Numerical Modelling of Wetland Hydrodynamics

by

Laura L. DePaoli

S.B., Civil and Environmental Engineering
Massachusetts Institute of Technology, June 1997

Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of

Master of Science in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

Author.....
Department of Civil and Environmental Engineering
May 7, 1999

Certified by.....
Heidi M. Nepf
Associate Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by.....
Andrew J. Whittle
Chairman, Departmental Committee on Graduate Studies

Numerical Modelling of Wetland Hydrodynamics

by

Laura L. DePaoli

Submitted to the Department of Civil and Environmental Engineering
on May 7, 1999, in partial fulfillment of the requirements for the degree of
Master of Science in Civil and Environmental Engineering

ABSTRACT

Wetlands enhance water quality via physical, chemical, and biological processes. Numerous mass balance studies have shown that suspended sediments, biological oxygen demand, nutrients, and metals are efficiently removed in natural and constructed wetlands by a variety of sink mechanisms, such as bacterial conversion, sorption, sedimentation, natural decay, volatilization, and chemical reactions. However, constructed wetland design for water treatment has been highly individualistic and treatment performance has been variable. Attempts are being made to create a more unified wetland design through a better understanding of the mechanisms in these systems. Clearly, highly efficient wetlands cannot be designed until we have gained a greater understanding of the governing processes in these systems.

The hydrodynamics of a wetland is one of the most influential factors governing its ability to enhance water quality; fluid dynamics controls the residence time of a wetland and thus the time available for water quality enhancement to take place. A depth-averaged hydrodynamic model was developed to study flow through natural and constructed free surface wetlands. This numerical model was used to investigate the effects of non-uniform plant distribution in constructed wetlands, the ability of deep zones or open-water zones to redistribute flow in these systems, and the bi-modal flow structure of natural wetlands.

Numerical experiments were performed on a typical constructed wetland (sloped rectangular basin). Non-uniform vegetation was characterized by either random sparseness or channels of sparse vegetation. Random sparseness in wetland vegetation created significant deviations from plug-flow. The hydraulic efficiency of the system decreased in proportion to increased sparseness. When channels of sparse vegetation were created, preferred flowpaths formed and the hydraulic efficiency of the system plummeted far below the efficiency of a wetland with the same plant coverage distributed randomly. Open-water zones with no rooted vegetation have been proposed by designers to redistribute this non-uniform flow, increase hydraulic detention time, and increase the overall mixing in wetlands. Hydrodynamic simulations of these zones support the claims of enhanced mixing and increased detention time, but not redistribution of flow.

Lastly, numerical experiments were performed on a generic basin geometry which simulated flow through a natural wetland (allows circulation). Two distinctly different flow structures exist for inertia-dominated and drag-dominated flow. Field data suggest that drag caused by wetland vegetation must be defined as a function of velocity in numerical models to predict this bi-modal flow structure accurately.

Thesis Supervisor: Heidi M. Nepf
Title: Associate Professor of Civil and Environmental Engineering

ACKNOWLEDGMENTS

Thanks first and foremost to Professor Heidi Nepf for providing me with the opportunity to work under her supervision and support. A special thanks to Chin-Hsien Wu for his insight and guidance in numerical modelling and throughout the development of my work. His assistance is very appreciated as well as the input and advice from fellow research group members, Hrund Andradottir and Paul Fricker.

Additional thanks are extended to Professor Harold Hemond for the use of his laboratory space and guidance with respect to the aquatic plant study. The XRF work in that study could not have been performed without the expertise of Dr. Daniel Brabander and Rachel Stanley.

Lastly, I am truly grateful for the financial support received from the National Science Foundation Hydrology Fellowship, Ralph M. Parsons Fellowship, NIEHS Grant no. P42-ES04675, and NSF Career Award EAR 9629259.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

CHAPTER 1: INTRODUCTION

1.1 Motivation

Natural wetlands have been used as convenient wastewater discharge sites since sewage has been collected, over 100 years ago. When monitoring at these discharge sites began in the 1960s and 1970s, an awareness of the water purification potential of wetlands began to emerge. Studies of wetlands' potential to treat wastewater began in Europe as early as 1952 and in the Western hemisphere during the 1970s. By 1985, worldwide acceptance of natural and constructed wetlands as a proven technology for water treatment was established (Kadlec & Knight 1996). Today, it is estimated that over 1000 constructed treatment wetlands have been implemented throughout the world (Wood 1995).

Wetlands accomplish water quality enhancement via physical, chemical, and biological processes. Numerous mass balance studies have shown that suspended sediments, biological oxygen demand, nutrients, and metals are efficiently removed in natural and constructed wetlands by a variety of sink mechanisms, such as bacterial conversion, sorption, sedimentation, natural decay, volatilization, and chemical reactions.

Constructed wetland design has been highly individualistic with variable treatment performance. Attempts are being made to create a more unified wetland design through a better understanding of the mechanisms in these systems. Most evaluations of the efficacy of natural and constructed wetlands are input-output analyses. Very little is understood about the hydrodynamic and biological properties within the black box. Clearly, highly efficient wetlands cannot be designed until we have gained a greater understanding of the governing processes in them (Wetzel 1993).

Experience through observation has demonstrated that flow in constructed wetlands is not uniform as is assumed in their design. The mixing characteristics of wastewater flowing through a wetland have been determined to be intermediate between plug-flow and well-mixed even for narrow wetlands (Kadlec, Bastiaens, & Urban 1993; Kadlec 1994; Wood 1995; Buchberger & Shaw 1995; Kadlec & Knight 1996). This is one of the largest factors responsible for decreased wetland performance. Non-uniform flow decreases the residence time in wetlands and thus the time available for reactions to

take place. If the hydrodynamics of wetlands can be linked with an accurate depiction of the biological and chemical reactions in wetlands, more informed decisions regarding constructed wetland design and natural wetland modification can be made in order to build more efficient and less expensive water treatment facilities.

1.2 Objectives

The objective of this research was to create and apply a depth-averaged hydrodynamic model to constructed and natural wetlands. Surface flow (free surface) wetlands are studied in particular because they are the most widely used constructed wetland type in the United States. By better understanding the hydrodynamics of these systems, they can be optimized for their water purifying potential.

The numerical model is used to investigate:

- the effects of non-uniform plant distribution on the flowfield of constructed wetlands,
- the ability of deep zones or open-water areas to redistribute flow, and
- the bi-modal flow structure of natural wetlands .

The hydrodynamic model created for use in this study can ultimately be developed into a water quality model to enable the examination of the fate and transport of contaminants and nutrients through free surface wetlands.

CHAPTER 2: TREATMENT WETLANDS

2.1 Introduction

Wetlands have long been thought of as roadblocks to development and swamps of little economic value. Prior to the mid-1970s, the drainage and destruction of wetlands were accepted practices in the United States and encouraged by government legislation to make space for agricultural, commercial, and residential development. Over half of the nation's original wetlands were destroyed before action to preserve them took place (Mitsch & Gosselink 1993).

In 1934, through the combined efforts of hunters, scientists, engineers, lawyers, and environmentalists, wetlands became recognized as valuable ecosystems and the U.S. government took the first step to preserving them (Mitsch & Gosselink 1993). Wetlands serve to protect fish and wildlife, produce and preserve fossil fuels, prevent floods, protect shorelines, recharge groundwater aquifers, and enhance water quality. Today, the values of wetlands are recognized and their preservation is standard policy.

2.2 Water Quality Enhancement

Natural wetlands have been used as convenient wastewater discharge sites for as long as sewage has been collected, over 100 years. When monitoring at these discharge sites began in the 1960s and 1970s, an awareness of the water purification potential of wetlands began to emerge.

Wetlands enhance water quality via physical, chemical, and biological processes. Three components are responsible for these processes: vegetation, microbial populations, and the air-water interface. The primary functions of vegetation are to create environments for microbial populations and to obstruct flow. Plants create and maintain a soil of decaying debris which provides a durable food supply and porous media for microbial growth. By obstructing flow, plants enhance settling of particulate matter from the water column. Plants can also assimilate nutrients and metals though this process is generally considered less important. Appendix A includes an exploratory study of the importance of plants in the uptake of arsenic in a specific wetland in MA. Microbial populations including microbes such as bacteria, algae, fungi, and protozoa, can remove or

transform organic substances or metallic ions. Lastly, the air-water interface allows gas exchange increasing dissolved oxygen concentrations and thereby enhancing the decomposition of organic compounds and oxidation of metallic ions (Hammer 1997).

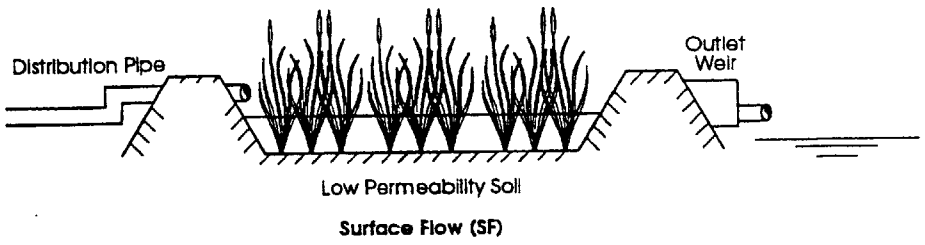
Research on constructing artificial wetlands for the purpose of water quality enhancement began in Europe as early as 1952 and in the Western hemisphere during the 1970s. By 1985, constructed and natural wetland systems had become accepted as proven technologies for improving and protecting surface water quality (Steiner & Freeman 1989; Moshiri 1993; Kadlec & Knight 1996; Hammer 1997; Vymazal et al. 1998). Today, full-scale wetland treatment systems (constructed and natural wetlands) are used routinely to treat municipal, industrial, and agricultural wastewater; agricultural and urban runoff; landfill leachate; and acid-mine drainage waters. However, the design of constructed wetlands, over 300 in North America and greater than 1000 worldwide, is still highly individualistic with variable treatment performance (Wood 1995).

2.3 Types of Constructed Wetlands

Constructed wetlands may be created for aesthetics, habitat, flood control, water treatment, or a variety of other purposes. For the purpose of this work, the constructed wetlands to which I refer are treatment wetlands, constructed for the primary function of water quality enhancement.

Two types of constructed wetlands are generally used for water treatment: surface flow and subsurface flow wetlands (Figure 2-1). Surface flow wetlands are shallow water bodies, less than one m deep, densely vegetated by a variety of plant species. Open-water areas (deep zones with no vegetation) may be incorporated into their design to optimize the hydraulics, wildlife enhancement, and pest control of these systems. Deep zones are examined from a hydrodynamic standpoint in Chapter 5. Subsurface flow wetlands use a bed of soil or gravel as a substrate for the growth of rooted wetland plants. Water flows by gravity, either horizontally or vertically through the bed substrate where it comes into contact with microbes living in association with the substrate and plant roots. This type of system is designed to minimize overland flow and is essentially flow through media (Kadlec & Knight 1996).

(a)



(b)

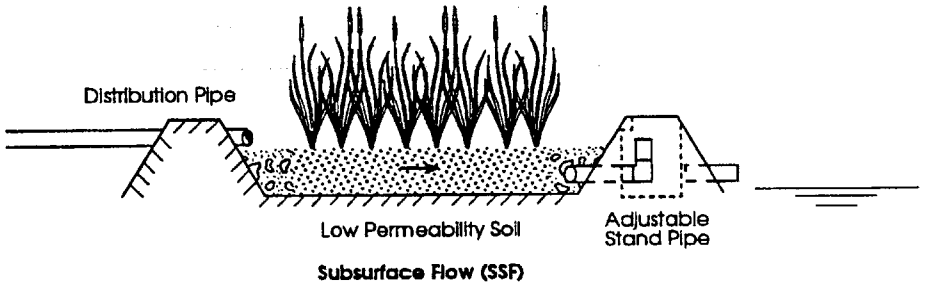


Figure 2-1. Basic Types of Constructed Treatment Wetlands: (a) surface flow, and (b) subsurface flow. Surface flow (free surface) wetlands are densely vegetated shallow water bodies. Subsurface flow wetlands use a bed of soil or gravel as a substrate for the growth of rooted wetland plants. Wastewater flows by gravity through the media; these systems are designed to minimize overland flow (Adapted from Kadlec & Knight 1996).

Europe, Australia, and South Africa predominantly use subsurface wetlands. The United States typically constructs surface flow wetlands for advanced secondary or tertiary treatment (Cooper & Findlater 1990; Reed & Brown 1992; Wood 1995; Kadlec & Knight 1996; Vymazal et al. 1998). For this reason, I have chosen to study surface flow wetlands and all further discussion unless noted otherwise will be with respect to surface flow wetlands.

2.4 Constructed Wetland Design

The design of constructed wetlands has been highly individualistic and their treatment performance has been variable. This is largely attributed to the lack of mechanism-based theory for wetlands; most work regarding constructed wetlands has been empirical or involves black box modelling. Despite this, some fundamentals of

constructed wetland design appear to be agreed upon. Many of the design parameters you will read about in this section directly address the need for uniform flow or plug-flow conditions for increased water quality potential.

2.4.1 Plug-flow

One of the most important aspects of constructed wetland design is hydraulic efficiency,

$$HE = \frac{t_{eff}}{t_{nom}} \tag{2.1}$$

where

HE represents the hydraulic efficiency of a system [-],

t_{eff} is the effective hydraulic detention time, average time water spends in a wetland, and

t_{nom} is the nominal detention time, the average time that water would spend in a wetland if the entire volume was part of the flowfield. The nominal detention time of a wetland is calculated with the following equation:

$$t_{nom} = \frac{V}{Q} \tag{2.2}$$

where

V is the volume of the wetland, and

Q is the flowrate into and out of the wetland [m^3/s]. Hydraulic efficiency plays such an important role in the ability of a wetland to enhance water quality because the longer the effective detention time (t_{eff}), the longer the time for physical, chemical, and biological reactions to take place.

