35: Steady state depth profile for various grid sizes

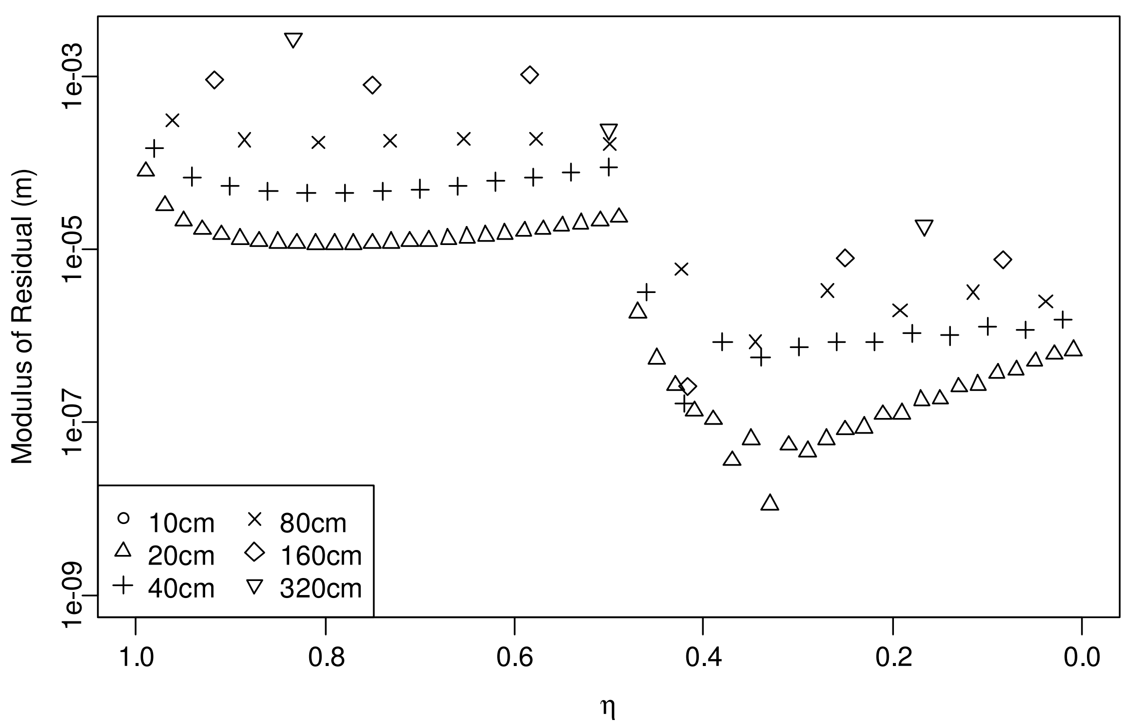

Figure 36: Residual with respect to 5cm grid by location,
all residuals for 10cm grid were zero

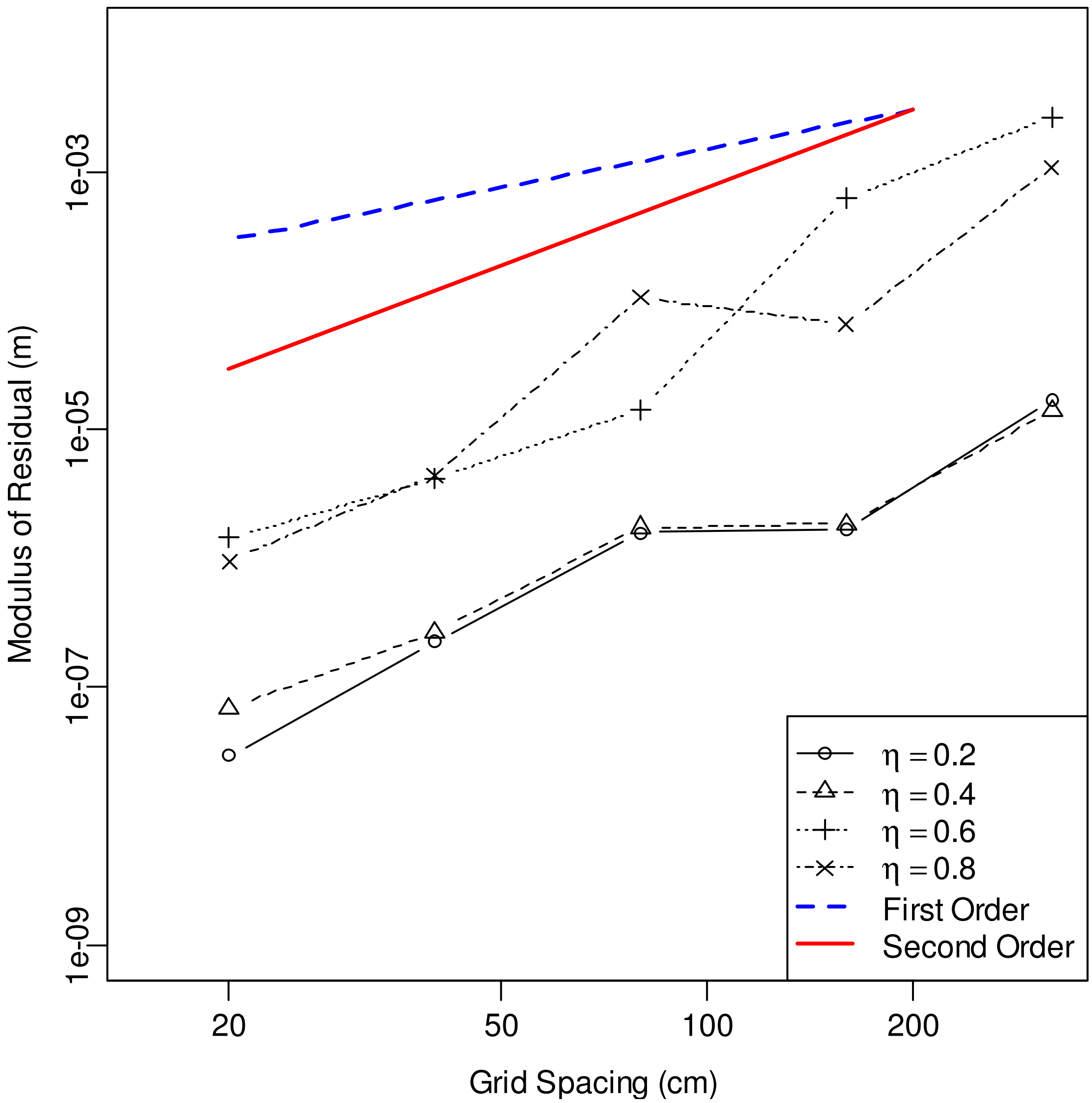

Figure 37: Grid refinement study

CHAPTER 5: COMPARISON WITH FIELD DATA

This chapter compares model results with field data from a monitoring site constructed on Loop 360, near Austin, Texas. The variable of interest remains the water depth on the highway, but measurements of this quantity are difficult to make. Indeed, one motivation for developing a model is to estimate quantities that are difficult to measure. What has been measured is the rainfall depth and runoff hydrograph at the monitoring site. The measured rainfall is taken as input and the variation of water depth through the storm is computed along with the runoff hydrograph. Reasonable agreement between the modeled and measured hydrographs lends credibility to the associated depth predictions.

5.1 Construction of Field Monitoring Site

The monitoring site, located on southbound Loop 360 near Austin, Texas (Figure 38), was initially established as a monitoring site for stormwater runoff in 2004. Later that year, the highway was repaved with PFC. Lower concentrations of total suspended solids and total heavy metals were observed in the runoff, which generated interest in additional research.

In the autumn of 2006 equipment for automatic sample collection was installed at the Loop 360 monitoring site. The field site was designed to measure the runoff hydrograph and to collect water quality samples. A drainage system was constructed using 4-inch PVC pipe to collect runoff from an 18m (60 ft) length of roadway and direct it to the sampler. A 6-inch H-flume was used to measure the flow rate from the drainage pipe. An ISCO 4230 bubbler flow meter measured the water depth in the H-flume and calculated the flow rate. An ISCO 3700 portable sampler used the flow rate to collect flow-weighted water samples. An ISCO 674 tipping bucket rain gage recorded rainfall. Both rainfall and runoff were recorded in five-minute intervals, rainfall as the total depth and runoff as the average flow rate. Refer to Stanard (2008) for additional details on the construction of the monitoring site and programming of the equipment.

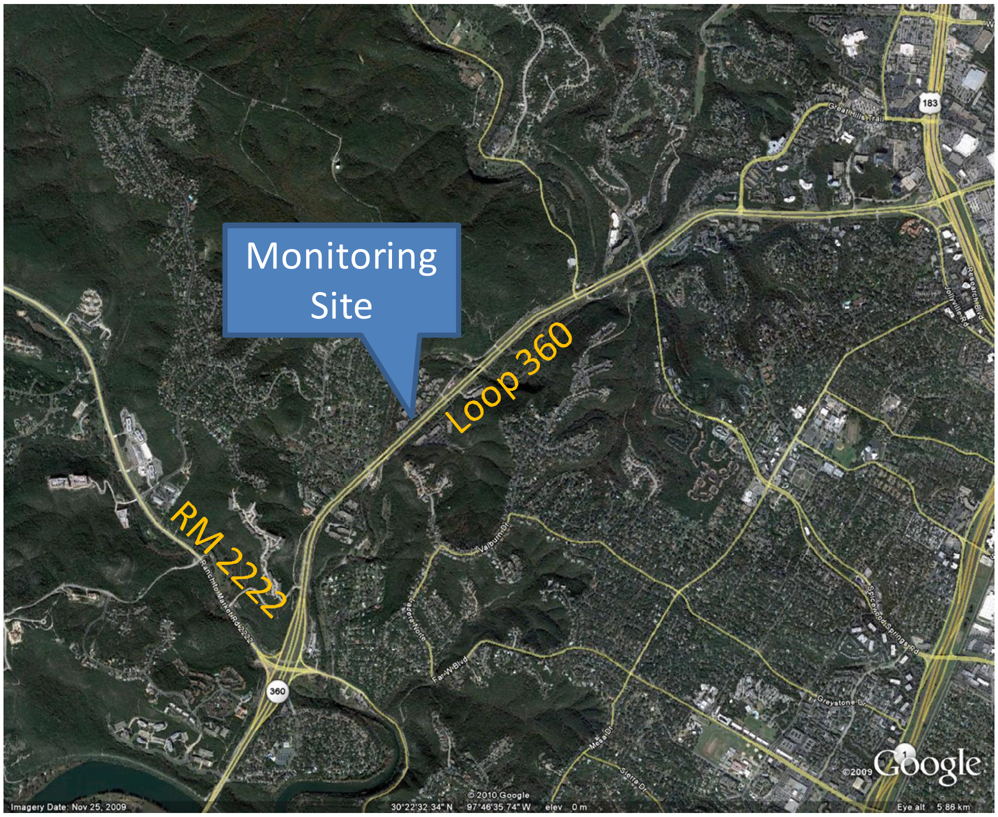

Figure 38: Aerial map of Loop 360 monitoring site (Google 2010)


Figure 39: Photograph of H-flume and drainage pipe at Loop 360 monitoring site

5.2 Model Inputs and Parameters

At the location of the monitoring site, Loop 360 is a four-lane divided highway. The monitoring site is situated on the right-hand shoulder of the south-bound traffic lanes. The traffic lanes (24ft) and right hand shoulder (10ft) slope to the driver's right-hand side at cross-slopes of 2% and 4%, respectively. The left shoulder (6ft) drains to the left at a cross-slope of 4%. The entire section has a longitudinal slope of 2.3%.

The roadway geometry for Loop 360 was used to develop input files for the model. The model domain was extended beyond the 60ft length monitored so that errors in the kinematic condition on the east and west boundaries would not influence the solution in the domain of interest. Kinematic boundary conditions were used on all four sides of the domain. In Figure 40, the middle third of the domain corresponds to the location of the drainage pipe at the monitoring site.

The storm event of July 20, 2007 was selected for simulation because it was a large enough to cause substantial sheet flow. The hydraulic conductivity and porosity for this simulation correspond to values measured by Klenzendorf (2010) for a nearby location on the same highway. Values of Manning's n have not been measured for PFC, but a value of 0.015 s / $m^{1/3}$ appears appropriate considering the analysis of Charbeneau et al. (2009). Table 4 summarizes the model parameters.

Table 4: Model Parameters for Loop 360 Monitoring Site

Parameter	Unit	Value
Roadway width	m	12.2
Domain length	m	36.6
Cross Slope	%	various
Hydraulic Conductivity	cm/s	3
PFC Thickness	cm	5
Porosity	--	0.2
Manning's n	$s/m^{1/3}$	0.015
Rainfall Rate	cm/hr	various

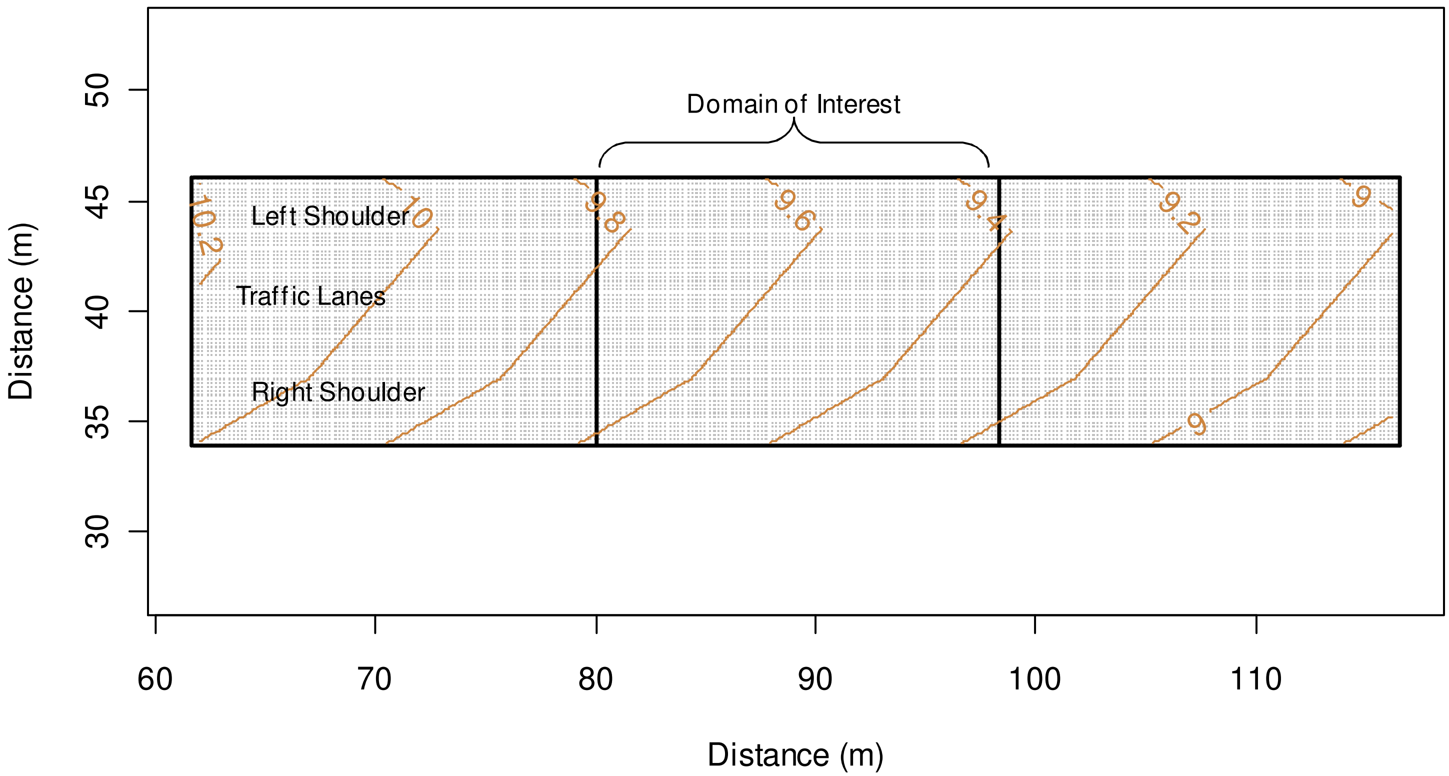

Figure 40: Simulation domain for Loop 360 monitoring site showing elevation contours (m) and location of grid cell centers

The storm of July 20, 2007 occurred during an unusually wet summer, and was a particularly large storm. A total of 48mm (1.9 in) of rainfall were recorded at the monitoring site over a 5.6 hour period. The peak rainfall depths on a five, fifteen and sixty minute basis were 6.6mm 18mm, and 39mm (0.26in, 0.71in, 1.56in), respectively. On a sixty minute basis, the storm corresponded to a return period of about 2 years (Chow et al., 1988 pg. 450) The highest five-minute rainfall intensity was 80mm/hr.

The field measurements provided the time at the end of five-minute periods for which the rainfall total was reported. This information was prepared for use in the model by computing the rainfall intensity (mm/hr or m/s) and inserting points at the beginning of each five-minute interval (Figure 41). The purpose of this approach was to facilitate use of a linear interpolation routine for selecting the proper rainfall rate for any time during the model simulation.

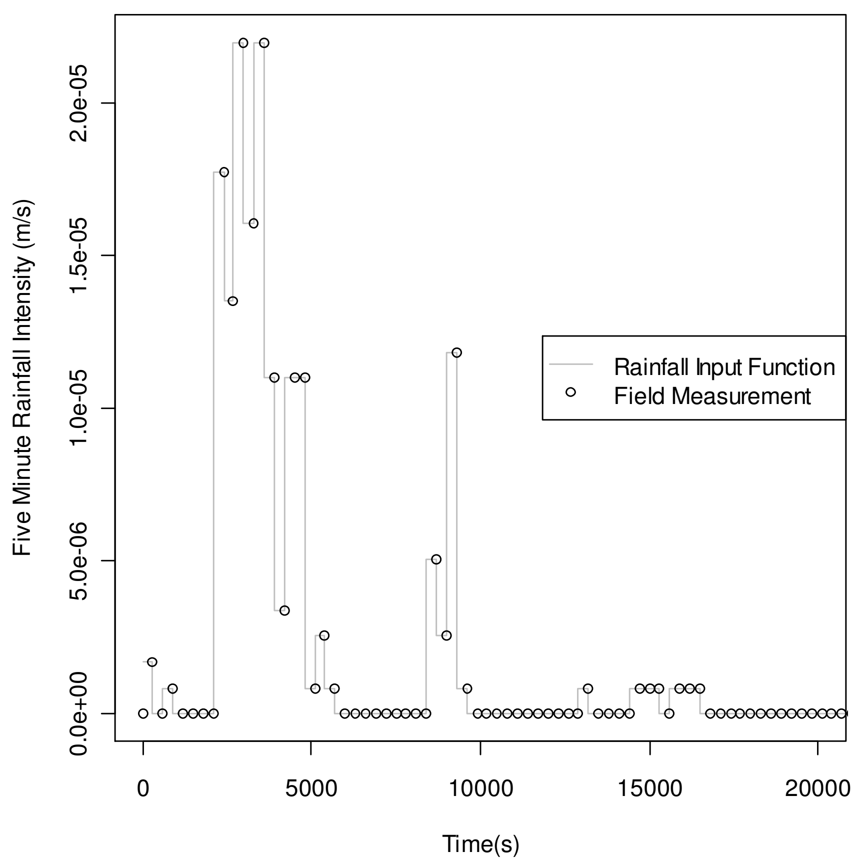

Figure 41: Measured rainfall and model input function for Loop 360 monitoring site on July 20, 2007

5.3 Results and Discussion for event of July 20, 2007

The rainfall function and other parameters were used as inputs for a simulation over 20,000 seconds. During the simulation, the runoff through the domain's southern boundary and was computed for each time step. The overall maximum depth and the maximum depth in the middle of the domain were also tracked throughout the simulation. This distinction in the depths was necessary due to oscillations near the boundary.

A model time step of 5s was used when the all of the drainage was contained within the pavement, but a step of 0.1s was needed during sheet flow for the model to remain stable. In order to make a fair comparison with the field measurements, the calculated flow rates were averaged over five minute intervals. A weighted average flow rate was used so that a five-minute interval containing two sizes of time step has the proper flow rate. These averaged flow rates showed generally good agreement with the field measurements (Figure 42). The model predicted peak flows of the proper time and magnitude, and the shape of the hydrograph generally matches the field observations.

The model predicted a peak flow 3.7 L/s, which is 97% of the measured value of 3.8 L/s. The difference between the modeled and measured flow rates (residual) had a mean -0.029L/s, median 0.021 L/s, standard deviation 0.24 L/s and standard error of the mean 0.029 L/s. The largest residuals were associated with high flow rates. This comparison suggests that the model parameters were consistent with field conditions and lends credibility to the associated depth predictions.

A plot of the model solution for maximum depth conditions shows sheet flow occurring in both traffic lanes and on the right hand shoulder (Figure 43). Within the domain of interest, the depth contours are parallel to the roadway centerline. This result is consistent with a straight road and constant slopes. Some oscillations in the depth contours appear outside of the domain of interest, especially near the western boundary. It is believed that these oscillations are related to using the kinematic outflow boundary condition from the east end of the domain on the inflow boundary at the west end.

During this simulation, maximum depth in the domain of interest was 0.05142m above the impervious layer, which represents a sheet flow depth of 1.4mm. This maximum occurred near the edge of the right traffic lane (Figure 44). The exact location was 3.2m from the southern edge of the domain; since the shoulder width is 3.05m, the maximum depth occurred 15cm from the shoulder. This peak occurred 1 hour after rainfall began (3599.9s) and during the peak rainfall intensity of 80 mm/hr.

The model results show that sheet flow begins 1.6m due south of the grade break for the left hand shoulder (Figure 44). Under most conditions, this break in slope acts as a no-flow boundary within the domain; the no flow condition is assumed here for purposes of comparison with the analytical model even though some flow does occur. At the peak rainfall rate for this storm, the analytical model (see Charbeneau & Barrett 2008 and Eck et al. 2010) predicts sheet flow at 2m down the drainage slope or 1.4m due south of the grade break (2% cross slope, 2.3% longitudinal slope; 3.048% drainage slope). This seems a reasonable match, considering that the numerical model is not at steady state, and that boundary condition is approximate.

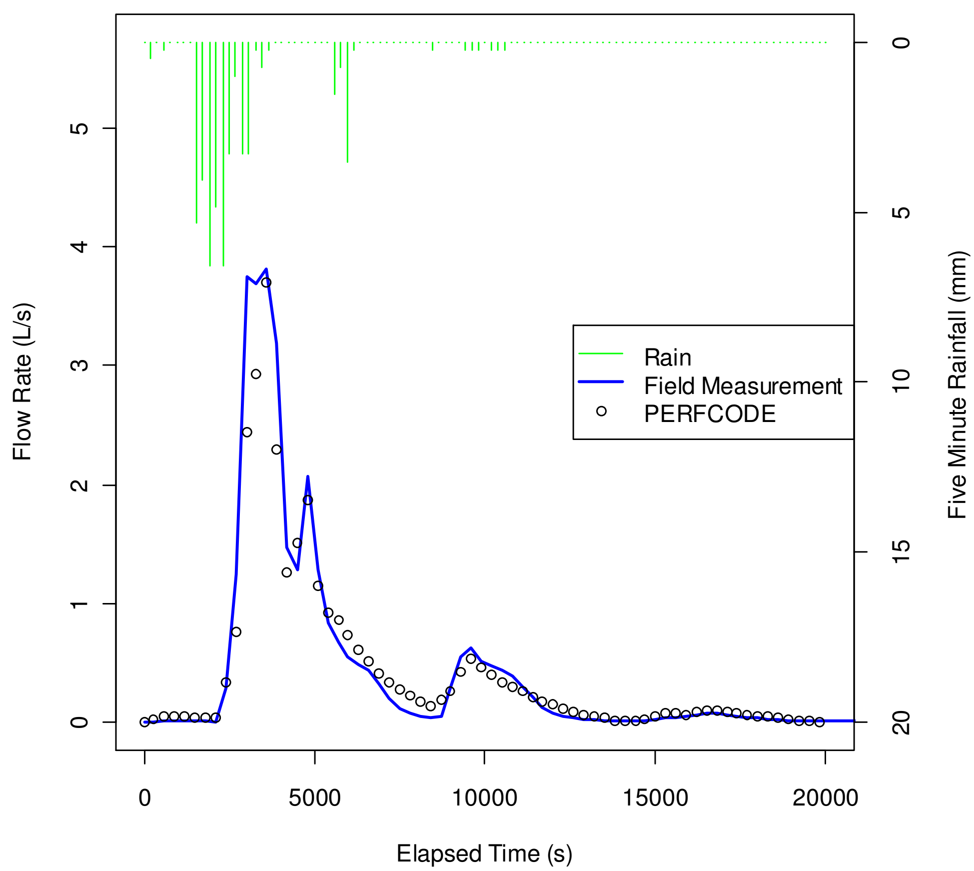

Figure 42: Comparison of modeled and measured hydrographs for storm of July 20, 2007

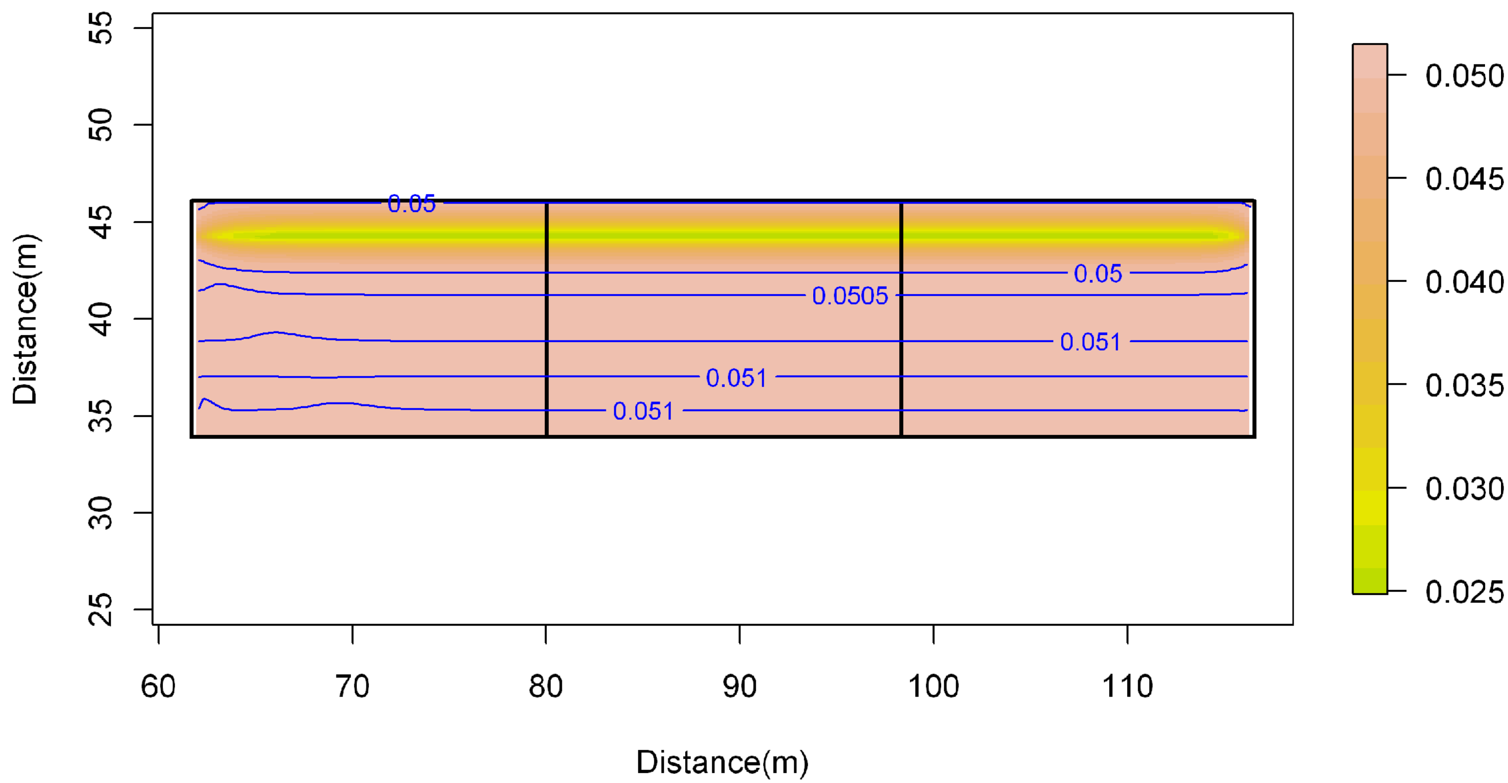

Figure 43: Water depth above impervious layer (m) for Loop 360 during maximum depth conditions on July 20, 2007. The PFC thickness was 0.05m: contours correspond to sheet flow conditions.

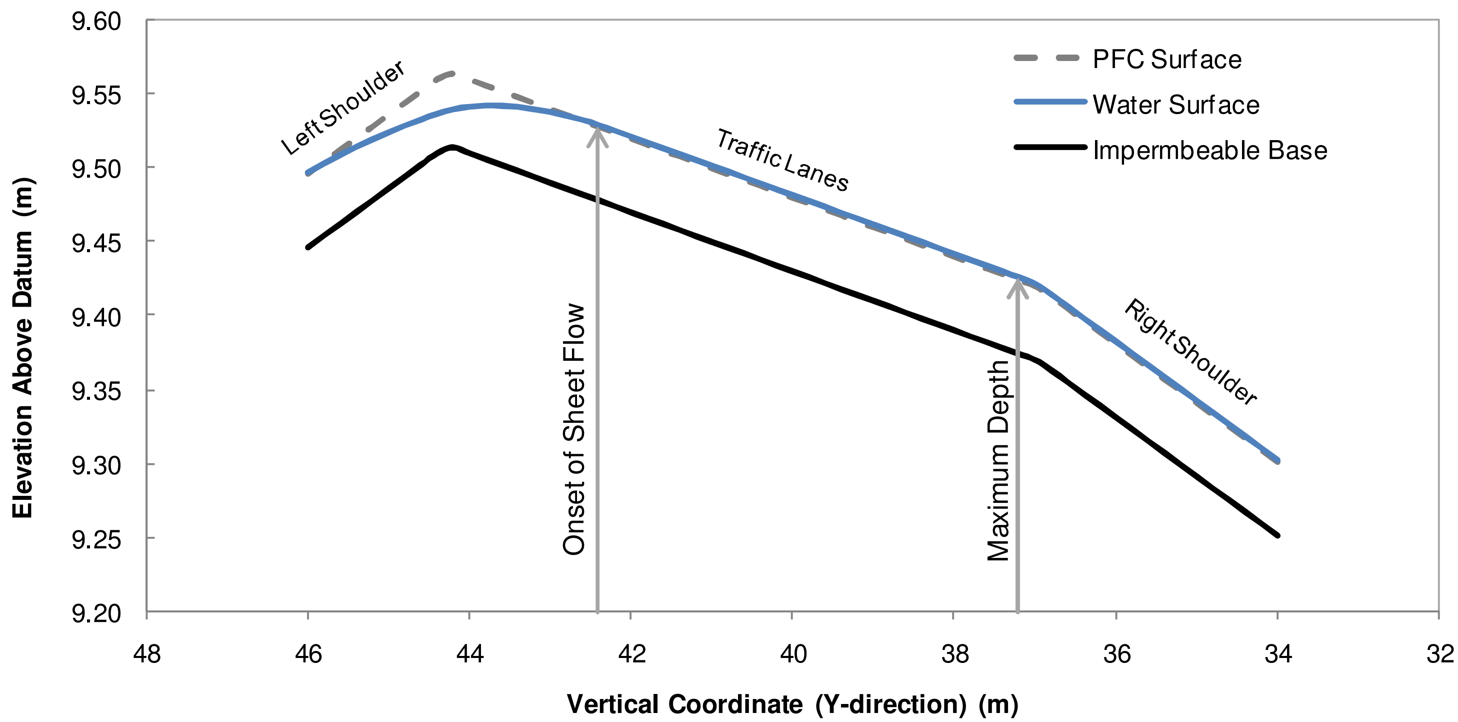

Figure 44: Profile through maximum depth section; the horizontal coordinate is 94.42m

In addition to examining water depths during an actual rainstorm, this example also provides an opportunity to illustrate the effect of using an under-relaxation factor in the non-linear iteration loop. Figure 45 shows how the solution at a grid cell just on the right shoulder evolves during a time step shortly after peak rainfall has started (time 2821.9s). At the previous time-step the traffic lanes have sheet flow and the shoulder is in PFC flow. The model is trying to determine if the shoulder is also now in sheet flow or if it remains in PFC flow. Without the under-relaxation, the solution bounces between inside and outside of the PFC surface, the grid cell shown has the largest error, and the solution does not converge for the time step. This 'hunting' behavior does not occur with the relaxation factor and the model concludes that the depth at this location remains in the PFC for this time-step.

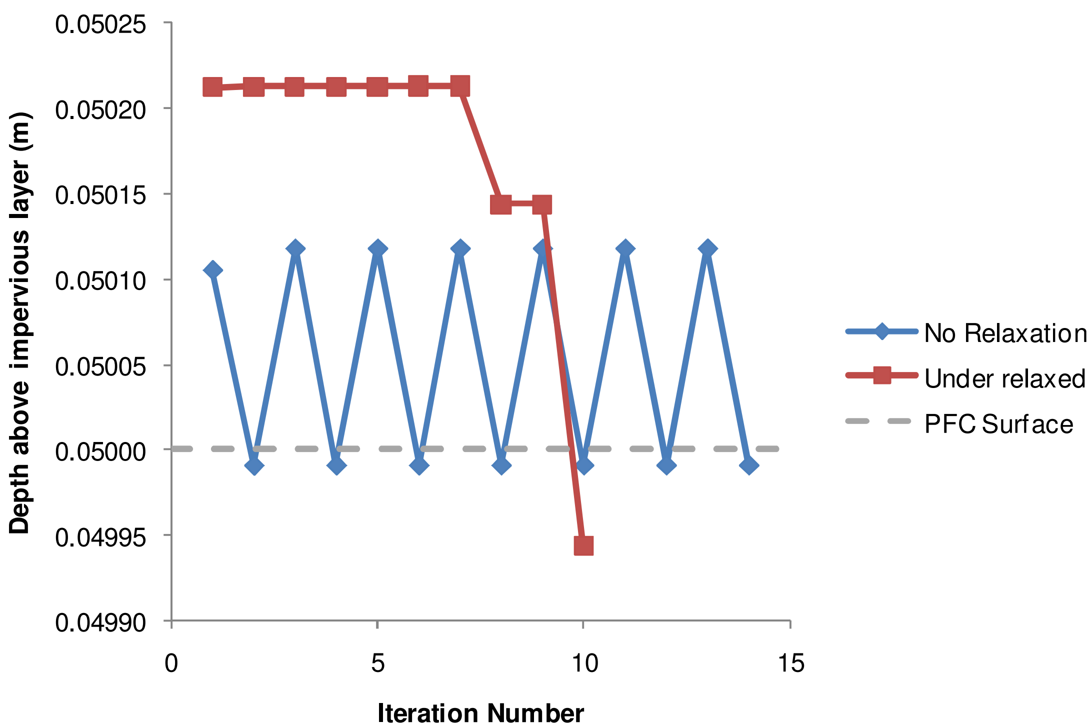

Figure 45: Solution history for an interior point (grid cell 2138) with and without under-relaxing the non-linear iteration

5.4 Loop 360 with and without PFC

One opportunity afforded by the simulation model is to compare results with and without PFC for the same storm event. Such an analysis gives direct insight about how PFC changes the drainage hydraulics as compared to conventional pavement and is the topic of this section. The same roadway geometry and simulation parameters used for the comparison with field measurements were used in this simulation, except that the thickness of the PFC layer was set to zero so that all drainage occurred as sheet flow.

The simulated hydrograph for Loop 360 without PFC is shown in Figure 46 along with the simulated hydrograph corresponding with a 5cm PFC layer. Both hydrographs have been time averaged over the reporting period for rainfall measurements (5 minutes). The absence of a PFC layer appears to make the hydrograph rise and fall faster, especially later in the storm (10,000s) when flow would be contained within the PFC. The PFC layer reduced the magnitude of this small peak by about 70% and delayed it five minutes, or one averaging period.

A PFC layer might be expected to delay the runoff hydrograph due to storage within the pavement, but that effect is not observed in this case. The high rainfall intensity quickly overwhelmed the capacity of the PFC layer, causing most of the drainage to occur as sheet flow so the hydrographs exhibit a similar shape.

The presence of a PFC layer reduced the sheet flow thickness during this event (Figure 47). The PFC layer prevented sheet flow entirely for the left part of the left lane and also on the left shoulder. In regions where sheet flow occurred over PFC, the PFC layer reduced the depth by an average of 0.35mm. Some small oscillations are noted in the sheet flow profile near the right shoulder and were associated with sharp change in cross slope.

In addition to reducing the magnitude of sheet flow on the highway, PFC also reduced the duration that sheet flow was present. Simulation results showed that sheet flow depths in excess of 0.1mm were present for about 1600 seconds when the PFC layer was present and for 8580 seconds without the PFC layer.

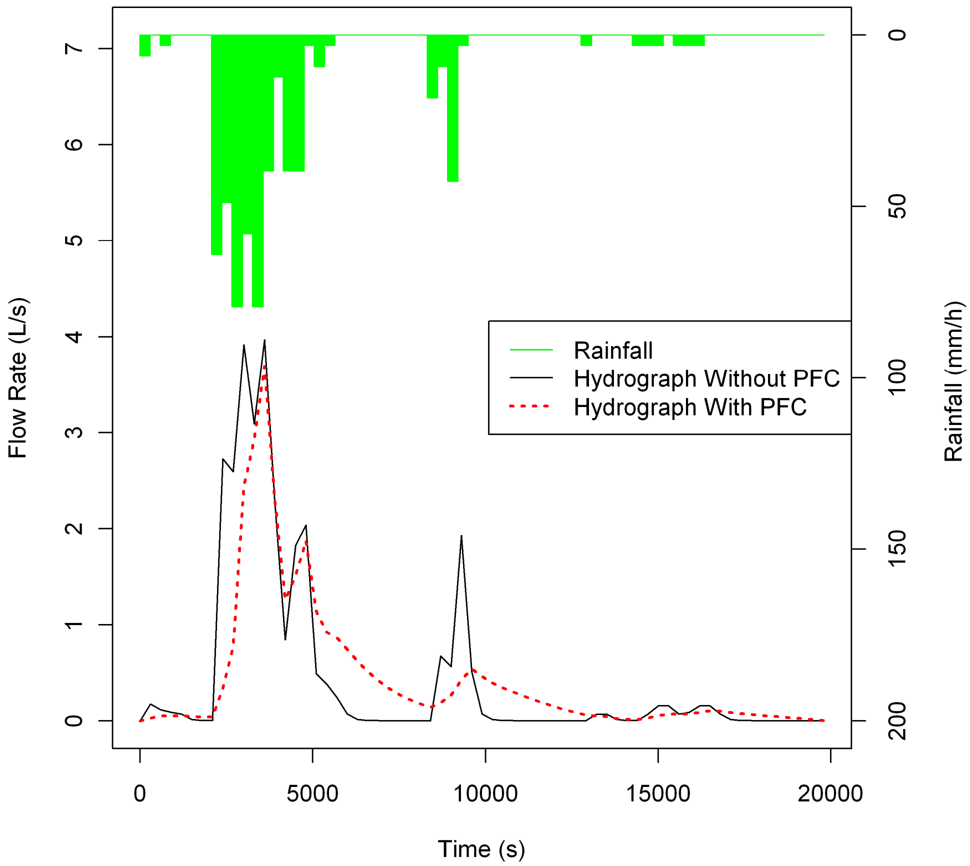

Figure 46: Comparison of modeled hydrographs with and without a PFC layer for Loop 360 on July 20, 2007. Plotted flow rates are five minute averages.

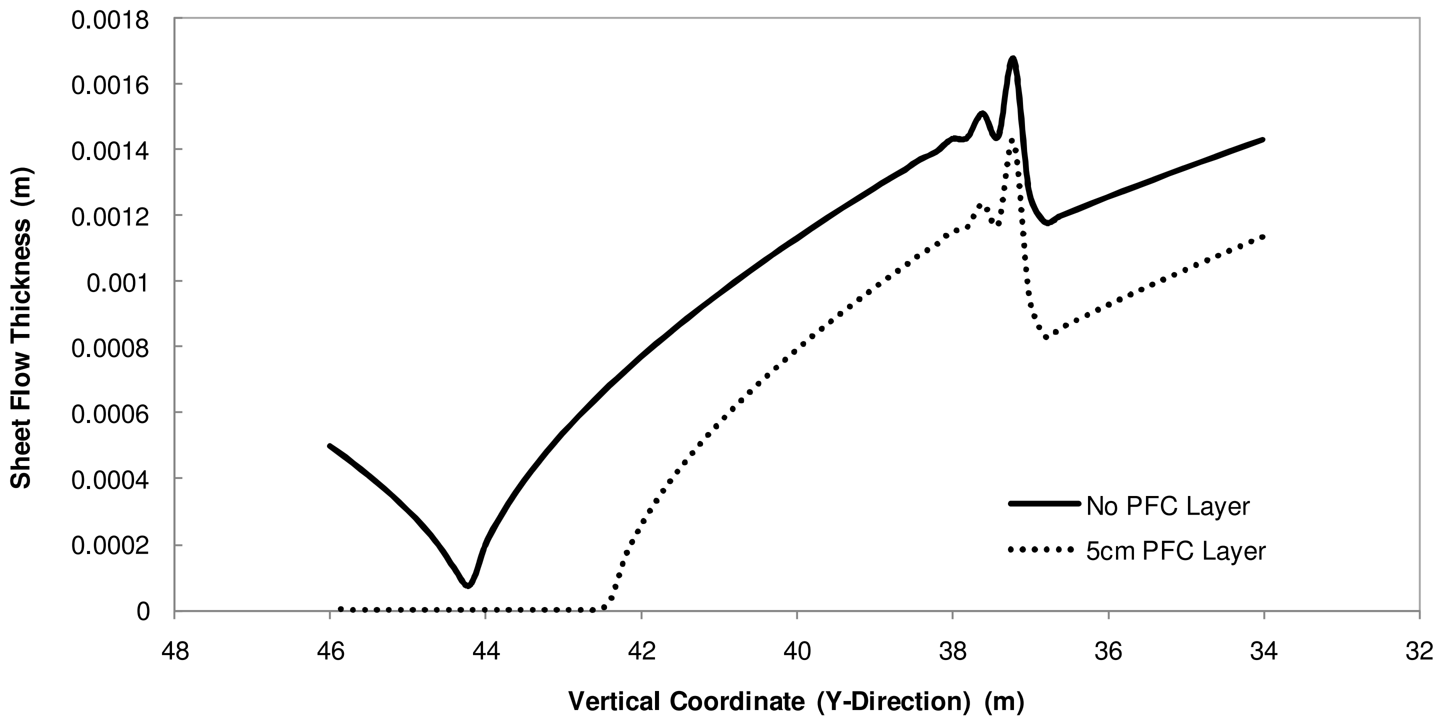

Figure 47: Comparison of sheet flow depths with and without a PFC layer horizontal coordinate of 94.42m at Loop 360 on July 20, 2007

5.5 Storm event of June 3, 2007

A comparison between model results and field measurements was made for a second storm event to confirm that the results obtained for July 20, 2007 were not coincidental. The event of June 3, 2007 was selected for analysis because the total rainfall depth was around 1-inch and because 90% of the rainfall was measured as runoff, a reasonable mass balance for field sampling. The measured rainfall data was prepared for simulation as outlined previously; all other simulation parameters remained the same.

The modeled hydrograph again shows reasonable agreement with the measured one (Figure 48). The model predicted a peak discharge of 2.6 L/s, which is 76% of the measured peak discharge of 3.4 L/s. Statistics of the residuals (the differences between modeled and measured values) are reported in Table 5. Compared to the July 20 event, the peak discharge was not modeled as well, but the statistics of the residuals were comparable between the events, suggesting that the model performed consistently in both cases.

A contour plot of the model domain during maximum depth conditions shows that sheet flow occurred over most of the roadway and that sheet flow depths were on the order of 1mm (Figure 49). The onset of sheet flow occurred 2.2m from the left hand shoulder and the maximum sheet flow depth of 1.3mm occurred near the right shoulder (Figure 50). These values compare favorably to the steady state model, which predicts sheet flow 3.4m from the left shoulder and a maximum sheet flow depth of 1.3mm.

Table 5: Summary of statistics of model residuals, all in units of L/s

Statistic	July 20, 2007	June 3, 2007
Mean	-0.029	0.016
Median	0.021	0.035
Standard Deviation	0.24	0.16
Standard Error of the Mean	0.029	0.02

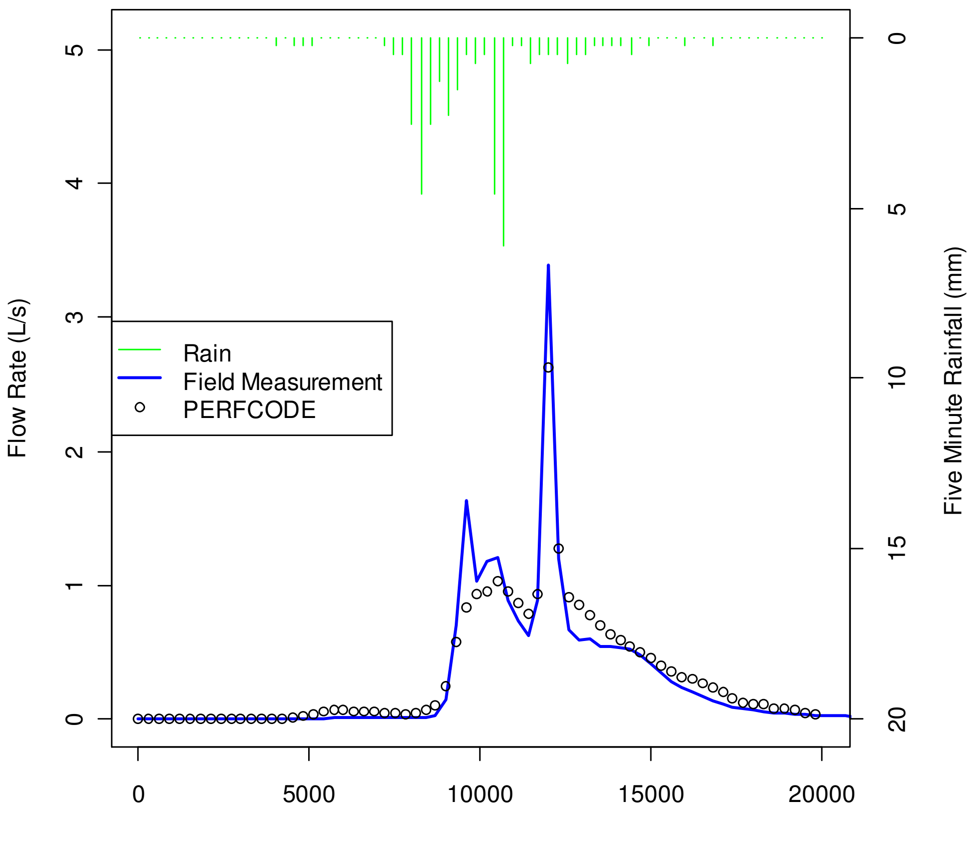

Figure 48: Comparison of modeled and measured hydrographs for June 3, 2007

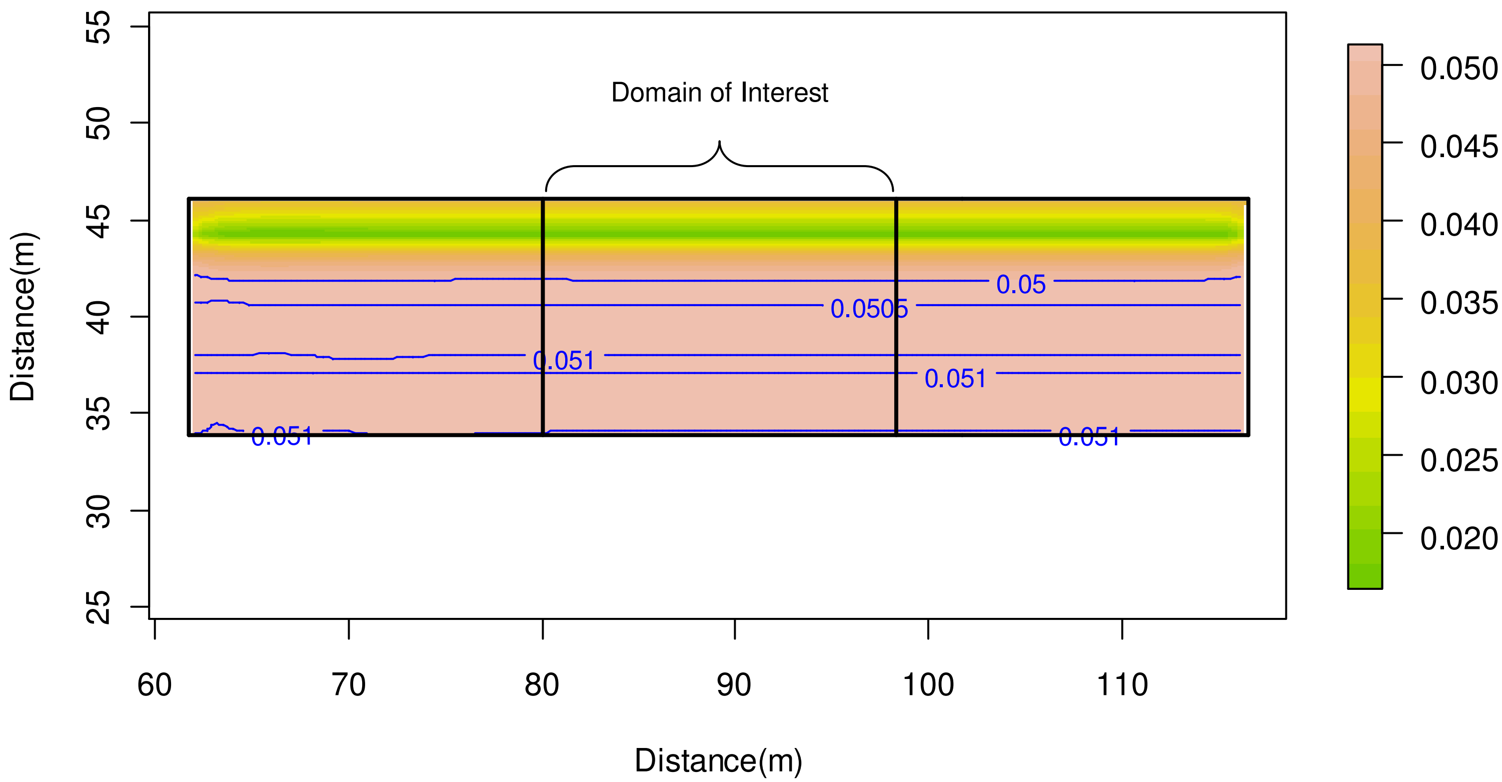

Figure 49: Water depth above impervious layer (m) for Loop 360 during maximum depth conditions on June 3, 2007. The PFC thickness was 0.05m; contours correspond to sheet flow conditions.

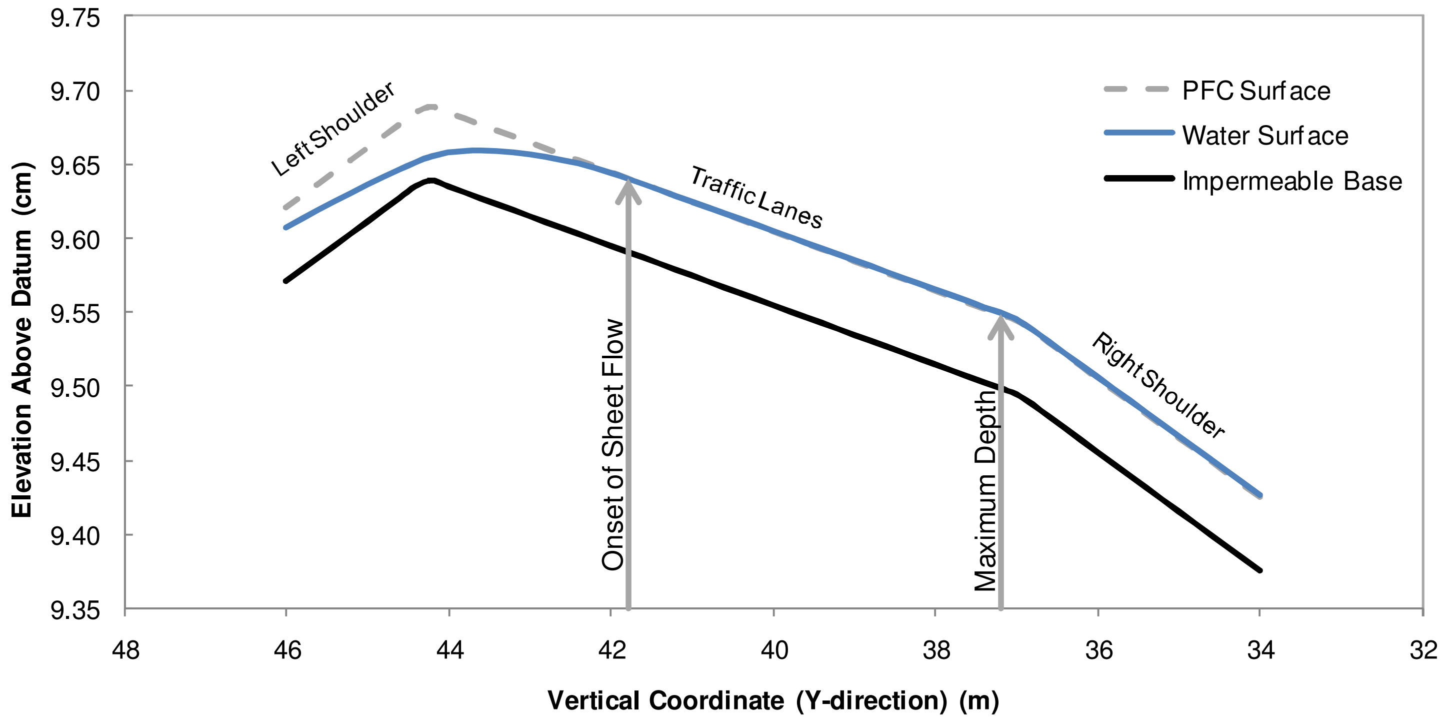

Figure 50: Profile through maximum depth section; the horizontal coordinate is 94.42m

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

6.1 Project Summary

This project has developed, validated, and applied a numerical model that couples the dynamics of overland flow with porous media flow for PFC roadways. The model represents overland flow using the 2-D diffusion wave approximation to the Saint-Venant equations. Porous media flow is described by the Boussinesq equation. Coupling these equations together facilitated water depth predictions at a fine spatial scale. This work has addressed the research objectives which were established in Chapter 1 and are repeated here for reference:

1. Identify governing equations for surface and subsurface flow for the geometry of interest
2. Develop a scheme to couple flow between the surface and subsurface
3. Implement the coupling scheme and numerical methods in a computer model that represents roadway geometry using a coordinate transformation
4. Validate the model using analytical solutions
5. Compare model predictions of runoff with values measured at an existing monitoring site

The governing equations for surface and subsurface flow have been identified and applied to roadway geometry. A scheme to couple the surface and subsurface flow components has been developed. The proposed scheme uses a mass balance approach and adjusts conveyance coefficients based on the flow conditions. A computer model has been developed and validated against steady state solutions that were obtained independently. Predictions of the runoff hydrograph were compared to measured values for the field monitoring site.

Several aspects of this work represent new and unique contributions to the fields of hydraulics and porous media flow:

- The model itself—PERFCODE—is a unique tool for understanding highway drainage. It builds on a long tradition of research in highway drainage hydraulics at The University of Texas at Austin.
- The way in which PFC flow and sheet flow are coupled within the model led to a better understanding of the interaction between PFC flow and sheet flow (see Eck et al. 2010).
- The ODE for PFC flow on a converging section has been derived and a numerical solution provided. The solution is useful for understanding how roadway geometry influences drainage behavior and for validating more comprehensive numerical treatments.
- A new boundary condition—the kinematic condition—for PFC flow has been developed and found to have reasonable agreement with field measurements.

6.2 Conclusions

Developing the simulation model and applying it to linear sections, converging sections, and the field monitoring site provided insight into the drainage behavior of PFC highways. Conclusions from this work are as follows:

- The kinematic boundary condition developed for PFC flow addresses an important gap in the literature of porous pavement hydraulics: the depth at the boundary can now be estimated for steady state or transient conditions. At the edge of pavement this condition gives a maximum depth in the PFC layer; but at the ends of the domain depth estimates are inconsistent with the domain interior, resulting in a boundary effect. The model domain should therefore be expanded to remove this effect from the area of interest. Use of this boundary condition yielded hydrographs that were consistent with field measurements.
- Predictions of runoff hydrographs for PFC roadways are available for the first time. These hydrographs show that PFC delays the initial discharge from the roadway compared to conventional pavement and that flow in a PFC layer requires a long time to reach steady state. For a constant rainfall case, PFC

delayed the initial discharge by 60 seconds and required 50 times more rainfall to reach steady state, though these values depend on problem parameters.

- One dimensional steady state equations remain a powerful tool for engineering design. For the storm investigated in Chapter 5, the 1D steady state equations predicted the location that sheet flow begins within 20cm of the PERFCODE's prediction. The location and magnitude of the maximum sheet flow depth were also closely predicted by the 1D steady state equations. This result confirms that the steady state equations (Charbeneau and Barrett, 2008 and Eck et al., 2010) are suitable for designing the PFC thickness on straight roads.
- The presence of a PFC layer did not affect the timing or magnitude of the peak discharge for the storm that was analyzed, but a later and smaller peak in the runoff hydrograph was delayed and reduced by the PFC layer. This result suggests that PFC has a negligible effect on the hydrology of large events, but can reduce the peak discharge of smaller events.
- During intense storms a PFC layer cannot prevent sheet flow altogether, but it can reduce the time during which sheet flow conditions persist. In the example studied, PFC reduced the duration of sheet flow conditions by about 80% and reduced the maximum sheet flow depth by 25%.

6.3 Recommendations for Future Work

Based on the research reported in this dissertation, several areas that should be considered for future research are as follows:

- The model required very small time-steps to simulate the measured rainfall. An infinite number of rainfall patterns are consistent with the five-minute rainfall data that was measured. Future work could include using a smoother rainfall function to see if the model's stability properties could be improved (e.g. take larger time-steps).
- Measured values of the hydraulic conductivity for PFC are at the high end of the acceptable range for Darcy's law on typical roadway slopes. Related

experimental and modeling efforts conducted by Klenzendorf (2010) used the Forchheimer equation to model flow through PFC and found the Forchheimer coefficients. Future work could update the model developed here to use Forchheimer's equation in place of Darcy's law. Such an update need only modify the subroutine for computing conveyance coefficients. Since the Darcy's law problem is already non-linear, the non-linearity introduced from Forchheimer's equation would be handled within the existing non-linear iteration loop.

- Small time steps (0.1s) were needed for non-smooth rainfall functions and high rainfall intensities. This small time step dramatically increased the time required for a model run. It also is based on the lowest common denominator—it is likely that larger time steps would be stable for part of the simulation time. An adaptive time stepping scheme could improve the run time while maintaining stability.
- The statistics of the residuals (modeled minus measured discharges) were similar for the two storms investigated. Future work should simulate additional storm events to further quantify the uncertainty in the model predictions.
- The model formulation is intended to allow simulations of more complex roadway geometry such as a superelevation transition or sag vertical curves. Although it is believed that major changes would not be required to deal with such geometries, they have not been attempted.

APPENDIX A: SUMMARY OF FORTRAN SOURCE CODE

The model described in this dissertation—PERFCODE—was implemented for computation in the Fortran 90/95 language and compiled for Microsoft Windows with the Lahey/Fujitsu Fortran compiler v5.5. The program runs as a console mode application (i.e. from the command prompt). This appendix describes (1) how to use model and (2) model limitations through a discussion of the model input files. A summary of the Fortran source code is given next, followed by a listing of the source code. The interested reader is encouraged to contact the author of this dissertation for an electronic copy of the model.

A.1 Model Limitations

PERFCODE has been designed to simulate highway drainage for a wide variety of conditions within certain limitations:

- The structure of the input files does not allow for a cross section that varies longitudinally (e.g. superelvation transition)
- Boundary conditions have not been developed for PFC on curbed sections

A.2 Running PERFCODE: Developing Input Files

PERFCODE is designed to simulate roadway drainage under a variety of conditions. Inputs to the model have been arranged into text files so that parameters can be changed without recompilation. In order apply the model to a situation of interest, input files must be developed. Model inputs and calculations use SI units.

The first input file contains basic simulation parameters and requires the most explanation. These parameters are read from Data File 1: Parameters.dat. As shown below, this file has several sections.

- *PFC properties* are listed first and these four properties are the only parameters of the mathematical model—these values must be accurate in order for simulation results to be consistent with physical observations. For the work of this dissertation, the hydraulic conductivity, porosity and pavement thickness were measured from core samples and the Manning's n value was inferred from an experimental study.

- The model uses different *time steps* for sheet flow and PFC flow conditions. The time of a model run must also be specified and care should be taken to select a simulation time that is consistent with the rainfall input.
- The *grid spacing* is controlled by selecting an approximate grid cell size. The size is approximate because the grid is creating using 'equal increments' see (Jeong et al. 2010). The size is also approximate because the user may specify a value that is not an exact divisor of the domain size (e.g. dy = 0.4m when the domain width is 5m). The quantities dx and dy should probably be called dxi and deta because the correspond to the cell size in the longitudinal and transverse directions (respectively).
- Several *tolerances* are needed including the maximum number of iterations, the required accuracy (eps is short for epsilon), and the relaxation factors for the non-linear iteration.
- The *initial condition* is simply the depth at the beginning of a simulation. A small value is used instead of zero because zero is a difficult number in floating point calculations.
- The *boundary condition* for each edge of the domain must also be specified
    - NO_FLOW is simply a no flow boundary
    - MOC_KIN means to use the method of characteristics to implement a kinematic boundary condition for PFC flow and sheet flow.
    - eastKIN means to use the MOC_KIN boundary from the east edge of the domain on the west end of the domain. This only makes

sense if the solutions on the east and west faces should be the same.

- 1D_FLOW means to use the one dimensional unsteady model as the boundary condition for the two dimensional domain. This boundary condition is experimental and not recommended for use.

Data File 1: Parameters.dat

```
Parameter Input file for PERFCODE
--------------------------------
PFC Properties
       0.01     <----- Hydraulic Conductivity [m/s]
       0.2      <----- Porosity
       0.05     <----- Pavement Thickness [m]
       0.015    <----- Manning's n  [ sec / m ^ (1/3) ]
Physical Constants
       9.81    <----- Gravitational Acceleration [m/s/s]
Time Steps
       5.      <----- time step for PFC flow [s]
       1.      <----- time step for sheet flow [s]
       8000    <----- Time to simulate [s]
Grid Spacing
       0.10    <----- preliminary value of dx [m]
       0.10    <----- preliminary value of dy [m]
Tolerances
       200     <----- qmax (maximum number of non-linear iterations)
       5000    <----- maxit ( maximum number of solver iterations)
       1.e-4   <----- eps_matrix
       1.e-3   <----- eps_itr
       1.e-3   <----- eps_ss
       1.      <----- Relaxation Factor for non-linear iteration
       0.2     <----- Relaxation factor for transition
Initial Condition
       1.e-10   <----- Initial depth [m]
Boundary Conditions ( legal values are: MOC_KIN, NO_FLOW, 1D_FLOW, eastKIN )
   NO_FLOW   <----- NORTH boundary of domain
   MOC_KIN   <----- SOUTH boundary of domain
   NO_FLOW   <----- EAST boundary of domain
   NO_FLOW   <----- WEST boundary of domain
```

Rainfall information is read from Data File 2: Rainfall.dat. The first line of the file is the number of rainfall records, which the program needs in order to read in the proper number of values. Note that the times move in 300s increments, consistent with the field monitoring data. The remaining lines of the file are not shown for brevity. A technical computing platform—such as the R Environment for Statistical Computing and Graphics or MATLAB—is useful for generating this file from a record of measured rainfall. In order to simulate a constant rainfall rate, only two records are required: time zero and some large time both with the same rainfall rate.

Data File 2: Rainfall.dat

```
208   <------ Number of rainfall records  20 July 2007
1,0,1.693333e-06  <--- Record, Time[s], Rainfall Rate [m/s]
2,299.99,1.693333e-06
3,300,0.000000e+00
4,599.99,0.000000e+00
5,600,8.466667e-07
6,899.99,8.466667e-07
7,900,0.000000e+00
8,1199.99,0.000000e+00
9,1200,0.000000e+00
.....
.....
```

Information about the horizontal alignment of the roadway is read from Data File 3: CL_Segments.dat. The information in this file pertains to the geometry of the roadway centerline. The variables correspond to Equation (3.22). This information can be specified directly as was done in this dissertation, or obtained by processing an output file from roadway design software such as GEOPACK as done by Jeong (2008).

Data File 3: CL_Segments.dat

```
1      <----- Number of Segments
Segment, xcc1, ycc1, dx, dy, R1, dR, W, theta1, dtheta,
1, 89.14400, -1000000, 0, 0, 1000040., 0.0, 12.192, 1.57080547, -1.82873E-05,
```

The vertical alignment of the roadway is specified by two different files. Cross section information is read from Data File 4: CrossSection.dat. Note that this file specifies relative elevations in the form of slopes, but not absolute elevations. The sum of the segment widths specified here should match the overall roadway width (W) that is given in Data File 3: CL_Segments.dat.

Data File 4: CrossSection.dat

```
Roadway Cross Section Input file for PERFCODE
       3         <------ Number of segments to define cross section
Segment        Slope  Width[m]
    1,     -0.04,   3.0480
    2,     -0.02,   7.3152
    3,      0.04,   1.8288

Note:  SLOPE is defined left to right with a negative slope
       corresponding to a loss of elevation moving from left to right.
       SEGMENTS are numbered from eta = 0 to eta = 1 so segment 1 is
       on the right end of the domain.
```

Elevations are obtained from Data File 5: LongProfile.dat. The elevations in this file correspond to the right edge of the pavement ($\eta = 0$). The structure of this file allows for more variations in longitudinal slope than were considered in this dissertation. By including more points in this file, different longitudinal geometries such as sag vertical curves can be represented.

Data File 5: LongProfile.dat

```
Longitudinal Profile Input file for PERFCODE
       2             <------ Number of points to define longitudinal profile
 Point No. Distance(m)  and Elevation(m)  ALONG ETA == 0
       1,     0.000000,        10.0000000 <--- West boundary of domain
       2,     18.28800,            9.579376   <--- East boundary of domain  2.3%
```

Once these data files have been formulated for the problem of interest, model runs can begin. Several output files are written during each model run and the content of these files is the subject of the next section.

A.3 PERFCODE Output Files

Output files are mostly formatted as .csv (comma separated values) so that results can be opened by a spreadsheet program or read into a technical computing environment. The primary output files are:

- details.csv contains summary information for each time step including the outflow hydrograph and other time history data.

- max_depth.csv contains the model solution for maximum depth conditions encountered during the simulation. The file is in vector form.
- params.csv is an echo of the model parameters used in the simulation
- PERFCODE_Run.txt is a log file with information about each iteration and each time step of the model run. Most warning messages during the simulation are directed to this file. If the simulation failed for some reason, this file is the first place to look for an explanation.

A.4 Fortran Source Code

In writing the code for the model, extensive use was made of Fortran *modules* for storing common variables and grouping procedures (functions and subroutines) thematically. Each module comprises its own source file, but may contain several procedures provided the procedures do not reference each other. Each module is compiled separately. When the main program is compiled, links to the requisite modules are made and the product is a single executable file. Table 6: shows the name and contents of each programming unit. The order of the source files in the table (after the main program) reflects the order in which the files must be compiled for proper linking. This table also serves as an index to the code listing. The interaction between the procedures is depicted graphically in Figure 51 on page 121.

Table 6: Fortran program and module listing

Program or Module Name	Source file and Page No.	Contents
PERFCODE	PERFCODE.f95 122	Main program (compiled last)
shared	shared.f95 152	Variables shared between different programming units

Program or Module Name	Source file and Page No.	Contents
pfc2Dfuns	pfc2Dfuns.f95 157	Function subprograms used in the 2D PFC drainage model: **F_LinearIndex** computes the linear index for each grid cell **F_por** computes the porosity factor (*pf*) for each grid cell **F_RHS_n** computes the contribution to the right hand side of the linear system due to time level n **F_RHS_n1** computes the contribution to the right hand side of the linear system due to time level n+1
utilities	utilities.f95 159	Functions and subroutines for general use **UNLINEARIZE** converts the solution from the linear form used in the matrix system into a two-dimensional array **BILINEAR_INTERP** performs bi-linear interpolation **F_LINTERP** Performs linear interpolation **F_L2_NORM** Computes the L2 norm of a vector **F_PYTHAGSUM** Computes the Pythagorean sum of two numbers **F_EXTRAPOLATE** Performs linear extrapolation
inputs	inputs.f95 169	Subroutines for reading the simulation parameters and rainfall information **GET_PARAMETERS** and **GET_RAINFALL**

Program or Module Name	Source file and Page No.	Contents
outputs	outputs.f95 172	Subroutines for generating selected outputs **ECHO_INPUTS** prints selected input parameters to the screen **WRITE_FLIPPED_MATRIX** creates comma seperated values (.csv) file of a matrix that has been 'flipped' to match the model domain (e.g. the 1,1 location is in the southwest corner) **WRITE_MATRIX** creates a .csv file of a matrix **WRITE_VECTOR** creates a .csv file of a vector **WRITE_SYSTEM** creates a .csv file of the bands and right hand side of the penta-diagonal matrix system
geom_funcs	geom_funcs.f95 177	Function sub-programs related to the curvilinear grid generation **F_L_xi** computes the metric coefficient for the length mapping **UNMAP_X** computes the x coordinate of a point in physical space from its coordinates in computational space **UNMAP_Y** computes the y coordinate of a point in physical space from its coordaintes in computational space

Program or Module Name	Source file and Page No.	Contents
ConvCoef	ConvCoef.f95 180	Subroutines related to computing the conveyance coefficients: **CONVEYANCE** computes the conveyance coefficient for a cell face **FrictionSlope** computes the friction slope at the center of each grid cell face
GridGen	GridGen.f95 188	Subroutines related to the grid generation scheme **GENERATE_GRID** reads the centerline geometry file and creates a curvilinear grid (horizontal coordinates) based on a given approximate grid spacing **SET_ELEVATIONS** reads the longitudinal profile from a file and assigns an elevation to each grid cell
Solvers	Solvers.f95 199	Subroutines related to solving linear systems: **DIAGDOM_PENTA** checks for diagonal dominance given the bands of a penta-diagonal matrix **GAUSS_SEIDEL_PENTA** uses the Gauss-Seidel method for iterative solution of a penta-diagonal system of linear equations. **THOMAS** uses the tri-diagonal matrix algorithm to solve a tri-diagonal linear system
pfc1Dfuns	pfc1Dfuns.f95 204	Functions used the 1D pfc flow model: **F_CC** computes the conveyance coefficient **F_por** computes the porosity function for a grid cell

Program or Module Name	Source file and Page No.	Contents
pfc1Dfuns2	pfc1Dfuns2.f95 205	Lower level functions used in the 1D pfc flow model: **F_hp_face** computes the saturated thickness at the cell face **F_hs_face** computes the sheet flow thickness at the cell face
pfc1Dsubs	pfc1Dsubs.f95 207	Subroutines used for the 1D flow model: **GRID_1D_SECTION** creates a grid for the 1D drainage path **pfc1Dimp** solves the 1D pfc drainge problem using the crank-nicolson implicit method. The routine only takes a single time-step.
pfc2Dsubs	pfc2Dsubs.f95 223	Subroutines related to the 2D pfc flow model: **SET_ABCDEF** fills the coefficients of the linear system for a single grid cell **SET_XYH** assigns values of x,y,and h for use in the bi-linear interpolation routine
BoundCond	BoundCond.f95 225	The subroutine **MOC_KIN**, which uses the method of characteristics to implement a kinematic boundary condition.