Since constructed wetland reactions, reactions enhancing water quality, are believed to be first-order, constructed wetlands are designed to perform as plug-flow. Plug-flow is uniform flow in which fluid particles pass through a basin and are discharged in the same sequence in which they entered. The fluid remains in the basin for a time equal to the nominal detention time (t_{nom}). Plug-flow is the optimal flow regime for constructed wetland design because a greater basin volume is necessary for other flow regimes to achieve the same water quality enhancement under identical operating conditions. A large

number of completely-mixed basins in series (approximately ten) can achieve the equivalent efficiency of plug-flow, but this approach is typically too costly to construct.

As a result, any deviation from plug-flow or the existence of dead zones, zones that do not communicate with the main flow, decreases the amount of time water resides in the wetland. For this case, design which assumed uniform flow would overestimate the water quality enhancement of the wetland. Being able to predict the hydrodynamics of a system using a hydrodynamic model will provide designers with knowledge enabling them to design more efficient constructed wetlands.

2.4.2 Basin geometry

Constructed wetlands are most commonly designed as rectangular-shaped basins. They are constructed with inflow and outflow applied over the entire width of the wetland. This design attempts to create a uniform flowfield across the width of the basin to reduce the existence of dead zones. A V-shaped basin pointing downstream may provide an alternative to a rectangular basin without causing dead zones, though this design has not been used very often in practice. Circular- or elliptical-shaped wetlands are likely to have dead zones and are not suggested.

The average water depth of a constructed wetland is approximately 0.5 m (single basin). See Table 2-1 for a compilation of recommended design criteria from multiple design references. Though many constructed wetlands consist of a single basin, there has been a recent trend toward the construction of multiple component systems in which several distinct basins are placed in series. The first compartment is typically a shallow basin with densely growing vegetation in 10 to 20 cm of water. The second compartment is a pond with depth of 0.75 to 1.0 m. Duckweed (non-rooted macrophytes) typically grow on the surface of this pond and submerged vegetation is planted in shallow regions to increase the microbial attachment area. The last component is another shallow basin with dense vegetation in 10 to 20 cm of water. The goal of multiple component systems is to utilize both aerobic and anaerobic conditions for a given constructed wetland system (Hammer 1995).

TABLE 2-1: RECOMMENDED DESIGN CRITERIA FOR SURFACE FLOW CONSTRUCTED WETLANDS				
Reference	Aspect Ratio	Detention Time (d)	Water Depth (m)	Bed Slope
Steiner & Freeman (1989)	> 10	-	-	<0.5 %
Watson & Hobson (1989)	>10	7 - 14	< 0.6	0 - 1 %
Reed & Brown (1992)	>5	-	0.3 - 0.45	0 - 1 %
Tchobanoglous (1993)	1	-	-	-
Witthar (1993)	-	5	< 0.46	-
Kent (1994)	-	-	0.5 - 0.6	-
Kadlec & Knight (1996)	2	14	< 0.4	-
Reed, Crites, & Middlebrooks (1995)	< 3		0.3	
Wood (1995)	2 - 10	5 - 14	0.1 - 0.5	-
Hammer (1997)	3 - 4	-	-	<0.05 %

2.4.3 Aspect ratio

The length-to-width (aspect) ratio of a wetland basin is important because it affects the flow distribution and potential for hydraulic short-circuiting, one portion of the flow moving through the basin in a time much less than $t_{nom,}$ the nominal detention time. Theoretically, a constructed wetland with a high aspect ratio is no better for treatment than one with a lower aspect ratio as long as flow is distributed uniformly. However, tracer studies have demonstrated that the flow hydraulics in constructed wetlands are not uniform in practice and long, narrow wetlands promote plug-flow behavior, though never reaching ideal plug-flow (Kadlec, Bastiaens, & Urban 1993; Kadlec 1994; Buchberger & Shaw 1995; Wood 1995; Kadlec & Knight 1996). For this reason, early constructed wetland designs created long and narrow basins with high aspect ratios often greater than ten.

Today, decisions regarding aspect ratio are a tradeoff between construction for plug-flow (enhanced treatment efficiency) and construction costs. Higher aspect ratios promote plug-flow behavior, but also increase the length of berms needed to enclose a given wetland area. Designers have recently begun utilizing smaller aspect ratios, 3 or 4, rather than those reported earlier in the literature of at least 10 (Buchberger & Shaw 1995). This trend from larger to smaller aspect ratios with time can be seen in Table 2-1.

2.4.4 Aquatic vegetation

A common obstacle to creating an efficient constructed wetland is establishing wetland plant communities evenly over the basin. When the wetland is flooded, some wetland species may die, leaving unvegetated areas. Several years may pass before these bare or thinly vegetated areas are covered by adapted wetland plants. Vegetation is almost certain to be uneven in coverage and will fill in and undergo species composition changes over time (Kadlec & Knight 1996). In addition, trails left by wildlife and humans also contribute to uneven vegetation.

In 1988, a survey of reed (*Phragmites australis*) growth in 81 subsurface flow treatment systems was performed in the United Kingdom (Parr 1990). Though such an in-depth study of aquatic vegetation for free surface flow wetlands is unavailable, the aforementioned study for subsurface wetlands can provide insight into the problems of plant coverage experienced in free surface flow wetlands.

The U.K. survey found generally poor reed establishment with only 35% of wetlands having over 90% coverage of reeds. The reed cover ranged from uniformly good to uniformly bad, but most basins had uneven coverage where patches of reeds grew sporadically. The main factors accounting for the variation in reed growth between wetlands were the age of the bed, the soil type, the type of feed, and type of reed material planted. It is approximated that one should be able to achieve 100% vegetative coverage in two years (Parr 1990).

Even and abundant vegetation is important because plants are responsible for much of the water quality enhancement in wetlands. Uneven coverage creates non-uniform flow and perhaps even short-circuiting if channels of sparse vegetation are created. Kadlec and Knight (1996) recommend plant densities of approximately 100,000 plants/hectare with one m spacing for free surface flow wetlands to insure good coverage of a wetland. In Chapter 5, the effects of uneven plant distributions in constructed wetlands are explored from a hydrodynamic standpoint.

2.4.5 Flow control strategies

To control flow through constructed wetlands, many features can be incorporated into constructed wetland design. Open-water zones, unable to support rooted vegetation,

are suggested to redistribute flow to plug-flow conditions and increase hydraulic residence time (Knight & Iverson 1990; Reed & Brown 1992; Reed, Crites, & Middlebrooks 1995; Kadlec & Knight 1996). Large slopes can cause short-circuiting, so slopes of less than 1% are recommended (Table 2-1). Finger dikes are often used to mitigate short-circuiting (Watson & Hobson 1989) as well as plant rows which run perpendicular to the direction of flow (Hammer 1997). Lastly, longitudinal baffles can increase the effective aspect ratio of a basin thus promoting plug-flow behavior.

2.5 Closing Remarks

The scientific community is getting closer to achieving a consensus on successful approaches to treatment wetland operation and management. Developing a better understanding of the mechanisms responsible for the water purification potential of wetlands has become a major effort. Both understanding the hydrodynamics, and biological and chemical reactions in wetlands will enable better informed constructed wetland design and natural wetland modification. Since constructed wetlands are driven by the need to achieve plug-flow for optimal water quality enhancement, it is imperative to study the hydrodynamics of constructed wetlands. Using a depth-averaged hydrodynamic model created for this purpose, I will showcase the benefit of a hydrodynamic approach to studying wetlands in Chapter 5 entitled "Model Applications".

CHAPTER 3: HYDRODYNAMIC MODEL

3.1 Overview

A numerical model of depth-averaged turbulent flow was developed in a boundary-fitted curvilinear coordinate system for application to surface flow wetlands. Depth-averaged models are often applied to free surface systems because of their efficiency and reasonable accuracy. These models are considered accurate when the width-to-depth ratio of a basin is large and the vertical variations in mean-flow quantities are insignificant (Ye & McCorquodale 1997). Though most wetlands have been modelled as series of mixed reactors or as plug-flow, tracer studies have shown that the flow dynamics in wetlands are much more complicated (Kadlec, Bastiaens, & Urban 1993; Kadlec 1994; Buchberger & Shaw 1995; Wood 1995; Kadlec & Knight 1996). Despite the fact that understanding the hydrodynamics of wetlands is crucial to understanding fate and transport in them, the application of hydrodynamic models to wetlands is limited. Wetland studies using hydrodynamic models include Guardo and Tomasello (1995), Barrett (1996), Prescott (1996), and Walker (1998).

This numerical model uses a finite difference approximation of the non-conservative, depth-averaged momentum and continuity equations. Inputs to the model include bathymetry, wind, bed friction, horizontal turbulent eddy viscosity, initial conditions, and boundary conditions. The model produces a depth-averaged velocity field and water surface elevation as output. The computer program is written in FORTRAN 90 and output files are directed to Matlab® for visualization of the results. The source code is presented in Appendix B.

Creating the hydrodynamic model involved several distinct steps. First, to capture complex basin geometry, body-fitted grid generation was employed instead of traditional stair-stepped grid generation. After creating a grid generation program, the governing equations had to be transformed to the orthogonal computational space created by the boundary-fitted grid. The transformed governing equations were then discretized and solved using a semi-implicit scheme which is second order in space and first order in time.

The intent in developing this model was to produce a tool for understanding the hydrodynamics in surface flow wetlands. More sophisticated schemes can and have been developed by numerical modelers, but they were not necessary for this study.

3.2 Model Capabilities

This model was created with flexibility in order to make it useful for work beyond that presented in this thesis. Despite this, all models possess constraints and for a given application this hydrodynamic model may be inappropriate. When using a numerical model, it is crucial to fully understand how it was developed and the inherent constraints associated with it.

This model can accommodate spatially heterogeneous bed stress to simulate plant heterogeneity and uneven multi-directional wind across a system. Transient precipitation can be applied to the basin surface and irregular bathymetry and geometry can be simulated. In addition, the user has the flexibility of assigning one of five boundary conditions to the inlet and outlet. It is not required that the same boundary conditions be used at both the inlet and outlet. Each boundary condition may be held constant or vary with time. The five boundary condition options include:

- water depth (h),
- water surface gradient ($\partial h/\partial x$),
- depth-averaged velocity in the x-direction ($\overline{u}$),
- velocity gradient ($\partial \overline{u}/\partial x$), and
- flowrate per unit width (q).

It is assumed that only one inlet and one outlet exist and they must be defined to be perpendicular to the x-coordinate system.

This model allows predictive design of constructed wetlands. The importance of each design variable can be evaluated and new design methods can be assessed for feasibility and efficiency. In addition, the hydrodynamic model created for this study can be developed into a water quality model to examine the fate and transport of contaminants and nutrients through free surface or surface flow wetlands. Most importantly, the use of this depth-averaged numerical model will provide a better description of the

hydrodynamics in wetland systems than the plug-flow approximation commonly used in wetland design.

3.3 Boundary-fitted Grid Generation

3.3.1 Motivation

Finite difference methods for solving differential equations require physical space to be discretized into a uniform orthogonal grid. It is imperative in the numerical solution of partial differential equations that boundary conditions be represented accurately since they are generally dominant in the characterization of the solution. The application of boundary conditions requires the boundaries of physical space to coincide with grid lines. In addition, accurate resolution of a solution by finite difference requires grid points to be clustered in regions of large gradients and economy requires them to be spread out in regions of small gradients. These requirements are generally incompatible with a Cartesian coordinate grid.

Prior to the 1980s, most models used a stair-stepped boundary to represent curvilinear boundaries (Figure 3-1). Using the stair-stepped technique will either result in poor resolution of a natural irregular boundary or necessitate a more refined grid space near boundaries to conform to the physical boundary increasing the computational time of the model. Thompson (1980) developed the boundary-fitted curvilinear coordinate method of grid generation which solves the problems of the stair-stepped grid method. This procedure eliminates boundary shape as a complicating factor by transforming non-uniform, non-orthogonal physical space (x,y) into uniform orthogonal computational space (ξ,η) (Figure 3-2). This method allows all computation to be performed on a transformed, orthogonal, computational grid regardless of the shape of the boundaries in physical space. Boundary conditions can be easily implemented since physical boundaries lie on the coordinate lines of computational space. In addition, a rectangular computational grid simplifies programming of the difference equations. There is a price to be paid; the governing equations become more complicated due to the transformation of coordinate systems.

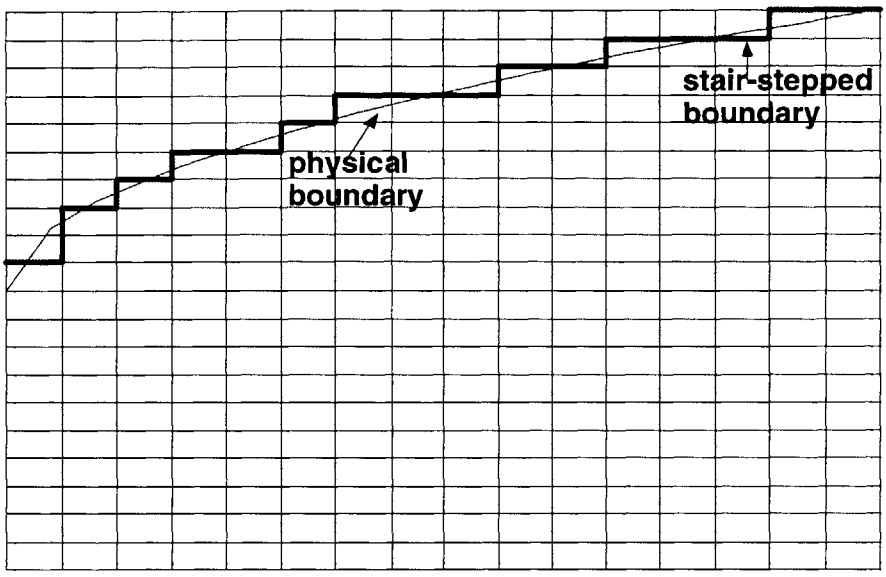


Figure 3-1. Stair-stepped Boundary. Use of a stair-stepped boundary to approximate a curvilinear boundary. Using this technique results in poor resolution of the boundary and perhaps inaccurate solutions. A more refined stair-stepped grid can produce better results, but only at a large computational cost.

The transformation relating physical space and computational space is specified by the direct transformation

$$\left.\begin{aligned}\xi &= \xi(x, y)\\ \eta &= \eta(x, y)\end{aligned}\right\} \qquad \textbf{(3.1a,b)}$$

and inversely by

$$\left.\begin{aligned}x &= x(\xi, \eta)\\ y &= y(\xi, \eta)\end{aligned}\right\}. \qquad \textbf{(3.2a,b)}$$

The determination of this transform is called grid generation. Once the transformation is determined, the governing differential equations of the system must be transformed from physical space (x,y) to computational space (ξ,η). The transformation of the governing equations for this model is performed in Section 3.5.

3.3.2 Orthogonality

At the beginning of any numerical study using boundary-fitted coordinates, it is necessary to decide whether the physical grid system should be orthogonal or non-

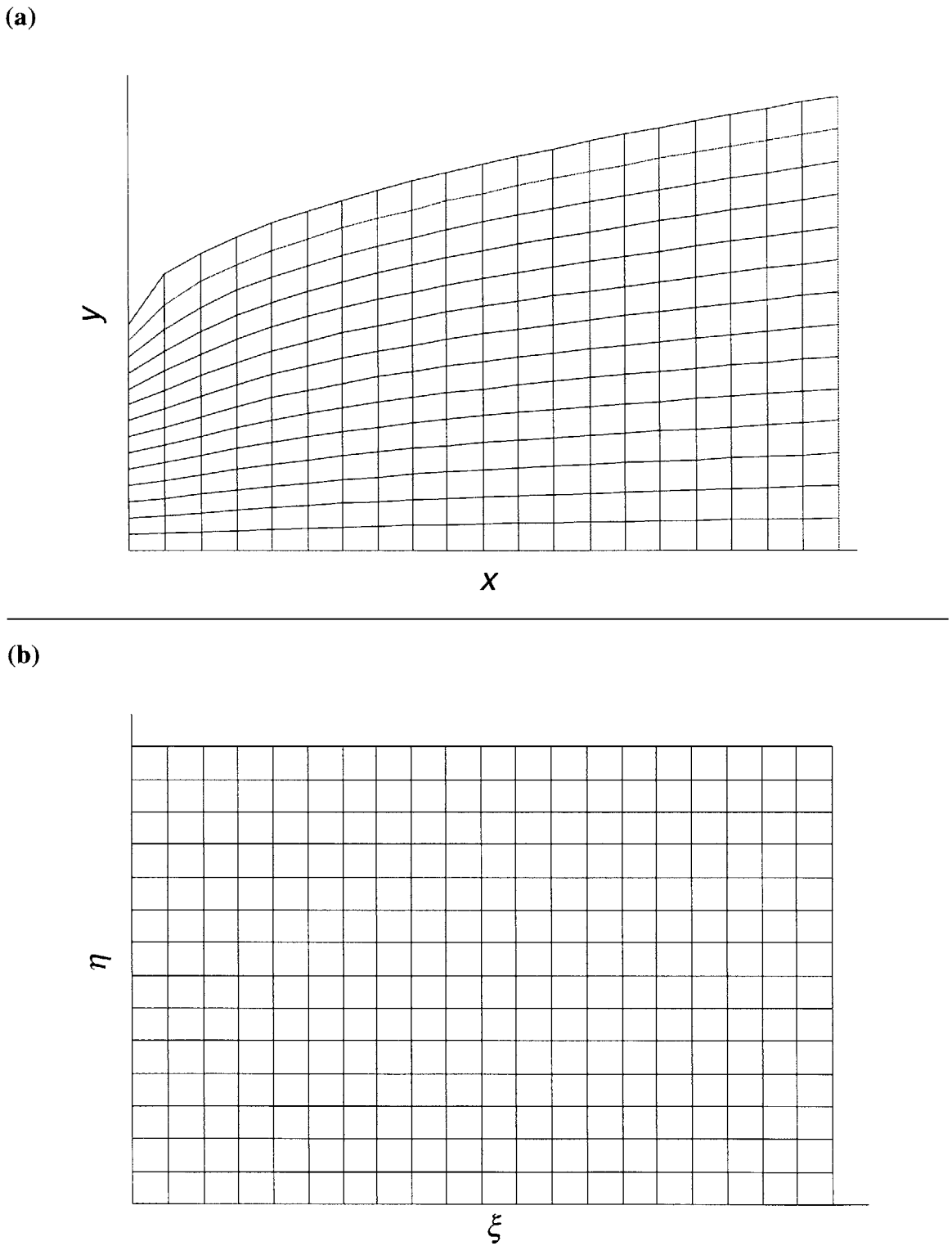


Figure 3-2. Body-fitted Grid: (a) physical space (x,y), and (b) computational space (ξ,η). Body-fitted grid generation allows boundary conditions to be easily implemented by transforming physical space into computational space, thus forcing physical boundaries to lie on coordinate lines in computational space. Each grid point in physical space corresponds to one grid point in computational space.

orthogonal. The former method is very attractive since the governing equations in the computational domain are then very similar to the standard Cartesian hydrodynamic expressions. Consequently, existing algorithms developed for Cartesian equations can be modified slightly and used. However, the orthogonality constraint limits the distribution of grid lines around complex boundaries. Non-orthogonal physical grid systems allow more flexibility in the internal grid distribution, but more complex governing equations are created (Barber et al. 1997).

This model employs a non-orthogonal mesh to take advantage of the additional flexibility in grid generation gained by the method. Since the computational grid has coordinate lines coincident with the physical boundaries, normal derivatives on these boundaries may be represented using finite differences between grid points in the computational domain regardless of the fact that the physical grid is not orthogonal at the boundary (Thompson 1980).

3.3.3 Grid generation using the Poisson equation

The body-fitted grid generation technique is based upon numerical generation of a curvilinear coordinate system (ξ,η) which has a coordinate line coincident with each boundary of physical space (x,y). This method is not limited to boundaries that can be represented by complex transformation. Various schemes are available to achieve mappings from physical space to computational space, including conformal mapping, shearing transforms, algebraic schemes, and elliptic partial differential equations. Regardless of the scheme, the grid must be smooth or the accuracy of the finite difference expressions will be poor.

The elliptic equation technique is considered the most flexible for generating well-ordered finite difference grids about arbitrary surfaces (Steger & Sorenson 1979). The curvilinear coordinates in this work were generated as a solution of the following Poisson equations, which have a solution completely defined by the boundaries of the physical region:

$$\left.\begin{aligned}\nabla^2\xi &= P(\xi,\eta)\\ \nabla^2\eta &= Q(\xi,\eta)\end{aligned}\right\} \qquad \textbf{(3.3a,b)}$$

where P and Q are non-homogeneous terms used to control interior grid lines. The Poisson equations generate smooth grids that permit a one-to-one mapping so grid lines of the same family do not cross. Using the Laplacian (P=Q=0) creates a system of coordinate lines that tend to be equally spaced in the absence of boundary curvature. However, due to the strong smoothing effect of the Laplacian, coordinate lines will become more closely spaced over convex boundaries and less so over concave boundaries (Thompson, Warsi, & Mastin 1985). Control of the interior grid to overcome this can be exercised by introducing non-zero P and Q in Eqs. 3.3. Negative Q will cause η-lines to move toward the η-line having the lowest value of η; positive Q will have the opposite effect. The magnitude of Q will control whether effects will be felt locally or widespread. Control of ξ-lines are similarly controlled by P, however, since the boundary values of the grid are fixed, P has the effect of controlling the intersection angle of the ξ- and η-lines.

Solving Eqs. 3.3 yield the direct transformation, Eqs. 3.1. However, the goal of grid generation is to define the computational grid and solve for the physical grid. In order to do this, the inverse transformation (Eqs. 3.2) must be solved for; the roles of the independent variables x and y must be interchanged with the dependent variables ξ and η in Eqs. 3.3 to achieve:

$$\left.\begin{aligned} \alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} &= -J^2(Px_\xi + Qx_\eta) \\ \alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} &= -J^2(Py_\xi + Qy_\eta) \end{aligned}\right\} \qquad \textbf{(3.4a,b)}$$

where

$\alpha = x_\eta{}^2 + y_\eta{}^2$,

$\beta = (x_\xi x_\eta + y_\xi y_\eta)$,

$\gamma = x_\xi{}^2 + y_\xi{}^2$, and $J = x_\xi y_\eta - x_\eta y_\xi$, the Jacobian of the transformation (Hoffman 1992). Eqs. 3.4 can be used to solve for a physical grid given a specified computational grid.

The system of equations (Eqs. 3.4) are solved directly to find a one-to-one mapping between the computational (ξ,η) mesh and the boundary-fitted physical (x,y) mesh. To ensure a reasonably smooth curvilinear grid, a computational mesh is chosen so that it can be molded to the shape of the flow domain without excessive stretching. For example, to create a body-fitted grid for the Upper Forebay of the Mystic Lakes in

Winchester, MA a non-rectangular computational grid was necessary. Figure 3-3 includes the physical and computational grid for this system.

Dirichlet boundary conditions are imposed through specification of the x and y coordinates of the physical boundaries of the body being fitted. An initial approximation for the interior grid point distribution is made using a normalization transformation in which the physical space in the x-coordinate is divided into equally spaced intervals and the y-coordinate space is divided into the desired number of equally spaced nodes at each x location. The parameters α, β, γ, and J are evaluated and the set of Eqs. 3.4 are solved directly with P=Q=0. Alpha, β, γ, and J are updated until the solution to the set of equations converges. Non-zero P and Q values may be needed to yield an acceptable interior grid distribution. For information on interior grid control refer to Hoffman (1992).

3.4 Governing Equations

The non-conservative form of the depth-averaged Navier Stokes equations were developed for use in this model. The non-conservative form allows one to solve for the primitive variables $\bar{u}$, $\bar{v}$, and h directly, requiring less computational time than the conservative form. For smooth solutions the non-conservative form applies, but the conservative form of these equations would be necessary if the model was used to solve for sharp discontinuous solutions (Vreugdenhil 1994). The following assumptions were made in developing these governing equations:

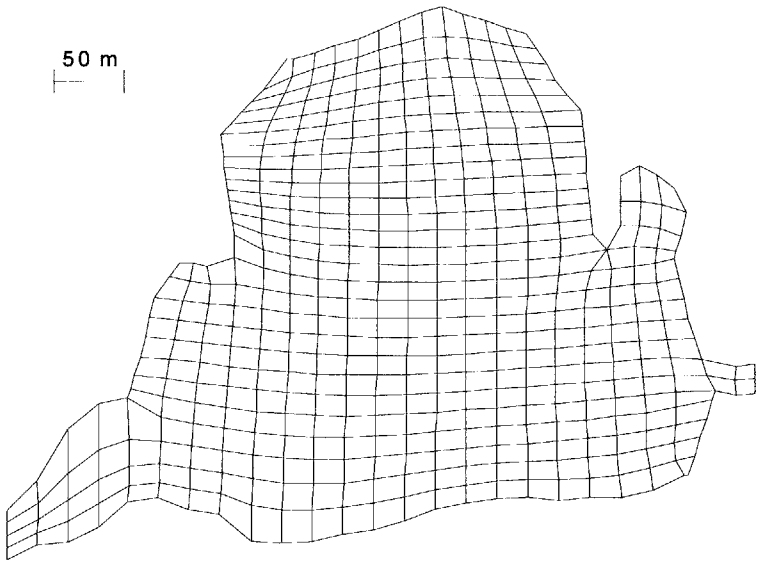
- incompressible Newtonian fluid,
- constant horizontal turbulent eddy viscosity,
- hydrostatic pressure distribution, and
- negligible Coriolis effects.

The governing equations in (x,y,t) Cartesian coordinates are:

Continuity equation (depth-integrated conservation of mass)

$$\frac{\partial h}{\partial t}+\frac{\partial(\bar{u}h)}{\partial x}+\frac{\partial(\bar{v}h)}{\partial y}-r=0 \qquad \textbf{(3.5)}$$

(a)



(b)

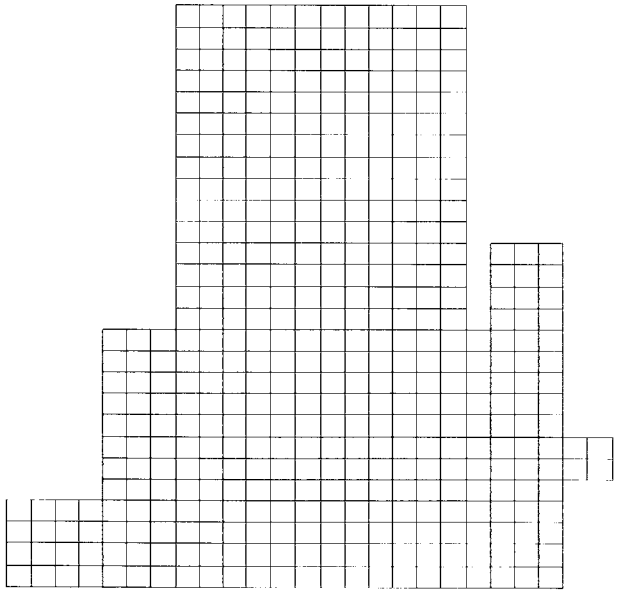


Figure 3-3. Body-fitted Grid of the Upper Forebay, Winchester, MA: (a) physical grid, and (b) computational grid. Non-rectangular computational grids are sometimes necessary to avoid excessive stretching of the physical grid.