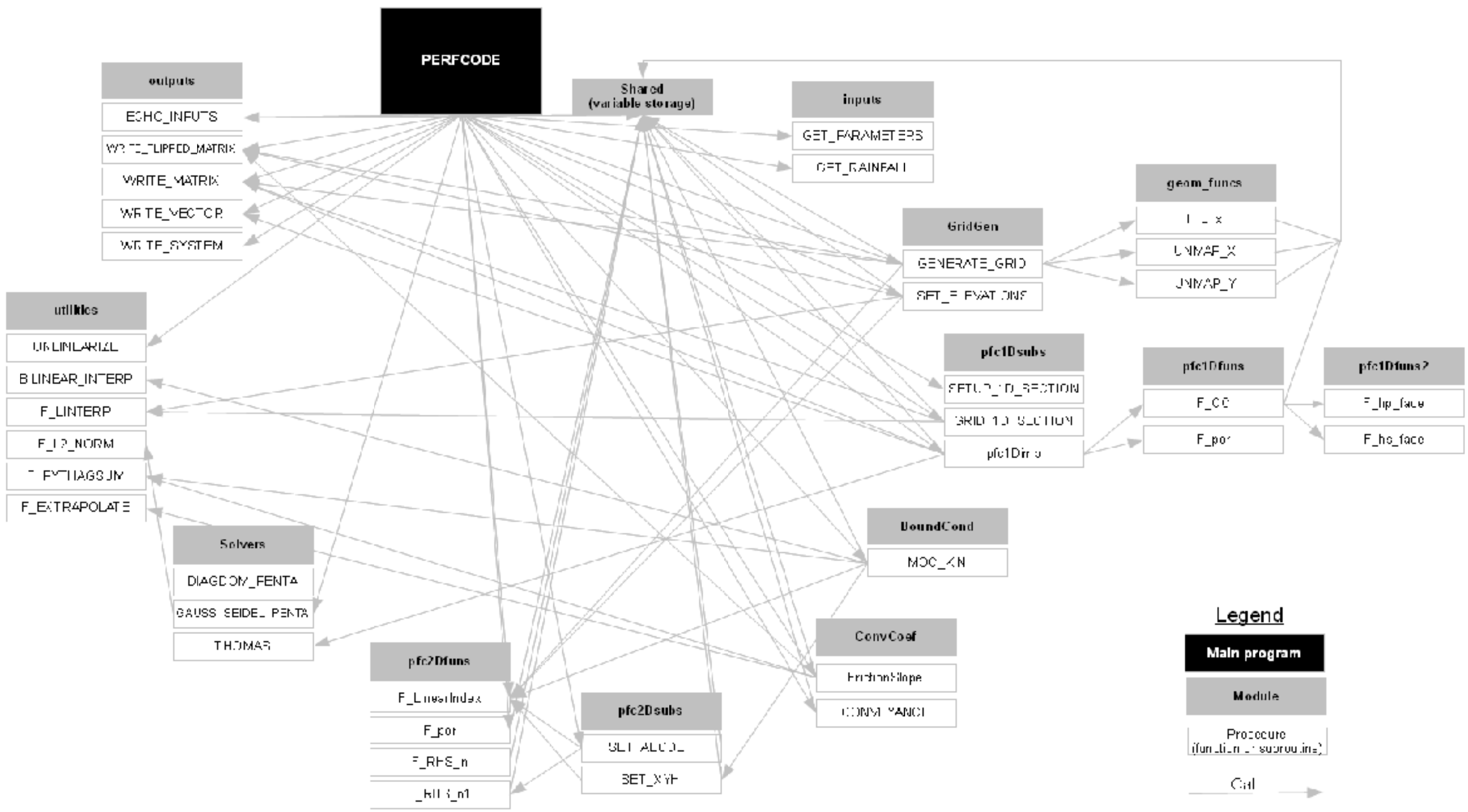

Figure 51: Calling tree for PERFCODE

Source File 1: PERFCODE.f95

```
! fortran_free_source
!
!
!   PPPP     EEEEEE   RRRR      FFFFFF      CCCC        OOOO     DDDDD     EEEEEE
!   P   P    E        R    R    F          C    C      O    O    D    D    E
!   P    P   E        R     R   F         C            O      O   D     D   E
!   P    P   E        R     R   F        C            O        O  D      D  E
!   P   P    E        R    R    F        C            O        O  D      D  E
!   PPPP     EEEEEE   RRRR      FFFFFF   C            O        O  D      D  EEEEEE
!   P        E        R    R    F        C            O        O  D      D  E
!   P        E        R     R   F        C            O        O  D      D  E
!   P        E        R      R  F         C            O      O   D     D   E
!   P        E        R      R  F          C    C      O    O    D    D    E
!   P        EEEEEE   R      R  F           CCCC        OOOO     DDDDD     EEEEEE
!
!
!   P E R m e a b l e   F r i c t i o n   C O u r s e   D r a i n g e   c o d E
!
!       Written By:     Brad Eck
!
!             Date:     April 2010
!
!=================================================================================
!       \\\\\\\\\\     P R O G R A M                            //////////
!       //////////                     D E S C R I P T I O N    \\\\\\\\\\
!=================================================================================
!
!
!       Purpose:        This program computes a 2D solution for unsteady
!                       drainage through a PFC.  The water THICKNESS in each
!                       cell is used as the primary variable.
!       IC:             Specified in input file
!       BCs:            Specified in input file
!       Linearization:  Picard Iteration (lag the coefficients)
!       Linear Solver:  Gauss-Seidel iteration
!
! Alphabetical list of variables used in the main program PERFCODE
!       (variables used in subroutines are described there)
!
!   A         -- lowest band of penta diagonal matrix
!   area      -- area of a grid cell
!   astat     -- array allocation statuses
!   B         -- subdiadonal band of penta diagonal matrix
!   b_pfc     -- thickness of the PFC layer
!   C         -- main diagonal of penta diagonal matrix
!   Ce        -- conveyance coefficient ( conv coef ) for the
!                EASTtern cell face at time level n
!   Ce1       -- conv coef for EASTern cell face at time level n + 1
!   Cn        -- conv coef for the NORTHern cell face at time level n
!   Cn1       --      ''              ''            ''  at time level n + 1
!   Cs        -- conv coef for the SOUTHern cell face at time level n
```

```
!   Cs1        --      ''            ''          ''  at time level n + 1
!   CV_Info    -- information about each grid cell (aka Control Volume)
!   Cw         -- conv coef for the WESTern  cell face at time level n
!   Cw1        --      ''            ''          ''  at time level n + 1
!   D          -- superdiagonal band of penta diagonal matrix
!   dist_lp    -- distance along longitudinal profile
!   diagdom    -- logical flag for test of diagonal dominance
!   ds         -- distance up characteristic in sheet flow moc bc
!   dt         -- time step for the simulation
!   dt_pfc     -- time step for PFC flow
!   dt_sheet   -- time step for sheet flow
!   dx         -- prelim. grid size for longitudinal direction
!   dx_moc     -- distance up drainage path in pfc moc bc
!   dy         -- prelim. grid size for transverse direction
!   E          -- uppermost band of penta diagonal matrix
!   east_bc    -- condition for east boundary
!   eps_matrix-- tolerance (epsilon) for matrix solver
!   eps_itr    -- tolerance for an iteration
!   eps_itr_tol-- selected tolerance for the iteration (based on transition)
!   eps_ss     -- tolerance for steady state (not used)
!   eta_cs     -- values of eta along the cross slope
!   eta_0_hp2_max-- max possible value for pfc moc bc
!   eta_cs_1D  -- values of eta for 1D model
!   eta1D      -- value of eta at each point in 1D domain
!   etaCV      -- value of eta at CV center for 1D grid
!   F_         -- the letter F with an underscore ( F_ ) denotes a
!                     function call and NOT an array
!   F          -- right hand side of linear system in pentadiagonal matrix
!   F1         -- contribution to F from time level n+1
!   Fn         -- contribution to F from time level n
!   g          -- constant of gravitational acceleration
!   grid       -- number of each grid cell
!   h0         -- initial depth (m)
!   h_bound    -- depth at boundary (returned by MOC_KIN or 1D_FLOW)
!   h_imid_j1_max-- solution when depth at middle of south boundary is max
!   h_imid_j1_max_hist
!   h_imid_max-- solution when depth in middle of domain is max
!   h_imid_max_hist
!   h_itr      -- matrix form of solution at level n+1
!   h_itr_vec  -- vector form of solution at time level n+1
!   h_max      -- solution at maximum depth
!   h_new_1d   -- solution at time level n+1 for 1D problem
!   h_old      -- solution at time level n
!   h_old_1d   -- solution at time level n for 1D problem
!   h_old_vec  --
!   h_pfc_min  -- minimum value for pfc flow thickness
!   h_Q_max    -- solution at maximum flow
!   h_temp_hist -- history of solution during an iteration
!   h_tmp_vec  --
!   hp1        -- depth at point 1 in pfc MOC bc
!   hp2        -- depth at point 2 in pfc MOC bc
!   hs1        -- sheet flow depth at point 1 in sheet flow moc bc
!   hs2        -- sheet flow depth at point 2 in sheet flow moc bc
!   i          -- array index ( longitudinally in the domain )
```

```
!   input_values-- array of values of the input variables
!   input_variables -- character array of input variables
!   imax      -- maximum value of the array index i
!   j         -- array index ( transverse in the domain )
!   jmax      -- maximum value of the array index j
!   K         -- the saturated hydraulic conductivity of the PFC
!   L2_history -- value of the L2 norm for each timestep
!   lng       -- curvilinear length of a grid cell at its center
!   lng_north -- curvilinear length of the northern face
!   lng_south -- curvilinear length of the southern face
!   loc       -- the location of the largest relative change in a time step
!   long_slope -- overall longitudinal slope
!   max_rec   -- maximum number of records (for pre-allocating arrays
!                where values are read in from a file )
!   max_time  -- longest time to simulate
!   maxdiff   -- the change in head at location LOC for timestep n
!   maxit     -- maximum number of matrix iterations
!   maxrelchng_ss-- maximum relative change for a timestep, for stdy state check
!   maxthk    -- maximum thickness fot the timestep
!   matrix_numits-- number of iterations to solve the matrix
!   n         -- index for time stepping
!   n_mann    -- Manning's roughness coefficient
!   north_bc  -- condition for north boundary
!   nlast     -- last timestep taken
!   nmax      -- maximum number of time steps in the simulation
!   numit     -- the number of iterations required for a timestep to converge
!   nr_cs     -- number of records in the cross slope file
!   nr_lp     -- number of records in the longitudinal profile file
!   nrr       -- number of rainfall records
!   out_time  --
!   pf        -- porosity factor ( includes effect of porosity
!                                   when pavement is not saturated )
!   pf_int    -- porosity factor as an integer
!   pf1       -- porosity factor for time level n+1
!   pf1_int   --  "      "                       ""   as integer
!   por       -- the effective porosity of the PFC
!   q         -- iteration index
!   qmax      -- maximum number of iterations
!   rain      -- rainfall rate for each timestep of the simulation
!   Qout      -- flow rate out the southern boundary for a timestep
!   rain_rate -- rainfall rate for each time increment in the
!                   rainfall input file
!   rain_time -- time column of rainfall input file
!   relax     -- relaxation factor for non-transition iterations
!   relaxation_factor -- underrelaxation factor for non-linear iteration
!   relax_tran -- relaxation factor for transition
!   relchng   -- the relative change between solns for an iteration or timestep
!   residual  -- difference between old and itr solutions
!   seg       -- properties of a centerline segment
!   Sfe_itr   -- friction slope at center of east  face at time level n+1
!   Sfe_old   -- friction slope at center of east  face at time level n
!   Sfn_itr   -- friction slope at center of north face at time level n+1
!   Sfn_old   -- friction slope at center of north face at time level n
!   Sfs_itr   -- friction slope at center of south face at time level n+1
```

```
!   Sfs_old   -- friction slope at center of south face at time level n
!   Sfw_itr   -- friction slope at center of west  face at time level n+1
!   Sfw_old   -- friction slope at center of west  face at time level n
!   slope_cs  -- slope column of cross section file
!   slope_cs_1d -- slope of 1D segment
!   sim_tim   -- character variable for time simulated
!   solver_numits-- number of iterations for the solver
!   south_bc  -- condition for south boundary
!   time      -- time at each timestep
!   time_simulated-- the time simulated
!   timestep_solver_numits --
!   transition -- logical to see if we're in a transition timestep
!   tolit     -- tolerence for iterations, used for relative (fractional) changes
!   TNE       -- total number of elements for 1D grid
!   v         -- linear index for domain
!   ve        -- linear index for cell to the east
!   v_in      -- linear index of adjacent inside cell
!   vmax      -- number of unknowns in the domain
!   west_bc   -- condition for west boundary
!   wid       -- curvilinear width of a grid cell at its center
!   wid_cs    -- width column of cross slope file
!   wid_cs_1d -- width of 1D segment
!   XCV       -- coordinate of CV center for 1D grid
!   Z         -- elevation at the cell center
!   Z_cs      -- elevation along the cross slope
!   Z_lp      -- elevation along longitudinal profile
!   ZCV       -- elevation of CV center for 1D grid

!==============================================================================
!      \\\\\\\\\\      B E G I N    P R O G R A M        //////////
!      //////////          P E R F C O D E               \\\\\\\\\\
!==============================================================================
program PERFCODE

!------------------------------------------------------------------------
!   >>>>>>>>>>          M O D U L E S                  <<<<<<<<<<
!------------------------------------------------------------------------
! Refer to the modules that are referred to by this code

USE SHARED      ! SHARED is used to store VARIABLES
USE INPUTS      ! INPUTS has subroutines
USE OUTPUTS     ! OUTPUTS has subroutines
USE ConvCoef    ! computes conveyance coefficinnts
USE SOLVERS     ! linear solvers
USE Utilities
USE gridgen
use pfc1Dsubs
use pfc2Dsubs
use pfc2Dfuns
use BoundCond


```

```
!-----------------------------------------------------------------------
!   >>>>>>>>>>          V A R I A B L E S                 <<<<<<<<<<
!-----------------------------------------------------------------------

implicit none

!  All variables are declared in module SHARED

!----------------------------------------------------------------------------
!   >>>>>>>>>>      P R O B L E M   S E T U P          <<<<<<<<<<
!----------------------------------------------------------------------------

!Create a file to store details of the run
open( unit = 100, file = 'PERFCODE_Run.txt', status = 'REPLACE' )

!----------------------------------
! Problem parameters file
!-----------------------------------
CALL GET_PARAMETERS( K, por, b_pfc, n_mann, g, dt_pfc, dt_sheet, max_time,  &
                     dx, dy, qmax, maxit, h0, eps_matrix, eps_itr, eps_ss,  &
                     relax, relax_tran,                                     &
                     north_bc, south_bc, east_bc, west_bc,                  &
                     animate, dt_ani                                          )


!----------------------------------------------------
! Rainfall file & maximum number of timesteps
!----------------------------------------------------

call GET_RAINFALL( max_rec, rain_time, rain_rate, nrr )

nmax = ( maxval( rain_time(1:nrr) ) / min( dt_pfc, dt_sheet ) ) !* 100


!--------------------------------------------------------
! GRID GENERATION
!-------------------------------------------------------
!   This subroutine takes the centerline geometry file that is generated
!   mannualy and creates a curvilinear grid.
!       INPUTS:  Preliminary grid spacing
!       OUTPUTS: Size of computational domain (imax & jmax )
!                Length, width and area of each grid cell ( module SHARED)
!                Coordinates of each CV center
call GENERATE_GRID( prelim_dx = dx , prelim_dy = dy )


! Reads in cross section and longitudinal files and computes elevations
!  of CV Centers

CALL SET_ELEVATIONS()
```

```
! Creates a grid for a 1D section in case a 1D boundary condition is used
call setup_1d_section()
CALL grid_1d_section( slope_in = slope_cs_1D  ,  &
                      width_in =   wid_cs_1D  ,  &
                           seg =       nr_cs  ,  &
                            dx = (( dx+dy ) / 2.)   )

!------------------------------------------------------------
! inputs summary
!------------------------------------------------------------
! make a list of input variables and values
input_variables = (/ 'K          ', &
                     'por        ', &
                     'b_pfc      ', &
                     'n_mann     ', &
                     'g          ', &
                     'dt_pfc     ', &
                     'dt_sheet   ', &
                     'max_time   ', &
                     'dx         ', &
                     'dy         ', &
                     'qmax       ', &
                     'maxit      ', &
                     'h0         ', &
                     'eps_matrix', &
                     'eps_itr    ', &
                     'eps_ss     ', &
                     'relax      ', &
                     'relax_tran' /)

!also collect and store values of input variales
input_values =  (/ K, por, b_pfc, n_mann, g, dt_pfc, dt_sheet, &
                  max_time, dx, dy, real(qmax), real(maxit),    &
                  h0, eps_matrix, eps_itr, eps_ss, relax, relax_tran /)


! Echo inputs to the screen, unit 6 by default
CALL ECHO_INPUTS( dev = 6 )
!also echo to log file
CALL ECHO_INPUTS( dev = 100 )


!----------------------------------------
!  Animation setup
!-----------------------------------------

if( animate .eqv. .TRUE. ) then

    animax = int( floor(max_time / dt_ani) )
    allocate( h_vec_ani ( vmax, animax  ) )
    allocate(   ani_lab (       animax  ) )
    allocate(   ani_time(       animax  ) )

endif
```

```
!------------------------------------------------------------------------------
!   >>>>>>>>>>     A L L O C A T E   A R R A Y S       <<<<<<<<<<
!------------------------------------------------------------------------------

! inialize as we go.

! VARIABLES IN MODULE SHARED

allocate(     h_old( imax, jmax ),  STAT = astat( 7) );       h_old = 0.0
allocate(     h_itr( imax, jmax ),  STAT = astat( 8) );       h_itr = 0.0
allocate(   Sfw_old( imax, jmax ),  STAT = astat( 9) );     Sfw_old = 0.0
allocate(   Sfe_old( imax, jmax ),  STAT = astat(10) );     Sfe_old = 0.0
allocate(   Sfs_old( imax, jmax ),  STAT = astat(11) );     Sfs_old = 0.0
allocate(   Sfn_old( imax, jmax ),  STAT = astat(12) );     Sfn_old = 0.0
allocate(   Sfw_itr( imax, jmax ),  STAT = astat(13) );     Sfw_itr = 0.0
allocate(   Sfe_itr( imax, jmax ),  STAT = astat(14) );     Sfe_itr = 0.0
allocate(   Sfs_itr( imax, jmax ),  STAT = astat(15) );     Sfs_itr = 0.0
allocate(   Sfn_itr( imax, jmax ),  STAT = astat(16) );     Sfn_itr = 0.0
allocate(     h_max( imax, jmax )                    );       h_max = 0.0
allocate(   h_Q_max( imax, jmax )                    );     h_Q_max = 0.0
allocate( h_imid_j1_max ( imax, jmax )              );h_imid_j1_max = 0.0
allocate( h_imid_max_hist( nmax ) );                 h_imid_max_hist = 0.0
allocate( h_imid_max( imax, jmax ) ) ;                    h_imid_max = 0.0

allocate( h_old_1d( TNE ) )
allocate( h_new_1d( TNE ) )


! Check allocation statuses
do i = 1, 20
    if( astat( i ) .NE. 0 ) then
        WRITE(100,*) 'PERFCODE: allocation problem!! &
                    & check shared variable:', i
    end if
end do

if( maxval(astat) .eq. 0 ) then
    WRITE(100,*) 'PERFCODE: allocation of shared variables sucessful'
endif


! VARIABLES IN THIS PROGRAM

allocate(   A( vmax ), stat = astat2( 1) );     A = 0.0
allocate(   B( vmax ), stat = astat2( 2) );     B = 0.0
allocate(   C( vmax ), stat = astat2( 3) );     C = 0.0
allocate(   D( vmax ), stat = astat2( 4) );     D = 0.0
allocate(   E( vmax ), stat = astat2( 5) );     E = 0.0
allocate(  Fn( vmax ), stat = astat2( 6) );     Fn= 0.0
allocate(  F1( vmax ), stat = astat2( 7) );     F1= 0.0
allocate(   F( vmax ), stat = astat2( 8) );     F = 0.0
```

```fortran
allocate(   h_itr_vec( vmax ),  stat = astat2( 9) );    h_itr_vec = 0.0
allocate(   h_tmp_vec( vmax ),  stat = astat2(10) );    h_tmp_vec = 0.0
allocate(   h_old_vec( vmax ),  stat = astat2(11) );    h_old_vec = 0.0
allocate(   h_new_vec( vmax ),  stat = astat2(12) );    h_new_vec = 0.0
allocate(   relchng   ( vmax ),  stat = astat2(13) );    relchng   = 0.0

allocate(     numit( nmax ),    stat = astat2(14) );      numit   = 0
allocate(       loc( nmax ),    stat = astat2(15) );      loc     = 0
allocate(   maxdiff( nmax ),    stat = astat2(16) );      maxdiff = 0.0
allocate(      Qout( nmax )   ); Qout = 0.0
allocate(   matrix_numits(nmax) )
allocate(  L2_History( nmax ) ); L2_History = 0.0


! Set indices for rain so that n-1 always works.  This is b/c
!  in Crank-Nicolson half of the rainfall rate is from time level
!  n  and half is from time level n-1
allocate(  rain( 0 : nmax-1 ),    stat = astat2( 0) )
allocate(  time( nmax ),    stat = astat2(17) )


allocate(   grid( jmax, imax ),     stat = astat2(19) )

allocate( maxthk( nmax ), stat = astat2(20) );             maxthk = 0.0

allocate( residual( vmax ), stat = astat2(21) );         residual = 0.0



allocate( h_temp_hist( vmax, qmax) ); h_temp_hist = 0.0

allocate( h_imid_j1_hist( nmax ), stat = astat(22) )
! Check allocation statuses
do i = 1, 29
    if( astat2( i ) .NE. 0 ) then
        WRITE(100,*) 'PERFCODE: allocation problem in main &
                                & program!!, check variable:', i
    end if
end do

if( maxval(astat2) .eq. 0 ) then
    WRITE(100,*) 'PERFCODE: allocation of main program variables sucessful'
endif


!--------------------------------------------------------------------------------
!   >>>>>>>>>>      P R O B L E M    S O L V I N G       <<<<<<<<<<
!--------------------------------------------------------------------------------

!  INITIAL CONDITIONS
!  set all all arrays to the initial depth value
h_old = h0
h_itr = h0   ! added this after b/c the first iteration kept failing
h_old_vec = h0
```

```
h_itr_vec = h0

h_old_1D = h0   ! initial depth for 1D boundary condition
h_new_1D = h0


WRITE(*,*) 'PERFCODE: starting time stepping loop,&
            & max time = ', max_time, ' seconds'

CALL SYSTEM_CLOCK( RUN_START_TIME, count_rate, count_max)



!       !open a file to store each timestep
!           open( unit = 50, file = 'timesteps.csv', status = 'REPLACE' )
!           write(50,5) ' n / v,', (v, v=1, vmax) !implied DO loop
!       5   format( A, 10000( I, ','))
!


! Set rainfall rate for begining of simulation
n=0
rain(n) = F_Linterp( 0.0              , &
                     rain_time(1:nrr), &
                     rain_rate(1:nrr), &
                     nrr                 )

!----------------------------------------------
! BEGIN TIME STEPPING
!----------------------------------------------

time_stepping: do while (time_simulated .LT. max_time )

!increment n and store the largest n we've gotten so far
n = n + 1
nlast = n

! Select the time step
if( maxval( h_old ) .GT. b_pfc * 0.95 ) then
    dt = dt_sheet
else
    dt = dt_pfc
endif

!Computed the time simulated
!   Do the accumulation with an internal write/read to
!    avoid accumulating the floating point errors

write( sim_time, 123 ) time_simulated
read( sim_time,  * ) time_simulated

123 format( F8.2 )

time_simulated = time_simulated + dt
```

```
time(n) = time_simulated
!Report which timestep we're in every 20 or so time steps
if( nint( real(n)/2. ) .gt. report ) then
    report = report + 1
    write(*,*) ' n = ', n, ' time = ', time_simulated,          &
                      'L_inf_norm = ', maxrelchng_ss,            &
                         'L2_norm = ', F_L2_Norm(relchng,vmax),  &
                            'Qout = ', Qout(n-1)
endif


!Come up with the rainfall rate for this timestep
rain(n) = F_Linterp( time_simulated  , &
                     rain_time(1:nrr), &
                     rain_rate(1:nrr), &
                     nrr                 )


!PART OF NON-LINEAR SYSTEM FROM TIME LEVEL n
!   FRICTION SLOPE
!   Compute friction slope magnitudes based on the converged thicknesses
!   from the previous time step
CALL FrictionSlope( 'old', Sfw_old, Sfe_old, Sfs_old, Sfn_old )



       ! Compute solution for 1D model to use as a boundary condition

! only invoke the 1D solver if called for by the boundary conditions

if( west_bc .eq. '1D_FLOW' .or. &
    east_bc .eq. '1D_FLOW'        ) then

       h_old_1d = h_new_1d

       CALL PFC1DIMP(  h_old = h_old_1d, &
                          dt = dt      , &
                        rain = rain(n) , &    !  Should probably add rain(n-1)
                       tolit = eps_itr , &
                        qmax = qmax    , &
                       h_new = h_new_1d, &
                        imax = TNE     , &
                    eta_0_BC = south_bc, &
                    eta_1_BC = north_bc   )


       ! Vet the solution to avoid a weird problem
       if( maxval( h_new_1d) .LT. TINY(h_new_1d(1) ) ) then
           write(100,*) 'PERFCODE: 1D Model zeroed out....stopping program'
           call write_vector( h_old_1d, TNE, 'h_old_1D.csv' )
           call write_vector( h_new_1d, TNE, 'h_new_1D.csv' )
           stop
       end if

```

```
endif

!---------------------------------------------
!   B O U N D A R Y   C O N D I T I O N S
!---------------------------------------------

! put east first so that west bc 'eastKIN' could copy it

!EASTERN BOUNDARY
i = imax
if( east_bc .eq. 'NO_FLOW' ) then
        do j = 2, jmax - 1
           pf = F_por( h_old( i, j ) )
           CALL Conveyance( 'west ', 'old', i, j, Cw )
           Ce = 0.0                            !<---- NO FLOW BOUNDARY
           CALL Conveyance( 'south', 'old', i, j, Cs )
           CALL Conveyance( 'north', 'old', i, j, Cn )
           v     = F_LinearIndex( i, j, jmax)
           Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )
        end do

elseif( east_bc .eq. '1D_FLOW') then

!open( unit = 66, file = 'eta_mapping.csv', status = 'REPLACE' )
!write( 66, * ) 'i,j,eta,eta_1D'

        do j = 1, jmax
            v = F_LinearIndex( i, j, jmax)
           eta_1D = F_LINTERP(        X = CV_Info( v ) % eta ,   &
                               known_X = eta_cs              ,   &
                               known_Y = eta_cs_1D           ,   &
                                     n = nr_cs + 1                      )
           h_bound= F_LINTERP(        X = eta_1D             ,   &
                                known_X = etaCV               ,   &
                                known_Y = h_new_1D            ,   &
                                      n = TNE                           )
            C(v) = 1.0
            F(v) = h_bound

!            write( 66, 660) i, j, CV_Info( v ) % eta, eta_1D

        end do

!close(66)


elseif( east_bc .eq. 'MOC_KIN' ) then

        do j = 2, jmax - 1
            CALL MOC_KIN_BC( i, j, rain(n), dt, 'east ', h_bound, 100 )
            v = F_LinearIndex( i, j, jmax )
            C(v) = 1.
            F(v) = h_bound
!            write(100,*) 'PERFCODE: east bc i=',i, 'j=',j, 'h_bound=',h_bound
```

```fortran
        end do


end if



!WESTERN BOUNDARY
i = 1
if( west_bc .eq. 'NO_FLOW' ) then
        do j = 2, jmax - 1
           ! Set porosity factor for this cell
           pf = F_por( h_old( i, j ) )
           ! Set the conveyance coefficients
           Cw = 0.0     !<---- NO FLOW BOUNDARY
           CALL Conveyance( 'east ', 'old', i, j, Ce )
           CALL Conveyance( 'south', 'old', i, j, Cs )
           CALL Conveyance( 'north', 'old', i, j, Cn )
           ! Compute the part if the right-hand-side that is from
           ! time level n
           v     = F_LinearIndex( i, j, jmax)
           Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )
        end do
elseif( west_bc .eq. '1D_FLOW' ) then
        do j = 1, jmax
            v = F_LinearIndex( i, j, jmax)
            eta_1D = F_LINTERP(        X = CV_Info( v ) % eta ,  &
                                 known_X = eta_cs             ,  &
                                 known_Y = eta_cs_1D          ,  &
                                       n = nr_cs + 1               )
            h_bound= F_LINTERP(        X = eta_1D             ,  &
                                 known_X = etaCV              ,  &
                                 known_Y = h_new_1D           ,  &
                                       n = TNE                     )
            C(v) = 1.0
            F(v) = h_bound
        end do

elseif( west_bc .eq. 'MOC_KIN' ) then
        write(*,*) 'PERFCODE: Boundary condition ', west_bc, &
                             'not supported for western boundary'

elseif( west_bc .eq. 'eastKIN' ) then

        do j = 2, jmax-1
            v = F_LinearIndex( i, j, jmax )
            ! index of corresponding eastern cell
            ve = F_LinearIndex( imax, j, jmax )
            ! Use solutions from east side on the west side
            C(v) = C(ve)
            F(v) = F(ve)
        end do

endif
```

```
!NORTHERN BOUNDARY
j = jmax
if( north_bc .eq. 'NO_FLOW' ) then
        do i = 2, imax - 1
           ! Set porosity factor for this cell
           pf = F_por( h_old( i, j ) )
           ! Set the conveyance coefficients
           CALL Conveyance( 'west ', 'old', i, j, Cw )
           CALL Conveyance( 'east ', 'old', i, j, Ce )
           CALL Conveyance( 'south', 'old', i, j, Cs )
           Cn = 0.0  !  <---- NO FLOW BOUNDARY
           ! Compute the part if the right-hand-side that is from
           ! time level n
           v     = F_LinearIndex( i, j, jmax)
           Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )
        end do

elseif( north_bc .eq. 'MOC_KIN' ) then
        do i = 2, imax - 1
            CALL MOC_KIN_BC( i, j, rain(n), dt, 'north', h_bound, 100)
            v = F_LinearIndex( i, j, jmax )
            C(v) = 1.
            F(v) = h_bound
        end do
!       ! Use the value of the next inside cell for cells
!       ! second from the end of the domain
!       i = 2
!       v = F_LinearIndex( i, j, jmax )
!       v_in = F_LinearIndex( i+1, j, jmax)
!       C(v) = 1.
!       F(v) = F( v_in)
!       i = imax - 1
!       v = F_LinearIndex( i, j, jmax )
!       v_in = F_LinearIndex( i-1, j, jmax)
!       C(v) = 1.
!       F(v) = F( v_in)
!
elseif( north_bc .eq. '1D_FLOW' ) then
        write(*,*) 'PERFCODE: Boundary condition ', north_bc, &
                             'not supported for northern boundary'

elseif( north_bc .eq. 'west_1D' .and. &
        west_bc .eq. '1D_FLOW'       ) then

      ! Put the answer for the northern most cell on the west end (i=1, j=jmax)
      !  in all of the northern cells

      do i = 2, imax - 1
             v = F_LinearIndex( i, j, jmax )
```

```
          v_in = F_LinearIndex( 1, jmax, jmax )
          C(v) = 1.
          F(v) = F(v_in)
      end do

end if


!SOUTHERN BOUNDARY
j = 1
if( south_bc .eq. 'NO_FLOW' ) then
        do i = 2, imax - 1
           pf = F_por( h_old( i, j ) )
           CALL Conveyance( 'west ', 'old', i, j, Cw )
           CALL Conveyance( 'east ', 'old', i, j, Ce )
           Cs = 0.0  !<------- NO FLOW BOUNDARY
           CALL Conveyance( 'north', 'old', i, j, Cn )
           v     = F_LinearIndex( i, j, jmax)
           Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )
        end do

elseif( south_bc .eq. 'MOC_KIN' ) then
        do i = 2, imax - 1
            CALL MOC_KIN_BC( i, j, rain(n), dt, 'south',  h_bound, 100)
            v = F_LinearIndex( i, j, jmax )
            C(v) = 1.
            F(v) = h_bound
        end do
!       ! Use the value of the next inside cell for cells
!       ! second from the west end of the domain
!       i = 2
!       v = F_LinearIndex( i, j, jmax )
!       v_in = F_LinearIndex( i+1, j, jmax)
!       C(v) = 1.
!       F(v) = F( v_in)
!       ! second from east end of domain
!       i = imax - 1
!       v = F_LinearIndex( i, j, jmax )
!       v_in = F_LinearIndex( i-1, j, jmax)
!       C(v) = 1.
!       F(v) = F( v_in )

elseif( south_bc .eq. '1D_FLOW' ) then
        write(*,*) 'PERFCODE: Boundary condition ', south_bc, &
                             'not supported for southern boundary'

elseif( south_bc .eq. 'west_1D'  .AND.  &
         west_bc .eq. '1D_FLOW'            ) then

        ! Put the answer for the southern most cell on the west end (v=1)
        !  in all of the southen cells

        do i = 2, imax - 1
            v = F_LinearIndex( i, j, jmax )
```

```
           v_in= F_LinearIndex( 1, 1, jmax )
            C(v) = 1.
            F(v) = F(v_in)
        end do

end if



!----------------------------
! C O R N E R   P O I N T S
!----------------------------
! only the 1D_FLOW condition is already handled for the corner points

! NORTH EAST CORNER
i = imax; j = jmax
if( north_bc .eq. 'NO_FLOW' .AND. east_bc .eq. 'NO_FLOW' ) then
           ! Set porosity factor for this cell
           pf = F_por( h_old( i, j ) )
           ! Set the conveyance coefficients
           CALL Conveyance( 'west ', 'old', i, j, Cw )
           Ce = 0.0  !  <---- NO FLOW BOUNDARY
           CALL Conveyance( 'south', 'old', i, j, Cs )
           Cn = 0.0  !  <---- NO FLOW BOUNDARY
           ! Compute the part of the right-hand-side that is from
           ! time level n
           v     = F_LinearIndex( i, j, jmax)
           Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )

elseif( north_bc .eq. 'MOC_KIN' .AND. east_bc .eq. 'NO_FLOW') then
           ! use the depth in the adjacent MOC_KIN cell
           v    = F_LinearIndex( i, j, jmax )
           v_in = F_LinearIndex( i-1, j, jmax )
           C(v) = 1.0
           F(v) = F( v_in )

elseif( north_bc .eq. 'NO_FLOW' .AND.  &
         east_bc .eq. 'MOC_KIN'          )  then

        ! is a problem when there are no grade breaks
        ! just value of adjacent no flow cell ??
        v    = F_LinearIndex( i, j, jmax )
        A(v) = -1.
        C(v) = 1.
        F(v) = 0.

elseif( Z( imax, jmax) .GE. Z(imax, jmax-1)   .AND. &
              north_bc .NE. 'NO_FLOW'                 ) then

              write( 100, *) ' North east corner drains to the south &
                             &consider NO_FLOW boundary for the north &
                                                    &side of the domain. '

elseif( Z( imax, jmax) .LT. Z(imax, jmax-1) .AND.  &
```

```
               east_bc .eq. 'MOC_KIN'                 ) then

       ! drainage is to the north and MOC KIN will work
       call MOC_KIN_BC( i, j, rain(n), dt, 'east ', h_bound, 100 )
       v = F_LinearIndex( i, j, jmax )
       C(v) = 1.
       F(v) = h_bound

end if



!   NORTH WEST CORNER POINTS
i = 1; j = jmax
if( north_bc .eq. 'NO_FLOW' .AND. west_bc .eq. 'NO_FLOW' ) then
          ! Set porosity factor for this cell
          pf = F_por( h_old( i, j ) )
          ! Set the conveyance coefficients
          Cw = 0.0  !  <---- NO FLOW BOUNDARY
          CALL Conveyance( 'east ', 'old', i, j, Ce )
          CALL Conveyance( 'south', 'old', i, j, Cs )
          Cn = 0.0  !  <---- NO FLOW BOUNDARY
          ! Compute the part if the right-hand-side that is from
          ! time level n
          v     = F_LinearIndex( i, j, jmax)
          Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )

elseif( north_bc .eq. 'MOC_KIN' .AND. west_bc .eq. 'NO_FLOW' ) then
          !use the depth from the adjaent MOC_KIN cell
          v    = F_LinearIndex( i, j, jmax )
          v_in = F_LinearIndex( i+1, j, jmax )
          C(v) = 1.0
          F(v) = F( v_in )

elseif( west_bc .eq. 'eastKIN' ) then

           v = F_LinearIndex( i, j, jmax )
           ! index of corresponding eastern cell
           ve = F_LinearIndex( imax, j, jmax )
           ! Use solutions from east side on the west side
           C(v) = C(ve)
           F(v) = F(ve)

end if



! SOUTH EAST CORNER
i = imax; j = 1
if( south_bc .eq. 'NO_FLOW' .and. east_bc .eq. 'NO_FLOW' ) then
          pf = F_por( h_old( i, j ) )
          ! Set the conveyance coefficients
          CALL Conveyance( 'west ', 'old', i, j, Cw )
          Ce = 0.0  !  <---- NO FLOW BOUNDARY
```

```
            Cs = 0.0  !  <---- NO FLOW BOUNDARY
            CALL Conveyance( 'north', 'old', i, j, Cn )
            ! Compute the part of the right-hand-side that is from
            ! time level n
            v     = F_LinearIndex( i, j, jmax)
            Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )

elseif( south_bc .eq. 'MOC_KIN' .AND. east_bc .eq. 'NO_FLOW' ) then
            ! use the depth in the adjacent MOC_KIN cell
            v    = F_LinearIndex( i, j, jmax )
            v_in = F_LinearIndex( i-1, j, jmax )
            C(v) = 1.0
            F(v) = F( v_in )

elseif( south_bc .eq. 'MOC_KIN' .AND. east_bc .eq. 'MOC_KIN' ) then

         call MOC_KIN_BC( i, j, rain(n), dt, 'east ', h_bound, 100 )
         v = F_LinearIndex( i, j, jmax )
         C(v) = 1.
         F(v) = h_bound

end if

! SOUTHWEST CORNER
i = 1; j = 1
if( south_bc .eq. 'NO_FLOW' .AND. west_bc .eq. 'NO_FLOW' ) then
            pf = F_por( h_old( i, j ) )
            ! Set the conveyance coefficients
            Cw = 0.0  !  <---- NO FLOW BOUNDARY
            CALL Conveyance( 'east ', 'old', i, j, Ce )
            Cs = 0.0  !  <---- NO FLOW BOUNDARY
            CALL Conveyance( 'north', 'old', i, j, Cn )
            ! Compute the part of the right-hand-side that is from
            ! time level n
            v     = F_LinearIndex( i, j, jmax)
            Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )

elseif( south_bc .eq. 'MOC_KIN' .AND. west_bc .eq. 'NO_FLOW' ) then
            ! use the depth in the adjacent MOC_KIN cell
            v    = F_LinearIndex( i, j, jmax )
            v_in = F_LinearIndex( i+1, j, jmax )
            C(v) = 1.0
            F(v) = F( v_in )

elseif( west_bc .eq. 'eastKIN' ) then

             v = F_LinearIndex( i, j, jmax )
             ! index of corresponding eastern cell
             ve = F_LinearIndex( imax, j, jmax )
             ! Use solutions from east side on the west side
             C(v) = C(ve)
             F(v) = F(ve)