X-momentum equation (depth-integrated conservation of x-momentum)

$$\frac{\partial \bar{u}}{\partial t} + \bar{u}\frac{\partial \bar{u}}{\partial x} + \bar{v}\frac{\partial \bar{u}}{\partial y} = -g\frac{\partial (z_b + h)}{\partial x} + \frac{\tau_{bx}}{\rho h} + \frac{\tau_{wx}}{\rho h} + \upsilon_t\left(2\frac{\partial^2 \bar{u}}{\partial x^2} + \frac{\partial^2 \bar{v}}{\partial x \partial y} + \frac{\partial^2 \bar{u}}{\partial y^2}\right) \quad \textbf{(3.6)}$$

where

$\tau_{bx} = -\frac{g}{C_f^{\,2}}\rho \bar{u}\sqrt{\bar{u}^2 + \bar{v}^2}$ (bed stress),

$\tau_{wx} = \rho_{air} C_w \bar{w}_{10}^2 \cos\Theta$ (surface stress from wind),

$\bar{u}$ and $\bar{v}$ are depth-averaged velocity components in the x and y directions respectively,

h is water depth,

r is rainrate [m/s],

z_b is bed elevation,

$g = 9.81$ m/s^2 (gravity),

$\rho = 998$ kg/m^3 (water density),

υ_t is horizontal turbulent eddy viscosity [m^2/s],

C_f is the Chezy coefficient [m$^{1/2}$/s],

$\rho_{air} = 1.2$ kg/m^3 (air density),

C_w is the wind drag coefficient [-],

$\bar{w}_{10}$ is average windspeed at 10 m above water level, and

Θ is the angle between the positive x-axis and the wind direction [°].

The momentum equation in the y-direction is analogous to that shown above for the x-direction. These governing equations are in the form presented in Barber et al. (1997).

3.4.1 Horizontal turbulent eddy viscosity (υ_t)

Since hydrodynamic modelling of wetlands is early in its development, limited information is available to estimate appropriate values for horizontal turbulent eddy viscosity. For a shallow river with slow current, the TABS Manual (a commercial open channel flow model manual), recommends a spatially uniform υ_t between 0.24 and 1.2 m^2/s (Barrett 1996). Barrett (1996) used a uniform eddy viscosity of 0.24 m^2/s to study

wetlands with the TABS model. Walker (1998) studied flow through stormwater wetlands using a depth-averaged numerical model applying a uniform horizontal eddy viscosity of 0.1 m^2/s.

Given the above information, a uniform υ_t of 0.1 m^2/s was adopted. In future work, this horizontal turbulent eddy viscosity value should be verified by either field measurements or through the development of a k-ε model. However, I performed a sensitivity analysis under typical constructed wetland conditions (rectangular channel: 300 m wide with bed slope=-5 x 10^{-4}, h=0.5 m, C_f=0.45 $m^{1/2}/s$) which shows that increasing υ_t by one order of magnitude from 0.1 m^2/s to 1 m^2/s yields a root mean square error (normalized by the maximum velocity) between the corresponding velocity profiles of 0.16%. Decreasing υ_t by one order of magnitude from 0.1 m^2/s to 0.01 m^2/s yields a normalized root mean square between the corresponding velocity profiles of 0.016%. The drag from the bed and vegetation is much more influential on the flowfield than horizontal turbulent eddy viscosity. So, although we do not expect υ_t to be uniform in reality, developing a better characterization of υ_t is not necessary; υ_t has little influence on the velocity field in a wetland with large drag.

3.4.2 Chezy coefficient (C_f)

Resistance to flow in vegetated wetlands can be attributed to bed friction and stem drag. Due to the complex nature of flow through vegetation, a strictly analytical relation for determining the flow resistance of vegetation has not been developed. Most of the work performed regarding flow through vegetation has been empirical or semi-empirical in nature and characterized with a single bulk roughness term.

The Manning equation was developed for one-dimensional, fully turbulent, steady, uniform flow in open channels to describe drag due to bottom-surface roughness. The equation takes the form:

$$V = \frac{R_h^{2/3} S_o^{1/2}}{n} \tag{3.7}$$

in which

V is velocity,

R_h is the hydraulic radius [m], cross-sectional area divided by wetted perimeter,

S_o is the slope of the energy line, and

n is the Manning resistance coefficient which is dependent on bed roughness [$s/m^{1/3}$].

The Manning equation was developed for situations in which frictional elements are restricted to channel bottoms. In wetlands, aquatic vegetation protrudes into the water column creating a more complex system in which the density of vegetation at each vertical location in the water column becomes important. As such, the Manning friction coefficient shows a dependence on vegetation density, water depth, and velocity in wetlands. This has raised serious doubts about the appropriateness of using the Manning equation in wetland systems. Despite its problems, the Manning coefficient, n, is still widely used to describe drag due to vegetation in wetland problems including Shih and Rahi (1982), Kadlec (1994), Guardo and Tomasello (1995), Reed, Crites, and Middlebrooks (1995), Barrett (1996), Kadlec and Knight (1996), and Feng and Molz (1997) .

For immediate practical purposes, a simple method of estimating n is needed. It is preferable to have a method in which n depends on vegetation density, water depth, and velocity. Such methods are not available, though formulas that account for vegetation density and water depth are available. Reed, Crites, and Middlebrooks (1995) present such a formula:

$$n = \frac{a}{h^{1/2}} \tag{3.8}$$

where

a is the vegetative resistance factor [$sm^{1/6}$], and

h is water depth [m].

The parameter a is defined as follows:

$a = 0.4$ sm$^{1/6}$ for sparse low-standing vegetation with $h > 0.4$ m

$a = 1.6$ sm$^{1/6}$ for moderately dense vegetation with $h = 0.3$ m

$a = 6.4$ sm$^{1/6}$ for very dense vegetation and litter layer with $h < 0.3$ m.

In most situations with typical emergent vegetation, it is acceptable to assume that a lies between 1 and 4 sm$^{1/6}$. This can yield a Manning coefficient (n) of up to 15 s/m$^{1/3}$ for thick vegetation with small depths, 100 times larger than estimations of n not specific to vegetation. It is not clear how this definition for the Manning coefficient was developed (Eq. 3.8).

Kadlec and Knight (1996) have also developed an empirical relationship for estimating n as a function of water depth and vegetation density using field observations from a number of published wetland studies:

$$n = \frac{b}{h^{1.7}} \qquad 0.1 \text{ m} < h < 1 \text{ m} \qquad \textbf{(3.9)}$$

with b ranging from 0.2 for sparse vegetation to 1.0 for dense vegetation.

The two relationships, Eq. 3.8 and Eq. 3.9, have been created with very limited information and are not a function of velocity, but they are currently the best methods for determining n for wetlands. In Chapter 5, the importance of developing methods of approximating drag as a function of velocity in wetlands is revealed.

Since the numerical model expresses bed stress in terms of the Chezy coefficient, the Manning number is converted into a Chezy coefficient with the following equation:

$$C_f = \frac{R_h^{1/6}}{n}. \qquad \textbf{(3.10)}$$

For wide channels, R_h approaches h and

$$C_f = \frac{h^{1/6}}{n}. \qquad \textbf{(3.11)}$$

Assuming h=0.5 m, Eqn. 3.9 yields a range of Manning coefficient values from approximately 0.65 to 3.25 s/m$^{1/3}$ for sparse to dense vegetation in a wetland. This range corresponds to Chezy coefficient values between 1.37 and 0.27 m$^{1/2}$/s respectively.

3.4.3 Wind drag coefficient (C_w)

Wind exerts a drag force on the surface of water bodies such as wetlands. The actual stress that the water surface feels is influenced by wind magnitude, the stability of the meteorological boundary layer over the water surface, the variability of the windspeed over the wetland, the length of the fetch, the degree of wave development, and the amount of wave energy dissipation at the shores of the lake. The wind stress,

$$\tau_w = \rho_{air} C_w \overline{w}_{10}^2, \tag{3.12}$$

defines the wind drag coefficient, C_w, which incorporates the variability induced by all influences other than wind magnitude (Fischer et al. 1979).

A great deal of effort has been exerted to determine C_w for different lake and open seas conditions. Wind drag coefficients for shallow lakes, less than 5 m in depth, differ greatly in comparison to the open seas (Hicks, Drinkrow, & Grauze 1974). In a wetland, it is especially difficult to model wind effects due to areas of emergent, and floating-leaved vegetation. Little work has been done to determine wind drag coefficients for wetlands; one can look at research on shallow lakes to obtain estimates for C_w for use in wetland models.

C_w ranges from 1.0 to 1.5 x 10^{-3} for wind speeds of 0 to 15 m/s for shallow water conditions. As water becomes very shallow, less than 2.5 m, Hicks, Drinkrow, and Grauze (1974) showed that C_w remains near 1.0 x 10^{-3} for all windspeeds because longer waves will no longer be able to fully develop. Since wetlands are typically less than 2.5 m in depth, a wind drag coefficient of 1.0 x 10^{-3} was used in the numerical model.

3.5 Transformation of the Governing Equations

Once the curvilinear coordinate system has been generated, the partial differential system, in this case the depth-averaged Navier Stokes equations (Eqs. 3.5 & 3.6), is transformed to the computational plane. A problem having simple equations but complex boundary conditions has been modified to a problem of complex equations and simple boundary conditions.

As you recall, the boundary-fitted curvilinear coordinates are (ξ,η). The governing equations (Eqs. 3.5 & 3.6) are transformed from the Cartesian coordinate space (x,y,t) to the curvilinear coordinate space (ξ,η,t) as follows:

Depth-averaged continuity equation

$$\frac{\partial h}{\partial t}+\frac{1}{J}\left(\frac{\partial Uh}{\partial \xi}+\frac{\partial Vh}{\partial \eta}\right)-r=0 \tag{3.13}$$

Depth-averaged momentum equations

$$\frac{\partial \phi}{\partial t}+\frac{1}{J}\left(\frac{\partial U\phi}{\partial \xi}+\frac{\partial V\phi}{\partial \eta}\right)=\frac{1}{J}\left\{\frac{\partial}{\partial \xi}\left[\frac{\Gamma_\phi}{J}\left(q_{11}\phi_\xi-q_{12}\phi_\eta\right)\right]+\frac{\partial}{\partial \eta}\left[\frac{\Gamma_\phi}{J}\left(q_{22}\phi_\eta-q_{12}\phi_\xi\right)\right]\right\}+S_\phi \tag{3.14}$$

where

$J = x_\xi y_\eta - x_\eta y_\xi$, the Jacobian matrix,

$U = \bar{u}y_\eta - \bar{v}x_\eta$, a contravariant velocity perpendicular to η,

$V = \bar{v}x_\xi - \bar{u}y_\xi$, a contravariant velocity perpendicular to ξ, and

$\Gamma_\phi=\upsilon_t$.

If one defines $\alpha=x_\eta^2+y_\eta^2$, $\beta=x_\xi x_\eta+y_\xi y_\eta$, and $\gamma=x_\xi^2+y_\xi^2$, then q_{11}, q_{12}, q_{22}, and the source term S_ϕ for the momentum equations are defined as follows:

X-momentum equation

$\phi=\bar{u}$,

$q_{11}=\alpha+y_\eta^2$,

$q_{12}=\beta+y_\eta y_\xi$,

$q_{22}=\gamma+y_\xi^2$, and

$$S_\phi=\frac{1}{J}\left[\left(\frac{x_\xi\Gamma_\phi}{J}T_1\right)_\eta-\left(\frac{x_\eta\Gamma_\phi}{J}T_1\right)_\xi\right]-\frac{g}{J}\left[y_\eta\left(z_b+h\right)_\xi-y_\xi\left(z_b+h\right)_\eta\right]+\frac{\tau_{bx}}{\rho h}+\frac{\tau_{wx}}{\rho h}$$

where $T_1 = y_\eta\bar{v}_\xi - y_\xi\bar{v}_\eta$.

Y-momentum equation

$\phi = \bar{v}$,

$q_{11} = \alpha + x_\eta^{\,2}$,

$q_{12} = \beta + x_\eta x_\xi$,

$q_{22} = \gamma + x_\xi^{\,2}$, and

$$S_\phi = \frac{1}{J}\left[\left(\frac{y_\eta \Gamma_\phi}{J} T_2\right)_\xi - \left(\frac{y_\xi \Gamma_\phi}{J} T_2\right)_\eta\right] - \frac{g}{J}\left[x_\xi (z_b + h)_\eta - x_\eta (z_b + h)_\xi\right] + \frac{\tau_{by}}{\rho h} + \frac{\tau_{wy}}{\rho h}$$

where $T_2 = x_\xi \bar{u}_\eta - x_\eta \bar{u}_\xi$ (Barber et al. 1997; Ye and McCorquodale 1997).

3.6 Numerical Scheme

The transformed governing equations (Eqs. 3.13 & 3.14) are solved using finite difference approximation. Most finite difference schemes use the method of lines which discretizes the governing equations of a system in two distinct steps: spatial and time discretization.

The governing equations are centrally discretized in space, $O(\Delta x^2)$ (Hoffman 1992). Central differencing is used because it is assumed that diffusion is important in wetland systems. Time is discretized using a semi-implicit Euler or forward-backward scheme, $O(\Delta t)$ (Vreugdenhil 1994). This scheme uses information at the new time level as soon as it is computed. The method remains effectively explicit in the sense that one does not need to solve the system of governing equations simultaneously; only one unknown appears in each differential equation. Although explicit methods are simpler than implicit, they are only conditionally stable which requires small values of Δx, Δy, and Δt for convergence and thus an increased computational time.

The Navier Stokes equations (Eqs. 3.5 & 3.6) contain various physical processes including advection, drag etc. It is non-trivial to develop a stability analysis for such complex governing equations in combination with a semi-implicit scheme. Yet, some criteria for choosing Δx, Δy, and Δt is desired. The following are provided as guidelines for choosing a Δx, Δy, and Δt which will yield a stable and convergent solution. These

stability criteria are based upon simpler governing equations and an explicit scheme centrally discretized in space and forward in time:

$$\left.\begin{aligned}\frac{\overline{u}\Delta t}{\Delta x} &\le 1\\ \frac{\overline{v}\Delta t}{\Delta y} &\le 1\end{aligned}\right\} \qquad \textbf{(3.15a,b)}$$

$$\frac{\Delta x^2 \Delta y^2}{2\upsilon_t \Delta t\left(\Delta x^2 + \Delta y^2\right)} \ge 1. \qquad \textbf{(3.16)}$$

Eqns. 3.15 do not permit advection to be larger than one grid interval (Δx or Δy) for a given timestep (Δt); the timestep and grid interval size are constrained accordingly. Eqn. 3.16 is the criteria given by the viscous forcing.

The choice of Δx, Δy, and Δt should not affect the numerical solution to a problem. It is best to choose the largest grid spacing and largest timestep which yields the same solution as a case in which the grid spaces and timestep are smaller. This will cut computational costs while insuring that the solution is stable.

An unstaggered grid (all variables are defined at each grid point) is used. Unstaggered grids have been known to create non-physical short-wave oscillations. These effects were not strongly evident in this work.

3.7 Boundary Conditions

To solve the transformed governing equations presented in the previous section, boundary conditions are required at each computational boundary. At solid walls or boundaries, the no-slip boundary condition for velocities is applied ($\overline{u} = \overline{v} = 0$). At the inlet and outlet, variables that are not designated by the user's boundary conditions are extrapolated from an interior point to the boundary (Molls & Chaudry 1995).

CHAPTER 4: MODEL VERIFICATION

4.1 Introduction

Numerical models are only useful if their accuracy can be verified. For this reason, the analytical solution to uni-directional depth-averaged flow through a rectangular channel was developed and compared with the numerical model results. The numerical model performs very well.

4.2 Analytical Solution

To confirm that the depth-averaged numerical model developed in this work accurately predicts hydrodynamic behavior, the analytical solution to uni-directional depth-averaged flow through a rectangular channel with bed slope was developed and compared with the model results.

The direction of flow is defined in the positive x-direction and is contained within a rectangular channel with infinite length and width (y_{max} - y_{min}). The depth-averaged momentum equations (Eqs. 4.1) and the continuity equation (Eq. 4.2)

$$\frac{\partial \overline{u}}{\partial t}+\overline{u}\frac{\partial \overline{u}}{\partial x}+\overline{v}\frac{\partial \overline{u}}{\partial y}=-g\frac{\partial (z_b+h)}{\partial x}+\frac{\tau_{bx}}{\rho h}+\frac{\tau_{wx}}{\rho h}+\upsilon_t\left(2\frac{\partial^2 \overline{u}}{\partial x^2}+\frac{\partial^2 \overline{v}}{\partial x\partial y}+\frac{\partial^2 \overline{u}}{\partial y^2}\right) \tag{4.1a}$$

$$\frac{\partial \overline{v}}{\partial t}+\overline{u}\frac{\partial \overline{v}}{\partial x}+\overline{v}\frac{\partial \overline{v}}{\partial y}=-g\frac{\partial (z_b+h)}{\partial y}+\frac{\tau_{by}}{\rho h}+\frac{\tau_{wy}}{\rho h}+\upsilon_t\left(\frac{\partial^2 \overline{v}}{\partial x^2}+\frac{\partial^2 \overline{u}}{\partial x\partial y}+2\frac{\partial^2 \overline{v}}{\partial y^2}\right) \tag{4.1b}$$

$$\frac{\partial h}{\partial t}+\frac{\partial (\overline{u}h)}{\partial x}+\frac{\partial (\overline{v}h)}{\partial y}=0 \tag{4.2}$$

combine to provide a complete mathematical description of depth-averaged incompressible Newtonian fluid flow. Notation in this chapter is identical to the notation in Chapter 3.

Assuming steady-state, uni-directional flow ($\overline{v}$=0), where $\overline{u}$ is not variable with x due to the infinite channel length ($\partial \overline{u}/\partial x$=0), Eqs. 4.1 and 4.2 simplify to become:

$$-g\frac{\partial (z_b+h)}{\partial x}+\frac{\tau_{bx}}{\rho h}+\frac{\tau_{wx}}{\rho h}+\upsilon_t\left(\frac{\partial^2 \overline{u}}{\partial y^2}\right)=0 \tag{4.3a}$$

$$\frac{\partial (z_b+h)}{\partial y}=0 \tag{4.3b}$$

$$\frac{\partial h}{\partial x} = 0 \tag{4.4}$$

where the bed and wind stress in the x-direction are:

$\tau_{bx} = -\frac{\rho g}{C_f^2}|\overline{u}|\overline{u}$ and,

$\tau_{wx} = \rho_{air} C_w \overline{w}_{10}^2 \cos\Theta$ as described in Chapter 3.

The next step is to integrate Eq. 4.3a in y. However, integrating over the bed stress term is a nontrivial task because $\overline{u}$ is a function of y. To avoid this complication, one can neglect the bed stress therefore neglecting the problematic term. Though this is not physically correct, the analytical solution still serves the purpose of verifying the accuracy of the numerical model.

Neglecting bed stress, integrating over y twice, and solving for $\overline{u}$ yields

$$\overline{u} = -\frac{\rho_{air} C_w}{2\rho h \upsilon_t}\overline{w}_{10}^2 y^2 \cos\Theta + \frac{g}{2\upsilon_t}\frac{\partial(z_b + h)}{\partial x} y^2 + C_1 y + C_2 \tag{4.5}$$

where C_1 and C_2 are integration constants. Applying a no-slip boundary condition, $\overline{u}$ =0 at y=y_{min} and y=y_{max},

$$\overline{u} = \left(-\frac{\rho_{air} C_w}{2ph\upsilon_t}\overline{w}_{10}^2 \cos\Theta + \frac{g}{2\upsilon_t}\frac{\partial(z_b + h)}{\partial x}\right)\left(y^2 - y_{\max} y + y_{\max} y_{\min} - y_{\min} y\right). \tag{4.6}$$

This solution is a parabolic velocity distribution across the width of the channel and is analogous to the solution for steady laminar flow between fixed parallel plates since υ_t was considered constant in this analysis.

4.3 Results

The numerical model created for this study produced results so similar to those of the analytical solution, Eq. 4.6, that plotting the analytical and numerical solution on the same graph resulted in complete overlapping of the analytical and numerical solutions. Since no difference can be seen between the two solutions, the numerical results are presented in the following figures to demonstrate the effects of different variables on the velocity profiles across the channel.

Bed slope (m), horizontal turbulent eddy viscosity (υ_t), and wind ($\overline{w}_{10}$) were varied and the numerical results were compared with the analytical solution. A channel of arbitrary length 500 m and width 300 m was studied. Numerical computations were performed on a 51 x 31 grid with square grid cells of length 10 m. Smaller grid spaces create the same results. A timestep of 1 s was used. Initial conditions for all cases are $\overline{u}$ =0, $\overline{v}$ =0, and a uniform water depth of 1.5 m. Upstream and downstream boundary conditions were defined as constant water depth boundaries of 1.5 m.

A baseline case is plotted on each figure as a point of comparison. The baseline case is characterized by a bed slope of -1 x 10^{-5}, water depth of 1.5 m, horizontal turbulent eddy viscosity of 10 m^2/s, and windspeed of 0 m/s. Since bed drag was ignored, a large value for υ_t was used to achieve a numerically stable solution. The baseline velocity profile across the channel is depicted in Figure 4-1. The baseline profile is represented by a solid line in each subsequent figure in this chapter as a point of reference.

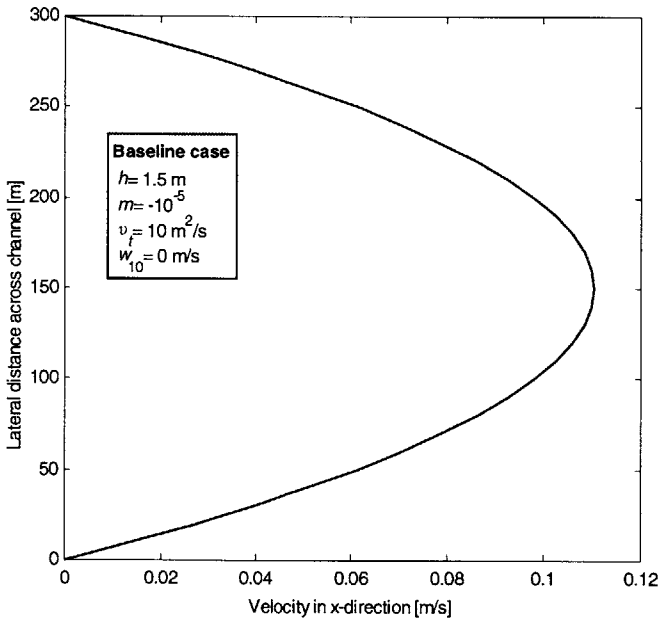


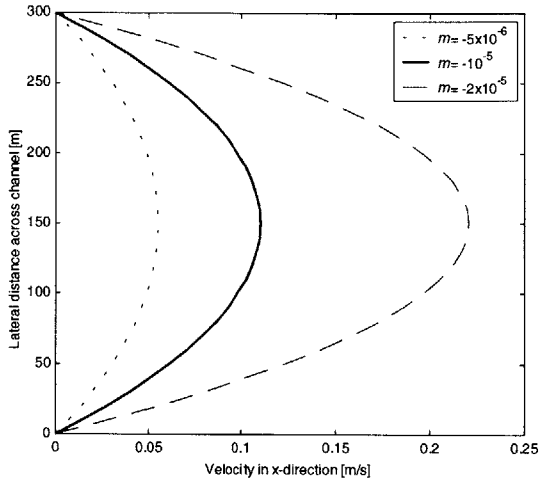
Figure 4-1. Baseline Case Velocity Profile in a Channel. This parabolic velocity profile across a rectangular channel of width 300 m is analogous to steady laminar flow between two fixed plates since υ_t was considered constant. The conditions under which this profile was created serve as the baseline conditions from which all other figures in this chapter were created. This baseline profile is depicted by a solid line in each subsequent figure.

Figure 4-2 displays the effects of varying bed slope, turbulent eddy viscosity, and wind on the velocity profiles in the channel. Increases in bed slope magnitude increase the velocity in the channel (Figure 4-2a). Turbulent eddy viscosity is negatively correlated with velocity (Figure 4-2b). Wind in the direction of flow increases velocity while wind blowing against the flow decreases velocity. Larger wind magnitudes create stronger velocity effects. All velocity profiles remain parabolic in shape and compare extremely well with the analytical results; a root mean square error of 0 was achieved to 14 significant digits (compiler precision).

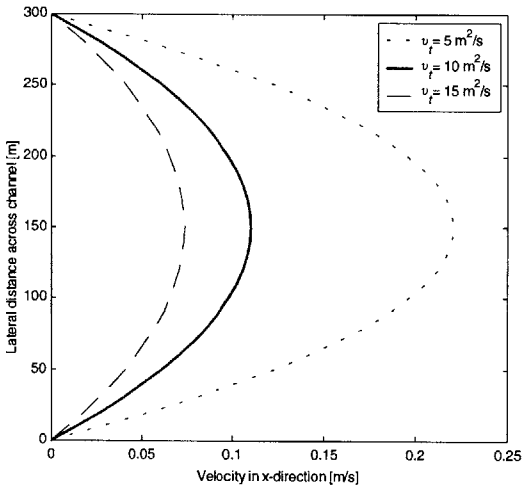
Thus far, the bed friction has been ignored to obtain a simple analytical solution to uni-directional channel flow. Although bed friction has been ignored in the analytical solution, using the numerical model to predict the effects of the Chezy coefficient on the uni-directional velocity profile is still beneficial. One can determine whether the model predicts behavior that is physically expected. In Figure 4-3, note that decreasing the Chezy coefficient results in decreasing velocity. This agrees with the governing equation. As the Chezy coefficient decreases the stress from the bed increases (C_f is in the denominator) which decreases the velocity. Any non-zero Chezy coefficient will create a smaller velocity than the baseline case in which bed friction was neglected.