```

```
end if


!---------------------------------------------
!  D O M A I N   I N T E R I O R
!---------------------------------------------
!   Compute the part of the right hand side of the linear system
!   that is from time level n (the stationary part that does not
!   change as the iteration progresses)
do j = 2, jmax -1;  do i = 2, imax - 1

    ! Set porosity factor for this cell
    pf  = F_por( h_old( i, j ) )
    ! Set the conveyance coefficients
    CALL Conveyance( 'west ', 'old', i, j, Cw )
    CALL Conveyance( 'east ', 'old', i, j, Ce )
    CALL Conveyance( 'south', 'old', i, j, Cs )
    CALL Conveyance( 'north', 'old', i, j, Cn )
    ! Compute the part of the right-hand-side that is from
    ! time level n
    v     = F_LinearIndex( i, j, jmax)
    Fn(v) = F_RHS_n( i, j, Cw, Ce, Cs, Cn, rain(n-1), pf, dt )

end do; end do


!----------------------------------------------
!ITERATIVE (LAGGED) PART OF NON-LINEAR SYSTEM
!----------------------------------------------

!zero out matrix iteration counter
timestep_solver_numits = 0

iteration: do q = 1, qmax

!   FRICTION SLOPE
!   compute friction slope magnitudes based on the thickness
!   from the previous iteration
CALL FrictionSlope( 'itr', Sfw_itr, Sfe_itr, Sfs_itr, Sfn_itr )


!   BOUNDARY CELLS
!WESTERN BOUNDARY
if( west_bc .eq. 'NO_FLOW' ) then
        i = 1
        do j = 2, jmax - 1
            pf = F_por( h_itr( i, j ) )
            Cw1 = 0.0             !<---------------No flow boundary
            CALL Conveyance( 'east ', 'itr', i, j, Ce1 )
            CALL Conveyance( 'south', 'itr', i, j, Cs1 )
            CALL Conveyance( 'north', 'itr', i, j, Cn1 )
            CALL set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )
        end do
end if
```

```fortran

!EASTERN BOUNDARY
if( east_bc .eq. 'NO_FLOW') then
        i = imax
        do j = 2, jmax - 1
            pf = F_por( h_itr( i, j ) )
            CALL Conveyance( 'west ', 'itr', i, j, Cw1 )
            Ce1 = 0.0               !<---------------No flow boundary
            CALL Conveyance( 'south', 'itr', i, j, Cs1 )
            CALL Conveyance( 'north', 'itr', i, j, Cn1 )
            CALL set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )
        end do
end if

!SOUTHERN BOUNDARY
if( south_bc .eq. 'NO_FLOW') then
        j = 1
        do i = 2, imax - 1
           ! Set porosity factor for this cell
           pf = F_por( h_itr( i, j ) )
           ! Set the conveyance coefficients
           CALL Conveyance( 'west ', 'itr', i, j, Cw1 )
           CALL Conveyance( 'east ', 'itr', i, j, Ce1 )
           Cs1 = 0.0  !  <---- NO FLOW BOUNDARY
           CALL Conveyance( 'north', 'itr', i, j, Cn1 )
           ! Fill in the linear system
           CALL set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )
        end do
end if


!NORTHERN BOUNDARY
if( north_bc .eq. 'NO_FLOW') then
        j = jmax
        do i = 2, imax - 1
           ! Set porosity factor for this cell
           pf = F_por( h_itr( i, j ) )
           ! Set the conveyance coefficients
           CALL Conveyance( 'west ', 'itr', i, j, Cw1 )
           CALL Conveyance( 'east ', 'itr', i, j, Ce1 )
           CALL Conveyance( 'south', 'itr', i, j, Cs1 )
           Cn1 = 0.0  !  <---- NO FLOW BOUNDARY
           ! Fill in the linear system
           CALL set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )
        end do
end if


!NORTH WEST CORNER
if( north_bc .eq. 'NO_FLOW' .AND. west_bc .eq. 'NO_FLOW' ) then
           i = 1; j = jmax
           ! Set porosity factor for this cell
           pf = F_por( h_itr( i, j ) )
           ! Set the conveyance coefficients
```

```
           Cw1 = 0.0  !  <---- NO FLOW BOUNDARY
           CALL Conveyance( 'east ', 'itr', i, j, Ce1 )
           CALL Conveyance( 'south', 'itr', i, j, Cs1 )
           Cn1 = 0.0  !  <---- NO FLOW BOUNDARY
           ! Fill in the linear system
           CALL set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )
end if

!NORTH EAST CORNER
if( north_bc .eq. 'NO_FLOW' .AND. east_bc .eq. 'NO_FLOW' ) then
           i = imax; j = jmax
           ! Set porosity factor for this cell
           pf = F_por( h_itr( i, j ) )
           ! Set the conveyance coefficients
           CALL Conveyance( 'west ', 'itr', i, j, Cw1 )
           Ce1 = 0.0  !  <---- NO FLOW BOUNDARY
           CALL Conveyance( 'south', 'itr', i, j, Cs1 )
           Cn1 = 0.0  !  <---- NO FLOW BOUNDARY
           ! Fill in the linear system
           CALL set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )
end if


! SOUTH WEST CORNER
if(  south_bc .eq. 'NO_FLOW'  .and. &
      west_bc .eq. 'NO_FLOW'          ) then

        i = 1; j = 1
        pf = F_por( h_itr( i, j ) )
        Cw1 = 0.0
        call conveyance( 'east ', 'itr', i, j, Ce1 )
        Cs1 = 0.0
        call Conveyance( 'north', 'itr', i, j, Cn1 )
        call set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )

end if


! SOUTH EAST CORNER
if(  south_bc .eq. 'NO_FLOW'  .and. &
      east_bc .eq. 'NO_FLOW'           ) then

        i = 1; j = 1
        pf = F_por( h_itr( i, j ) )
        call conveyance( 'east ', 'itr', i, j, Cw1 )
        Ce1 = 0.0
        Cs1 = 0.0
        call Conveyance( 'north', 'itr', i, j, Cn1 )
        call set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )

end if


! INTERIOR of DOMAIN
```

```
do j = 2, jmax - 1;     do i = 2, imax - 1

    ! set porosity factor for this cell
    pf = F_por( h_itr( i, j ) )
    ! These things Do change as the iteration progresses
    CALL Conveyance( 'west ', 'itr', i, j, Cw1 )
    CALL Conveyance( 'east ', 'itr', i, j, Ce1 )
    CALL Conveyance( 'south', 'itr', i, j, Cs1 )
    CALL Conveyance( 'north', 'itr', i, j, Cn1 )
    ! Fill in the linear system
    CALL set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rain(n) )

end do;     end do


! TRANSITION CHECK
!   test to see if there is a transition to or from sheet flow
!   happening during this timestep.   Use under-relaxtion to
!   control oscillations during a transition timestep.

transition = .false.
do j = 1, jmax
    do i = 1, imax
        ! integers used to assure correct behavor when  equal
        pf_int = nint( F_por( h_old(i,j) ) )
        pf1_int= nint( F_por( h_itr(i,j) ) )
        if( pf_int .NE. pf1_int ) then
            transition = .true.
        endif
    end do
end do

if( transition .eqv. .true. ) then
    relaxation_factor = relax_tran
    eps_itr_tol = eps_itr * 10.
else
    relaxation_factor = relax
    eps_itr_tol = eps_itr
endif


! CALL diagdom_penta( A, B, C, D, E, n, LB, UB, diagdom)
! WRITE(100,*) 'Timestep ', n, 'Iteration ', q, &
!               'Is matrix diagonally dominant?', diagdom


! Confirm that there is a value of C for all of the rows
! this is mostly a check to see that the corner points of
! the domain had values put in.
do v = 1, vmax
    if( abs( C(v) ) .LT.  TINY(  C(v) ) ) then
        write(100,*) ' No value of C: v = ', v, 'C(v)=', C(v)
        write(*,*) 'STOPPING PROGRAM'
        STOP
```

```fortran
    end if
end do



!CALL SOLVER
! gauss_seidel_penta(A,B,C,D,E,F,n,LB,UB,tolit,maxit,Xold,Xnew)
CALL GAUSS_SEIDEL_penta( A, B, C, D, E, F, vmax, jmax, jmax, eps_matrix, maxit,&
                         h_itr_vec, h_tmp_vec, 100, solver_numits )


! Compute residual and relative change for this iteration.  This took
!  some careful thought to handle both filling and draining cases.
!  Relative change is used when the solution is far from zero
!  and absolute change (residual) is used near zero.


! Should put residual/ relchng computation block into a subroutine.

do v = 1, vmax

    if( h_tmp_vec(v) .GT. TINY( h_tmp_vec(v) ) ) then

            ! Compute residual for this iteration
            residual(v) =  h_tmp_vec(v) - h_itr_vec(v)

            ! Handle a result that is effectively zero by
            ! using an absolute tolerance instead of
            ! a relative one
            if( h_tmp_vec(v) .LE. h_pfc_min  .and. &
                residual (v) .LE. eps_itr_tol     ) then

                    relchng(v) = 0.0

            else
                    relchng (v) =  residual (v) / h_itr_vec(v)
            endif

    elseif( h_tmp_vec(v) .LE. TINY( h_tmp_vec(v) ) ) then

            ! the model is saying the cell is empty,
            ! so force the solution to be zero
            h_tmp_vec(v) = 0.0
            ! compute the residual
            residual(v) =  h_tmp_vec(v) - h_itr_vec(v)
            ! For the zero case, use an absolute rather than
            ! relative tolerance by setting the value of relchng
            ! below the tolerance instead of computing it.
            if( abs( residual(v) ) .LE. eps_itr_tol ) then

                    relchng(v) = 0.0
            endif
    endif

```

```
end do


! Store solution history during iteration in case
!  the model fails to converge
h_temp_hist( :, q ) = h_tmp_vec


!Output the biggest change for this iteration
WRITE(100,*) 'PERFCODE: Iteration q =',         q                 , &
                'Solver Interations =',     solver_numits          , &
                        'L_inf_norm  =', maxval( abs( relchng ) )   , &
                         'At Cell v  =', maxloc( abs( relchng ) )   , &
                          ' L2 Norm  =', F_L2_NORM( relchng, vmax ) , &
                        'eps_itr_tol =', eps_itr_tol

! CONVERGENCE TEST
! Exit iteration loop if this timestep has converged
if( maxval( abs ( relchng ) )    .le. eps_itr_tol  .AND. &
    F_L2_NORM   ( relchng, vmax) .le. eps_itr_tol           ) then
    WRITE(100,*) 'Time step n = ', n, time(n),'sec ' , &
                      'rain(n) = ', rain(n)          , &
              ' converged in q = ', q, ' iterations.' , &
              ' maxdepth=', maxval( h_tmp_vec ),        &
              ' max 1D =', maxval( h_new_1D ) , 'min 1D =', minval( h_new_1D)

    WRITE(100,*)''
    !output results for each timestep for checking purposes
!    write(50,2) n, h_itr_vec(:)
    EXIT iteration
endif



!update iteration variables
h_itr_vec = h_itr_vec + relaxation_factor * residual



! un-linearize the thicknesses back to a matrix h_itr_vec ---> h_itr
call unlinearize( h_itr_vec, imax, jmax, vmax, h_itr )


end do iteration


!Give Error if Iteration fails to converge and write some diagnostics
if (q .gt. qmax) then
    WRITE(*,*) ' Iteration failed to converge for time level n = ', n
    !output the coefficient matrix and main diagonal
    call write_system( A, B, C, D, E, F, vmax, 'ABCDEF.csv' )
    call write_flipped_matrix( h_old, imax, jmax, 'h_old.csv' )
    call write_matrix( h_temp_hist, vmax, qmax, 'h_temp_hist.csv')
```

```
    call WRITE_VECTOR( residual, vmax, 'residual_iteration.csv')
    call WRITE_VECTOR( relchng, vmax, 'relchng_iteration.csv')
!    call put_bands(a, b, c, d, e, vmax, lb, ub, amatrix)
!    call write_matrix( amatrix, vmax, vmax, 'amatrix.csv')
    EXIT time_stepping
end if


! Compute Change for this time step
!Time stepping residual (re-uses the arrays)
residual = h_tmp_vec - h_old_vec

! compute relative change for this timestep
do v = 1, vmax
    if( abs(residual(v)) .LT. TINY(residual(v)) ) then
            ! The converged solution is zero
            relchng(v) = 0.0
    else
            !the solution is non-zero, compute as ususal
            relchng(v) = residual(v) / h_old_vec(v)
    endif
end do

maxrelchng_ss = maxval ( ABS( relchng ) )

!call WRITE_VECTOR( relchng, vmax, 'relchng_time.csv')



!Update the old and new solutions
!At the end of the iteration, we have found values for the
!next time step.
h_new_vec  = h_tmp_vec

!but when we go back to the top of the loop, the old is what we just found
h_old_vec  = h_new_vec
!and now we need to unlinearize the h_old values
call unlinearize( h_old_vec, imax, jmax, vmax, h_old )






!--------------------------------------------------------------------
!  Summary Info for this timestep
!-------------------------------------------------------------------

numit         ( n ) = q
loc           ( n ) = maxloc ( abs( relchng ), dim = 1 )
maxdiff       ( n ) = relchng ( loc ( n ) )
maxthk        ( n ) = maxval( h_old_vec )
L2_History    ( n ) = F_L2_Norm( relchng, vmax )
h_imid_j1_hist( n ) = h_old( imax/2, 1 )
```

```
h_imid_max_hist(n)  = maxval( h_old( imax/2 , :) )

! Compute the flow into the southern boundary for this time step
!  (assume that we can neglect the drainage area of the last row)
j = 2
do i = 1, imax
    CALL Conveyance( 'south', 'itr', i, j, Cs1 )
    Qout(n) = Qout(n) + Cs1 * area(i,j) *        &
              (  ( h_itr(i, j-1) - h_itr(i,j) ) &
               + (     Z(i, j-1) -     Z(i,j) )   )
end do


!SELECTIVELY STORE MODEL RESULTS
! MAXIMUM DEPTH
! Check to see if this was the maximum time-step and store if so
if( maxval( h_old_vec) .GT. maxval( h_max ) ) then
        call unlinearize( h_old_vec, imax, jmax, vmax, h_max )
endif

! MAXIMUM DISCHARGE
if( Qout(n) .GT. maxval( Qout(1:n-1) ) ) then
        call unlinearize( h_old_vec, imax, jmax, vmax, h_Q_max )
endif

! MAXIMUM MID DOMAIN DISCHARGE DEPTH
if( h_imid_j1_hist( n ) .GT. maxval( h_imid_j1_hist(1:n-1) )) then
        call unlinearize( h_old_vec, imax, jmax, vmax, h_imid_j1_max )
endif

! MAXIMUM MID DOMAIN DISCHARGE DEPTH
if( h_imid_max_hist( n ) .GT. maxval( h_imid_max_hist(1:n-1) )) then
        call unlinearize( h_old_vec, imax, jmax, vmax, h_imid_max )
endif


! ANIMATION
!  Decide if the results from this timestep should be stored for
!  animation output.  Take the time, divide by the animation step,
!  round to the lowest integer and then convert to integer
if( animate .eqv. .true. ) then

    if( int( floor( time(n) / dt_ani ) )  .gt. ani ) then
        ! set the value of ani
        ani = ani + 1
print *, 'n = ', n, 'ani=', ani
        ! store the solution for this step
        h_vec_ani( :, ani ) = h_old_vec
        ! also store a label
        write( sim_time2, 123 ) time_simulated
        ani_lab( ani ) = 'h'//sim_time2//'s'
        ani_time(ani ) = time_simulated
    endif

```

```
endif

!STEADY-STATE CHECK  ( disabled in favor of setting
!                       the time for the simulation to run )
!IF (          maxrelchng_ss .le. eps_ss .AND. &
!     F_L2_NORM( relchng, v) .le. eps_ss         ) then
!  WRITE(*,*) 'Simulation reached steady state after', n, &
!               &'time steps or', time_simulated, 'seconds'
!  EXIT time_stepping
!end if

end do time_stepping

!for outputting each timestep
! close(50)

!close log file
close(100)

!-------------------------------------------------------------------------------
!   >>>>>>>>>>      P O S T   P R O C E S S I N G      <<<<<<<<<<
!-------------------------------------------------------------------------------

!-------------------------------------------------------------------------------
!   >>>>>>>>>>     W R I T E    O U T P U T    F I L E S       <<<<<<<<<<
!-------------------------------------------------------------------------------
!Set date and time stamps

CALL SYSTEM_CLOCK( RUN_END_TIME, COUNT_RATE, COUNT_MAX )
call DATE_AND_TIME(FILE_DATE,FILE_TIME)
call CPU_TIME(cputime)

!-------------------------------------------------------------------------------
! file to show 1D solution along i = imax / 2; j = 1:jmax

OPEN(UNIT = 10, FILE = 'PERFCODE.csv', STATUS='REPLACE')
WRITE(10,*) 'Output From PERFCODE.f95'
WRITE(10,*) 'Timestamp,', FILE_DATE,' ', FILE_TIME,','
do i = 1, 18
    write( 10, * ) input_variables(i), ',', input_values(i), ','
end do
write(10,*) 'north_bc,', north_bc
write(10,*) 'south_bc,', south_bc
write(10,*) 'east_bc,', east_bc
write(10,*) 'west_bc,', west_bc

WRITE(10,200) 'Average Rainfall Intensity (m/s),', &
               sum( rain(1:nlast )) / time_simulated
WRITE(10,200) 'Average Rainfall Intensity (cm/hr),',&
               sum( rain(1:nlast )) / time_simulated * 3600. * 100.
```

```
WRITE(10,200) 'Final Time (sec),', time_simulated
WRITE(10,201) 'Number of cells longitudinally,', imax
WRITE(10,201) 'Number of cells transversly,'   , jmax
WRITE(10,201) 'Total Number of Grid Cells,', vmax
WRITE(10,200) 'CPU Time (seconds),', cputime
WRITE(10,200) 'Run Time (seconds),', &
               real(run_end_time - run_start_time)/real(count_rate)
WRITE(10, *) '************************** &
              &1D MODEL OUTPUT IN [ SI ] UNITS &
              &**********************************,'
i = imax / 2
write(10,*) ' i = ', i,','
write(10, *) 'j,eta,Z,PFC_Surf,h,Head,Surf_Thk.mm,'
do j = 1, jmax
    v = F_LinearIndex( i, j, jmax)
    write(10, 2) j, CV_Info(v)%eta, Z(i,j), Z(i,j) + b_pfc,      &
                                 h_old(i,j), Z(i,j) + h_old(i,j), &
                               ( h_old(i,j) - b_pfc ) * 1000.
end do



!------------------------------------------------------------------------------
! 3d plotting output for maximum depth
!    ( contour plots of the resuls are made from  this file )
open( unit = 10, file = 'max_depth.csv', status = 'replace' )

write( 10, * ) 'v, X, Y, Z, h,'

do j = 1, jmax
    do i = 1, imax
        v = F_LinearIndex( i, j, jmax)
        write(10, 2) v, CV_Info (v ) % X,                          &
                        CV_Info( v ) % Y, Z(i,j), h_max(i,j)
    end do
end do

close( 10 )



!-------------------------------------------------------------------------------
! 3d plotting output for maximum discharge
!    ( contour plots of the resuls are made from  this file )
open( unit = 10, file = 'max_Q.csv', status = 'replace' )

write( 10, * ) 'v, X, Y, Z, h,'

do j = 1, jmax
    do i = 1, imax
        v = F_LinearIndex( i, j, jmax)
        write(10, 2) v, CV_Info (v ) % X,                          &
                        CV_Info( v ) % Y, Z(i,j), h_Q_max(i,j)
    end do
```

```
end do

close( 10 )
!-------------------------------------------------------------------------------
! 3d plotting output for maximum mid-domain outlet depth
!    ( contour plots of the resuls are made from  this file )
open( unit = 10, file = 'max_imidj1depth.csv', status = 'replace' )

write( 10, * ) 'v, X, Y, Z, h,'

do j = 1, jmax
    do i = 1, imax
        v = F_LinearIndex( i, j, jmax)
        write(10, 2) v, CV_Info (v ) % X,                    &
                        CV_Info( v ) % Y, Z(i,j), h_imid_j1_max(i,j)
    end do
end do

close( 10 )
!------------------------------------------------------------------------------
! 3d plotting output for maximum mid-domain outlet depth
!    ( contour plots of the resuls are made from  this file )
open( unit = 10, file = 'max_imiddepth.csv', status = 'replace' )

write( 10, * ) 'v, X, Y, Z, h,'

do j = 1, jmax
    do i = 1, imax
        v = F_LinearIndex( i, j, jmax)
        write(10, 2) v, CV_Info (v ) % X,                    &
                        CV_Info( v ) % Y, Z(i,j), h_imid_max(i,j)
    end do
end do

close( 10 )



!---------------------------------------------------------------------------------
! Write parameters to a seperate file for convenicence
open( unit = 15, file = 'params.csv', status = 'REPLACE' )
write( 15, 155 ) input_variables(:), 'north_bc', 'south_bc', 'east_bc', 'west_bc'
write( 15, 156 ) input_values(:), north_bc, south_bc, east_bc, west_bc
close(15)

155  format (  22( A, ',') )
156  format (  18( E, ','), 4 ( A, ',') )
!--------------------------------------------------------------------------
!Write time history to a file
!  ( hydrographs and anything else time-dependant
!    is plotted from this file )

OPEN( UNIT = 20, FILE = 'details.csv', STATUS='REPLACE')
WRITE(20,*) 'Timestamp,', FILE_DATE, ' ', FILE_TIME, ','
```

```
DO i = 1, 18
    WRITE( 20, * ) input_variables(i), ',', input_values(i), ','
END DO
write(20,*) 'north_bc,', north_bc
write(20,*) 'south_bc,', south_bc
write(20,*) 'east_bc,', east_bc
write(20,*) 'west_bc,', west_bc
WRITE(20,*) 'imax,', imax, ','
WRITE(20,*) 'jmax,', jmax, ','
WRITe(20,*) 'vmax,', vmax, ','
WRITE(20,*) '-----,'
WRITE(20,*) 'Timestep,Iterations,MaxRelChng,MaxLocn,' , &
                                'L2_Norm,Rain.mmphr,' , &
                           'MaxThk.cm,Time,Qout.Lps,' , &
                                    'h_imid_j1_hist,' , &
                                    'h_imid_max_hist,'
DO n = 1, nlast
WRITE(20,300) n, numit(n), maxdiff(n), loc(n)         , &
              L2_History(n), rain(n)*1000.*3600.      , &
              maxthk(n)*100., time(n), -Qout(n)*1000. , &
              h_imid_j1_hist(n), h_imid_max_hist(n)
end do
close(20)


!----------------------------------------------------------------------------
! Output depth grid for last timestep

! an internal write statement to store the value of the REAL variable
! "time_simulated" in the CHARACTER variable "out_time"
write( out_time, 111 ) time_simulated

call write_flipped_matrix( h_old, imax, jmax, 'h_old'//out_time//' sec.csv' )

!-----------------------------------------------------------------------------
! Output iteration history for the last time-step

call write_matrix( h_temp_hist, vmax, qmax, 'h_temp_hist'//out_time//' sec.csv')

!-----------------------------------------------------------------------
! Animation output

if( animate .eqv. .TRUE. ) then

!Animation results
open( unit = 70, file = 'animate.csv', status = 'REPLACE' )
write( 70, 700) 'v,X,Y,Z,', ani_lab(:)
do j = 1, jmax
    do i = 1, imax
        v = F_LinearIndex( i, j, jmax)
        write(70, 2) v, CV_Info( v ) % X,                 &
                        CV_Info( v ) % Y, Z(i,j), h_vec_ani( v, :)
    end do
end do
```

```fortran
close( 70 )

700 format( (A, 10000( A, ',') ) )

!Also sperately output the list of animation lables
open( unit = 71, file = 'ani_labs.csv', status = 'REPLACE' )
write( 71, *) 'ani,lab,time,'
do ani = 1, animax
    write( 71, 711 ) ani, ani_lab(ani), ani_time(ani)
end do
close( 71 )

end if


711 format( (I, ','), (A, ','), (F8.2, ',') )

!-----------------------------------------------------------------------------------
! Output grid numbering scheme to a file
! store grid numbering scheme and write it to a file
do j = 1, jmax
    do i = 1, imax
        grid( j, i) = F_LinearIndex( i, j, jmax )
    end do
end do

open( unit = 30, file = 'grid.csv', status = 'REPLACE' )
do j = jmax, 1, -1
    WRITE(30, 400 ) grid( j, : )
end do
close(30)

!---------------------------------------------------------------------------------
!Format statements

2       FORMAT( I, ',', 10000 ( E, ',') )
10      FORMAT('   ', ( i3, '    '),  ( F10.3, '  ') , F10.6 )
111     FORMAT( f9.2 )
200     FORMAT ( A, ( E, ',') )
201     FORMAT ( A, ( I, ',') )
300     FORMAT ( 2 (     I, ','),      F12.7, ',' , &    ! n, numit, maxdif
                         I, ',' ,          E, ',' , &    ! loc, L2_History
                 2 ( F12.8, ','), ( F12.3, ',' ), 3 ( F12.8 ,',')     )  ! rain, maxthk, time, Qout, h_imid_j1hist, h_imid_max_hist
400     FORMAT( 10000 ( I, ',' ) )
401     FORMAT( (I, ',') , 2( F12.7, ',' ) )
660     FORMAT(  2( I, ','), 2( F12.7, ',') )
!----------------------------------------------------------------------------------
end program PERFCODE
!==================================================================================
!       \\\\\\\\\\         E N D   P R O G R A M         //////////
!       //////////           P E R F C O D E             \\\\\\\\\\
!==================================================================================
```

Source File 2: shared.f95

```fortran
! fortran_free_source
!
!  This module is part of PERFCODE, written by Bradley J. Eck.
!
!   File Date:  5 April 2010
!
!   Purpose:  This module declares variables to be used globally
!
!     Notes:  - Variable organization tries to mirror program
!               the organization of the program
!             - See begining of main program for alphabetical
!               listing of variables with descriptions
!             - Use ONLY statement in subroutines to restrict
!               access to variables in this module
!================================================================
!       \\\\\\\\\\                                  //////////
                             MODULE SHARED
!       //////////                                  \\\\\\\\\\
!================================================================
implicit none
save

!--------------------------------------------------
!PARAMETERS INPUT FILE
!--------------------------------------------------
!   PFC Properties
REAL :: K       ! Hydraulic Conductivity [m/s]
REAL :: por     ! Porosity [--]
REAL :: b_pfc   !PFC Thickness [ m ]
REAL :: n_mann  !Manning's n [ s / m ^(1/3) ]
!   Physical constants
REAL :: g       ! Gravitational Acceleration [m/s/s]
! Time Steps
REAL :: dt_pfc, dt_sheet, max_time
! Grid Spacing
REAL :: dx, dy
!Tolerances
INTEGER :: qmax, maxit
REAL :: eps_matrix, eps_itr, eps_ss
REAL :: relax, relax_tran
!Initial Condition
real :: h0 ! initial depth in meters
!Boundary Conditions
character( len=7 ) :: north_bc, south_bc, east_bc, west_bc
!Animation Options
logical :: animate ! at all and for this step
real :: dt_ani
!-------------------------
! OTHER PARAMETERS
!-------------------------
```

```
INTEGER, PARAMETER :: max_rec = 1000
REAL, PARAMETER :: h_pfc_min  = 1.e-10   ! use this instead of TINY

!---------------------------
! RAINFALL
!---------------------------
INTEGER :: nrr  ! Number of rainfall records
REAL, DIMENSION( max_rec ) :: rain_time, rain_rate


!---------------------------------------------------------
!  GRID GENERATION
!---------------------------------------------------------

!----------------------
!  Derived data types
type CLSEG  !describes a centerline segment
    real xcc1, ycc1, dx, dy, R1, dR, W, theta1, dtheta, arclen
end type CLSEG

type gridcell   ! Summary information for a grid cell
    integer :: i, j, segment
    real    :: xi, eta
    real    :: X , Y
end type gridcell

! allocatable variables of derived types
type(CLSEG), allocatable, dimension(:) :: seg
type(gridcell) , allocatable, dimension(:) :: CV_Info  !17
!-------------------------

!Array sizes
integer :: imax, jmax, vmax

! Grid numbering scheme
integer, allocatable, dimension(:,:) :: grid

! Geometric Arrays
REAL, ALLOCATABLE, DIMENSION(:,:) :: lng, wid, area, Z
REAL, ALLOCATABLE, DIMENSION(:,:) :: lng_south, lng_north


!-------------------------------------------------
!  ELEVATIONS
!------------------------------------------------

!CROSS SECTION  ( Transverse direction)
! input file
integer :: nr_cs
REAL ::  slope_cs(10), wid_cs(10)

! derived values
real, dimension( 11 ) :: eta_cs=0., Z_cs=0.

```

```fortran
!LONGITUDINAL PROFILE
integer :: nr_lp
real, dimension(100) :: dist_lp, Z_lp
real :: long_slope  !longitudinal slope at each end of domain


!  1D GRID GENERATION
integer, TARGET :: TNE
REAL, ALLOCATABLE, DIMENSION(:), TARGET :: EDX, XCV, ZCV, etaCV
! 1D boudary conditions
real, allocatable, dimension(:) :: h_old_1d, h_new_1d
real, allocatable, dimension(:) :: slope_cs_1D, wid_cs_1d, eta_cs_1D


!------------------------------------------------------------
! INTERMEDIATE VARIABLES
!-------------------------------------------------------------

! ARRAY INDICES AND LIMITING VALUES
integer :: i, j, v, q, n
integer :: ve
integer :: v_in !global index of 'inside' adjacent cell
integer :: nmax     ! maximum number of time steps
integer :: nlast    !the last timestep taken

! TIME STUFF
REAL :: dt
REAL, ALLOCATABLE, DIMENSION(:) :: rain  !rainfall depth for each time step
REAL, ALLOCATABLE, DIMENSION(:) :: time
real :: time_simulated = 0.
character( len = 9 ) :: out_time   ! Characters to for internal writes to store
character( len = 8 ) :: sim_time   ! simulation time w/o floating point error 99999.99
character( len = 8 ) :: sim_time2

! FRICTION SLOPES, POROSITY FUNCTIONS, AND CONVEYANCE COEFFICIENTS
!  'old' means time level 'n'
!  'itr' or '1' means time level n+1
REAL, ALLOCATABLE, DIMENSION(:,:), TARGET :: Sfw_old, Sfe_old, Sfs_old, Sfn_old
REAL, ALLOCATABLE, DIMENSION(:,:), TARGET :: Sfw_itr, Sfe_itr, Sfs_itr, Sfn_itr
REAL :: pf, pf1
REAL :: Cw , Ce , Cs , Cn
REAL :: Cw1, Ce1, Cs1, Cn1

! BOUNDARY CONDITION STUFF
real :: eta_1D
real :: hs1, hs2, ds      ! Sheet flow MOC
real :: hp1, hp2, dx_moc ! PFC flow MOC
real :: h_bound          ! depth at boundary (returnd by MOC_KIN or 1D_FLOW
real :: eta_0_hp2_max    ! max possible value for the MOC BC

! CONVERGENCE TESTING
logical :: transition
real :: relaxation_factor
```

```fortran
REAL :: eps_itr_tol
integer :: pf_int, pf1_int  ! use integers to detect transition
REAL, ALLOCATABLE, DIMENSION(:) :: residual, relchng
real :: maxrelchng_ss

! LINEAR SYSTEM
!  Bands
REAL, ALLOCATABLE, DIMENSION(:) :: A, B, C, D, E, Fn, F1, F
!  Test for diagonal Dominance
logical diagdom
!  Square matrix for outputting/use with library solvers

!--------------------------------------------------------
! THE SOLUTION (at various stages and in various formats)
!--------------------------------------------------------

! Vector Form
REAL, ALLOCATABLE, DIMENSION(:) :: h_itr_vec, h_tmp_vec
REAL, ALLOCATABLE, DIMENSION(:) :: h_old_vec, h_new_vec

! Vector form, within a timestep (during an iteration)
real, allocatable, dimension(:,:) :: h_temp_hist

! Vector form, at intervals for animation
!   rows --> grid cells
!   cols --> times
REAL, ALLOCATABLE, DIMENSION(:,:) :: h_vec_ani

! Matrix Form
REAL, ALLOCATABLE, DIMENSION(:,:), TARGET :: h_old, h_itr

! Matrix form, at special times
real, allocatable, dimension(:,:) :: h_max, h_Q_max
real, allocatable, dimension(:,:) :: h_imid_j1_max, h_imid_max

!-----------------------------------------------------
! SUMMARY INFORMATION
!-----------------------------------------------------

! Input variables and values
character( len=10), dimension(18) :: input_variables
real, dimension(18) :: input_values

! Information about each timestep
INTEGER, ALLOCATABLE, DIMENSION(:) :: numit, loc
REAL, ALLOCATABLE, DIMENSION(:) :: maxdiff
real, allocatable, dimension(:) :: maxthk
integer, allocatable, dimension(:) :: matrix_numits
real, allocatable, dimension(:) :: Qout, L2_History
integer :: solver_numits, timestep_solver_numits

! time history of the depth at i=imax/2 j=1
real, allocatable, dimension( : ) :: h_imid_j1_hist, h_imid_max_hist

```

```
!-----------------------------------------------------
!  MISCELLANEOUS  (gotta love this category)
!------------------------------------------------------

integer, dimension(60) :: astat=0     ! for keeping track of allocation statuses
integer, dimension( 30 ) :: astat2(0:29) = 0

CHARACTER(8)  FILE_DATE
CHARACTER(10) FILE_TIME

! Routine timing
REAL    :: cputime
integer :: run_start_time, run_end_time, count_rate, count_max

integer :: report = 1  ! determine if we should write out the timestep.

! For animation output

integer :: ani = 0  ! use this like 'report'
integer :: animax   ! maximum value of ani, compute from max_time / ani_step
character( len = 10 ), allocatable, dimension(:) :: ani_lab   ! labels for animtaion output
real, allocatable, dimension(:) :: ani_time
character(len = 10) :: lab

!===============================================================================
!       \\\\\\\\\\                                           //////////
                               END MODULE SHARED
!       //////////                                           \\\\\\\\\\
!===============================================================================
```