In this chapter, the numerical model has been used to examine the effects of varying bed slope, horizontal turbulent eddy viscosity, wind, and the Chezy coefficient. Except for the results varying Chezy coefficient, all numerical results have been verified with the analytical solution to uni-directional depth-averaged channel flow and proven to be accurate.

(a)



(b)



(c)

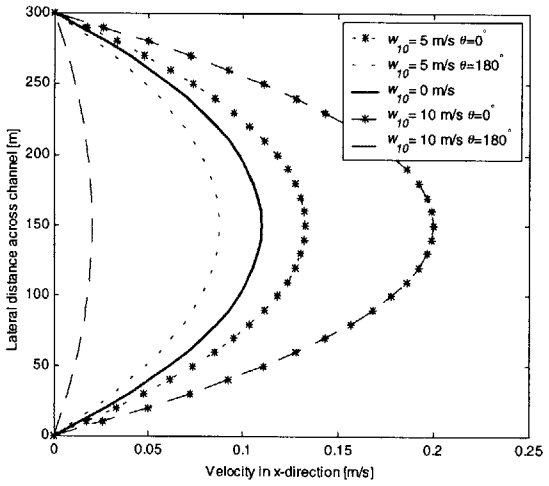


Figure 4-2. Velocity Profiles in Rectangular Channel Under Varying Conditions: (a) bed slope, (b) horizontal turbulent eddy viscosity, and (c) wind. Larger magnitude bed slope increases velocity. Turbulent eddy viscosity is negatively correlated with velocity. Wind in the direction of the flow(Θ=0°) increases velocity while wind in the opposite direction(Θ=180°) reduces velocity. Larger wind magnitudes create stronger effects. These results have been verified with the analytical solution.

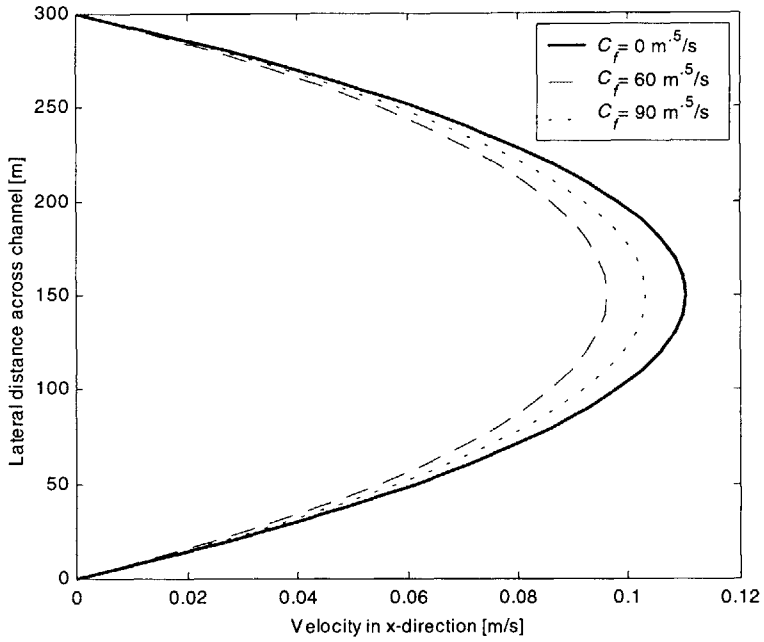


Figure 4-3. Varying Chezy Coefficient. A non-zero Chezy coefficient will reduce the velocity profile across the rectangular channel. A larger Chezy coefficient, however, increases velocity compared to a smaller Chezy coefficient. This is a direct result of C_f being in the denominator for the bed stress formulation. A larger C_f decreases the magnitude of the bed stress.

CHAPTER 5: MODEL APPLICATIONS

5.1 Constructed Wetlands: Uneven Vegetation Cover

5.1.1 Motivation

One of the most important aspects of constructed wetland design is hydraulic efficiency, the ratio of effective hydraulic detention time to nominal detention time. Hydraulic efficiency plays such an important role in wetland design because longer detention times allow additional time for physical, chemical, and biological reactions to take place thus increasing the ability of a wetland to enhance water quality. As a result, deviations from uniform flow or the existence of dead zones, zones that do not communicate with the main flow, decrease the efficiency of wetlands.

Design which assumes uniform flow overestimates the effective hydraulic detention time of a wetland and thus overestimates water quality enhancement. Therefore, it is crucial to understand the hydrodynamics of wetlands to optimize the design of such systems.

A common obstacle to establishing an efficient constructed wetland is establishing evenly distributed wetland plant communities. Uniform vegetative cover prevents poor flow distribution and is an important factor in the success of a constructed wetland. However, vegetation is almost certain to be uneven in coverage and will fill in and undergo species composition changes over time. When a wetland is flooded, some wetland species may die leaving unvegetated areas. Several years may pass before these bare or thinly vegetated areas are covered by adapted wetland plants spreading by rhizomatous growth (Kadlec & Knight 1996). In addition, trails left by wildlife and humans also create areas of sparse vegetation resulting in non-uniform flow in the system.

Parr (1990) performed a survey of reed (*Phragmites australis*) growth in 81 subsurface flow wetlands in the United Kingdom. Reed establishment was generally poor; only 35% of the wetlands had over 90% coverage of reeds. The main factors accounting for the variation in reed growth between wetlands were the age, soil type, type of feed, and type of reed material planted. It is estimated that one should be able to achieve 100% coverage in two years, but this is highly dependent on planting method. For example,

using seeds to establish wetland plant communities is extremely difficult. Though such an in-depth study of vegetation cover for free surface flow wetlands was unavailable, Parr's (1990) U.K. study on subsurface flow wetlands provides insight into the problem of plant coverage in constructed wetlands.

To gain a better understanding of the effects of uneven plant distribution in constructed wetlands, numerical experiments were performed using the hydrodynamic model developed as part of this thesis. These experiments not only provide insight into the applicability of such a model on wetlands, but also provide a better understanding of how uneven plant distribution may affect the hydrodynamics in constructed wetlands.

The following numerical experiments were performed on a rectangular constructed wetland with typical design characteristics. The wetland was prescribed an aspect ratio of 3:1 (600 m by 200 m), a slope of -5 x 10^{-4} (.05%), constant horizontal turbulent eddy viscosity of 0.1 m^2/s, and water depth of 0.3 m. Inflow and outflow are applied over the entire width of the wetland. The focus of this study, uneven plant distribution, was simulated by spatially varying the Chezy coefficient in the wetland. Chezy coefficients were estimated using both the Reed, Crites, and Middlebrooks (1995), Eqn. 3.8, and the Kadlec and Knight (1996), Eqn. 3.9, formulations for determining Chezy coefficients. By spatially varying C_f values, patchiness is simulated within the plant stands of the constructed wetland.

5.1.2 Numerical experiment: Random plant distribution (80% coverage)

To begin, it is important to confirm that the model accurately predicts plug-flow or uniform flow for moderately dense vegetation (C_f=0.25 $m^{1/2}/s$) in a wetland with uniform cover. A grid size of 10 m by 10 m and a timestep of 0.1 s were used. A constant water depth (h=0.3 m) was assigned as the upstream boundary condition and the downstream boundary condition was assigned as no-flux ($\partial h/\partial x$=0). These numerical conditions are constant throughout this section. Figure 5-1 shows the velocity profile across the wetland as a result of this simulation. The result, as expected, is a uniform velocity distribution of magnitude 3.06 x 10^{-3} m/s (0.306 cm/s) across the width of the constructed wetland

(plug-flow behavior). Typical constructed wetland velocities are in the range 10^{-1} to 10^{-3} m/s (Witthar 1993; Barrett 1996; Kadlec & Knight 1996).

As you may recall from Chapter 4, sloped rectangular channels created parabolic velocity profiles in contrast with the uniform velocity profile seen in Figure 5-1. In the case of wetlands, Chezy coefficient values are two order of magnitudes smaller (C_f=0.25 $m^{1/2}/s$) than under non-vegetated conditions such as those in Chapter 4 (C_f=30 to 90 $m^{1/2}/s$). The transition from a parabolic velocity profile to a uniform one is a result of the transition from the turbulence-dominated flow in Chapter 4 and bed drag-dominated flow in the case of wetlands. When the viscous forcing becomes negligible, the forcing on the system is uniform (bed slope and vegetation drag) producing uniform flow.

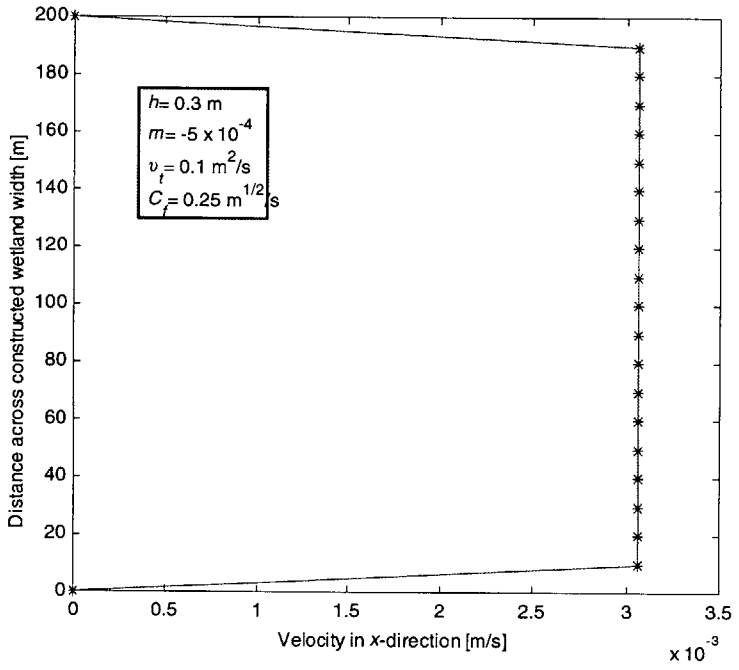


Figure 5-1. Velocity Distribution for Constructed Wetland with Uniform Vegetation. A rectangular-shaped constructed wetland with a width of 200 m, slope of .05%, υ_t=0.1 m^2/s, water depth of 0.3 m, and uniform C_f of 0.25 $m^{1/2}/s$, simulating moderately dense vegetation, exhibits plug-flow characteristics.

Uniform vegetation coverage takes years to achieve and in some cases is never achieved due to fluctuations in water level, etc. A more realistic depiction of wetland vegetation coverage was simulated in which 80% of the basin had coverage of moderate vegetation (C_f =0.25 $m^{1/2}/s$) and the other 20% had very sparse vegetation (C_f =1.5 $m^{1/2}/s$). As you may recall, Parr (1990) used 90% coverage as a benchmark for good coverage in subsurface wetlands. Only 35% of the wetlands evaluated in that study achieved this coverage. In light of this, 80% coverage appears to be a satisfactory assumption for coverage in constructed wetlands.

Each grid cell was assigned either moderate vegetation (80% of the grid cells) or very sparse vegetation (20% of the grid cells). Though heterogeneity of plant cover is believed to be O(1 m), the heterogeneity was assigned O(10 m) due to computational constraints. However, results in this section should be self-similar to heterogeneity generated on different scales. The very sparse vegetation was assigned randomly in the wetland using a random number generator (Figure 5-2).

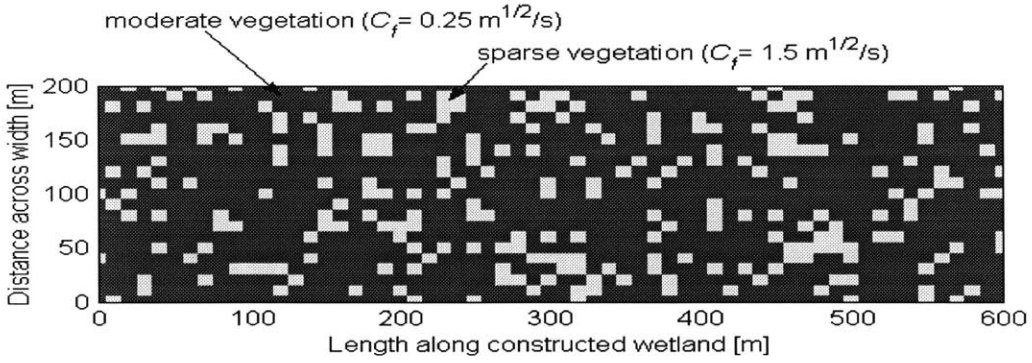


Figure 5-2. Random Plant Distribution (80% Coverage). Each grid cell was assigned either moderate vegetation or sparse vegetation. Sparse vegetation was assigned using a random number generator.

The effects of the non-homogeneous plant cover are evident. Velocities in areas of sparse vegetation were as much as seven times higher than the velocities in areas of moderately dense vegetation. Figure 5-3 shows the normalized deviation, δ, from the uniform velocity profile generated with uniform plant cover

$$\delta = \frac{\overline{u}_{heterogeneous} - \overline{u}_{uniform}}{\overline{u}_{uniform}} \tag{5.1}$$

where $\overline{u}_{heterogeneous}$ is the velocity in the wetland with 80% random coverage, and $\overline{u}_{uniform}$ is the velocity in the wetland with uniform plant cover. Transects across the wetland were taken at the intervals x= 100, 200, 300, 400, and 500 m. Large deviations from the uniform profile correspond to areas with sparse vegetation.

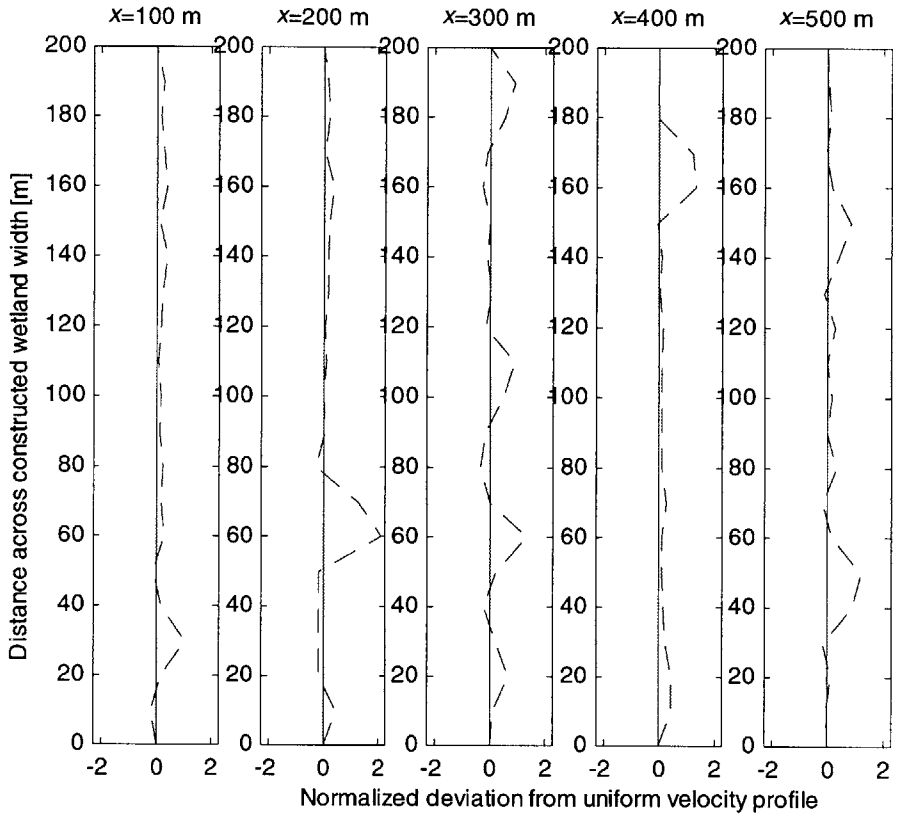


Figure 5-3. Deviation from Uniform Flow: 80% vegetation cover. Normalized deviations from the velocity profile for uniform vegetation cover at x= 100, 200, 300, 400, and 500 m. Eighty percent cover is still considered good, but deviations from plug-flow are approximately 44% (normalized root mean square) and lead to a decreased hydraulic efficiency of 81%.

To determine the effect these deviations will have on the hydraulic efficiency of the wetland, I evaluated the effects of the random plant heterogeneity as a decrease in bulk drag. The mean velocity across the transects of the wetland with heterogeneous vegetation is 0.376 cm/s (standard deviation 5.82 x 10^{-3} cm/s), a 23% increase from the velocity seen under uniform vegetation cover (0.306 cm/s). This velocity increase yields a decreased hydraulic efficiency.

The effective detention time in the constructed wetland is approximated as

$$t_{eff} = \frac{L}{\overline{u}_{avg}} \tag{5.2}$$

where

L is the length of the constructed wetland, and

$\overline{u}_{avg}$ is the average velocity across the wetland. The effective residence time of the constructed wetland with uniform vegetation is 2.27 days. With 20% random sparseness in the vegetation of the wetland, t_{eff} was reduced to 1.84 days, a 19% decrease from the uniform vegetation conditions. This translates to a hydraulic efficiency of 81% (eqn. 2.1).

This numerical study shows that as uniform vegetation cover becomes sparse in a random fashion, the bulk Chezy coefficient of the system will increase proportionally and the hydraulic efficiency of the system will decrease proportionally. This is a direct result of the force balance in the system:

$$-\frac{\partial(z_b + h)}{\partial x} = \frac{\overline{u}_{avg}^{\,2}}{hC_f^2}. \tag{5.3}$$

5.1.3 Numerical experiment: Channels of sparse vegetation (80% coverage)

In the previous case, random sparseness in the aquatic vegetation of wetlands was simulated. As the vegetation coverage decreases, the chance of connectivity amongst sparse patches increases. Channelization defines the case in which areas of sparse vegetation connect to create pathways of low resistance, ultimately creating short-circuiting in a wetland. Short-circuiting is a term for cases in which a portion of water passes through a basin in a time much less than the nominal detention time. Channels of sparse vegetation are also commonly created by the trails of wildlife and humans.

A simulation was performed given 80% coverage again, but in this case with channelization. Vegetation is no longer assigned randomly; it is given a channel structure which can be seen in Figure 5-4. Areas of moderate plant coverage were assigned C_f =0.25 $m^{1/2}/s$, and areas of very sparse vegetation were assigned C_f=1.5 $m^{1/2}/s$. The resultant steady-state velocity field is superimposed in the figure; velocities in the sparse channels are as high as 0.02 m/s (2 cm/s). An average detention time for parcels in the channel system can be calculated as

$$t_{channel} = \frac{L}{\overline{u}_{avg}} \tag{5.4}$$

where $t_{channel}$ is the average detention time in the channel system,

L is the pathlength, and

$\overline{u}_{avg}$ is the average channel velocity. The paths of low resistance (channels of sparse vegetation) provide a pathway for incoming water to pass through this wetland in as little as 0.345 days or 8.27 hours (short-circuiting) as opposed to the plug-flow conditions of uniform plant coverage with a residence time of 2.27 days (calculated in the previous section). The hydraulic efficiency (Eqn. 2.1) for this system is 15%, very poor.

Both the case of random plant distribution and the case of channels of sparse vegetation were simulated under identical conditions with 80% plant cover. However, the hydraulic efficiency of the two cases are quite different. When the wetland had ransom sparseness, the hydraulic efficiency was 81%. When the wetland had channels of sparse vegetation, the hydraulic efficiency was much lower, 15%. This emphasizes the disastrous consequences of channelization in constructed wetlands.

5.1.4 Numerical experiment: Open-water zones (deep zones)

As demonstrated in the previous sections, the efficiency of wetlands can be severely undermined due to uneven vegetation. The utilization of open-water zones,

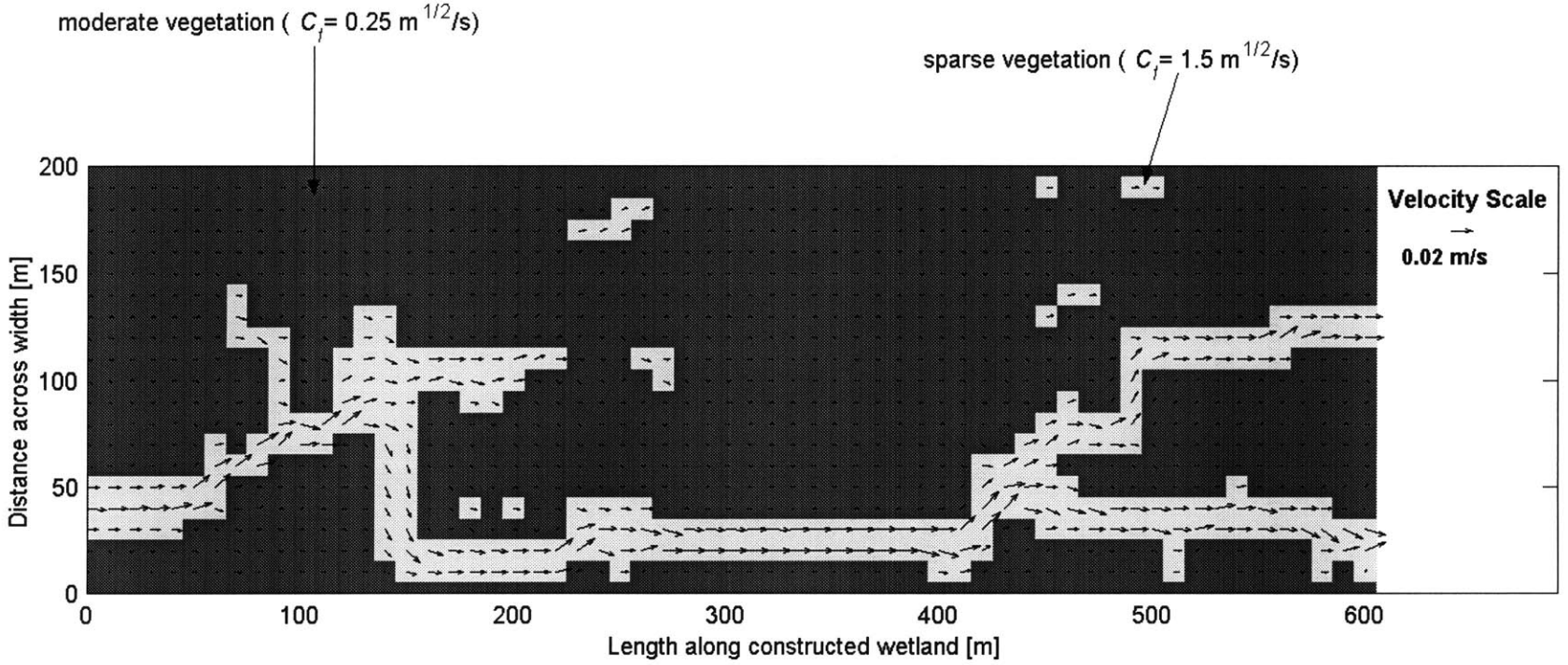


Figure 5-4. 80% Vegetation Cover with Channelization. Light areas indicate zones of sparse vegetation in the constructed wetland. Dark areas are zones of moderately dense vegetation. Flow through the system has short-circuited as a result of the preferred pathway of least resistance. The hydraulic efficiency of this wetland is very poor, 15%.

zones too deep to support rooted vegetation, is highly recommended to redistribute flow in constructed wetlands with uneven vegetation causing a return to plug-flow conditions (Knight & Iverson 1990; Reed & Brown 1992; Reed, Crites, & Middlebrooks 1995; Kadlec & Knight 1996).

Open-water zones, also called deep zones, are approximately 1 m deeper than the rest of the wetland system and have been credited with enhancing atmospheric oxygen diffusion, providing a deep water refuge for fish (pest control) and other wildlife, providing convenient sampling stations for operational research, and most importantly, redistributing flow, increasing hydraulic residence time, and increasing the overall mixing in the wetland (Knight & Iverson 1990; Reed & Brown 1992; Reed, Crites, & Middlebrooks 1995; Kadlec & Knight 1996).

Though deep zones have been cited in many design texts as a method to redistribute flow, modelling of the flow in these zones has not been performed. The best description of the physics governing the redistribution of flow in deep zones is provided by Kadlec and Knight (1996) in which they state that unvegetated cross-ditches create a low resistance path for water to move laterally thus redistributing flow to uniform conditions.

In an attempt to validate the claim that open-water zones can redistribute flow, the constructed wetland with channelized vegetation depicted in Figure 5-4 was modified by adding three deep zones perpendicular to the primary flow direction and centered at x=110, 310, and 510 m. The open-water zones for this experiment are scaled on those presented in Knight and Iverson (1990) (Figure 5-5). The deep zones have a width of 30 m parallel to the flow and length of 200 m perpendicular to the flow, spanning the wetland width. The depth of the open-water zones is approximately 1 m below that of the surrounding sloped bed surface (h=0.3 m). The deepest section of the zones have flat bottoms 10 m in width and 180 m in length (sloped sides of 0.1).

Both the physical bathymetry and decreased resistance to flow (lack of vegetation) of the open-water zones affect the hydrodynamics in the wetland. A Manning coefficient

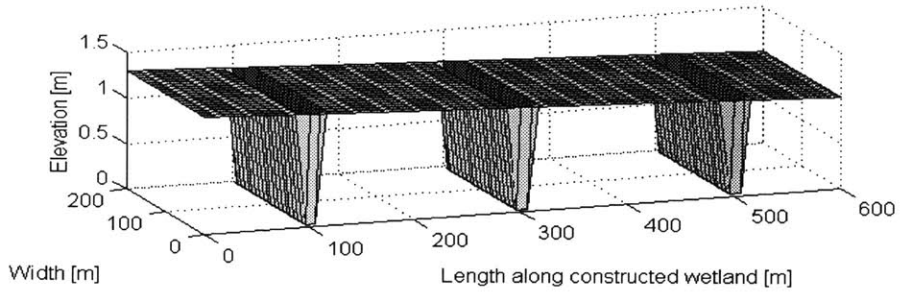


Figure 5-5. Wetland Bed with Open-water Zones. Open-water zones at x=110, 310, and 510 m are oriented perpendicular to flow. These zones scale on those presented in Knight and Iverson (1990). They span a width of 30 m, length of 200 m, and are 1 m deeper than the surrounding sloped bed.

representative of a sluggish channel with no vegetation, C_f=20 $m^{1/2}/s$, was assigned to the open-water zones.

The resulting velocity field is superimposed on a map of vegetation distribution (Figure 5-6). It appears that the open-water zones do not necessarily redistribute flow across the wetland. Flow does not appear to expand creating a uniform flow as is described by Kadlec and Knight (1996). Instead, lateral circulation is created in the deep zones, but otherwise the velocity field structure is not different from that of the channelized sparse vegetation in Figure 5-4 without the open-water zones. It should be noted that a more resolved grid must be used to simulate flow through open-water zones before these exploratory results can be confirmed.

These results appear to support the argument that deep zones provide extra detention time due to the increased volume of water that has become part of the flow path. The degree of mixing of slow and fast parcels of water may also be attributed to open-water zones; water enters the deep zones both through the channel and moderately vegetated areas and then mixes due to circulation in the zones. However, this numerical experiment suggests that open-water zones are not responsible for redistributing flow to plug-flow conditions though further study is necessary.