Source File 3: pfc2Dfuns.f95

```fortran
! fortran_free_source

! This module holds external procedures (subroutine and functions)
!  for the pfc2D model (PERFCODE).
! Using module creates an explicit interface for the procedures


module pfc2Dfuns

implicit none

contains

!   1. F_LinearIndex
!   2. F_por
!   3. F_RHS_n
!   4. F_RHS_n1


!============================================================
Function F_LinearIndex( i, j, jmax )
!   Converts grid index to one-dimensional storage location
implicit none
integer, intent( in ) :: i, j, jmax
integer               :: F_LinearIndex
F_LinearIndex = ( i - 1) * jmax + j
end Function F_LinearIndex
!=============================================================


!===========================================================================
!Function to switch the porosity on/off if the water is in/out of the pavement
FUNCTION F_por(h)
USE shared, only: b_pfc, por
IMPLICIT NONE
REAL h, F_por
if     ( h >= b_pfc ) then
            F_por = 1.
ELSEIF ( h < b_pfc  )  then
            F_por = 1./por
end if
END Function F_por
!=======================================================================

Function F_RHS_n( i, j, Cw, Ce, Cs, Cn, rr, pf, dt ) Result( Fn )
!   Computes the RHS of the linear system for time level n
use shared, only: h_old, Z, imax, jmax
implicit none
! Arguments
integer, intent( in ) :: i, j
real   , intent( in ) :: Cw, Ce, Cs, Cn, rr, pf, dt
```

```fortran
! Internal variables
!   added a bunch of dummy variables with if statements to have this function
!   also work at the boundaries.
real :: Fn
real :: hw, he, hn, hs, Zw, Ze, Zs, Zn


! Thicknesses
if( i == 1 )    then; hw = 0.0; else; hw = h_old(i-1,j); endif
if( j == 1 )    then; hs = 0.0; else; hs = h_old(i,j-1); endif
if( j == jmax) then; hn = 0.0; else; hn = h_old(i,j+1); endif
if( i == imax) then; he = 0.0; else; he = h_old(i+1,j); endif
! Elevations
if( i == 1 )    then; Zw = 0.0; else; Zw = Z(i-1,j); endif
if( j == 1 )    then; Zs = 0.0; else; Zs = Z(i,j-1); endif
if( j == jmax) then; Zn = 0.0; else; Zn = Z(i,j+1); endif
if( i == imax) then; Ze = 0.0; else; Ze = Z(i+1,j); endif

!Compute the RHS from time level n
    Fn     = h_old(i,j)    +                          &
             pf * dt / 2. * (  Cw * hw  +  Cs * hs &
                              + Cn * hn  +  Ce * he &
                              + Cw * Zw  +  Cs * Zs &
                              + Cn * Zn  +  Ce * Ze &
                              -  (Cw + Cs + Cn + Ce) * h_old(i,j)       &
                              -  (Cw + Cs + Cn + Ce) *     Z(i,j)       &
                              + rr                                       )
end function F_RHS_n
!===========================================================================


Function F_RHS_n1( i, j, Cw1, Ce1, Cs1, Cn1, rr, pf, dt ) Result (F1)
!   Computes the part of the RHS due to time level n+1
use shared, only: Z, imax, jmax
implicit none
!   Arguments
integer, intent( in ) :: i,j
real    , intent( in ) :: Cw1, Ce1, Cs1, Cn1, rr, pf, dt
!   Internal Variables
real :: F1
real :: Zw, Ze, Zs, Zn

! Elevations
if( i == 1 )    then; Zw = 0.0; else; Zw = Z(i-1,j); endif
if( j == 1 )    then; Zs = 0.0; else; Zs = Z(i,j-1); endif
if( j == jmax) then; Zn = 0.0; else; Zn = Z(i,j+1); endif
if( i == imax) then; Ze = 0.0; else; Ze = Z(i+1,j); endif

F1          = pf * dt / 2. * (  Cw1 * Zw  +  Cs1 * Zs &
                              + Cn1 * Zn  +  Ce1 * Ze &
                              - (Cw1 + Cs1 + Cn1 + Ce1)* Z(i,j)    &
                              + rr                                  )
end function F_RHS_n1
!=================================================================================
```

```



end module pfc2Dfuns
```

Source File 4: Utilities.f95

```
! fortran_free_source

!=====================================================================
!=====================================================================
!    \\\\\\\\\\   B E G I N   M O D U L E         ///////////
!    //////////                 U T I L I T I E S  \\\\\\\\\\\
!=====================================================================
module utilities
implicit none
contains

! This module holds subroutines and functions for various jobs:
!    1.  Subroutine  GET_BANDS
!    2.  Subroutine  PUT_BANDS
!    3.  Subroutine  UNLINEARIZE
!    4.  Subroutine  BILINEAR_INTERP
!    5.  Function    F_LINTERP
!    6.  Function    F_L2_NORM
!    7.  Function    F_PYTHAGSUM
!    8.  Function    F_EXTRAPOLATE
!======================================================================


!======================================================================
!    \\\\\\\\\\      B E G I N    F U N C T I O N   ///////////
!    //////////       G E T _ B A N D S             \\\\\\\\\\\
!======================================================================
!
!    PURPOSE:    Extracts the five bands from a penta-diagonal matrix.
!
SUBROUTINE GET_BANDS( COEF, N, LB, UB, A, B, C, D, E)
!
! COEF -- Penta-diagonal coefficient matrix.
!    N  -- number of unknowns (size of system)
!   LB  -- lower bandwidth
!   UB  -- upper bandwidth
!   A,B -- lower bands of the penta-diagonal matrix
!    C  -- main diagonal
!   D,E -- upper bands of the penta-diagonal matrix
!---------------------------------------------------------------------
!VARIABLE DECLARATIONS
!    Arguments
```

```
integer, intent( in ) :: N, LB, UB
real,    intent( in ) :: COEF( N, N )
real,    intent( out) :: A(N), B(N), C(N), D(N), E(N)
!   Internal variables
integer ::  i   !looping variable
!-----------------------------------------------------------------------

! Lowermost subdiagonal
do i = LB+1, n
    A(i) = coef(i,i-LB)
end do

! Subdiagonal
do i = 2, n
    B(i) = coef(i,i-1)
end do

! Main Diagonal
do i = 1, n
     C(i) = coef(i,i)
end do

! Super diagonal
do i = 1, n-1
     D(i) = coef(i,i+1)
end do


! Uppermost diagonal
do i = 1, n - UB
     E(i) = coef(i,i+ub)
end do
!-----------------------------------------------------------------------
end subroutine GET_BANDS
!======================================================================
!   \\\\\\\\\\     E N D      S U B R O U T I N E ///////////
!   //////////      G E T _ B A N D S           \\\\\\\\\\\
!======================================================================

!======================================================================
!   \\\\\\\\\\    B E G I N   S U B R O U T I N E ///////////
!   //////////      P U T  _ B A N D S          \\\\\\\\\\\
!======================================================================
!
!   PURPOSE:    Puts the five bands into a square matrix.
!
SUBROUTINE PUT_BANDS(A, B, C, D, E, N, LB, UB, COEF)
!
!  A,B -- lower bands of the penta-diagonal matrix
!   C  -- main diagonal
!  D,E -- upper bands of the penta-diagonal matrix
!   N  -- number of unknowns (size of system)
!  LB  -- lower bandwidth
!  UB  -- upper bandwidth
```

```
! COEF -- Penta-diagonal coefficient matrix.
!-----------------------------------------------------------------------
!VARIABLE DECLARATIONS
!   Arguments
integer, intent( in ) :: N, LB, UB
real,    intent( in  ) :: A(N), B(N), C(N), D(N), E(N)
real,    intent( out ) :: COEF( N, N )
!   Internal variables
integer ::  i   !looping variable
!----------------------------------------------------------------------
! Fill coefficient matrix with zeros

coef(:,:) = 0.0

! Lowermost subdiagonal
do i = LB+1, n
    coef(i,i-LB) = A(i)
end do

! Subdiagonal
do i = 2, n
    coef(i,i-1) = B(i)
end do

! Main Diagonal
do i = 1, n
     coef(i,i) = C(i)
end do

! Super diagonal
do i = 1, n-1
     coef(i,i+1) = D(i)
end do


! Uppermost diagonal
do i = 1, n - UB
     coef(i,i+ub) = E(i)
end do
!----------------------------------------------------------------------
end subroutine PUT_BANDS
!=======================================================================
!   \\\\\\\\\\     E N D      S U B R O U T I N E ///////////
!   //////////       P U T _ B A N D S            \\\\\\\\\\\
!=======================================================================


!=======================================================================
!   \\\\\\\\\\     B E G I N     S U B R O U T I N E ///////////
!   //////////       U N L I N E A R I Z E           \\\\\\\\\\\
!=======================================================================
subroutine unlinearize( vector, imax, jmax, vmax, matrix )

!   Puts unknowns in linear (vector) form into matrix form
```

```
!   Assumes column-wise ordering from southwest corner of domain
use pfc2Dfuns, only: F_LinearIndex
implicit none
!Arguments
integer,                       intent( in ):: imax, jmax, vmax
real, dimension(vmax),         intent(in)  :: vector
real, dimension(imax, jmax),   intent(out) :: matrix
!Internal Variables
integer                                    :: i, j, v

DO j = 1, jmax
    DO i = 1, imax
        v = F_LinearIndex( i, j, jmax )
        matrix(i,j) = vector( v )
    end do
end do

end subroutine unlinearize
!=======================================================================
!   \\\\\\\\\\      E N D        S U B R O U T I N E ///////////
!   //////////      U N L I N E A R I Z E           \\\\\\\\\\\
!=======================================================================

!=======================================================================
!   \\\\\\\\\\   B E G I N    S U B R O U T I N E    //////////
!   //////////      B I L I N E A R _ I N T E R P    \\\\\\\\\\
!=======================================================================
subroutine BILINEAR_INTERP       (  X , Y , Z ,   &
                                    x1, y1, z1,   &
                                    x2, y2, z2,   &
                                    x3, y3, z3,   &
                                    x4, y4, z4,   &
                                    dev, error       )

! Finds the value of Z at the point X,Y using Finite Element
! style interpolation with a Bi-linear element.  The physical
! coordinates (x,y,z) are mapped into ksi, eta space that
! ranges from -1 to 1.  The values of ksi and eta for the point
! X, Y are found by solving the non-linear system using the
! Newton-Raphson method.
!
!

implicit none
!VARIABLE DECLARATIONS
!   Arguments
real, intent( in  ) :: X, Y         ! Coordinates of point where Z is desired
real, intent( out ) ::       Z      ! Unknown function value
real, intent( in  ) :: x1, y1, z1   ! Coordinates of point 1
real, intent( in  ) :: x2, y2, z2   !   "    "       point 2
real, intent( in  ) :: x3, y3, z3   !   "    "       point 3
real, intent( in  ) :: x4, y4, z4   !   "    "       point 4
integer, optional   :: dev          ! output device for writing errors
logical, optional   :: error
```

```fortran

!   Internal variables
real :: ksi, eta                       ! mapped coordinates of XY
real :: X_guess, Y_guess               ! Values of X and Y computed from ksi and eta
real :: delta_ksi, delta_eta           ! incremental change in values over iteration
real :: J_11, J_12, J_21, J_22         ! elements of the jacobian matrix
real :: PSI_1, PSI_2, PSI_3, PSI_4     ! Shape functions for BiLinear element
real, parameter :: tolit = 1.e-5      ! iteration tolerance
integer, parameter :: qmax = 10       ! maximum number of iterations
integer             :: q               ! looping variable
integer             :: device          ! output device

!-------------------------------------------------------------------------------

! Default values for output device
if( present( dev ) .EQV. .FALSE. ) then
        device = 6
else
        device = dev
end if


! STEP 1:  Find the value of ksi and eta that correspond to the point X,Y
!  initial guess for ksi and eta is in the middle of the element ( 0,0 )
ksi = 0.0
eta = 0.0

Map: do q = 1, qmax

        !   Values of the shape functions at the point (X, Y)
        PSI_1 = 0.25 * ( 1. - ksi ) * ( 1. - eta )
        PSI_2 = 0.25 * ( 1. + ksi ) * ( 1. - eta )
        PSI_3 = 0.25 * ( 1. + ksi ) * ( 1. + eta )
        PSI_4 = 0.25 * ( 1. - ksi ) * ( 1. + eta )

        !  figure out value of X and Y using ksi and eta
        X_guess = x1*PSI_1 + x2*PSI_2 + x3*PSI_3 + x4*PSI_4
        Y_guess = y1*PSI_1 + y2*PSI_2 + y3*PSI_3 + y4*PSI_4


        !compute values of jacobian
        !J_11 = d X_guess / d ksi
        J_11 =   x1 / 4. * ( eta - 1.  ) &
               + x2 / 4. * ( 1.  - eta ) &
               + x3 / 4. * ( eta + 1.  ) &
               - x4 / 4. * ( eta + 1.  )

        !J_12 = d X_guess / d eta
        J_12 =   x1 / 4. * ( ksi - 1.  ) &
               - x2 / 4. * ( ksi + 1.  ) &
               + x3 / 4. * ( ksi + 1.  ) &
               + x4 / 4. * (  1. - ksi )

        !J_21 = d Y_guess / d ksi
```

```
        J_21 =   y1 / 4. * ( eta - 1.  ) &
               + y2 / 4. * (  1. - eta ) &
               + y3 / 4. * ( eta + 1.  ) &
               - y4 / 4. * ( eta + 1.  )

        !J_22 = d Y_guess / d eta
        J_22 =   y1 / 4. * ( ksi - 1.  ) &
               - y2 / 4. * ( ksi + 1.  ) &
               + y3 / 4. * ( ksi + 1.  ) &
               + y4 / 4. * ( 1.  - ksi )

        !Manual solution of 2 x 2 system:  J * delta_ksi/eta = X/Y_guess - X/Y
        delta_ksi = (   (X_guess - X)*J_22        &
                      - (Y_guess - Y)*J_12    ) / &
                    (   J_11 * J_22               &
                      - J_12 * J_21           )

        delta_eta = (   (Y_guess - Y)*J_11        &
                      - (X_guess - X)*J_21    ) / &
                    (   J_11 * J_22               &
                      - J_12 * J_21           )


!write(device,*) 'BILINER_INTERP q=', q, 'delta_ksi =', delta_ksi, ' delta_eta', delta_eta

        ! update vales of ksi and eta
        ! rembeber delta = ksi_q - ksi_q+1
        ksi = ksi - delta_ksi
        eta = eta - delta_eta

        !Convergence Test
        if( abs( delta_ksi ) .LT. tolit .AND.   &
            abs( delta_eta ) .LT. tolit         ) then

                exit Map
        endif

end do Map



!report mapping result
!write( device, * ) 'BILINEAR_INTERP: Mapping result: ksi =', ksi, ' eta = ', eta

! assume no error and change if there is one
if( present( error) .eqv. .TRUE. ) then
    error = .FALSE.
end if

! Give Error if iteration fails to converge
if( q .GT. qmax ) then
    write( device, * ) 'BILINEAR_INTERP: Mapping iteration failed. ksi =', ksi, ' eta = ', eta
```

```

    if( present( error) .eqv. .TRUE. ) then   !assign an error if the variable was provided.
        error = .TRUE.
    end if
end if


! Confirm that mapped point lies inside the range of the datapoints
if( abs( ksi ) .GT. 1. + tolit  .OR. &
    abs( eta ) .GT. 1. + tolit        )  then
    write( device, * ) 'BILINEAR_INTERP: Desired point lies outside &
                        & known points: ksi =', ksi, ' eta = ', eta
    if( present( error) .eqv. .TRUE. ) then   !assign an error if the variable was provided.
        error = .TRUE.
    end if
end if

! STEP 2: Having found the values of ksi and eta that correspond
!         to the point (X, Y) compute the value of Z at that location.

!   Values of the shape functions at the point (X, Y)
PSI_1 = 0.25 * ( 1. - ksi ) * ( 1. - eta )
PSI_2 = 0.25 * ( 1. + ksi ) * ( 1. - eta )
PSI_3 = 0.25 * ( 1. + ksi ) * ( 1. + eta )
PSI_4 = 0.25 * ( 1. - ksi ) * ( 1. + eta )

! Value of Z at the point (X, Y)
Z = z1*PSI_1 + z2*PSI_2 + z3*PSI_3 + z4*PSI_4

!write( device, *) 'BILINEAR_INTERP: PSI_1=', PSI_1, &
!                                  ' PSI_2=', PSI_2, &
!                                   'PSI_3=', PSI_3, &
!                                   'PSI_4=', PSI_4, &
!                                      ' Z=', Z

!---------------------------------------------------------------------
end subroutine BILINEAR_INTERP
!=====================================================================
!   \\\\\\\\\\    E N D    S U B R O U T I N E        //////////
!   //////////       B I L I N E A R _ I N T E R P    \\\\\\\\\\
!=====================================================================


!=====================================================================
!   \\\\\\\\\\      B E G I N    F U N C T I O N   ///////////
!   //////////       F _ L I N T E R P             \\\\\\\\\\\
!=====================================================================
! linear interpolation function
function F_linterp( X, known_X, known_Y, n) Result( Y )
!VARIABLE DECLARATIONS
!   Arguments
integer, intent( in ) :: n
```

```fortran
real    , intent( in ) :: X
real, dimension(n), intent(in) :: known_X, known_Y
!   Internal Variables
real :: Y
integer :: i1, i2, j, im
!-----------------------------------------------------------------
! bi-section method to find the right place in the table
! initialize indices
i1 = 1
i2 = n


if( known_X(n) .GT. known_X(1)  ) then
!ASCENDING ORDER
    do j = 1, 1000
        if ( i2 - i1 .gt. 1 ) then
            im = (i1+i2)/2       !midpoint
            if ( X .eq. known_X(im) ) then
                i1 = im
                i2 = im + 1
            elseif( X .gt. known_X(im) ) then
                i1 = im
            elseif( X .lt. known_X(im) ) then
                i2 = im
            endif
        else
            exit
        end if
    end do

elseif( known_X(n) .LT. known_X(1) )then
!DESCENDING ORDER
    do j = 1, 1000
        if ( i2 - i1 .gt. 1 ) then
            im = (i1+i2)/2       !midpoint
            if ( X .eq. known_X(im) ) then
                i1 = im
                i2 = im + 1
            elseif( X .gt. known_X(im) ) then
                i2 = im
            elseif( X .lt. known_X(im) ) then
                i1 = im
            endif
        else
            exit
        end if
    end do

end if


!        WRITE(*,*) 'j=', j, 'im=', im, 'i1=', i1, 'i2=', i2

if( j .eq. 1000 ) then
```

```
    write( *,* ) 'F_LINTERP: Arrays too large for this routine, &
                 & increase number of searching steps and recompile.'
endif


! bounds found; compute interpolated value
Y  = (X - Known_X(i1)) / &
     (Known_X(i2) - Known_X(i1)) * &
     (Known_Y(i2) - Known_Y(i1)) + Known_Y(i1)


!-----------------------------------------------------------------------
end function  F_LINTERP
!=======================================================================
!   \\\\\\\\\\      E N D   F U N C T I O N      ///////////
!   //////////       F _ L I N T E R P           \\\\\\\\\\\
!=======================================================================


!=======================================================================
!   \\\\\\\\\\      B E G I N   F U N C T I O N   ///////////
!   //////////       F _ L 2 _ N O R M           \\\\\\\\\\\
!=======================================================================
function F_L2_NORM( vector, n ) result( L2 )
!   Computes the L2 norm of a real-valued vector with n elements
!
! Variable Declarations
implicit none
! Arguments
integer,                   intent( in ) :: n
real    , dimension( n ), intent( in ) :: vector
! Internal Variables
real :: L2
real, dimension( n ) :: squares
integer :: i
!----------------------------------------------------------------------
do i = 1, n
    squares(i) = vector(i) ** 2
end do

L2 = sqrt( sum( squares(:) ) )
!----------------------------------------------------------------------
end function F_L2_NORM
!=======================================================================
!   \\\\\\\\\\      E N D   F U N C T I O N      ///////////
!   //////////       F _ L 2 _ N O R M           \\\\\\\\\\\
!======================================================================


!==============================================================
!   \\\\\\\\\\  B E G I N   F U N C T I O N      ///////////
!   //////////       F _ P y t h a g S u m       \\\\\\\\\\\
!==============================================================
Function F_PythagSum( x, y)
```

```
! Computes the pythagorean sum of twovariables
implicit none
REAL :: x, y, F_PythagSum
F_PythagSum = sqrt( x**2 + y**2 )
end function F_PythagSum
!=====================================================================
!   \\\\\\\\\\   E N D      F U N C T I O N      ///////////
!   //////////      F _ P y t h a g S u m        \\\\\\\\\\\
!====================================================================


!====================================================================
!   \\\\\\\\\\  B E G I N    F U N C T I O N      ///////////
!   //////////      F _ E X T R A P O L A T E    \\\\\\\\\\\
!====================================================================
Function F_Extrapolate( X, x1, y1, x2, y2) RESULT(Y)
!   Finds the value of Y corresponding to the location X
!   on the line passing through ( x1, y1) and (x2, y2)
!
!   Called from:    convcoef@frictionslope


implicit none
REAL, intent(in) :: X, x1, y1, x2, y2
REAL :: Y
REAL :: slope, intercept
slope = (y2 - y1) / (x2 - x1)
intercept = y1 - slope * x1
Y = slope * X + intercept
end function F_Extrapolate
!=====================================================================
!   \\\\\\\\\\   E N D      F U N C T I O N      ///////////
!   //////////      F _ E X T R A P O L A T E    \\\\\\\\\\\
!====================================================================


!=====================================================================
end module utilities
!=====================================================================
!   \\\\\\\\\\   E N D      M O D U L E          ///////////
!   //////////      U T I L I T I E S            \\\\\\\\\\\
!====================================================================
!=====================================================================
```

Source File 5: inputs.f95

```fortran
! fortran_free_source

!   Purpose:    This module contains subroutines to read input files


module inputs

implicit none

contains

!   1. GET_PARAMETERS
!   2. GET_RAINFALL

!===================================================================
!   \\\\\\\\\\      B E G I N    S U B R O U T I N E ///////////
!   //////////         G E T _ P A R A M E T E R S   \\\\\\\\\\\
!==================================================================
!
!   Purpose:    This subroutine reads problem parameters
!               from a user selected input file.
subroutine GET_PARAMETERS( K, por, b_pfc, n_mann, g, dt_pfc, dt_sheet, max_time, &
                           dx, dy, qmax, maxit, h0, eps_matrix, eps_itr, eps_ss, &
                           relax, relax_tran,                &
                           north_bc, south_bc, east_bc, west_bc,                 &
                           animate, dt_ani )
!
!   K   --  Darcy Hydraulic Conductivity
!   por --  Effective porosity of the PFC
!  b_pfc--  Thickness of the pfc
! n_mann--  Manning's n
!   g   --  gravitational acceleration
!-------------------------------------------------------------------
! VARIABLE DECLARATIONS
!Arguments
REAL, intent( out ) :: K, por, b_pfc, n_mann, g, dt_pfc, dt_sheet, max_time, dx, dy
integer, intent(out) :: qmax, maxit
real, intent( out ) :: h0, eps_matrix, eps_itr, eps_ss, relax, relax_tran
character(7), intent(out) :: north_bc, south_bc, east_bc, west_bc
logical, intent( out)  :: animate
real, intent( out ) :: dt_ani
! Internal variables
CHARACTER(20) infile    ! the file name to read parameters from
CHARACTER(5) :: dummy_line
!---------------------------------------------------------------
! Executable

! default value for input file
```

```
infile = 'parameters.dat'

! Prompt the user for the input file
WRITE(*,*) 'Enter filename or press / for ', infile
READ(*,*) infile

!read the file
OPEN( UNIT=8,   FILE = infile, ACTION = 'read', STATUS = 'old' )

READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * ) dummy_line
! PFC Properties
READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * )   K
READ( unit=8, fmt = * )   por
READ( unit=8, fmt = * )   b_pfc
READ( unit=8, fmt = * )   n_mann

!Physical Constants
READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * )   g

!Timesteps
READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * )   dt_pfc
READ( unit=8, fmt = * )   dt_sheet
READ( unit=8, fmt = * ) max_time

! preliminary grid spacing
READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * ) dx
READ( unit=8, fmt = * ) dy

! tolerences
READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * ) qmax
READ( unit=8, fmt = * ) maxit
READ( unit=8, fmt = * ) eps_matrix
READ( unit=8, fmt = * ) eps_itr
READ( unit=8, fmt = * ) eps_ss
READ( unit=8, fmt = * ) relax
READ( unit=8, fmt = * ) relax_tran


! inital depth
READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * ) h0

! Boundary conditions
READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * ) north_bc
READ( unit=8, fmt = * ) south_bc
READ( unit=8, fmt = * ) east_bc
READ( unit=8, fmt = * ) west_bc
```

```
!Animation options
READ( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * ) animate
READ( unit=8, fmt = * ) dt_ani

close( 8 )

!---------------------------------------------------------------------
end subroutine GET_PARAMETERS
!=====================================================================
!   \\\\\\\\\\      E N D      S U B R O U T I N E ///////////
!   //////////         G E T _ P A R A M E T E R S   \\\\\\\\\\\
!=====================================================================



!=====================================================================
!   \\\\\\\\\\      B E G I N   S U B R O U T I N E ///////////
!   //////////         G E T _  R A I N F A L L     \\\\\\\\\\\
!=====================================================================
!
!   Purpose:    This subroutine reads the rainfall record
!               from a user selected input file.

SUBROUTINE GET_RAINFALL(max_rec, rain_time, rain_rate, nrr)

INTEGER, intent( in) :: max_rec ! maximum allowable number of rainfall records
INTEGER, intent( out ) :: nrr   !Number of rainfall records
REAL, DIMENSION( max_rec), intent( inout) :: rain_time, rain_rate

! Internal variables
CHARACTER(30) infile    ! the file name to read parameters from
integer :: i, j     ! looping variables


!-------------------------
! read in the rainfall data

! default value for input file
infile = 'rainfall.dat'

! Prompt the user for the input file
WRITE(*,*) 'Enter filename or press / for ', infile
READ(*,*) infile

OPEN( UNIT=8,   FILE = infile, ACTION = 'read', STATUS = 'old' )

! Rainfall Rate
read( unit=8, fmt = * ) nrr

if ( nrr .gt. size ( rain_time ) ) then
    print *, 'GET_RAINFALL: Too many rainfall records--increase array size and recompile'
```

```fortran
else
    do i = 1, nrr
        READ( unit = 8 , fmt = * ) j, rain_time(j), rain_rate(j)
    end do
end if

close( 8 )

!-----------------------------------------------------------------
end subroutine GET_RAINFALL
!=================================================================
!   \\\\\\\\\\      E N D      S U B R O U T I N E ///////////
!   //////////        G E T _ R A I N F A L L      \\\\\\\\\\\
!=================================================================


end module inputs
```

Source File 6: Outputs.f95

```fortran
! fortran_free_source

!   Purpose:    This module contains subroutines to output information

!===========================================================================
!   \\\\\\\\\\                                      //////////
                        MODULE outputs
!   //////////                                      \\\\\\\\\\
                        implicit none

                        contains
!===========================================================================
!   1. ECHO_INPUTS
!   2. WRITE_FLIPPED_MATRIX
!   3. WRITE_MATRIX
!   4. WRITE_VECTOR
!   5. WRITE_SYSTEM

!=================================================================
!   \\\\\\\\\\      B E G I N    S U B R O U T I N E ///////////
!   //////////        E C H O _ I N P U T S        \\\\\\\\\\\
!=================================================================
!
!   Purpose:    This subroutine echos the input data to the
!               specified device in comma seperated values format.
subroutine ECHO_INPUTS( dev )
use shared, only: K, por, b_pfc, n_mann, g
```

```
!----------------------------------------------------------------------
! VARIABLE DECLARATIONS
!   Arguments
integer, intent( In ) :: dev    ! The device number that the output
!integer, intent( in ) :: nrr    ! number of rainfall records
!REAL, intent( in ) :: K, por, b_pfc, n_mann, g
!REAL, intent( in ) :: rain_time(:), rain_rate(:)
!real, intent( in ) :: dt
!   Internal variables
!integer :: i

!----------------------------------------------------------------------
write(dev, *  ) 'SUMMARY OF INPUT DATA,'
write(dev, 200) 'Hydraulic Conductivity (m/s),', K
write(dev, 200) 'Effective Porosity,', por
write(dev, 200) 'PFC Thickness (m),', b_pfc
write(dev, 200) "Manning's n,", n_mann
write(dev, 200) 'Gravitational Acceleration (m/s/s),', g

!--------------------------------------------------------------------------------
! Format Statements
200     FORMAT (' ', A, ( F10.6, ',') )

!-------------------------------------------------------------------
end subroutine ECHO_INPUTS
!===================================================================
!   \\\\\\\\\\      E N D        S U B R O U T I N E ///////////
!   //////////          E C H O _ I N P U T S         \\\\\\\\\\\
!==================================================================

!===================================================================
!   \\\\\\\\\\    B E G I N    S U B R O U T I N E    ///////////
!   ////////// W R I T E _ F L I P P E D _ M A T R I X \\\\\\\\\\\
!==================================================================
subroutine write_flipped_matrix( array, imax, jmax, outfile )
!   Writes matrix in 'flipped' form so it corresponds to
!       the physical geometry.  This means the (1,1) entry
!       appears at the bottom left corner of the ouput file.
!-----------------------------------------------------------
! VARIABLE DECLARATIONS
!   Arguments
integer imax, jmax
real array( imax, jmax)
character(len=*):: outfile  !assumed length specifier *
!   Internal variables
integer :: i, j
integer :: ilist( imax ), jlist( jmax )
!-------------------------------------------------------
! Create lists of indices
do i = 1, imax
    ilist(i) = i
end do

do j = 1,  jmax
```

```fortran
    jlist(j) = j
end do

print *, 'WRITE_FLIPPED_MATRIX: writing the file ', outfile

! Write the length array in upside down form so it corresponds to
!   to the physical geometry.
OPEN( UNIT=9,   FILE = outfile,  STATUS = 'REPLACE' )

! First line
write(9, 1) ' j \ i ', ilist(:)

! and the rest
do j = jmax, 1, -1
        write(9, 2) jlist( j ), array(:,j)
end do

close( 9 )

!---------------------------------------------------------------
! Format statements
1       format( A, ',', 10000 ( I, ',') )
2       format( I, ',', 10000 ( F12.7, ',') )

!-------------------------------------------------------------------
end subroutine write_flipped_matrix
!===================================================================
!   \\\\\\\\\\    E N D        S U B R O U T I N E    ///////////
!   ////////// W R I T E _ F L I P P E D _ M A T R I X \\\\\\\\\\\
!==================================================================


!===================================================================
!   \\\\\\\\\\    B E G I N    S U B R O U T I N E     ///////////
!   //////////        W R I T E _ M A T R I X          \\\\\\\\\\\
!==================================================================
subroutine write_matrix( array, imax, jmax, outfile )
!   Writes matrix in usual form so the (1,1) entry
!       appears at the top left corner of the ouput file.
!-----------------------------------------------------------
! VARIABLE DECLARATIONS
!   Arguments
integer imax, jmax
real array( imax, jmax)
character(len=*):: outfile  !assumed length specifier *
!   Internal variables
integer :: i, j
integer :: ilist( imax ), jlist( jmax )
!-------------------------------------------------------
! Create lists of indices
do i = 1, imax
    ilist(i) = i
end do

```

```
do j = 1,  jmax
    jlist(j) = j
end do

print *, 'WRITE_MATRIX: writing the file ', outfile

!
OPEN( UNIT=9,   FILE = outfile,  STATUS = 'REPLACE' )


do i = 1, imax
         write(9, 4) array(i,:)
end do

close( 9 )

!--------------------------------------------------------------
! Format statements
1       format( A, ',', 10000 ( I, ',') )
2       format( I, ',', 10000 ( F12.7, ',') )
3       format(         10000 ( F12.7, ',') )
4       format(         10000 (   E  , ',') )

!------------------------------------------------------------------
end subroutine write_matrix
!==================================================================
!   \\\\\\\\\\    E N D       S U B R O U T I N E    ///////////
!   //////////      W R I T E _ M A T R I X          \\\\\\\\\\\
!=================================================================

!==================================================================
!   \\\\\\\\\\    B E G I N    S U B R O U T I N E     ///////////
!   //////////       W R I T E _ V E C T O R          \\\\\\\\\\\
!=================================================================
subroutine write_vector( array, imax,  outfile )
!   Writes matrix in usual form so the (1,1) entry
!       appears at the top left corner of the ouput file.
!----------------------------------------------------------
! VARIABLE DECLARATIONS
!   Arguments
integer imax
real array( imax )
character(len=*):: outfile  !assumed length specifier *
!   Internal variables
integer :: i
!------------------------------------------------------
! Create lists of indices

print *, 'WRITE_VECTOR: writing the file ', outfile


!
OPEN( UNIT=9,   FILE = outfile,  STATUS = 'REPLACE' )

```

```
! and the rest
do i = 1, imax
        write(9, 3 ) array(i)
end do

close( 9 )

!-------------------------------------------------------------------
! Format statements
3       format( ( E, ',') )

!-------------------------------------------------------------------
end subroutine write_vector
!===================================================================
!   \\\\\\\\\\    E N D       S U B R O U T I N E    ///////////
!   //////////      W R I T E _ V E C T O R          \\\\\\\\\\\
!==================================================================


!======================================================================
subroutine WRITE_SYSTEM( A, B, C, D, E, F, n, outfile )
integer, intent( in ) :: n
real, dimension( n ), intent( in ) :: A, B, C, D, E, F
character( len=*) :: outfile

integer :: i


open( unit = 11, file = outfile, status = 'REPLACE' )
write( 11, *) 'v, A, B, C, D, E, F,'
do i = 1, n
    write( 11, 3 ) i, A(i), B(i), C(i), D(i), E(i), F(i)
end do
close( 11 )


!----------------------------------------------------------------------
! Format statements
3   format( (I, ','),  6( E, ',' ) )
!----------------------------------------------------------------------
end subroutine WRITE_SYSTEM
!======================================================================


!============================================================================
!   \\\\\\\\\                                     ///////////
                        END MODULE outputs
!   /////////                                     \\\\\\\\\\\\
!============================================================================
```

Source File 7: geom_funcs.f95

```fortran
! fortran_free_source

!==============================================================================
!   \\\\\\\\\\                                          //////////
                        MODULE geom_funcs
!   //////////                                          \\\\\\\\\\
                        implicit none

                        contains
!==============================================================================
!   1.  F_L_xi
!   2.  unmap_x
!   3.  unmap_y

!=============================================================================
Function F_L_xi(xi, eta, seg) Result(L_xi) !xcc1, ycc1, dx, dy, R1, dR, W, theta1, dtheta)
!   Computes the METRIC COEFFICIENT for the length mapping.
!Function F_length_xi(xi, eta, xcc1, ycc1, dx, dy, R1, dR, W, theta1, dtheta)
! GEOMETRY MAPPING FUNCTIONS from Geometry.xlsb
use shared, only: CLSEG
implicit none
!   Arguments
real xi, eta
type(CLSEG) :: seg
!   Result
real L_xi
!   Internal Variables
real angle, dx_dxi, dy_dxi
real xcc1, ycc1, dx, dy, R1, dR, W, theta1, dtheta
!-------------------------------------------------------------------------
! Assign parts of the derived type to local variables
!  to keep the formulas cleaner
xcc1    = seg%xcc1
ycc1    = seg%ycc1
dx      = seg%dx
dy      = seg%dy
R1      = seg%R1
dR      = seg%dR
W       = seg%W
theta1 = seg%theta1
dtheta = seg%dtheta

! compute intermediate variables
Angle = theta1 + xi * dtheta

dx_dxi = dx + dR * Cos(Angle) - dtheta * Sin(Angle) * &
         (R1 + W * (eta - 0.5) + xi * dR)

dy_dxi = dx + dR * Sin(Angle) + dtheta * Cos(Angle) * &
         (R1 + W * (eta - 0.5) + xi * dR)
```

```

! Calculate metric coefficient
L_xi = sqrt( (dx_dxi ** 2 + dy_dxi ** 2) )

End Function F_L_xi
!=============================================================================

Function unmap_x(xi, eta, seg) Result( X )
!
use shared, only: CLSEG
implicit none
!   Arguments
real xi, eta
type(CLSEG) :: seg
!   Result
real X
!   Internal Variables
real xcc1, dx, R1, dR, W, theta1, dtheta
!------------------------------------------------------------------------
! Assign parts of the derived type to local variables
!  to keep the formulas cleaner
xcc1   = seg%xcc1
dx     = seg%dx
R1     = seg%R1
dR     = seg%dR
W      = seg%W
theta1 = seg%theta1
dtheta = seg%dtheta

! Compute the X coordinate
X = (xcc1 + xi * dx) + &
          (R1 + xi * dR + (eta - 0.5) * W) * Cos(theta1 + xi * dtheta)

end function unmap_x
!===============================================================================


Function unmap_y(xi, eta, seg) result( Y )

use shared, only: CLSEG
implicit none
real xi, eta
type(CLSEG) :: seg
!   Result
real Y
!   Internal Variables
real ycc1, dy, R1, dR, W, theta1, dtheta
!------------------------------------------------------------------------
! Assign parts of the derived type to local variables
!  to keep the formulas cleaner
ycc1   = seg%ycc1
dy     = seg%dy
R1     = seg%R1
dR     = seg%dR
```

```fortran
W      = seg%W
theta1 = seg%theta1
dtheta = seg%dtheta


Y = (ycc1 + xi * dy) + &
          (R1 + xi * dR + (eta - 0.5) * W) * Sin(theta1 + xi * dtheta)

end function unmap_y
!==============================================================================









!============================================================================
!   \\\\\\\\\                                       ///////////
                      END MODULE geom_funcs
!   /////////                                       \\\\\\\\\\\\
!============================================================================
```

Source File 8: ConvCoef.f95

```
! fortran_free_source


module ConvCoef

implicit none

contains


!   1. CONVEYANCE
!   2. FrictionSlope

!=====================================================================
!   \\\\\\\\\\  B E G I N     S U B R O U T I N E //////////
!   //////////      C O N V E Y A N C E          \\\\\\\\\\
!=====================================================================
!
!   Purpose:    This subroutine computes the conveyance coefficient
!               for a given cell face...look out, its fancy!
SUBROUTINE CONVEYANCE( face, sol, i, j, CC )
USE shared, ONLY: Sfw_old, Sfe_old, Sfs_old, Sfn_old,    &
                  Sfw_itr, Sfe_itr, Sfs_itr, Sfn_itr,    &
                  h_old, h_itr, wid, Z, K, n_mann,       &
                  b_pfc, lng, lng_south, lng_north,      &
                  h_pfc_min, area
implicit none
!VARIABLE DECLARATIONS
!   Arguments
character(5), intent( in )  :: face     ! Which face?
character(3), intent( in)   :: sol      ! Computed based on which solution?
integer, intent( in )       :: i, j     ! of which cell?
REAL, intent(out)           :: CC       ! the conveyance coefficient
!   Internal Variables
real   :: hp, hs ! the thickness in the pavement and on the surface
REAL :: distin, distout  ! size of cells for scaling purposes (will be length or width depeding on which direction we're going.
REAL :: fluxdist    ! the distance (size) of the cell face that the flux applies t
REAL :: hin, hout     ! thickness at CV center
REAL :: zin, zout     ! elevation at CV center
REAL :: head_at_face, Zface !HEAD and ELEVATION at the face
REAL, POINTER, DIMENSION(:,:) :: h  ! pointer to the thickness array
REAL, POINTER, DIMENSION(:,:) :: Sfw, Sfe, Sfs, Sfn ! points to magnitude of friction slope at compass face.
REAL :: Sf !the friction slope for the particular face that we're working with
logical :: error !make sure the result is reasonable
!---------------------------------------------------------------------
! Compute based on the old or iterative thickness?
if ( sol .EQ. 'old' ) then
    ! thickness array
    h => h_old
```

```
    ! friction slope arrays
    Sfw => Sfw_old
    Sfe => Sfe_old
    Sfs => Sfs_old
    Sfn => Sfn_old
elseif( sol .eq. 'itr' ) then
    h => h_itr
    Sfw => Sfw_itr
    Sfe => Sfe_itr
    Sfs => Sfs_itr
    Sfn => Sfn_itr
endif

! set internal/generic variables based on cell face
if ( face .EQ. 'west ' ) then
    distin = lng( i, j)
    distout= lng( i-1, j)
    hin = h(i,j)
    hout= h(i-1,j)
    zin = Z(i,j)
    zout= Z(i-1,j)
    fluxdist = wid(i,j)
    Sf = Sfw( i, j)
elseif( face .eq. 'east ' ) then
    distin = lng( i, j)
    distout= lng( i+1, j)
    hin = h(i,j)
    hout= h(i+1,j)
    zin = Z(i,j)
    zout= Z(i+1,j)
    fluxdist = wid(i,j)
    Sf = Sfe( i, j)
elseif( face .EQ. 'south' ) then
    distin = wid( i, j)
    distout= wid( i, j-1)
    hin = h(i,j)
    hout= h(i,j-1)
    zin = Z(i,j)
    zout= Z(i,j-1)
    fluxdist = lng_south(i,j)
    Sf = Sfs(i,j)
elseif( face .EQ. 'north' ) then
    distin = wid( i, j)
    distout= wid( i, j+1)
    hin = h(i,j)
    hout= h(i,j+1)
    zin = Z(i,j)
    zout= Z(i,j+1)
    fluxdist = lng_north(i,j)
    Sf = Sfn(i,j)
endif
!Compute the total head at the cell face
head_at_face = ( (hin+zin)*distout + (hout+zout)*distin )   &
                         / ( distin + distout)
```

```
!Elevation at the cell face
Zface        = ( zin*distout + zout*distin ) / ( distin + distout)
!compute the thicknesses
hp = MIN ( b_pfc, head_at_face - Zface        )
hs = MAX ( 0.   , head_at_face - Zface - b_pfc)


!Force hp to stay positive
if( hp .LT. 0.0 ) then
    hp = TINY(h_pfc_min)
end if

! Compute the Conveyance coefficient
! would really like to just one statement to calc the conv coef
!   but sqrt(Sf) sometimes gives problems, even when there is no
!   sheet flow, so this if block hopefully avoids the problem

if( hs .GT. 0.) then
    !Sheet flow occurs and compute CC as usual
    CC = ( K * hp + 1./n_mann*hs**(5./3.)/sqrt(Sf) )  *  &
            ( 2.*fluxdist / ( distout + distin ) ) / Area(i,j)
else
    !Sheet flow does not occur and CC only depends on subsurface
    CC = ( K * hp    )  *  &
            ( 2.*fluxdist / ( distout + distin ) ) / Area(i,j)
end if


! ERROR CHECKING FOR CONVEYANCE COEFS
if( CC .GT. HUGE(CC) .OR. CC .LT. -HUGE(CC) ) then
    error = .true.
else
    error = .false.
endif

!Output the parts of the calculation if the error is true
if( error .eqv. .true. ) then
    write(*,*) 'Problem with conveyance coefficient!'
    print *, 'i = ', i, ' j = ', j, ' Face = ', face, ' Soln = ', sol
    print *, '       K = ', K
    print *, '      hp = ', hp
    print *, '  n_mann = ', n_mann
    print *, '      hs = ', hs
    print *, '      Sf = ', Sf
    print *, 'fluxdist = ', fluxdist
    print *, ' distout = ', distout
    print *, '  distin = ', distin
    print *, '    Area = ', Area(i,j)
    print *, '      CC = ', CC
    write(*,*) 'Stopping Program'
    STOP
endif
```

```
! print the inputs for checking
!       print *, ''
!       print *, 'i = ', i, ' j = ', j, ' Face = ', face, ' Soln = ', sol
!       print *, '       K = ', K
!       print *, '      hp = ', hp
!       print *, '  n_mann = ', n_mann
!       print *, '      hs = ', hs
!       print *, '      Sf = ', Sf
!       print *, 'fluxdist = ', fluxdist
!       print *, ' distout = ', distout
!       print *, '  distin = ', distin
!       print *, '    Area = ', Area(i,j)
!       print *, '      CC = ', CC
!





END subroutine conveyance

!======================================================================
!   \\\\\\\\\\  E N D        S U B R O U T I N E //////////
!   //////////     C O N V E Y A N C E          \\\\\\\\\\
!======================================================================

!======================================================================
!   \\\\\\\\\\  B E G I N    S U B R O U T I N E //////////
!   //////////    F R I C T I O N   S L O P E    \\\\\\\\\\
!======================================================================
!   Purpose:    This subroutine computes the magnitude of the friction
!               slope at the cell faces.
!               The arguments specifcy whether to use the OLD or ITR
!               solution array in the calculations and the arrays for storing the results.
!
!
!       ---x---|---x---     Key:  * is CV Center
!       |      |      |           x normal component of friction slope
!       |  *   O   *  |             computed here by central difference
!       |      |      |           O the four normal components are
!       ---x---|---x---             tangent here and so are averaged
!PERFCODE   PERFCODE.f95    Main program

SUBROUTINE FrictionSlope( sol, Sfw, Sfe, Sfs, Sfn )
use SHARED, only: h_old, h_itr, Z, lng, wid, imax, jmax

use outputs, only: write_flipped_matrix
use utilities, only: F_PythagSum, F_Extrapolate
!----------------------------------------------------------------------
!VARIABLE DECLARATIONS
implicit none
!   Arguments
```

```fortran
character(3), intent( in ) :: sol
REAL, DIMENSION(imax,jmax), intent(out), optional :: Sfw, Sfe, Sfs, Sfn
!   Internal Variables
REAL, DIMENSION(imax, jmax) :: HD      ! Total HEAD at cell centers
REAL, DIMENSION(:,:), pointer :: h     ! Pointer to array of thicknesses
REAL, DIMENSION(imax,jmax) :: Sf_norm_west, Sf_norm_east, Sf_norm_south, Sf_norm_north
REAL, DIMENSION(imax,jmax) :: Sf_tan_west, Sf_tan_east, Sf_tan_south, Sf_tan_north
REAL, ALLOCATABLE, DIMENSION(:,:), target :: h_dry  ! for computing pavement slopes
INTEGER :: i, j         !array idices

!------------------------------------------------------------------------------
! choose which thickness array to use for estimating the friction slope
if ( sol .EQ. 'old' ) then
    h => h_old
elseif( sol .eq. 'itr' ) then
    h => h_itr
elseif( sol .eq. 'dry' ) then
    allocate( h_dry( imax, jmax ) )
    h_dry = 0.0
    h => h_dry
endif

! compute the total head
HD = h + z !total head is thickness plus elevation

!initialize arrays to zero
!       Sf_norm_west = 0.0
!       Sf_norm_east = 0.0
!       Sf_norm_south= 0.0
!       Sf_norm_north= 0.0
!       Sf_tan_west = 0.0
!       Sf_tan_east = 0.0
!       Sf_tan_south= 0.0
!       Sf_tan_north= 0.0
!
!-------------------------------------------------------------------------
!   C O M P O N E N T   N O R M A L   T O   E A C H   F A C E
!-------------------------------------------------------------------------

!

! WEST
do j = 1, jmax
    ! Domain interior, by central differences
    do i = 2, imax
        Sf_norm_west(i,j) = ( ( HD(i,j) - HD(i-1,j)  ) / 0.5 / ( lng(i-1,j) + lng(i,j) ) )
    end do
    ! Western boundary of domain by extrapolation
    i = 1
    Sf_norm_west(i,j) = F_Extrapolate( 0.,                                          &
                                 lng(i,j)          , Sf_norm_west(i+1,j),   &
```

```
                                    lng(i,j)+lng(i+1,j), Sf_norm_west(i+2,j)  )
end do

!EAST
do j = 1, jmax
    ! Domain interior, by central differences
    do i = 1, imax - 1
        Sf_norm_east(i,j) = (  ( HD(i+1,j) - HD(i,j) ) / 0.5 / ( lng(i+1,j) + lng(i,j) ) )
    end do
    ! Eastern boundary of domain by extrapolation
    i = imax
    Sf_norm_east(i,j) = F_Extrapolate( 0.,                                  &
                                    lng(i,j)           , Sf_norm_east(i-1,j), &
                                    lng(i,j)+lng(i-1,j), Sf_norm_east(i-2,j)  )
end do


!SOUTH
do i = 1, imax
    ! Domain interior, by central differences
    do j = 2, jmax
        Sf_norm_south(i,j) = ( ( HD(i,j-1) - HD(i,j) ) / 0.5 / ( wid(i,j-1) + wid(i,j) ) )
    end do
    ! Southern boundary of domain by extrapolation
    j = 1
    Sf_norm_south(i,j) = F_Extrapolate( 0.,                                  &
                                    wid(i,j)           , Sf_norm_south(i,j+1), &
                                    wid(i,j)+wid(i,j+1), Sf_norm_south(i,j+2)  )

end do

!NORTH
do i = 1, imax
    ! Domain interior, by central differences
    do j = 1, jmax - 1
        Sf_norm_north(i,j) = (  ( HD(i,j) - HD(i,j+1) ) / 0.5 / ( wid(i,j+1) + wid(i,j) ) )
    end do
    ! Northern oundary of domain by extrapolation
    j = jmax
    Sf_norm_north(i,j) = F_Extrapolate( 0.,                                  &
                                    wid(i,j)           , Sf_norm_north(i,j-1), &
                                    wid(i,j)+wid(i,j-1), Sf_norm_north(i,j-2)  )
end do

!------------------------------------------------------------------------
!   C O M P O N E N T   T A N G E N T   T O   E A C H   F A C E
!       A N D   M A G N I T U D E  A T   E A C H   F A C E
!------------------------------------------------------------------------
! component of friction slope that is TANGENT to each cell face
! computed by averaging the four nearest locations where the
! component is normal to a face.
```

```
!
!WEST
do j = 1, jmax
    do i = 2, imax
        Sf_tan_west(i,j) =  ( (Sf_norm_north(i,j) + Sf_norm_south(i,j))*lng(i-1,j)   &
                                   +(Sf_norm_north(i-1,j) + Sf_norm_south(i-1,j))*lng(i,j) ) &
                                  / ( 2. * ( lng(i,j) + lng(i-1, j) ) )
        Sfw(i,j) = F_PythagSum( Sf_norm_west (i,j), Sf_tan_west (i,j) )
    end do
end do

!EAST
do j = 1, jmax
    do i = 1, imax - 1
        Sf_tan_east(i,j) =  ( (Sf_norm_north(i,j) + Sf_norm_south(i,j))*lng(i+1,j)    &
                                   +(Sf_norm_north(i+1,j) + Sf_norm_south(i+1,j))*lng(i,j) ) &
                                  / ( 2. * ( lng(i,j) + lng(i+1, j) ) )
        Sfe(i,j) = F_PythagSum( Sf_norm_east (i,j), Sf_tan_east (i,j) )
    end do
end do

!SOUTH
do i = 1, imax
    do j = 2, jmax
        Sf_tan_south(i,j) =  ( ( Sf_norm_east(i,j) + Sf_norm_west(i,j) ) * wid(i,j-1)   &
                                    +( Sf_norm_east(i,j-1)+Sf_norm_west(i,j-1))* wid(i,j  ) ) &
                                  / ( 2. * ( wid(i,j) + wid(i,j-1) ) )
        Sfs(i,j) = F_PythagSum( Sf_norm_south(i,j), Sf_tan_south(i,j) )
    end do
end do

!NORTH
do i = 1, imax
    do j = 1, jmax - 1
        Sf_tan_north(i,j) = ( ( Sf_norm_east(i,j) + Sf_norm_west(i,j) ) * wid(i,j+1)   &
                                   +( Sf_norm_east(i,j+1)+Sf_norm_west(i,j+1))* wid(i,j  ) ) &
                                  / ( 2. * ( wid(i,j) + wid(i,j+1) ) )
        Sfn(i,j) = F_PythagSum( Sf_norm_north(i,j), Sf_tan_north(i,j) )
    end do
end do


! deallocate space for h_dry
if( sol .eq. 'dry' ) then
    deallocate( h_dry )
endif
```

```fortran




!
!        i = 50; j = 51
!       write(100,*) 'i,j=', i, j
!       write(100,*) 'Sf_norm_east', Sf_norm_east(i,j)
!       write(100,*) 'Sf_tan_east', Sf_tan_east(i,j)
!       write(100,*) 'HD(i,j)', HD(i,j)
!       write(100,*) 'HD(i+1,j)', HD(i+1,j)
!       write(100,*) 'HD(i,j+1)', HD(i,j+1)
!       write(100,*) 'HD(i,j-1)', HD(i,j-1)
!       write(100,*) 'Sf_norm_north(i,j)', Sf_norm_north(i,j)
!       write(100,*) 'Sf_norm_north(i+1,j)', Sf_norm_north(i+1,j)
!       write(100,*) 'Sf_norm_south(i,j)', Sf_norm_south(i,j)
!       write(100,*) 'Sf_norm_south(i+1,j)', Sf_norm_south(i+1,j)
!
!

end subroutine FrictionSlope

!======================================================================
!   \\\\\\\\\\     E N D     S U B R O U T I N E //////////
!   //////////     F R I C T I O N   S L O P E   \\\\\\\\\\
!======================================================================





end module ConvCoef

```

Source File 9: GridGen.f95

```
! fortran_free_source
!
!  This module is part of PERFCODE, written by Bradley J. Eck.
!
!
!
! This module contains subroutines related to generating the computational
!   grid.  The subroutines are:
!       1. Generate_Grid  Computes the length, width, and area
!                           of each grid cell.
!       2. Assign_Elevations  Gives an elevation to each grid cell center
!
!
!
!   External code required by this module includes the modules

!==============================================================================
!   \\\\\\\\\\                                        //////////
                          MODULE GridGen
!   //////////                                        \\\\\\\\\\
                          implicit none

                          contains
!==============================================================================

!   1. GENERATE_GRID
!   2. SET_ELEVATIONS
!==============================================================================
!   \\\\\\\\\                                                     ///////////
                  subroutine Generate_Grid( prelim_dx, prelim_dy )
!   /////////                                                    \\\\\\\\\\\\
!==============================================================================
!   Purpose:    Read entries of a file into a derived data type
!               and print the entries to the screen

USE shared,     ONLY: seg, area, lng, lng_south, lng_north, wid,    &
                      imax, jmax, vmax, CV_Info, astat, gridcell
USE outputs,    ONLY: WRITE_MATRIX, WRITE_FLIPPED_MATRIX
USE geom_funcs, ONLY: F_L_xi, UNMAP_X, UNMAP_Y
USE pfc2Dfuns,  ONLY: F_LinearIndex
!--------------------------------------------------------------------------------
implicit none

! VARIABLE DECLARATIONS
!   Agruments
real, intent(in) :: prelim_dx, prelim_dy
!   Internal Variables
INTEGER, parameter :: max_rec=144 ! maximum allowable number of rainfall records
integer :: N_seg
CHARACTER(len=20) infile     ! the file name to read parameters from
integer :: i, j, v      ! looping variables
```

```fortran

character :: TRASH

real :: xi, eta, X, Y
real :: eta_s, eta_n

integer :: N_xi, N_eta, Seg_num


! for writing border info to file
integer :: factor = 5          ! how many times more border points than CVs?
integer :: ijf ! dummy variable for i or j times the factor
real, allocatable, dimension(:) :: NX, NY, SX, SY, EX, EY, WX, WY

! for writing grid to a file
integer :: res
real, allocatable, dimension(:,:) :: X_gl_long, Y_gl_long, X_gl_tran, Y_gl_tran


!----------------------------------------------------------------------------
! read in the geometry data

! default value for input file
infile = 'CL_Segments.dat'

! Prompt the user for the input file
WRITE(*,*) 'Enter filename or press / for ', infile
READ(*,*) infile

OPEN( UNIT=8,   FILE = infile, ACTION = 'read', STATUS = 'old' )

! Rainfall Rate
read( unit=8, fmt = * ) N_seg


!if ( nrr .gt. size ( rain_time ) ) then
!    print *, 'Too many rainfall records--increase array size and recompile'
!else

read( unit=8, fmt = * ) trash



allocate( seg( N_seg ) )

do i = 1, N_seg
    READ( unit = 8 , fmt = * ) j, seg(j)%xcc1, seg(j)%ycc1, seg(j)%dx, &
                                  seg(j)%dy,   seg(j)%R1,   seg(j)%dR, &
                                  seg(j)%W,    seg(j)%theta1, seg(j)%dtheta
        ! seg(j) = CLSEG(xcc1, ycc1, dx, dy, R1, dR, W, theta1, dtheta, 0. ) ! use 0. as placeholder for length
end do

close( 8 )
```

```
! estimate the length of each segment by evaluating the metric coefficients
do i = 1, N_seg
    seg(i)%arclen = F_L_xi( 0.5, 0.5, seg(i) ) * 1.0   ! L_xi * Delta_xi
end do


print *, 'Segment    ArcLength'
do i = 1, N_seg
    print *, i, seg(i)%arclen
end do

!total length
print *, '  Total Length: ', sum( seg(:)%arclen )

! average length
print *, 'Average Length: ', sum( seg(:)%arclen ) / real( N_seg )


! Number of elements per segment
!nint = nearest integer
N_xi = nint(  sum( seg(:)%arclen)  / real( N_seg ) / prelim_dx )
N_eta= nint(  sum( seg(:)%W     )  / real( N_seg ) / prelim_dy )
print *, 'N_xi = ', N_xi, '    N_eta = ', N_eta

! size of computational domain
imax = N_xi * N_seg
jmax = N_eta
vmax = imax * jmax


!-------------------------------------------------------
! ALLOCATE ARRAYS

allocate(    lng( imax, jmax),  STAT = astat( 1) )
allocate(    wid( imax, jmax),  STAT = astat( 2) )
allocate(   area( imax, jmax),  STAT = astat( 3) )


allocate(   lng_south( imax, jmax ),    STAT = astat( 6) )
allocate(   lng_north( imax, jmax ),    STAT = astat( 7) )

allocate ( CV_Info( vmax ), STAT = astat(17) )
!--------------------------------------------------------------

! Now compute length, width, and area of each cell
! should confirm that all widths are the same

do j = 1, jmax
!    print *, 'j = ', j
!    print *, 'i   Segment'
    do i = 1, imax
        ! Determine which segment we're in
```

```fortran
        ! the intrinic function CEILING is like ROUNDUP in excel
        Seg_Num = ceiling( real(i) / real( imax ) * real( N_seg ) )
        ! Compute values of xi for the cell that we're in
        if ( i .LE. N_xi ) then
                xi =   i * 1. / N_xi - 1. / N_xi / 2.
        else
                xi = ( i - ( Seg_Num - 1 ) * N_xi ) * 1. / N_xi - 1. / N_xi / 2.
        end if
        ! value of eta
        eta = 1. / N_eta * j  -  1. / N_eta / 2.
        ! Physical Coordinates of CV
        X = unmap_x( xi = xi, eta = eta, seg = seg( Seg_Num ) )
        Y = unmap_y( xi = xi, eta = eta, seg = seg( Seg_Num ) )
        ! store the summary information for this cell
        v = F_LinearIndex( i, j, jmax )
        CV_Info(v) = gridcell( i, j, Seg_Num, xi, eta, X, Y )
        ! now compute the quantities of interest for each cell.
!        print *, i, Seg_Num
        lng (i,j) = F_L_xi( xi, eta, seg( Seg_Num ) ) * 1. / N_xi
        wid (i,j) = seg( Seg_Num)%W / N_eta
        area(i,j) = lng(i,j) * wid(i,j)
        ! compute lengths for north and south faces of cell
        !   south
        eta_s = eta - 1./N_eta * 1./2.
        lng_south(i,j) = F_L_xi( xi, eta_s, seg( Seg_Num) ) * 1./N_xi
        !   north
        eta_n = eta + 1./N_eta * 1./2.
        lng_north(i,j) = F_L_xi( xi, eta_n, seg( Seg_Num) ) * 1./N_xi
    end do
end do


! Output the arrays to respective files
!  subroutine write_flipped_matrix( array, imax, jmax, outputfile )
call write_flipped_matrix(  lng, imax, jmax, 'length.csv' )

call write_flipped_matrix(  wid, imax, jmax,  'width.csv' )

call write_flipped_matrix( area, imax, jmax,   'area.csv' )

call write_flipped_matrix( lng_south, imax, jmax, 'lng_south.csv')

!Write the CV info file
open( unit=40, file = 'CV_info.csv', status = 'REPLACE')
write(40,*) 'v,i,j,segment,xi,eta,X,Y,'
do v = 1, vmax
    WRITE(40,44) v, CV_info(v)
end do

44  format( 4(I, ','), 4(E, ','))


!-----------------------------------------------------------------------------
!   >>>>>>>>>>     W R I T E   B O U N D A R Y   C O O R D S    <<<<<<<<<<
```

```
!---------------------------------------------------------------------------
! was going to make this a subroutine, but it seemed easier to add it here

! Allocate arrays
ijf = imax * factor
allocate( NX( ijf) )
allocate( NY( ijf) )
allocate( SX( ijf) )
allocate( SY( ijf) )

ijf = jmax * factor + 1
allocate( EX( ijf ) )
allocate( EY( ijf ) )
allocate( WX( ijf ) )
allocate( WY( ijf ) )

! NORTH and SOUTH borders
!re-calc N_xi so as not to change the following formula
N_xi = N_xi * factor
do i = 1, imax * factor
    Seg_Num = ceiling( real(i) / real( imax * factor ) * real( N_seg ) )
    ! Compute values of xi
    if ( i .LE. N_xi ) then
            xi =   i * 1. / N_xi - 1. / N_xi / 2.
    else
            xi = ( i - ( Seg_Num - 1 ) * N_xi ) * 1. / N_xi - 1. / N_xi / 2.
    end if
    ! NORTH -- Physical Coordinates on border
    eta = 1.0
    NX(i) = unmap_x( xi = xi, eta = eta, seg = seg( Seg_Num ) )
    NY(i) = unmap_y( xi = xi, eta = eta, seg = seg( Seg_Num ) )
    ! SOUTH -- Physical Coordinates on border
    eta = 0.0
    SX(i) = unmap_x( xi = xi, eta = eta, seg = seg( Seg_Num ) )
    SY(i) = unmap_y( xi = xi, eta = eta, seg = seg( Seg_Num ) )
end do

! EAST and WEST borders
do j = 1, jmax * factor + 1
    eta = ( j - 1. ) / ( jmax * factor )
    ! WEST
    xi = 0.0
    WX(j) = unmap_x( xi = xi, eta = eta, seg = seg( 1 ) )
    WY(j) = unmap_y( xi = xi, eta = eta, seg = seg( 1 ) )
    ! EAST
    xi = 1.0
    EX(j) = unmap_x( xi = xi, eta = eta, seg = seg( N_seg ) )
    EY(j) = unmap_Y( xi = xi, eta = eta, seg = seg( N_seg ) )
end do

! Write the borders to files
open( unit = 40, file = 'NS_borders.csv', status = 'REPLACE')
WRITE(40, *) 'NX, NY, SX, SY,'
do i = 1, imax * factor
```

```
    write(40, 4) NX(i), NY(i), SX(i), SY(i)
end do
close(40)

open( unit = 41, file = 'EW_borders.csv', status = 'REPLACE')
write( 41, * ) 'EX, EY, WX, WY,'
do j = 1, jmax * factor + 1
    write(41, 4) EX(j), EY(j), WX(j), WY(j)
end do
close(41)

4 format( 4 (E, ',') )


!--------------------------------------------------------------------------
!   >>>>>>>>>   WRITE GRID FOR PLOTTING                      <<<<<<<<<
!--------------------------------------------------------------------------
!
! envisioning the use of R's MATPLOT command, write a matrix of X coords
! and a matrix of Y coords
! we plot a bunch of points

res = 4   !parameter to control the resolution of of the plotting
           ! this is analogous to the variable 'factor'  how many
           ! points do you want per CV?

allocate( X_gl_long( imax * res, jmax + 1 ) )
allocate( Y_gl_long( imax * res, jmax + 1 ) )

! redefine N_xi to reflect the amplified number of points for the grid
N_xi = imax / N_seg * res

! LONGITUDINAL GRID LINES
do j = 1, jmax + 1
    !value of eta is constant for each j
    eta = ( j - 1. ) / jmax
    do i = 1,  imax * res
        ! figure out which segment we're in
        Seg_Num = ceiling( real(i) / real( imax * res ) * real( N_seg ) )
        ! Compute values of xi
        if ( i .LE. N_xi ) then
            xi =  ( i - 1. ) / ( N_xi )
        else
            xi = ( i - ( ( Seg_Num - 1. ) * N_xi ) ) * 1. / N_xi
        end if
        X_gl_long(i, j)  = unmap_x( xi = xi, eta = eta, seg = seg( Seg_Num ) )
        Y_gl_long(i, j)  = unmap_y( xi = xi, eta = eta, seg = seg( Seg_Num ) )
    end do
end do


call WRITE_MATRIX( X_gl_long, imax * res, jmax + 1, 'X_gl_long.csv' )
call WRITE_MATRIX( Y_gl_long, imax * res, jmax + 1, 'Y_gl_long.csv' )

```

```fortran

!TRANSVERSE GRID LINES (these are just straight and so only require two points)
allocate( X_gl_tran( 2, imax + 1 ) )
allocate( Y_gl_tran( 2, imax + 1 ) )

! put N_xi back to what its proper value
N_xi = imax / N_seg

do j = 1, 2
    eta = j - 1.
    do i = 1, imax + 1
        ! figure out which segment we're in
        Seg_Num = ceiling( real(i) / real( imax + 1 ) * real( N_seg ) )
        ! Compute values of xi
        if ( i .LE. N_xi ) then
            xi =  ( i - 1. ) / ( N_xi )
        else
            xi = ( i - 1.  - ( Seg_Num - 1. ) * N_xi ) / N_xi
        end if
        X_gl_tran(j, i)  = unmap_x( xi = xi, eta = eta, seg = seg( Seg_Num ) )
        Y_gl_tran(j, i)  = unmap_y( xi = xi, eta = eta, seg = seg( Seg_Num ) )
    end do
end do


call WRITE_MATRIX( X_gl_tran, 2, imax + 1 , 'X_gl_tran.csv' )
call WRITE_MATRIX( Y_gl_tran, 2, imax + 1, 'Y_gl_tran.csv' )




!------------------------------------------------------------------------------
! Deallocate needed?

deallocate( NX, NY, SX, SY, EX, EY, WX, WY )
deallocate( X_gl_long, Y_gl_long, X_gl_tran, Y_gl_tran )

!==============================================================================
!   \\\\\\\\\                                       ///////////
                    end subroutine Generate_Grid
!   /////////                                       \\\\\\\\\\\\
!==============================================================================



!==============================================================================
!   \\\\\\\\\                                       ///////////
                   subroutine Set_Elevations( )
!   /////////                                       \\\\\\\\\\\\
!=============================================================================
!   Purpose:    Gives an elevation to each node of the grid
!
! VARIABLE DECLARATIONS
!   Arguments
```

```
!
!
!   Internal Variables
!
!
!   Assign from a cross section:  read in the cross section
!
!
!
!
!
!

use SHARED, only: Z, imax, jmax, lng_south, CV_Info, seg,  &
                  nr_cs, slope_cs, wid_cs, eta_cs, Z_cs, nr_lp, dist_lp, Z_lp

use utilities, only: F_Linterp
use outputs,   only: write_flipped_matrix
use pfc2dfuns,  only: F_LinearIndex
implicit none

!       !CROSS SECTION  ( Transverse direction)
!       ! input file
!       integer :: nr_cs
!       REAL ::  slope_cs(10), wid_cs(10)
!
!       ! derived values
!       real, dimension( 11 ) :: eta_cs=0., Z_cs=0.
!
!       !LONGITUDINAL PROFILE
!       integer :: nr_lp
!       real, dimension(100) :: dist_lp, Z_lp
!

CHARACTER(20) infile    ! the file name to read from
CHARACTER(3) dummy_line
INTEGER :: i, j, v      ! looping variables


!CROSS Section Input File

real :: tot_wid
real :: CL_wid
! Generating elevations
real :: dist_along_lp, Z_eta_0, Z_add



!------------------------------------------------------------------------
! C R O S S   S E C T I O N
!------------------------------------------------------------------------
! default value for input file
infile = 'CrossSection.dat'

```

```
! Prompt the user for the input file
WRITE(*,*) 'Enter filename or / for ', infile
READ(*,*) infile

! Read the file
OPEN( UNIT=8,   FILE = infile, ACTION = 'read', STATUS = 'old' )

!Cross Seection geometry
read( unit=8, fmt = * ) dummy_line
READ( unit=8, fmt = * ) nr_cs
read( unit=8, fmt = * ) dummy_line
if( nr_cs .gt. size( slope_cs) ) then
    print *, 'SET_ELEVATIONS: Too many records in', infile, &
             'increase array size and recompile'

else
   do i = 1, nr_cs
       READ( unit=8, fmt = * ) j, slope_cs(j), wid_cs(j)
   end do
end if

! Close the input file
close(8)

! Echo to screen
PRINT *, 'CROSS SECTION INPUTS '
WRITE(*,*)  ' Segment    Slope       Width  '
WRITE(*,*)  '=================================='
!            '----|----|----|----|----|----|----|----|----|----|----|----|
!                 5    10   15   20   25   30   35   40   45   50   55   60
DO j = 1, nr_cs
       WRITE(*,10) j, slope_cs(j), wid_cs(j)
END DO


! Compute eta and elevation from widths and slopes
!    Given:  widths and slopes  slope_cs   wid_cs, nr_cs
!    Find : Z vs eta

tot_wid = sum( wid_cs(1:nr_cs) )

CL_wid = seg(1) % W

! Check tot_wid for consistency with the width given in CL segments
if(    abs(  tot_wid  -  CL_wid )    .GE. 1.e-3 ) then
    write( *,*) ' Cross Section Width =', tot_wid
    write( *,*) ' Centerline Width = ', CL_wid
    write(*,*) ' SET_ELEVATIONS: Total width specified in '//infile//&
               &'is inconsistent with the centerine geometry...Stopping Program'

    STOP

end if

```

```fortran


eta_cs(1) = 0.
  z_cs(1) = 0.   !<---dummy value here, elevations are made relative to eta=0

! Compute etas and elevations
do i = 2, nr_cs + 1
    eta_cs(i) = eta_cs(i-1) +   wid_cs(i-1) / tot_wid
      Z_cs(i) =   z_cs(i-1) - slope_cs(i-1) * wid_cs(i-1)
end do

! print the results to confirm
print *, ' CROSS SECTION POINTS '
WRITE(*,*) ' Point        Eta       Elevation '
WRITE(*,*)  '=================================='
do i = 1, nr_cs + 1
    write(*,10) i, eta_cs(i), Z_cs(i)
end do


!-----------------------------------------------------------------------
!     L O N G I T D I N A L   P R O F I L E
!-----------------------------------------------------------------------
! default value for input file
infile = 'LongProfile.dat'

! Prompt the user for the input file
WRITE(*,*) ''
WRITE(*,*) 'Enter filename or press / for ', infile
READ(*,*) infile

! Read the file
OPEN( UNIT=8,   FILE = infile, ACTION = 'read', STATUS = 'old' )

read( unit=8, fmt = * ) dummy_line
read( unit=8, fmt = * ) nr_lp    !number of rows to define cross section
read( unit=8, fmt = * ) dummy_line

if ( nr_lp .gt. size ( dist_lp ) ) then
    print *, 'SET_ELEVATIONS: Too many records in', infile, &
               'increase array size and recompile'
else
    do i = 1, nr_lp
        READ( unit = 8 , fmt = * ) j, dist_lp(j), Z_lp(j)
    end do
end if


close( 8 )

! Echo to screen
PRINT *, 'LONGITUDINAL PROFILE '
WRITE(*,*)  ' Point    Distance     Elevation  '
WRITE(*,*)  '=================================='
```

```
!             '----|----|----|----|----|----|----|----|----|----|----|----|
!                  5    10   15   20   25   30   35   40   45   50   55   60
DO i = 1, nr_lp
      WRITE(*,10) i, dist_lp(i), Z_lp(i)
END DO

write(*,*) ''

!--------------------------------------------------------------------------------
! I N T E R P O L A T E   E L E V A T I O N S
!--------------------------------------------------------------------------------
allocate(         Z( imax, jmax ) );           Z = 0.0  !,  STAT = astat( 4)


do i = 1, imax
    ! Compute distance along longitudinal profile at eta = 0
    dist_along_lp = sum( lng_south(1:i,1 ) ) - lng_south(i,1)/2.
    ! Compute elevation at eta = 0 for this column of grid cells
    Z_eta_0 = F_Linterp(      x = dist_along_lp  ,    &
                        Known_X = dist_lp        ,    &
                        Known_Y =    Z_lp        ,    &
                              n =   nr_lp                )
    do j = 1, jmax
        v = F_LinearIndex( i, j , jmax )
        Z_add  =       F_Linterp(       X = CV_Info(v)%eta , &
                                  known_X = eta_cs         , &
                                  known_Y = Z_cs           , &
                                        n = (nr_cs + 1)        )
        Z(i,j) = Z_eta_0  + Z_add
    end do
end do


! Output matrix of cell elevations
CALL WRITE_FLIPPED_MATRIX( Z, imax, jmax, 'Z.csv')

!--------------------------------------------------------------------------------
! F O R M A T   S T A T E M E N T S
!--------------------------------------------------------------------------------

10     FORMAT('   ', ( i3, '    '),  ( F10.3, '  ') , F10.6, i   )

!==============================================================================
!   \\\\\\\\\                                      ///////////
                  end subroutine Set_Elevations
!   /////////                                      \\\\\\\\\\\\
!==============================================================================

!==============================================================================
!   \\\\\\\\\                                      ///////////
                       END MODULE GridGen
!   /////////                                      \\\\\\\\\\\\
!==============================================================================
```

Source File 10: solvers.f95

```fortran
! fortran_free_source
!
!  This module contains subroutines for a few linear solvers
!=========================================================================
!   \\\\\\\\\\                                       //////////
                          MODULE solvers
!   //////////                                       \\\\\\\\\\
                          implicit none

                          contains
!=========================================================================
!   Subroutines related to solving linear systems:
!   1. DIAGDOM_PENTA checks for diagonal dominance
!        given the bands of a penta-diagonal matrix
!   2. GAUSS_SEIDEL_PENTA uses the Gauss-Seidel method
!       for iterative solution of a penta-diagonal system
!       of linear equations.
!   3. THOMAS uses the tri-diagonal matrix algorithm to solve
!        a tri-diagonal linear system



!===================================================================
!   \\\\\\\\\\      B E G I N    S U B R O U T I N E ///////////
!   //////////         D I A G D O M _ P E N T A     \\\\\\\\\\\
!===================================================================
!
!   Purpose:    Checks to see if a penta-diagonal matrix is
!               diagonally dominant.  Knowing this helps select
!               a solver.  The routine operates only on the bands
!               of the coefficent matrix.
subroutine diagdom_penta( A, B, C, D, E, n, LB, UB, diagdom)
!  A,B -- lower bands of the penta-diagonal matrix
!   C  -- main diagonal
!  D,E -- upper banks of the penta-diagonal matrix
!   n  -- number of unknowns (size of system)
!  LB  -- lower bandwidth
!  UB  -- upper bandwidth
! tolit-- iteration tolerence
! diagdom-- a logical that stores the result.
!-------------------------------------------------------------------
! VARIABLE DECLARATIONS

! Arguments
integer, intent(in)  :: n, LB, UB
real, intent( in )   :: A(n), B(n), C(n), D(n), E(n)
logical, intent(out) :: diagdom
! Internal Variables
integer              :: k
real                 :: T1, T2, T4, T5
real                 :: tot
```

```fortran


!----------------------------------------------------------------

! set the logical to true, the following loop changes it if
! a row is not diagonally dominant.
diagdom = .true.

do k = 1, n
    ! compute the magnitude of each term in the row of the matrix
    if( k-LB .LT. 1 ) then; T1 = 0. ; else; T1 = abs( A(k) ); endif
    if( k    .EQ. 1 ) then; T2 = 0. ; else; T2 = abs( B(k) ); endif
    if( k    .EQ. n ) then; T4 = 0. ; else; T4 = abs( D(k) ); endif
    if( k+UB .GT. n ) then; T5 = 0. ; else; T5 = abs( E(k) ); endif
    ! Test for diagonal dominance
    tot = T1 + T2 + T4 + T5
    if( tot .GT. abs( C(k) ) ) then
        write(*,*) 'Row ', k, 'of the matrix is not diagonally dominant'
        diagdom = .false.
    endif
enddo
!------------------------------------------------------------------
end subroutine diagdom_penta
!=====================================================================
!   \\\\\\\\\\      E N D      S U B R O U T I N E ///////////
!   //////////        D I A G D O M _ P E N T A    \\\\\\\\\\\
!=====================================================================

!=====================================================================
!   \\\\\\\\\\      B E G I N   S U B R O U T I N E ///////////
!   //////////  G A U S S _ S E I D E L _ P E N T A  \\\\\\\\\\\
!=====================================================================

! CAUTION -- This routine DOES NOT check convergence criteria
!            so it possible to converge to the wrong answer.


subroutine gauss_seidel_penta( A, B, C, D, E, F, n , LB, UB, &
                               tolit, maxit, Xold, Xnew, dev, numits )
!  A,B -- lower bands of the penta-diagonal matrix
!   C  -- main diagonal
!  D,E -- upper banks of the penta-diagonal matrix
!   F  -- right hand side (force vector) of linear system
!   n  -- number of unknowns (size of system)
!  LB  -- lower bandwidth
!  UB  -- upper bandwidth
! tolit-- iteration tolerence
! maxit-- maximum number of iterations allowed
! Xold -- initial guess
! Xnew -- converged solution
! dev  -- device for outputting information from the solver
!numits-- number of iterations required to converge
!----------------------------------------------------------------------
!DECLARATIONS
```

```
use utilities, only: F_L2_NORM
!arguments
integer, intent(IN) :: n, LB, UB
real, intent(in )   :: A(n), B(n), C(n), D(n), E(n), F(n)
real, intent(in )   :: tolit
integer, intent(in) :: maxit
real, intent(in)    :: Xold(n)
real, intent(out)   :: Xnew(n)
integer, intent(in) :: dev
integer, intent(out):: numits  ! number of iterations required
! internal variables
integer :: k     ! array index
integer :: m     ! iteration index
real :: T1, T2, T4, T5  ! Terms in the equation
real :: relchng(n)  !relative change between iterations
real :: Xtmp(n)     !temporary array to store the progressive solutions

!-------------------------------------------------------------------------

!store the starting guess in the temporary array
Xtmp = Xold

! Perform the iterative solution
do m = 1, maxit
!    write(*,*) ' iteration Number = ', m
!    WRITe(*,*) 'Row, T1, T2, T4, T5, Xnew'
    do k = 1, n
        ! compute terms in the expression using if statements to
        ! sort out which terms apply based on the indices
        if( k-LB .LT. 1 ) then; T1 = 0. ; else; T1 = A(k)*Xnew(k-LB); endif
        if( k    .EQ. 1 ) then; T2 = 0. ; else; T2 = B(k)*Xnew(k-1 ); endif
        if( k    .EQ. n ) then; T4 = 0. ; else; T4 = D(k)*Xtmp(k+1 ); endif
        if( k+UB .GT. n ) then; T5 = 0. ; else; T5 = E(k)*Xtmp(k+UB); endif
        ! Compute
        Xnew( k ) = 1./C(k) * ( F(k) - T1 - T2  - T4 - T5 )
!         write(*,10) k, T1, T2, T4, T5, Xnew( k )
    end do
    !compute relative change for this iteration
    do k = 1, n
        relchng(k) = ( Xnew(k) - Xtmp(k) ) / Xtmp(k)
    end do
    ! check for convergence
!    write(dev,*) 'GAUSS_SIEDEL_PENTA: Iteration', m, ', Max rel change:', maxval(abs(relchng))
    if( maxval( abs( relchng ) ) .LT. tolit .AND. &
          F_L2_Norm( relchng, n) .LT. tolit          ) then
!            write(dev,*) 'GAUSS_SIEDEL_PENTA: Iterations required to converge: ', m
            numits = m
            exit ! exit iteration loop
!    elseif( maxval( abs(relchng) ) .GT. tolit ) then
     else
            Xtmp = Xnew
    endif
```

```
!end iteration loop
end do
! Print message if maximum number of iterations was exceeded
if ( m .gt. maxit ) then
    write(*,*) 'GAUSS_SEIDEL_PENTA: maximum number of iterations &
                 &exceeded; program will terminate.'
    STOP
endif

10 format( I, 10f12.7 )

!---------------------------------------------------------------------------
end subroutine gauss_seidel_penta
!============================================================================
!   \\\\\\\\\\         E N D   S U B R O U T I N E    //////////
!   //////////    G A U S S _ S E I D E L _ P E N T A   \\\\\\\\\\
!============================================================================




!======================================================================
!       \\\\\\\\\\   B E G I N   S U B R O U T I N E   //////////
!       //////////         T H O M A S                 \\\\\\\\\\
!======================================================================
SUBROUTINE THOMAS(A,B,C,D,X,N)
integer :: N
REAL A(N), B(N), C(N), D(N), X(N), Q(n+1), G(n+1)
REAL :: WI
integer :: i,j
!
!       Purpose:        Solve a system of linear equations that appear
!                       as a tri-diagonal matrix.
!
!       Source:         This algorithm was handed out in class.
!       Written By:     Brad Eck
!       Revision 0:     Original coding on 19 Feb 09
!
!       A -- Main diagonal
!       B -- Superdiagonal
!       C -- Subdiagonal
!       D -- RHS vector
!       X -- Solution vector
!       N -- number of unknowns
!-----------------------------------------------------------------------

WI=A(1)
G(1)=D(1)/WI
DO I=2,n
        Q(I-1) = B(I-1)/WI
        WI = A(I) - C(I) * Q(I-1)
        G(I) = ( D(I) -C(I) * G(I-1))/WI
END DO

```

```fortran
X(N) = G(N)

DO I=2,n
        J = N - I + 1
        X(J) = G(J) - Q(J) * X(J+1)
END DO

END SUBROUTINE THOMAS

!===========================================================================
!       \\\\\\\\\\    E N D   S U B R O U T I N E    //////////
!       //////////         T H O M A S               \\\\\\\\\\
!==========================================================================

!============================================================================
!   \\\\\\\\\\\\\\\\\\                    //////////////////////
                            END MODULE solvers
!   //////////////////                    \\\\\\\\\\\\\\\\\\\\\\
!============================================================================


```

Source File 11: pfc1Dfuns2.f95

```
!fortran_free_source

! need a seperate module for these last two functions b/c
! the function F_CC calls both of them, and they cannot be
! in the same module
module pfc1dfuns2
implicit none
contains

!==========================================================================
!function to determine thickness in the pavement at the cell face
FUNCTION F_hp_face(dxin, hin, zin, dxout, hout, zout, b)
implicit none
!INPUTS
REAL :: dxin, dxout  ! size of cells
REAL :: hin, hout    ! thickness at CV center
REAL :: zin, zout    ! elevation at CV center
REAL :: b            ! pavement thickness
REAL :: F_hp_face
!DUMMY
REAL :: head_at_face, Zface !HEAD and ELEVATION at the face
!
head_at_face = ( (hin+zin)*dxin + (hout+zout)*dxout )  &
                         / ( dxin + dxout)
Zface        = ( zin*dxin + zout*dxout ) / ( dxin + dxout)
F_hp_face = MIN ( b, head_at_face - Zface )
END function
!==========================================================================
!function to determine the thickness on the surface at the cell face
FUNCTION F_hs_face(dxin, hin, zin, dxout, hout, zout, b)
implicit none
!INPUTS
REAL :: dxin, dxout  ! size of cells
REAL :: hin, hout    ! thickness at CV center
REAL :: zin, zout    ! elevation at CV center
REAL :: b            ! pavement thickness
REAL :: F_hs_face
!DUMMY
REAL :: head_at_face, Zface !HEAD and ELEVATION at the face
!
head_at_face = ( (hin+zin)*dxin + (hout+zout)*dxout )  &
                         / ( dxin + dxout)
Zface        = ( zin*dxin + zout*dxout ) / ( dxin + dxout)
F_hs_face = MAX ( 0., head_at_face - Zface - b )
END FUNCTION
!==========================================================================

end module pfc1dfuns2

```

Source File 12: pfc1Dfuns.f95

```fortran
! fortran_free_source

module pfc1Dfuns

implicit none

contains



!============================================================================
! function to compute the conveyance coef at the western face
! the convention used here is that 'in' refers to cell 'i'
! and 'out' refers to cell 'i-1', which is the western cell

FUNCTION F_CC( xin,  dxin,  hin, zin, &
              xout, dxout, hout, zout )  Result( CC )
use shared, only: K, n_mann, b_pfc, h_pfc_min
use pfc1dfuns2
REAL :: xin,  xout   ! coordinate of the ith cell and the WESTERN cell center
REAL :: dxin, dxout  ! cell sizes
REAL :: hin,  hout   ! thicknesses at cell center
REAL :: zin,  zout   ! elevations  at cell center
REAL :: CC   !, F_hp_face, F_hs_face  <----these now in a module
! dummy vars
REAL :: hpw !thickness in the PAVEMENT at the western face
REAL :: hsw !thickness on the SURFACE at the western face
REAL :: Sfw !magnitude of hydraulic gradient at the western face
logical :: error

! Intermediate quantities
hpw =  F_hp_face(dxin, hin, zin, dxout, hout, zout, b_pfc)
hsw =  F_hs_face(dxin, hin, zin, dxout, hout, zout, b_pfc)
Sfw =  sqrt( ( ( hout + zout - hin - zin ) * 2. / &
                (dxout + dxin) ) ** 2 )

! Set hpw to small but positive and with enough range
!   left to allow further calcs.zero if negative
if( hpw .LT. TINY( hpw ) ) then
    hpw   =  h_pfc_min
end if

!Conveyance coefficient itself
if( hsw .GT. 0.0 ) then
   CC  = 1. / abs( xin - xout ) *                        &
            (   K * hpw +                                &
              1./ n_mann * hsw ** (5./3.) /  sqrt(Sfw) )
else
! only PFC flow
   CC  = 1. / abs( xin - xout ) *                        &
            (   K * hpw   )
```

```fortran
end if


! ERROR CHECKING FOR CONVEYANCE COEFS
if( CC .GT. HUGE(CC) .OR. CC .LT. -HUGE(CC) ) then
    error = .true.
else
    error = .false.
endif

!Output the parts of the calculation if the error is true
if( error .eqv. .true. ) then
    write(*,*) 'Problem with 1D conveyance coefficient!'
    print *, '       K = ', K
    print *, '      hp = ', hpw
    print *, '  n_mann = ', n_mann
    print *, '      hs = ', hsw
    print *, '      Sf = ', Sfw
    print *, '     xin = ', xin
    print *, '    xout = ', xout
    print *, '      CC = ', CC
    write(*,*) 'Stopping Program'
    STOP
endif

END FUNCTION
!======================================================================

!======================================================================
!Function to switch the porosity on/off if the
!  water is in/out of the pavement
FUNCTION F_por(h)
USE shared, only: b_pfc, por
IMPLICIT NONE
REAL h, F_por
if      ( h >= b_pfc ) then
                        F_por = 1.
ELSEIF  ( h < b_pfc )  then
                        F_por = 1./por
end if
END function F_por
!======================================================================




!======================================================================
!       \\\\\\\\\\                                      //////////
                      END MODULE pfc1Dfuns
!       //////////                                      \\\\\\\\\\\
!======================================================================
```

Source File 13: pfc1Dsubs.f95

```fortran
! fortran_free_source

module pfc1Dsubs

implicit none

contains






!==========================================================================================
!        \\\\\\\\\\       B E G I N    S U B R O U T I N E      //////////
!        //////////      S E T U P    1 D    S E C T I O N      \\\\\\\\\\

!==========================================================================================
SUBROUTINE setup_1D_section( )
!    does the setup work for looking at this as a 1D section
USE shared,    ONLY: Z_lp, nr_lp, dist_lp, slope_cs_1D,          &
                     wid_cs_1d, eta_cs_1d, nr_cs, long_slope,    &
                     slope_cs, wid_cs
USE utilities, ONLY: F_PythagSum
!-----------------------------------------------------------------
integer :: i
!-----------------------------------------------------------------
! Compute longitudinal slope
!   ( assumed to be constant thorought the domain)
long_slope = (    Z_lp( nr_lp ) -    Z_lp( 1 ) ) / &
             ( dist_lp( nr_lp ) - dist_lp( 1 ) )


! Using the longitudinal slope and cross slope,
! compute the slopes and segment widths for the 1D profile

allocate( slope_cs_1d( nr_cs) )
allocate(   wid_cs_1d( nr_cs) )
allocate(   eta_cs_1d( nr_cs+1) )

write(*,*) ''
write(*,*) '  1D CROSS SECTION   '
write(*,*) ' Segment   Slope   Width '
write(*,*) '============================'
do i = 1, nr_cs
    !For the slope, compute the magnitude of the resultant slope
    ! using pythagorean sum and then use the intrinsic SIGN function
    ! to give the resultant the same sign as the cross slope.
    slope_cs_1D(i) = SIGN (  F_PythagSum( long_slope, slope_cs(i) ) , &
                                                     slope_cs(i)       )
    ! Compute 1D width using similar triangles
```

```fortran
    wid_cs_1D(i) = slope_cs_1D(i) / slope_cs(i) * wid_cs(i)
    ! Print the results as we go
    write(*,10) i, slope_cs_1D(i), wid_cs_1d(i)
end do
write(*,*) ''



eta_cs_1d(1) = 0.

! Compute etas and elevations
do i = 2, nr_cs + 1
    eta_cs_1d(i) = eta_cs_1d(i-1) +   wid_cs_1d(i-1) / sum( wid_cs_1D(1:nr_cs) )
end do

! print the results to confirm
print *, ' 1D CROSS SECTION POINTS '
WRITE(*,*) ' Point         Eta         '
WRITE(*,*)  '==================================='
do i = 1, nr_cs + 1
    write(*,10) i, eta_cs_1d(i)
end do


!--------------------------------------------------------------------------------
! Format Statements
10      FORMAT('   ', ( i3, '    '),  ( F10.3, '  ') , F10.6 )

!--------------------------------------------------------------------------------
end subroutine setup_1D_section
!================================================================================
!       \\\\\\\\\\      E N D       S U B R O U T I N E   //////////
!       //////////     S E T U P  1 D  S E C T I O N      \\\\\\\\\\
!================================================================================

!=================================================================================
!       \\\\\\\\\\      B E G I N    S U B R O U T I N E   //////////
!       //////////     G R I D   1 D   S E C T I O N      \\\\\\\\\\
!=================================================================================
SUBROUTINE grid_1d_section( slope_in, width_in, seg, dx )
!---------------------------------------------------------------------------
use shared, only: TNE, XCV, ZCV, EDX, etaCV
use pfc1Dfuns
use utilities, only: F_Linterp
use outputs, only: WRITE_MATRIX

!Define variables

    IMPLICIT NONE

    !CONSTANTS
    INTEGER, intent(in) :: seg
    real, intent(in) :: dx

```

```fortran
    !ARRAYS
    REAL, dimension( seg ), intent(in) :: slope_in,  width_in
!calculation variables
    real, dimension(seg) :: slope, width
    INTEGER, dimension( seg ) :: ne, ir
    INTEGER :: gb
    INTEGER :: i, n, s, start, finish
    REAL, ALLOCATABLE :: xface(:), zface(:)
    real, allocatable :: seg_X(:), seg_Z(:)
    REAL :: DX1
!------------------------------------------------------
! Need to reverse slope and width arrays based on
!   the design of this subroutine.
! Indices for Reverse arrays (uses an implied DO loop )
ir = (/ ( i, i = seg, 1, -1  ) /)
    do i = 1, seg
        slope(i) = slope_in( ir(i) )
        width(i) = width_in( ir(i) )
           ne(i) = NINT( width(i) / dx )
    end do
!-----------------------------------------------------------------
!Compute derivative quanties   & allocate remaining arrays
    gb = seg - 1
    TNE = sum(ne) + gb
    allocate( XFACE(TNE+1), &
              ZFACE(TNE+1), &
                XCV(TNE)  , &
                ZCV(TNE)  , &
                EDX(TNE)  , &
              etaCV(TNE)         )
!-------------------------------------------------------------------
! compute the points for the boundaries and the CV centers
    XFACE( 1 ) = 0. !could use a different starting point
    EDX(:) = dx     ! all elemnts are the same size
```

```fortran
    do i = 1, TNE
        XFACE( i+1 ) = XFACE( i ) + EDX( i )
        XCV  ( i   ) = (  XFACE( i ) + XFACE( i+1 )  ) / 2.
        etaCV( i   ) = 1. - XCV( i ) / sum( width(:) )
    end do
!-----------------------------------------------------------------------
! interpolate elevations of the points

    allocate ( seg_X(seg+1), seg_Z(seg+1) )

    seg_x(1) = 0.
    seg_Z(1) = 10.

    do i = 1, seg
        seg_x(i+1) = seg_X(i) + width(i)
        seg_Z(i+1) = seg_Z(i) + width(i) * slope(i)
    end do

    ! first cross section by interpolation
    ZFACE( 1 ) = seg_z( 1 )
    do i = 1, TNE
        ZFACE( i+1 ) =  F_linterp( XFACE( i+1 ) , seg_X, seg_Z, seg+1 )
        ZCV  ( i   ) =  F_linterp( XCV  ( i   ) , seg_X, seg_Z, seg+1 )
    end do

!-----------------------------------------------------------------------
! output the resulting arrays

! VECTOR FORM
    OPEN( UNIT = 20, FILE = 'grid_1D_section.csv', STATUS = 'REPLACE' )
    WRITE(20,*) 'XFACE, ZFACE, CV, XCV, ZCV, EDX, etaCV'
    do i = 1, TNE
        WRITE(20,100) XFACE( i ), ZFACE( i ), i, XCV( i ), ZCV( i ), EDX(i), etaCV(i)
    end do
    WRITE(20,99) XFACE( TNE+1 ), ZFACE( TNE+1)
    close(20)

!---------------------------------------------------
!Format statements
90      FORMAT( i, F12.6 )
99      FORMAT( 2( F12.6, ',' ) )
100     FORMAT( 2( F12.6, ',' ), 1(I, ','), 5( F12.6, ',')  )


!-----------------------------------------------------------------------------
    end subroutine GRID_1D_SECTION
!================================================================================
!       \\\\\\\\\\          E N D   S U B R O U T I N E    //////////
!       //////////       G R I D  1 D  S E C T I O N       \\\\\\\\\\
!================================================================================
```

```
!==============================================================================
!   \\\\\\\\\\\\        B E G I N     P R O G R A M     //////////
!   ////////////            P F C 1 D I M P             \\\\\\\\\\
!==============================================================================
!       Purpose:        This program computes a 1D solution for unsteady
!                       drainage through a PFC.  The water THICKNESS in each
!                       cell is used as the primary variable.
!       History:        Source code revised from previous program that used
!                       an explicit method, and revised again to use as a
!                       subroutien within the 2D model
!       IC:             The depth on input to the subroutine
!       BCs:            Various
!       Linearization:  Picard Iteration (lag the coefficients)
!       Linear Solver:  Thomas Alogorithm used for this 1D case
!
!       Externals:      1.
!                       2.


SUBROUTINE pfc1Dimp( h_old, dt, rain, tolit, qmax, &
                     h_new, imax, eta_0_BC, eta_1_BC )
!bring in related modules
use shared,    only: K, g, b_pfc, n_mann, por, XCV, ZCV, EDX, &
                     etaCV, relax, relax_tran, h_pfc_min, eta_0_hp2_max
use utilities, only: F_Linterp, F_L2_NORM
use solvers,   only: Thomas
use pfc1Dfuns
use outputs,   only: WRITE_MATRIX, write_vector

!-------------------------------------------------------------------------------
!VARIABLE DECLARATIONS

implicit none
! NOTE:  Because of ONLY statement for modules, only the names variables
!        are used in this subroutine.  This allows re-using variables names
!        thus the variable 'h_old' in this subroutine is not the same as
!        the 'h_old' matrix in PERFCODE.

! Arguments
integer,                intent(in)  :: imax     ! imax == TNE from 1D gridding
real, dimension(imax), intent(in)  :: h_old
real,                   intent(in)  :: dt
real,                   intent(in)  :: rain
real,                   intent(in)  :: tolit
integer,                intent(in)  :: qmax
real, dimension(imax), intent(out):: h_new
character( len = 7 ) :: eta_0_BC, eta_1_BC

!SCALERS
INTEGER :: i, n, q
REAL    :: PF, PF1, Cw, Ce, Ce1, Cw1
REAL    :: hp, hs
REAL    :: cputime
real, dimension(:), pointer ::  X, Z, DX
```

```fortran
!guess values for water thickness in cm
REAL    :: h_itr(imax)
!linear system (tri-diagonal)
REAL    :: main(imax), super(imax), sub(imax), RHS(imax)
!solution
REAL    :: h_temp (imax) !,  h_new  (imax)
!convergence test
REAL    :: relchg(imax)
!post processing
REAL    :: head  (imax), hygrad (imax)               ! head and hydraulic gradient
REAL    :: q_pav (imax), q_surf (imax), q_tot(imax) !fluxes
!summary info
integer, parameter :: nmax = 2
INTEGER :: numit  (nmax), loc(nmax)  ! number of iterations at each timestep & locaiton of max change
REAL    :: maxdiff(nmax)             ! max change and its location
!FUNCTIONS
!       REAL    :: F_por, F_CC
!CHARACTERS
CHARACTER(8) DATE
CHARACTER(10) TIME
logical :: transition
real, dimension( imax, qmax) :: h_temp_hist, h_itr_hist !for monitoring the solution through the iteration
real :: relaxation_factor    !  Relaxation Factor
real, dimension( imax ) :: residual
real :: b  ! an extra value for b_pfc
real :: hs1, hs2, ds  ! Sheet flow MOC
real :: hp1, hp2, dx_moc ! PFC flow MOC
real :: cross_slope
integer, dimension(1) :: i_Zmax   !index of point with highest elevation
real :: eps_itr_tol
integer :: nip    ! number of interpolation points
!------------------------------------------
! Set pointers
X => XCV
Z => ZCV
DX => EDX
b = b_pfc
n = 1
i_Zmax = maxloc( Z )
!-------------------------------------------------------------------------------
!SPATIAL GRID  (from GRID_1D_SECTION
```

```
! INITIALIZE ARRAYS
    !iteration array
    h_itr = h_old

    ! Linear system
    main  = 0.
    super = 0.
    sub   = 0.
    rhs   = 0.

    ! Summary arrays
    numit   = 0.
    maxdiff = 0.
    loc     = 0.

!--------------------------------------------------------------------------------
! Solution using Crank-Nicolson with tri-diagonal matrix algorithm
! main, super & sub are diagonals of the coefficient matrix
! RHS is the right hand side of the linear system

open( unit = 50, file = 'PF_smry.csv',  status = 'REPLACE')
write(50,54) 'n/i', (/ (i,  i = 1, imax) /)

open( unit = 110, file = '1DRunDetails.txt', status = 'REPLACE')

54 format( ( A, ','), 10000( I, ',') )



!iteration loop
do q = 1, qmax


! UPSTREAM BOUNDARY
i = 1
! First cell
 PF  = F_por( h_old(i) )
 PF1 = F_por( h_itr(i) )

BC1: if( eta_1_BC .EQ. 'NO_FLOW') then
  ! NO FLOW BOUNDARY  Cw ---> 0
    Cw  = 0.
    Ce  = F_CC( X(i  ), DX(i  ), h_old(i  ), Z(i),     &
                X(i+1), DX(i+1), h_old(i+1), Z(i+1) )
    !these coefficints are updated as the iteration progresses
    Cw1 = 0.
    Ce1 = F_CC( X(i  ), DX(i  ), h_itr(i  ), Z(i),     &
                X(i+1), DX(i+1), h_itr(i+1), Z(i+1) )
    !Diagonals of coefficient matrix
    main (i) =  1  +  dt / 2 * PF1 * Cw1 / DX(i) &
                   +  dt / 2 * PF1 * Ce1 / DX(i)
    super(i) =      - dt / 2 * PF1 * Ce1 / DX(i)
```

```
    sub  (i) =      - dt / 2 * PF1 * Cw1 / DX(i)    ! will be zero
    !Right hand side of linearized system
              ! part that came from n level
    RHS  (i) =    h_old(i) + dt / 2.   * PF *                   &
            !  (  Cw / DX(i) * (  h_old(i-1) + Z(i-1)           &  !enforce BC
            !                    - h_old(i  ) - Z(i  ) )        &  !enforce BC
              ( + Ce / DX(i) * (  h_old(i+1) + z(i+1)           &
                                - h_old(i  ) - Z(i  ) )         &
               + rain  )                                        &
               ! part from n+1 level
               + dt / 2. * PF1 *                                &
           !   (   Cw1 / DX(i) * ( Z(i-1) - Z(i) )              &   !enforce BC
               ( + Ce1 / DX(i) * ( Z(i+1) - Z(i) )              &
                + rain   )

elseif( eta_1_BC .EQ. 'MOC_KIN') then
        ! METHOD OF CHARCTERISTICS KINEMATIC BOUNDAARY
        ! cross slope is defined positive for use in SQRT
        cross_slope = ( Z(i+1) - Z(i) ) / ( X(i+1) - X(i) )
        ! If it comes out negative, a different BC is needed
        if( cross_slope .LT. 0.0) then
            write(110,*) 'Different BC needed at eta = 1'
        endif
        if( h_old(i) .LE. b_pfc) then
                ! PFC FLOW MOC BC
                dx_moc =  K * (cross_slope) * dt / por
                !Interpolate up the drainage slope...make sure we have enough points
                nip = NINT(dx_moc / DX(1) ) + 2
                write(110,*) 'eta_1_bc X=', (xcv(1) + dx_moc), &
                                    'nip=', nip,               &
                                     'KX=',   XCV(1:nip),      &
                                     'KY=', h_old(1:nip)
                hp1 = F_Linterp(       X = (xcv(1) + dx_moc), &
                                 Known_X = XCV( 1:nip)      , &
                                 Known_Y = h_old(1:nip)     , &
                                       n = nip                 )
                if( hp1 .LT. 0.0) then
                        write(100,*) 'PFC1DIMP: eta_1_bc hp1=', hp1
                        stop
                endif
                hp2 = hp1 + rain * dt / por
                if( rain .LT. TINY ( rain )) then
                    ! Rainfall rate is effectively zero
                    hp2 = hp2
                else
                    ! Rainfall is non-zero, set a maximum value for hp2
                    ! the total drainage distance is the sum from the highest point in
                    ! the 1D domain to the end, this is why MAXLOC is used.
                    hp2 = min( hp2, sum( DX( i : i_Zmax(1) ))*rain/K/cross_slope)
                endif
                ! Fill in linear system
                main(i) = 1.
```

```
                    RHS (i) = hp2
! write(100,*) 'PFC1DIMP: eta_BC = MOC_KIN i=',i, 'dx_moc=', dx_moc, 'hp1=', hp1, 'hp2=', hp2
            else
                    !SHEET FLOW MOC BC
                    hs2 = h_old(i) - b_pfc
                    ! Handle zero rainfall
                    if( rain .LT. TINY( rain ) ) then
                        ! no increase in flow rate along drainge path
                        ! ds is arbitray, so use the PFC value
                        ds =  K * (cross_slope) * dt / por
                    else
                        ds = sqrt( cross_slope ) / n_mann / rain *    &
                             ( ( hs2 + rain * dt )**(5./3.) - hs2**(5./3.) )
                    endif
                    ! Interpolate up the slope to find hs1
                    nip = NINT(ds / DX(1) ) + 2
                    write(110,*) 'eta_1_bc X=',(xcv(1) + ds), 'nip=', nip, 'KX=', XCV(1:nip), 'KY=', h_old(1:nip)
                    hs1 = F_Linterp(       X = (xcv(1) + ds),    &
                                     Known_X = XCV( 1:nip)  , &    !(/ 0.      , DX(i)  /) ,&
                                     Known_Y =  h_old(1:nip), &
                         !(/ h_old(i), h_old(i+1) /), &
                                           n = nip ) - b_pfc
                    ! Handle return to sheet flow
                    if( hs1 .GT. 0.0 ) then
                        ! we have sheet flow
                        main(i) = 1.
                        RHS (i) =  b_pfc + ( hs1**(5./3.) + ( hs2 + rain*dt )**(5./3.) - hs2**(5./3.) )**0.6
                    else
                        ! upstream point has sheet flow
                        ! use the PFC characteristic
                        main(i) = 1.
                        RHS (i) = hs1 + b_pfc + rain * dt / por
                    end if
              end if
endif BC1

!Interior of domain
do i = 2, imax - 1
    PF  = F_por( h_old(i) )
    PF1 = F_por( h_itr(i) )
!        FUNCTION F_CC(    xin,  dxin,  hin, zin, &
!                          xout, dxout, hout, zout )
    !these coefficients are stationary (time level n)
    Cw  = F_CC( X(i  ), DX(i  ), h_old(i  ), Z(i),     &
                X(i-1), DX(i-1), h_old(i-1), Z(i-1) )
    Ce  = F_CC( X(i  ), DX(i  ), h_old(i  ), Z(i),     &
                X(i+1), DX(i+1), h_old(i+1), Z(i+1) )
    !these coefficints are updated as the iteration progresses
    Cw1 = F_CC( X(i  ), DX(i  ), h_itr(i  ), Z(i),     &
                X(i-1), DX(i-1), h_itr(i-1), Z(i-1) )
```

```
    Ce1 = F_CC( X(i  ), DX(i  ), h_itr(i  ), Z(i),      &
                X(i+1), DX(i+1), h_itr(i+1), Z(i+1) )


    !Diagonals of coefficient matrix
    main (i) =  1  +  dt / 2 * PF1 * Cw1 / DX(i) &
                   +  dt / 2 * PF1 * Ce1 / DX(i)
    super(i) =      - dt / 2 * PF1 * Ce1 / DX(i)
    sub  (i) =      - dt / 2 * PF1 * Cw1 / DX(i)
    !Right hand side of linearized system
                ! part that came from n level
    RHS  (i) =    h_old(i) + dt / 2.  * PF *                       &
               (  Cw / DX(i) * (  h_old(i-1) + z(i-1)              &
                                - h_old(i  ) - Z(i  ) )            &
                + Ce / DX(i) * (  h_old(i+1) + z(i+1)              &
                                - h_old(i  ) - Z(i  ) )            &
                + rain  )                                          &
                ! part from n+1 level
                + dt / 2. * PF1 *                                  &
               (   Cw1 / DX(i) * ( z(i-1) - z(i) )                 &
                 + Ce1 / DX(i) * ( z(i+1) - Z(i) )                 &
                 + rain   )
 end do

! DOWNSTREAM BOUNDARY
i = imax
! use BC from input argument
BC0:if( eta_0_BC .EQ. 'NO_FLOW') then
    !NO FLOW BOUNDARY ---> Ce == 0
    !these coefficients are stationary (time level n)
    Cw  = F_CC( X(i  ), DX(i  ), h_old(i  ), Z(i),      &
                X(i-1), DX(i-1), h_old(i-1), Z(i-1) )
    Ce  = 0.0
    !these coefficints are updated as the iteration progresses
    Cw1 = F_CC( X(i  ), DX(i  ), h_itr(i  ), Z(i),      &
                X(i-1), DX(i-1), h_itr(i-1), Z(i-1) )
    Ce1 = 0.0
    !Diagonals of coefficient matrix
    main (i) =  1  +  dt / 2 * PF1 * Cw1 / DX(i) &
                   +  dt / 2 * PF1 * Ce1 / DX(i)
    super(i) =      - dt / 2 * PF1 * Ce1 / DX(i)
    sub  (i) =      - dt / 2 * PF1 * Cw1 / DX(i)
    !Right hand side of linearized system
                ! part that came from n level
    RHS  (i) =    h_old(i) + dt / 2.  * PF *                       &
               (  Cw / DX(i) * (  h_old(i-1) + z(i-1)              &
                                - h_old(i  ) - Z(i  ) )            &
            !   + Ce / DX(i) * (  h_old(i+1) + z(i+1)              &     !enforce BC
           !                    - h_old(i  ) - Z(i  ) )            &     !enforce BC
                + rain  )                                          &
                ! part from n+1 level
                + dt / 2. * PF1 *                                  &
               (   Cw1 / DX(i) * ( z(i-1) - z(i) )                 &
           !     + Ce1 / DX(i) * ( z(i+1) - Z(i) )                 &     !enforce BC
```

```
                 + rain   )
elseif( eta_0_BC .EQ. 'MOC_KIN') then
      ! METHOD OF CHARCTERISTICS KINEMATIC BOUNDAARY
      cross_slope = ( Z(i-1) - Z(i) ) / ( X(i) - X(i-1) )    ! cross slope is positive downwars for use in SQRT
      if( cross_slope .LT. 0.0) then
          write(110,*) 'Different BC needed at eta = 0'
      endif
      if( h_old(i) .LE. b_pfc) then
              ! PFC FLOW MOC BC
              dx_moc =  K * (cross_slope) * dt / por       !
              nip = NINT(dx_moc / DX(imax) ) + 2
              write(110,*) 'eta_0_BC X=', (XCV(imax) - dx_moc)  ,  &
                                    'nip=', nip,                   &
                                     'KX=', XCV( imax-nip+1:imax) ,  &
                                     'KY=', h_old( imax-nip+1 : imax)

              hp1 = F_Linterp(      X = (XCV(imax) - dx_moc)   ,  &
                                 Known_X = (XCV( imax-nip+1:imax)),  &
                                 Known_Y = h_old( imax-nip+1 : imax) , &
                                       n = nip )
              if( hp1 .LT. 0.0) then
                      write(100,*) 'PFC1DIMP: eta_0_bc hp1=', hp1
                      stop
              endif
              hp2 = hp1 + rain * dt / por
              if( rain .LT. TINY ( rain )) then
                  ! Rainfall rate is effectively zero
                  hp2 = hp2
              else
                  ! Rainfall is non-zero, set a maximum value for hp2
                  ! the total drainage distance is the sum from the highest pt
                  ! inthe 1D domain to the end, this is why MAXLOC is used.
                  eta_0_hp2_max = sum( DX( i_Zmax(1) : i ))*rain/K/cross_slope
                  hp2 = min( hp2, eta_0_hp2_max)
              endif
              ! Fill in linear system
              main(i) = 1.
              RHS (i) = hp2
       else
              !SHEET FLOW MOC BC
              hs2 = h_old(i) - b_pfc
              ! Handle zero rainfall
              if( rain .LT. TINY( rain ) ) then
                  ! no increase in flow rate along drainge path
                  ! ds is arbitray, so use the PFC value
                  ds =  K * (cross_slope) * dt / por
              else
                  ds = sqrt( cross_slope ) / n_mann / rain *    &
                       ( ( hs2 + rain * dt )**(5./3.) - hs2**(5./3.) )
              endif
              ! Interpolate up the slope to find hs1
              nip = NINT(ds / DX(imax) ) + 2
              write(110,*) 'eta_0_BC X=', (XCV(imax) - ds)         , &
```

```
                              'nip=', nip                   , &
                               'KX=', XCV( imax-nip+1:imax)    , &
                               'KY=', h_old( imax-nip+1 : imax)

                hs1 = F_Linterp(       X = (XCV(imax) - ds)           , &
                                 Known_X = (XCV( imax-nip+1:imax))    , &
                                 Known_Y = h_old( imax-nip+1 : imax) , &
                                       n = nip ) - b_pfc
                ! Handle return to sheet flow
                if( hs1 .GT. 0.0 ) then
                    ! we have sheet floe
                    main(i) = 1.
                    RHS (i) =  b_pfc + ( hs1**(5./3.) + ( hs2 + rain*dt )**(5./3.)
- hs2**(5./3.) )**0.6
                else
                    ! upstream point has sheet flow
                    ! use the PFC characteristic
                    main(i) = 1.
                    RHS (i) = hs1 + b_pfc + rain * dt / por
                end if
            end if
   end if BC0



! TRANSITION CHECK
!   test to see if there is a transition to or from sheet flow
!   happening during this timestep.   Use under-relaxtion to
!   control oscillations during a transition timestep.

transition = .false.
do i = 1, imax
    pf = F_por( h_old(i) )
    pf1= F_por( h_itr(i) )
    if( pf .GT. pf1 .OR. pf .LT. pf1) then
        transition = .true.
        write(110,*) 'PERFCODE: transition for cell i=', i, 'pf=', pf, 'pf1=', pf1
    endif
end do

if( transition .eqv. .true. ) then
    relaxation_factor = relax_tran
    eps_itr_tol = tolit * 10.
else
    relaxation_factor = relax
    eps_itr_tol = tolit
endif


        !Solve linear system
        CALL THOMAS(main, super, sub, RHS, h_temp, imax)



```

```fortran
! Compute residual and relative change for this iteration.  This took
!  some careful though to handle both filling and draining cases.
!  relative change is used when the solution is far from zero
!  and absolute change (residual) is used near zero.

do i = 1, imax

    if( h_temp(i) .GT. TINY( h_temp(i) ) ) then

            ! Compute residual for this iteration
            residual(i) =  h_temp(i) - h_itr(i)

            ! Handle a result that is effectively zero by
            ! using an absolute tolerance instead of
            ! a relative one
            if( h_temp(i) .LE. h_pfc_min  .and. &
                residual (i) .LE. eps_itr_tol     ) then

                    relchg(i) = 0.0

            else
                    relchg (i) =  residual (i) / h_itr(i)
            endif

    elseif( h_temp(i) .LE. TINY( h_temp(i) ) ) then

            ! the model is saying the cell is empty,
            ! so force the solution to be zero
            h_temp(i) =  0.0
            ! compute the residual
            residual(i) =  h_temp(i) - h_itr(i)
            ! For the zero case, use an absolute rather than
            ! relative tolerance by setting the value of relchng
            ! below the tolerance instead of computing it.
            if( abs( residual(i) ) .LE. eps_itr_tol ) then

                    relchg(i) = 0.0
            endif
    endif

end do


if( maxval( h_temp) .LT. TINY( h_temp(1) )   ) then
       write(*,*) 'PFC1DIMP: Zeroed out.  Writing system and stopping program'
       open( unit = 10, file = '1Dsystem.csv', status = 'REPLACE' )
       write(10,*) 'i,sub,main,super,rhs,h_temp,'
       do i = 1, imax
             write(10, 10) i, sub(i), main(i), super(i), RHS(i), h_temp(i)
       end do
       close( 10 )

       call write_vector( h_old, imax, 'h_old_1d.csv')
       STOP
```

```
10  FORMAT (  (I, ','), 5(E, ',') )

end if


!perform usual iteration check
IF ( maxval ( ABS( relchg ) ) .le. eps_itr_tol .AND.  &
      F_L2_NORM( relchg, imax )    .le. eps_itr_tol       ) then
        !  WRITE(*,*) 'Time step n = ', n,' converged in q = ', q, ' iterations.'
          EXIT
      end if




! Smith page 32

h_itr = h_itr + relaxation_factor * residual

!Store the result of this iteration
h_temp_hist( : , q ) = h_temp
h_itr_hist ( : , q ) = h_itr



 WRITE(110,*) 'ITERATION q=', q                          , &
              'Max Change of', maxval( abs( relchg) ), &
                      'at i=', maxloc( abs( relchg)) , &
                 'h_temp(i)=', h_temp( maxloc( abs( relchg)) ) , &
                   'L2_Norm=', F_L2_NORM( relchg, imax)



!end iteration loop
end do

!update the old and new solutions
!At the end of the iteration, we have found values for the
!next time step.

h_new  = h_temp


!Store summary info for this timestep
numit  ( n ) = q
loc    ( n ) = maxloc ( abs( relchg ), dim=1 )
maxdiff( n ) = relchg ( loc ( n ) )

!Give Error if Iteration fails to converge
if (q .gt. qmax) then
  WRITE(*,*)   'PFC1DIMP: Iteration failed to converge. '
  write(100,*)  'PFC1DIMP: Iteration failed to converge. '
  CALL WRITE_MATRIX( h_temp_hist, imax, qmax, 'h_temp_hist_1D.csv')
  CALL WRITE_MATRIX( h_itr_hist , imax, qmax, 'h_itr_hist_1D.csv')
```

```
 
!          EXIT
end if
 
 
close(50)   ! pf summary file
close(110)  ! Run details file
!-------------------------------------------------------------------------------
! POST PROCESSING
 
!Compute head
head(:) = h_new(:) + Z(:)
 
!Compute hydraulic gradient and flux ( both positive downwards)
 
!Upstream boundary node (using a 1-sided approximation)
i = 1
hygrad(i) =  ( head(i) - head(i+1) )  /  ( X(i+1) - X(i) )
!In the pavement
hp = min( h_new(i), b )
q_pav (i) = K * hp * hygrad(i)
!on the surface
hs = max( 0., h_new(i) - b )
q_surf(i) = 1. / n_mann * hs ** (2./3.) * sqrt( abs( hygrad(i) ) ) * hs
q_tot(i) = q_pav(i) + q_surf(i)
 
do i = 2, imax - 1
    !hydrualic gradient
    hygrad(i) =  ( head(i-1) - head(i+1) )  /  ( X(i+1) - X(i-1) )
    !thickness in the pavement
    hp = min( h_new(i), b )
    q_pav (i) = K * hp * hygrad(i)
    !thickness on the surface
    hs = max( 0., h_new(i) - b )
    q_surf(i) = 1. / n_mann * hs**(2./3.) * sqrt( abs( hygrad(i) ) ) * hs
!              WRITE(*,*) 'i = ', i, 'hs = ', hs, 'q_surf =', q_surf(i)
    q_tot(i) = q_pav(i) + q_surf(i)
end do
 
! DOWNSTREAM BOUNDARY ( 1 sided approximation)
i = imax
!hydrualic gradient
hygrad(i) =  ( head(i-1) - head(i) )  /  ( X(i) - X(i-1) )
!thickness in the pavement
hp = min( h_new(i), b )
q_pav (i) = K * hp * hygrad(i)
!thickness on the surface
hs = max( 0., h_new(i) - b )
q_surf(i) = 1. / n_mann * hs ** (2./3.) * sqrt( abs( hygrad(i) ) ) * hs
q_tot(i) = q_pav(i) + q_surf(i)
 
!------------------------------------------------------------------------------
!Write results to a file
 
```

```
        call DATE_AND_TIME(DATE,TIME)
        call CPU_TIME(cputime)

        OPEN(UNIT = 10, FILE = 'pfc1Dimp.csv', STATUS='REPLACE')
        WRITE(10,*) 'Output From pfc1Dimp.f95'
        WRITE(10,*) 'Timestamp,', DATE,' ', TIME,','
        WRITE(10,*) 'Upstream Boundary == Fixed Value'
        WRITE(10,*) 'Downstream Boundary == Sf = So'
        WRITE(10,200) 'Hydraulic Conductivity (cm/s),', k
        WRITE(10,200) 'Rainfall Intensity (cm/hr),', rain * 3600.
        WRITE(10,200) 'PFC Thickness (cm),', b
        WRITE(10,200) 'Final Time (sec),', ( n-1 ) * dt
        WRITE(10,200) 'Time step (seconds),', dt
        WRITE(10,200) 'Grid spacing (cm),', dx(5)
        WRITE(10,*) 'Number of elements,', imax - 2
        WRITE(10,200) 'CPU Time (seconds),', cputime
        WRITE(10,*) '************************** &
                    &MODEL OUTPUT IN [ SI  ] UNITS &
                    &***********************************'
        WRITE(10,*) 'X, eta,    Z, PFC Surface, Thickness, Head,', &
                    'Hydraulic Gradient, Pavement Flux,'         , &
                    ' Surface Flux, Total Flux,'
        do i = 1, imax
                WRITE(10,100) X(i), etaCV(i), Z(i), Z(i) + b,  &
                              h_new(i), Head(i), hygrad(i),    &
                              q_pav(i), q_surf(i), q_tot(i)
        END do
        CLOSE(10)

!-------------------------------------------------------------------------------
!Write calculation summary to file

    OPEN( UNIT = 20, FILE = '1Ddetails.csv', STATUS='REPLACE')
    WRITE(20,*) 'Timestamp,', DATE, ' ', TIME, ','
    WRITE(20,*) '-----,'
    WRITE(20,*) 'Timestep, Iterations, MaxRelChng, MaxLocn'
    DO n = 1, nmax
        WRITE(20,300) n, numit(n), maxdiff(n), loc(n)
    end do
    close(20)

!------------------------------------------------------------------------------
!Format statements

100     FORMAT (100 ( F14.7, ',' ) )        !  Formatting for the actual output
200     FORMAT ( A, F10.4 )
300     FORMAT ( 2 (I, ','), E, ',', I, ',' )

!------------------------------------------------------------------------------
        END subroutine pfc1Dimp
!===============================================================================
!       \\\\\\\\\\      E N D      S U B R O U T I N E  //////////
!       //////////          P F C 1 D I M P             \\\\\\\\\\
!===============================================================================
```

```fortran
END MODULE pfc1Dsubs
```

Source File 14: pfc2Dsubs.f95

```fortran

! fortran_free_source

! This module holds external procedures (subroutine and functions)
!  for the pfc2D model (PERFCODE).
! Using module creates an explicit interface for the procedures

!=========================================================================
!     \\\\\\\\\\\  B E G I N                        //////////
                                    MODULE pfc2Dsubs
!     //////////                                     \\\\\\\\\\
!=========================================================================

IMPLICIT NONE

CONTAINS

!===================================================================
SUBROUTINE set_ABCDEF( i, j, Cw1, Ce1, Cs1, Cn1, pf, dt, rr )
!   Fills the arrays of the linear system for Cell v
USE shared,    ONLY: A, B, C, D, E, Fn, F1, F, jmax
USE pfc2Dfuns, ONLY: F_LinearIndex, F_RHS_n1
implicit none
!   Arguments
integer, intent( in ) :: i, j
real   , intent( in ) :: Cw1, Ce1, Cs1, Cn1, pf, dt, rr
!   Internal Variables??
integer :: v
!real, external :: F_RHS_n1
!------------------------------------------------------------
! Linear Index
v   = F_LinearIndex( i, j, jmax)

! Bands of penta-diagonal matrix
A(v) = - dt / 2. * pf * Cw1
B(v) = - dt / 2. * pf * Cs1
C(v) =   dt / 2. * pf * ( Cw1 + Cs1 + Cn1 + Ce1 ) + 1.
D(v) = - dt / 2. * pf * Cn1
```

```
E(v) = - dt / 2. * pf * Ce1
! Right-hand-side
! Portion from time level n+1
F1(v) = F_RHS_n1( i, j, Cw1, Ce1, Cs1, Cn1, rr, pf, dt )
!The complete right hand side has contributions from
! time level n and time level n+1
F(v) = Fn(v) + F1(v)
!----------------------------------------------------------------------
end subroutine set_ABCDEF
!=======================================================================
!   \\\\\\\\\\   E N D   S U B R O U T I N E      //////////
!   //////////    S E T _ A B C D E F             \\\\\\\\\\\
!========================================================================

!=======================================================================
!   \\\\\\\\\\   B E G I N   S U B R O U T I N E   //////////
!   //////////    S E T _ X Y H                   \\\\\\\\\\\
!========================================================================
subroutine SET_xyh( i , j, xx, yy , hh)
!   Assigns values to X, Y, and Z for pointing to the bi-linear
!    interpolatoin subroutine
use shared, only: jmax, h_old, Z, CV_Info
use pfc2dfuns, only: F_LinearIndex
!VARIABLE DECLARATIONS
!   Arguments
integer,intent( in ) :: i, j        ! Grid indices
real, intent( out ) :: xx, yy, hh   ! physical coordinates
!Internal variables
integer :: v
!-----------------------------------------------------------------------
v = F_LinearIndex( i, j, jmax )
xx = CV_Info( v ) % X
yy = CV_Info( v ) % Y
hh = h_old( i, j )

!----------------------------------------------------------------------
end subroutine SET_xyh
!=======================================================================
!   \\\\\\\\\\   E N D      S U B R O U T I N E   //////////
!   //////////    S E T _ X Y H                   \\\\\\\\\\\
!========================================================================


!========================================================================
!     \\\\\\\\\\\                                  //////////
                        END MODULE pfc2Dsubs
!     //////////                                   \\\\\\\\\\
!========================================================================
```

Source File 15: BoundCond.f95

```fortran
! fortran_free_source

!==========================================================================
!   \\\\\\\\\\                                      //////////
                      MODULE BoundCond
!   //////////                                      \\\\\\\\\\
                      implicit none
                      contains
!==========================================================================

!=================================================================
!   \\\\\\\\\\      B E G I N    S U B R O U T I N E ///////////
!   //////////         M O C _ K I N _ B C          \\\\\\\\\\\
!================================================================

!   inputs:  everything
!  outputs:  the depth in the boundary cell

subroutine MOC_KIN_BC( i, j, rain, dt, side,  h_bound, dev)

use shared    , only: K, por, b_pfc, n_mann, CV_Info, wid, &
                     imax, jmax, lng, wid, h_old, Z, eta_0_hp2_max
use pfc2dsubs, only: set_xyh
use pfc2Dfuns, only: F_LinearIndex
use utilities, only: BILINEAR_INTERP, F_PythagSum


integer, intent( in ) :: i, j
real, intent ( in )    :: dt       ! timestep
character(5), intent(in) :: side ! which side of the domain are we working on
real, intent( in ) :: rain        ! rainfall rate for this timestep
real, intent( out )    :: h_bound
integer, optional :: dev  !device for outputing errors

real, dimension( 2 ) :: ksi_ii1  !vector in the ksi direction from point i to i+1
real, dimension( 2 ) :: eta_jj1  !vector in the eta direction from point j to j+1
real, dimension( 2 ) :: S_ksi    ! slope vector in the ksi direction
real, dimension( 2 ) :: S_eta    ! slope vector in the eta direction
real, dimension( 2 ) :: S_drain
real, dimension( 2 ) :: S_drain_unit  ! slope vector for drainage slope
real :: drain_slope  ! magnitude of drainge slope

integer :: v
integer :: vi1  !value of v for the cell i+1
integer :: vj1  !value of v for the cell j+1
integer :: vjm1 !value of v for the cell j-1
integer :: vim1 !value of v for the cell i-1
integer :: v1, v2, v3, v4  ! global index for interpolation points
! Get a vector that points up the drainage slope from the point i,j


```

```
! Bilinear Interpolation
real :: XX, YY        ! Coordinates of point where depth is interpolated
real :: x1, y1, h1    ! Coordinates of point 1  Interpolation points
real :: x2, y2, h2    !    "    "       point 2
real :: x3, y3, h3    !    "    "       point 3
real :: x4, y4, h4    !    "    "       point 4

! Method of Characteristics
!   PFC
real :: dx_moc, hp1, hp2, hp2_max
!   Sheet flow
real :: ds, hs1, hs2


integer :: device
logical :: bilin_err

!--------------------------------------------------------------------------------

! Default values for output device
if( present( dev ) .EQV. .FALSE. ) then
        device = 6
else
        device = dev
end if

!-----------------------------------------------------------------------
!   D R A I N A G E   S L O P E   C A L C U L A T I O N S
!-----------------------------------------------------------------------

! setup to figure out the slope components in
! the i (ksi)  and j (eta) directions
v    = F_LinearIndex( i  , j  , jmax )
vi1  = F_LinearIndex( i+1, j  , jmax )
vj1  = F_LinearIndex( i  , j+1, jmax )
vjm1 = F_LinearIndex( i  , j-1, jmax )
vim1 = F_LinearIndex( i-1, j  , jmax )



!---------------------------------------------------
! Compute unit vectors in the  longitudinal (ksi)
!    and tranverse (eta) directions.  If statements
!    are careful around the boundaries
!---------------------------------------------------


! LONGITUDINAL DIRECTION (ksi)
if( side == 'north' .or. side == 'south' ) then
    ksi_ii1 = (/  CV_info(vi1)%X - CV_Info(v)%X ,   &
                  CV_info(vi1)%Y - CV_Info(v)%Y      /)

elseif( side == 'east ' )  then
    ksi_ii1 = (/  CV_info(v)%X - CV_Info(vim1)%X ,   &
```

```
                   CV_info(v)%Y - CV_Info(vim1)%Y      /)

endif


! TRANSVERSE DIRECTION (eta)

if( side == 'south' ) then

        eta_jj1 =   (/  CV_info(vj1)%X - CV_Info(v)%X ,   &
                        CV_info(vj1)%Y - CV_Info(v)%Y      /)

elseif( side == 'north' ) then

        eta_jj1 = - (/  CV_info(vjm1)%X - CV_Info(v)%X,   &
                        CV_info(vjm1)%Y - CV_Info(v)%Y     /)


elseif( side == 'east ' ) then

    if( j  /= jmax ) then
        ! j+1 is OK
        eta_jj1 = (/  CV_info(vj1)%X - CV_Info(v)%X ,   &
                      CV_info(vj1)%Y - CV_Info(v)%Y      /)
    elseif( j == jmax ) then
        ! special treatment for jmax
        eta_jj1 = (/  CV_info(v)%X - CV_Info(vjm1)%X ,   &
                      CV_info(v)%Y - CV_Info(vjm1)%Y      /)
    endif

endif


!write(device,*) 'Direction Vectors: ksi_ii1 = ', ksi_ii1,  &
!                                   ' eta_jj1 = ', eta_jj1

!Make the direction vectors of unit length
ksi_ii1 = ksi_ii1 / F_PythagSum( ksi_ii1(1), ksi_ii1(2) )
eta_jj1 = eta_jj1 / F_PythagSum( eta_jj1(1), eta_jj1(2) )

!write(device,*)'Direction UNIT Vectors: ksi_ii1 = ', ksi_ii1, &
!                                        ' eta_jj1 = ', eta_jj1


!-----------------------------------------------------------
! Compute a slope vector for each direction by
!   estimating the magnitude and using the unit vectors
!   obtained above for the directions
!-----------------------------------------------------------

! LONGITUDINAL DIRECTION (ksi)
if(  side == 'north' .or. side == 'south' ) then

    S_ksi = ksi_ii1 * ( Z( i+1, j   ) - Z( i, j ) ) / lng( i, j )
```

```
elseif( side == 'east ' ) then
    S_ksi = ksi_ii1 * ( Z( i, j   ) - Z( i-1, j ) ) / lng( i, j )
end if

! TRANSVERSE DIRECTION (eta)
if( side == 'south') then
    S_eta =   eta_jj1 * ( Z( i  , j+1 ) - Z( i, j ) ) / wid( i, j )
elseif( side == 'north' ) then
    S_eta = - eta_jj1 * ( Z( i  , j-1 ) - Z( i, j ) ) / wid( i, j )
elseif( side == 'east ' ) then
    if( j /= jmax ) then
        S_eta =   eta_jj1 * ( Z( i  , j+1 ) - Z( i, j ) ) / wid( i, j )
    elseif( j == jmax ) then
        S_eta =  eta_jj1 * (  Z( i, j) - Z( i, j-1) ) / wid( i, j )
    endif
end if

!--------------------------------------------------
! Compute vector for the drainage slope ( S_drain )
! and its magnitude, and unit vector for direction
!-------------------------------------------------

! compute drainage slope vector
S_drain = S_ksi + S_eta
! and the magnitude
drain_slope = F_PythagSum( S_drain(1), S_drain(2) )
! and a drainage slope unit vector
S_drain_unit = S_drain / drain_slope

!write( device, *) 'Slope Vectors:  S_ksi = ', S_ksi, ' S_eta', S_eta

!---------------------------------------------------------------------------
!    I N T E R P O L A T I O N    P O I N T S
!---------------------------------------------------------------------------
```

```
! now we can figure out which points to use for the
! bilinear interploation routine. Points must be specified
! counter-clockwise around the perimeter:
!
!       4-----3
!       |     |
!       |     |
!       1-----2

if    ( side == 'south'  .AND.  S_drain_unit(1) .LE. 0. )  then
        ! This is the southern boundary and
        ! The domain slopes from left to right
        !Point 1
        call set_xyh( i-1, j  , x1, y1, h1 )
        !Point 2
        call set_xyh( i  , j  , x2, y2, h2 )
        !Point 3
        call set_xyh( i  , j+1, x3, y3, h3 )
        !Point 4
        call set_xyh( i-1, j+1, x4, y4, h4)

elseif( side == 'south'  .AND.  S_drain_unit(1) .GE. 0.0 ) then
        ! This is the southern boundary and
        ! The domain slopes from right to left
        ! Point 1
        call set_xyh( i  , j  , x1, y1, h1 )
        ! Point 2
        call set_xyh( i+1, j  , x2, y2, h2 )
        !Point 3
        call set_xyh( i+1, j+1, x3, y3, h3 )
        !Point 4
        call set_xyh( i  , j+1, x4, y4, h4 )

elseif( side == 'north' .AND. S_drain_unit(1) .LE. 0.0 ) then
        ! This is the northern boundary and
        ! The domain slopes from left to right
        call set_xyh( i-1, j-1, x1, y1, h1 )
        call set_xyh( i  , j-1, x2, y2, h2 )
        call set_xyh( i  , j  , x3, y3, h3 )
        call set_xyh( i-1, j  , x4, y4, h4 )

elseif( side == 'north' .AND. S_drain_unit(1) .GE. 0.0 ) then
        ! This is the northen boundary and
        ! The domain slopes from right to left
        call set_xyh( i  , j-1, x1, y1, h1 )
        call set_xyh( i+1, j-1, x2, y2, h2 )
        call set_xyh( i+1, j  , x3, y3, h3 )
        call set_xyh( i  , j  , x4, h4, h4 )


elseif( side == 'east ' .AND. S_drain_unit(2) .GE. 0.0 ) then
        ! This is the eastern boundary and
        ! and uphill is the positive Y direction
```

```
        call set_xyh( i-1, j  , x1, y1, h1 )
        call set_xyh( i  , j  , x2, y2, h2 )
        call set_xyh( i  , j+1, x3, y3, h3 )
        call set_xyh( i-1, j+1, x4, y4, h4 )
elseif( side == 'east ' .AND. S_drain_unit(2) .LT. 0.0 ) then
        ! This is the eastern boundary and
        ! and uphill is the negative Y direction
        call set_xyh( i-1, j-1, x1, y1, h1 )
        call set_xyh( i  , j-1, x2, y2, h2 )
        call set_xyh( i  , j  , x3, y3, h3 )
        call set_xyh( i-1, j  , x4, y4, h4 )
endif
!------------------------------------------------------------------
!   M E T H O D    O F    C H A R A C T E R I S T I C S
!------------------------------------------------------------------
! Reset v to confirm we're in the right cell
v = F_LinearIndex( i, j, jmax )
MOC:if( h_old(i,j) .LE. b_pfc) then
!------------------
!PFC FLOW
!-----------------
        ! Sheet flow has not started yet
        ! use MOC to estimate the solution at the next time step
        ! figure out how far up the drainage slope to go
        dx_moc =  K * (drain_slope) * dt / por
!        write( device, * ) 'MOC_KIN_BC: i = ', i, ' j = ', j, 'pfc char len = ', dx_moc
        ! and the coordinates of this location
        XX = CV_Info( v ) % X  +  dx_moc * S_drain_unit( 1 )
        YY = CV_Info( v ) % Y  +  dx_moc * S_drain_unit( 2 )
        ! use bilinear interpolation to find the
        ! thickness (hp1) at this location
        call BILINEAR_INTERP(  XX, YY, hp1,   &
                               x1, y1, h1 ,   &
                               x2, y2, h2 ,   &
                               x3, y3, h3 ,   &
                               x4, y4, h4 ,   &
                               device, bilin_err     )
        ! value at next time step
        hp2 = hp1 + rain * dt / por
        ! set maximum value for hp2 (1D flow)
        if( rain .LT. TINY ( rain )) then
                ! Rainfall rate is effectively zero
                hp2 = hp2   ! Eqv to hp2 = hp1
```

```
        else
                ! Rainfall is non-zero, set a maximum value for hp2
                hp2 = min( hp2, sum(wid(i,:))*rain/K/drain_slope)
                ! Use hp1 (basically zero rainfall) if there
                !  is a decrease in depth
                if( hp2 .LT. hp1 ) then
                        hp2 = hp1
                end if
        endif

!
!      ! Error checking for eastern boundary
!      if( i == imax ) then
!         if( j == jmax -5 .or. j == jmax/2 .or. j == 5 ) then
!
!            write( device, *)   'MOC_KIN: i =', i ,          &
!                                         'j =', j ,          &
!                                   'S_drain =', S_drain,     &
!                              ' drain_slope =', drain_slope, &
!                             ' S_drain_unit =', S_drain_unit
!            write(device,*) 'Bilinear Interpolation'
!            write(device,*) '       X,          Y,              h,'
!            write(device,32) 0, XX, YY, hp1
!            write(device,32) 1, x1, y1,  h1
!            write(device,32) 2, x2, y2,  h2
!            write(device,32) 3, x3, y3,  h3
!            write(device,32) 4, x4, y4,  h4
!         end if
!      endif
!

        ! error checking for interpolation
        if( bilin_err .eqv. .true. ) then
            write(device,*) 'MOC_KIN_BC: Bilinear interpolation error &
                            & for grid cell i = ', i, ' j = ', j
            write(device,*)         'S_drain=', S_drain,      &
                               ' drain_slope=', drain_slope,  &
                              ' S_drain_unit=', S_drain_unit
            write(device,*)              'hp2=', hp2    ,  &
                                      'dx_moc=', dx_moc,   &
                                         'hp1=', hp1    ,  &
                                        'rain=', rain   ,  &
                                          'dt=', dt     ,  &
                                         'por=', por

            write(device,* ) 'Interploation points/result:'
            write(device,* ) '       X,          Y,              h,'
            write(device,32) 0, XX, YY, hp1
            write(device,32) 1, x1, y1,  h1
            write(device,32) 2, x2, y2,  h2
            write(device,32) 3, x3, y3,  h3
            write(device,32) 4, x4, y4,  h4

        end if
```

```fortran



        if( i == imax/2  .OR. j == jmax/2  ) then
           write(device,*) 'PFC Flow MOC BC: i=', i      , &
                                            'j=', j      , &
                                          'hp2=', hp2    , &
                                       'dx_moc=', dx_moc, &
                                          'hp1=', hp1    , &
                                         'rain=', rain   , &
                                           'dt=', dt     , &
                                          'por=', por
        endif
        h_bound = hp2
else
!-----------------
!SHEET FLOW
!-----------------
        hs2 =  h_old(i,j) - b_pfc
        ! Handle Zero Rainfall
        if( rain .LT. TINY( rain ) ) then
            ! there is no increase in flow rate along the drainage path
            !  and ds becomes arbitrary so use the characteristic length for PFC flow
            ! b/c you might need it later
            ds = K * ( drain_slope ) * dt / por
        else
            ds = sqrt( drain_slope ) / n_mann / rain *      &
                 ( ( hs2 + rain*dt )**(5./3.) - hs2**(5./3.) )
        end if
       ! interpolate up the drainage slope to find hs1
        XX = CV_Info( v ) % X  +  ds * S_drain_unit( 1 )
        YY = CV_Info( v ) % Y  +  ds * S_drain_unit( 2 )
        ! use bilinear interpolation to find the thickness (hs1) at this location
        call  BILINEAR_INTERP(  XX, YY, hs1,  &
                                x1, y1, h1 ,  &
                                x2, y2, h2 ,  &
                                x3, y3, h3 ,  &
                                x4, y4, h4 ,  &
                                device, bilin_err )
        if( bilin_err .eqv. .true. ) then
            write(device,*) 'MOC_KIN_BC: Bilinear interpolation error &
                            & for grid cell i = ', i, ' j = ', j

              write(device,*) '       X,          Y,               h,'
              write(device,32) 0, XX, YY, hs1
              write(device,32) 1, x1, y1,  h1
              write(device,32) 2, x2, y2,  h2
              write(device,32) 3, x3, y3,  h3
              write(device,32) 4, x4, y4,  h4

        end if

        ! subtract off the pavement thickness
```

```
        hs1 = hs1 - b_pfc
        !Handle return to PFC flow
        if( hs1 .GT. 0. ) then
            !we have sheet flow
            !Output some summary info
            if( i == imax/2 .or. j == jmax / 2 ) then
               write(device,*) 'Sheet Flow MOC BC: i=',   i, &
                                                  'j=',   j, &
                                                'hs2=', hs2, &
                                                 'ds=',  ds, &
                                                'hs1=',  hs1
            endif
            !checking for good values of inputs
            if( hs1 .LT. 0.  .OR.  hs2 .LT. 0.  .OR. rain .LT. 0.) then
                write(device,*) 'Sheet Flow MOC BC: i=',i, 'j=',j, &
                                 'hs1=', hs1, 'hs2=',hs2, 'rain=',rain
            end if
            ! return value for the boundary
            h_bound =  b_pfc + (   hs1**(5./3.) +              &
                                 ( hs2 + rain*dt )**(5./3.) - &
                                   hs2**(5./3.) )**0.6
        else
            ! the upstream point does not have sheet flow
            ! use PFC characterisitic
            h_bound = hs1 + b_pfc + rain * dt / por
        end if

end if MOC



!-----------------------------------------------------------------
! Format statements
31  format( 3( F12.7, '  ') )
32  format( I3, '  ', 3(F12.7, '  ') )

!-----------------------------------------------------------------
end subroutine MOC_KIN_BC
!=================================================================
!   \\\\\\\\\\    E N D      S U B R O U T I N E     ///////////
!   //////////         M O C _ K I N _ B C          \\\\\\\\\\\
!=================================================================





!==========================================================================
!   \\\\\\\\\                                       ///////////
                        END MODULE BoundCond
!   /////////                                       \\\\\\\\\\\\
!==========================================================================
```

REFERENCES

Anderson, M.P. and Woessner, W.W. (1992), Applied Groundwater Modeling: Simulation of Flow and Advective Transport, Academic Press, San Diego.

Bird, R.B. Stewart, W.E. and Lightfoot, E.N. (1960). Transport Phenomena, John Wiley and Sons, Inc., Madison, WI.

Beavers, G.S., and Joseph, D.D. (1967), Boundary conditions at a naturally permeable wall. J. Fluid Mech. 30:197–207.

Bear, Jacob. (1972), Dynamics of Fluids in Porous Media, Elsevier, New York.

Barrett, Michael (2006). Stormwater Quality Benefits of a Porous Asphalt Overlay. Center for Transportation Research, Austin, Texas. Report No. FHWA/TX-07/0-4605-2.

Barrett, M.E., Klenzendorf, J.B., Eck, B. J., and Charbeneau, R.J. (2009), Water Quality and Hydraulic Properties of the Permeable Friction Course, Proceedings of the World Environmental and Water ResourcesConference 2009, Kansas City, MO, May 17-21, 2009.

Berbee, R., G. Rijs, R. de Brouwer, and L. van Velzen (1999), Characterization and Treatment of Runoff from Highways in the Netherlands Paved with Impervious and Pervious Asphalt, Water Environment Research, 71(2), 183-190.

Charbeneau, R. J. (2000), Groundwater Hydraulics and Pollutant Transport, Waveland Press, Long Grove, IL.

Charbeneau, R.J. and Barrett, M.E. (2008), Drainage Hydraulics of Permeable Friction Courses, Water Resources Research 44, W04417.

Charbeneau, R. J., Jeong, J. and Barrett, M.E. (2009). Physical Modeling of Sheet flow on Rough Impervious Surfaces, Journal of Hydraulic Engineering, Vol 135. No. 6.

Chow, V.T., D.R. Maidment, and L.W. Mays (1988), Applied Hydrology, McGraw-Hill, New York.

Eck, B.J., Barrett, M.E. and R.J. Charbeneau (2010), Note on Modeling Surface Discharge from Permeable Friction Courses, Water Resources Research (Under Review).

Dabaghmeshin, M. (2008), Modeling the Transport Phenomena within the Arterial Wall: Porous Media Approach. Thesis for the degree of Doctor of Science. Lappeenranta University of Technology, Lappeenranta, Finland. Accessed Online (18 Nov 08): https://oa.doria.fi/bitstream/handle/10024/42280/isbn9789522146274.pdf?sequence=2

Daluz Vieira, J.H. (1983), Conditions Governing the Use of Approximations for the Saint-Venant Equations for Shallow Surface Water Flow. Journal of Hydrology, 60: 43-58.

Ergun, S. (1952), Fluid Flow Through Packed Columns, Chemical Engineering Progress, Vol 48, No.2, pp 89-94.

Ferziger, J.H. and Peric, M. (2002), Computational Methods for Fluid Dynamics, Springer, Berlin.

Furman, A, (2008), Modeling Coupled Surface-Subsurface Flow Processes: A Review. Vadose Zone Journal, 7:741-756.

Google Inc. (2010). Google Earth (Version 5.1.3533.1731) [Software]. Available from http://earth.google.com/

Halek, V. and J. Svec. (1979), Groundwater Hydraulics. Elsevier, New York.

He, Z., Wu, W. and Wang, Sam S. Y. (2008), Coupled Finite-Volume Model for 2D Surface and 3D Subsurface Flows. Journal of Hydrologic Engineering, Vol. 13 No. 9.

Irmay, S. (1967), On the Meaning of the Dupuit and Pavlovskii Approximations in Aquifer Flow, Water Resources Research Vol. 3, No. 2, pp 599-608.

Jeong, J. (2008), A Hydrodynamic Diffusion Wave Model for Stormwater Runoff on Highway Surfaces at Superelevation Transitions. Dissertation. University of Texas at Austin.

Jeong, J. and Charbeneau, R. J., (2010), Diffusion Wave Model for Simulating Stormwater Runoff on Highway Pavement Surfaces at Superelevation Transition, Journal of Hydraulic Engineering, (In Press).

Klenzendorf, J. B. (2010), Hydraulic Conductivity Measurement of Permeable Friction Course (PFC) Experiencing Two-Dimensional Nonlinear Flow Effects. Dissertation. University of Texas at Austin.

Kreyzig, E. (1999), Advanced Engineering Mathematics, 8th Edition. John Wiley and Sons, New York.

Kollet, S. J., and Maxwell, R. M. (2006), Integrated surface-groundwater flow modeling: A free-surface overland flow boundary condition in a parallel groundwater flow model. Adv. Water Resour.,129, 945–958.

Kovacs, G. (1981), Seepage Hydraulics. Elsevier, New York.

Li, D. and Engler, T.W., (2001), Literature Review on Correlations of the Non-Darcy Coefficient. SPE 70015, in: Proceedings of the SPE Permian Basin Oil and Gas Recovery Conference, Midland, Texas, USA, May 15-16.

Liang, D., Falconer, R.A., and Lin, B. (2007), Coupling surface and subsurface flows in a depth averaged flood wave model. Journal of Hydrology, 337:147-158.

Loaiciga, H. A. (2005), Steady state phreatic surfaces in sloping aquifers, Water Resources Research 41, W08402, doi:10.1029/2004WR003861.

NCHRP: National Cooperative Highway Research Program (2009), Construction and Maintenance Practices for Permeable Friction Courses, Report 640, Transportation Research Board, Washington, D.C..

Ranieri, V. (2002), Runoff Control in Porous Pavements, Transpation Research Record.1789, pp.46-55.

Refsgaard, J.C., and B. Storm. (1995), MIKE-SHE. p. 809–846. In V.P. Singh (ed.) Computer models of watershed hydrology. Water Resour. Publ., Highlands Ranch, CO.

Ruth, D. and Ma, H. (1992), On the Derivation of the Forchheimer Equation by Means of the Averaging Theorem. Transport in Porous Media 7: 255-264.

Simpson, M.J., Clement, T.P. and Gallop, T.A. (2003), Laboratory and Numerical Investigation of Flow and Transport Near a Seepage-Face Boundary. Ground water. Vol. 41 No.5 pp690-700.

Stanard, C. E. (2008), Stormwater Quality Benefits of a Permeable Friction Course. Master's Thesis. Univeristy of Texas at Austin. Available Online: http://www.crwr.utexas.edu/reports/2008/rpt08-3.shtml

Street, R.L. (1973), The Analysis and Solution of Partial Differential Equations, Brooks/Cole, Monterey, California.

Tan, S.A., T.F. Fwa, and K.C. Chai (2004), Drainage consideration for Porous Asphalt Surface Course Design, in Transportation Research Record 1868, pp 142-149.

Thauvin, R. and Mohanty, K.K. (1998), Network Modeling of Non-Darcy Flow Through Porous Media, Transport in Porous Media 31: 19-37.

Ward, J.C. (1964), Turbulent Flow in Porous Media, Journal of Hydraulics Division, ASCE Vol 90 #HY5, pp 1-12.

White, F.M. (1999), Fluid Mechanics, Fourth Edition. WCB/McGraw-Hill.

Woolhiser, D.A. and Liggett, J.A. (1967), Unsteady, One-Dimensional Flow over a Plane—the Rising Hydrograph, Water Resources Research, Vol. 3 No. 3 753-771.

Yates, S.R., A.W. Warrick, & D.O. Lomen (1985), Hillside Seepage: An Analytical Solution to a Nonlinear Dupuit-Forchheimer Problem, Water Resources Research 21(3) 331-336.

Zeng, Z and Grigg, R. (2006), A Criterion for Non-Darcy Flow in Porous Media, Transport in Porous Media 63: 57-69.