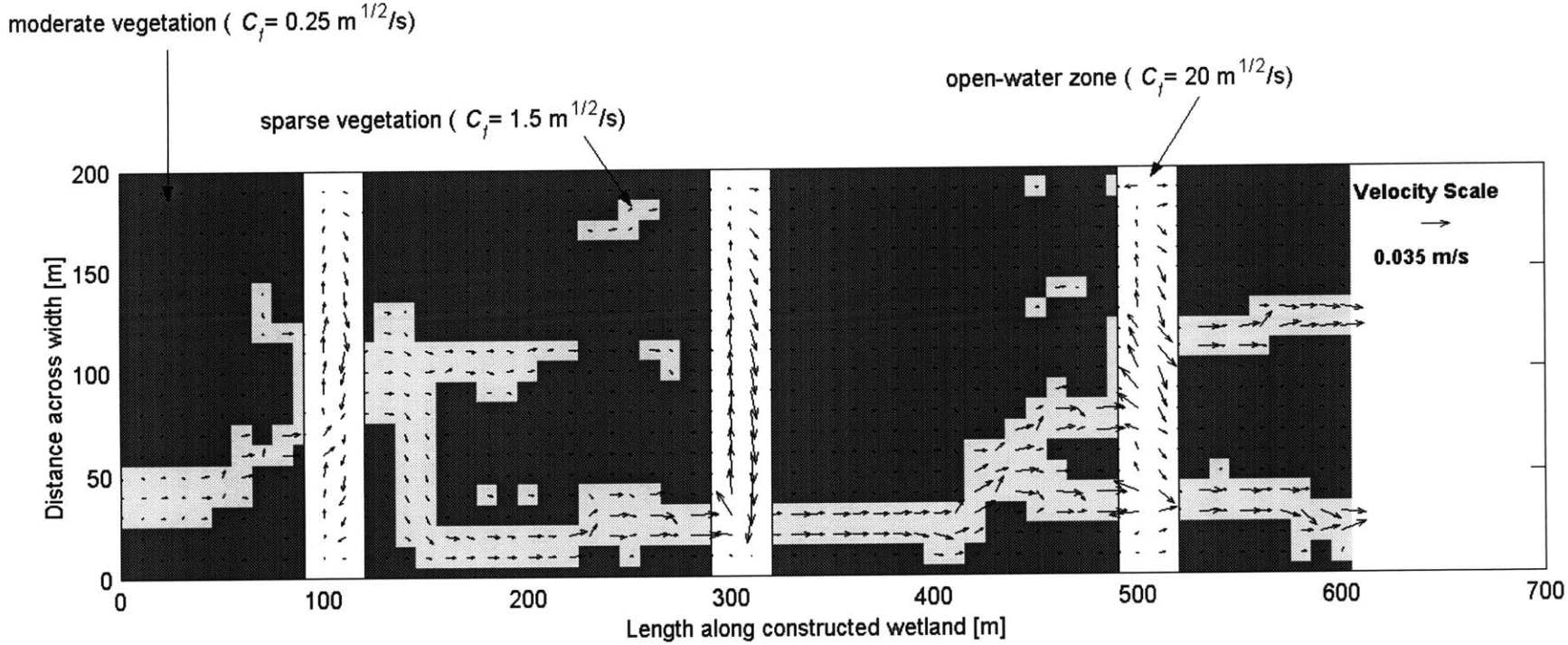


Figure 5-6. Flow Through Open-water Zones. Open-water zones are suggested in wetland design to redistribute flow across the wetland, increase hydraulic detention time, and increase mixing. These results, depicting deep zones as C_f=20 m$^{1/2}$/s, appear to support the case for increased detention time and increased mixing, but not redistribution of flow.

5.1.5 Summary

Using hydrodynamic models or hydrodynamic models coupled with fate and transport, constructed wetland designers can gain insight that may be extremely useful in designing more efficient wetlands. In this exercise, I was able to demonstrate how modelling uneven vegetation in constructed wetlands can provide useful information to a designer.

When uneven plant coverage was introduced, the flowfield deviated from plug-flow. Randomly sparse vegetation effectively increases the bulk C_f of a wetland. As the wetland becomes increasingly sparse, the drag and the hydraulic efficiency of the system decreases proportionally. Channels created much larger problems than random sparseness. The effective residence time of a wetland with channelized vegetation was six times less than the nominal detention time of the system (hydraulic efficiency of 15%). A wetland under identical conditions except with randomly distributed sparse vegetation possessed a hydraulic efficiency of 81%. Lastly, open-water zones created zones of lateral circulation, thereby increasing mixing and the effective residence time in the constructed wetland. Open-water zones did not appear to redistribute flow to plug-flow conditions as is generally described though the physics of this system requires further investigation. Information provided by this type of modelling can help designers better evaluate the cost-effectiveness of defensive mechanisms against short-circuiting.

In this study, vegetation was structured randomly or as channels. To model vegetation more realistically, it may be useful to draw from techniques developed for the field of groundwater hydrology. Developing methods to better describe soil heterogeneity is a topic of extensive study in groundwater hydrology because soil heterogeneity has large effects on flow through porous media. Geostatistical concepts have proven useful in representing the heterogeneous nature of soil (Domenico & Schwartz 1998).

Geostatistical models of hydraulic conductivity (capacity of a medium to transmit water) represent heterogeneity by a small number of statistical parameters such as mean, variance, and correlation length scales. The correlation length is a measure of the spatial persistence of zones of similar properties. Though some sparse vegetation may be completely random, it is likely that vegetation is correlated such that two measurements

taken close together are quite similar, but as measurements are separated by larger distances they become much less similar. Applying geostatistical models to describe the heterogeneity of vegetation in wetlands may prove to be useful and warrants further study.

5.2 Flow Through Natural Wetlands

In some cases, natural wetlands are used for water treatment instead of constructed wetlands. This may reduce construction costs, but the use of a natural wetland requires the use of an existing water course. Natural wetland geometries promote more circulation than the typical rectangular constructed wetland shape. This may lead to a less than ideal layout from a water treatment standpoint.

This study examines flow through typical natural wetlands. The generic geometry used in this study consists of a narrow inlet and outlet and wide rectangular basin (Figure 5-7). This wetland shape was chosen because it was considered the simplest geometry which would emulate the circulation seen in natural wetlands. The inflow and outflow were not placed in the center of the wetland basin because no additional information is gained from such a configuration due to symmetry. The basin has a water depth of 1.5 m, no bed slope, and a υ_t of 0.1 m^2/s. These conditions apply throughout this study.

The generic geometry used in this study is scaled on the Upper Forebay (UFB) of the Mystic Lakes located in Winchester, MA. This natural free surface wetland has been the subject of many environmental studies because it receives elevated levels of toxic heavy metals via its inflow. Field studies from the Upper Forebay will be used to support the numerical results for the generic natural wetland.

5.2.1 Bi-modal flow behavior

The flow structure of a natural wetland is the principal factor governing its water quality enhancement potential. Longer residence times create additional time for the physical, chemical, and biological reactions which enhance water quality to take place. At first glance it may seem that the circulation in natural wetlands will increase the detention time of a wetland and improve water quality. In practice, this is not true. Water in the

circulation zones is unavailable to exit the wetland forcing water from other areas to exit the natural wetland in a time less than the nominal detention time.

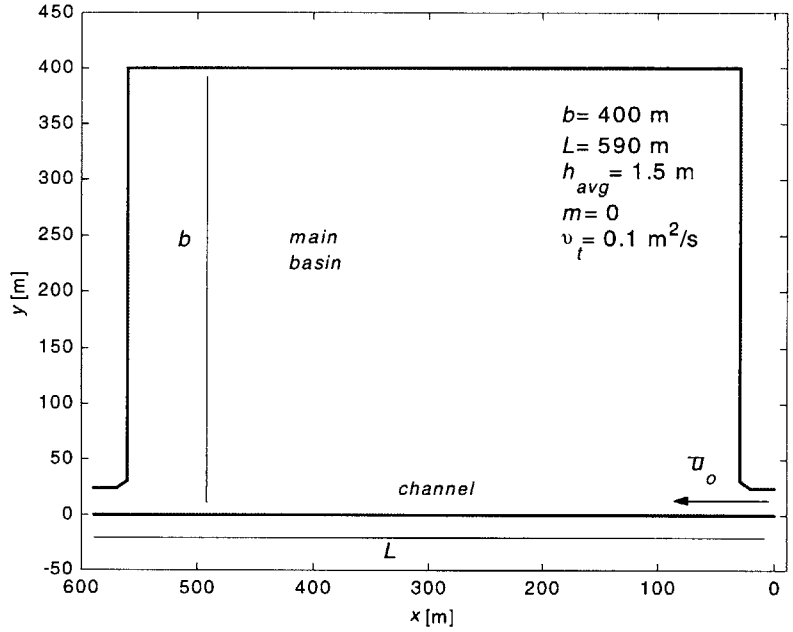


Figure 5-7. Generic Geometry of a Natural Wetland. Natural wetlands are often characterized by narrow inlets and outlets and much wider main basins. The above basin shape was chosen because it is a simple geometry which possesses these characteristics. This natural wetland is scaled on the dimensions of the Upper Forebay of the Mystic Lakes, a site of field experiments which are used to support the numerical results in this study.

Andradottir (1997) describes flow through natural wetlands as bi-modal. During low flows, a wetland is described as well-mixed. During high flows, such as during a storm, inflow momentum controls circulation in a wetland causing short-circuiting. This theory was developed through a combination of field experiments and conceptual modelling.

To better understand these two flow regimes from a hydrodynamic standpoint, I non-dimensionalized the steady-state momentum equation in the primary direction of flow, the x-direction:

$$\left(\overline{u}_* \frac{\partial \overline{u}_*}{\partial x_*}\right) = -\frac{g h_{avg}}{\overline{u}_o^2}\left(\frac{\partial h_*}{\partial x_*}\right) - \frac{gL}{C_f^2 h_{avg}}\left(\frac{\overline{u}_*^2}{h_*}\right) + \left[\frac{2\upsilon_t}{\overline{u}_o L}\left(\frac{\partial^2 \overline{u}_*}{\partial x_*^2}\right) + \frac{\upsilon_t L}{\overline{u}_o b^2}\left(\frac{\partial^2 \overline{u}_*}{\partial y_*^2}\right)\right] \quad \textbf{(5.5)}$$

where h_{avg} is average water depth in the basin,

L is the basin length,

b is the basin width, and

$\overline{u}_0$ is the depth-averaged inlet velocity (Figure 5-7). The variables are normalized as follows: $\overline{u}_* = \overline{u} / \overline{u}_o, h_* = h / h_{avg}, x_* = x / L$, and $y_* = y / b$. The bed slope of the wetland is assumed to be zero to simplify the parameter analysis.

The non-dimensional scale factor of the pressure term is Fr^{-2} (Froude number^{-2}). Under typical wetland conditions of $\overline{u}_o$=$O(10^{-3}$ to 10^{-1} m/s) and h_{avg}=$O(10^{-1}$ to 1 m), Fr=$O(10^{-4}$ to $10^{-1})$. Under typical Upper Forebay conditions of $\overline{u}_o$=$O(10^{-3}$ to 1 m/s) and h_{avg}=1.5 m, Fr=$O(10^{-4}$ to $10^{-1})$. This implies that the pressure forcing term of the momentum equation is always important under these conditions (Fr^{-2}=$O(10^2$ to 10^8)).

The non-dimensional scale factors for the turbulence terms are proportional to the Re^{-1} (Reynolds number^{-1}). These scale factors are negligible, $O(10^{-4}$ to $10^{-1})$, for the typical conditions described above.

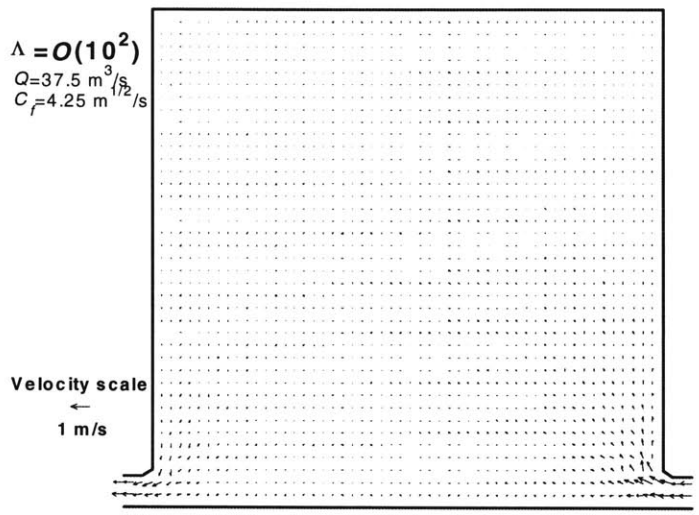
The remaining non-dimensional scale factor precedes the drag term and is defined as

$$\Lambda = \left(\frac{gL}{C_f^2 h_{avg}} \right). \tag{5.6}$$

Lambda is a function of C_f and basin geometry. In the hydrodynamic model, C_f was defined purely as a function of water depth and plant density, though it is also a function of velocity. This will have important consequences which are explored later. When Λ>>1, bed drag dominates the inertial forcing in the system. Alternatively, if Λ<<1, the inertial forcing dominates the bed drag. Recall, the primary balancing force in both regimes would be the pressure forcing. These two scenarios create two very different flow regimes similar to the bi-modal behavior described by Andradottir (1997). Figure 5-8 depicts the two flow regimes numerically simulated for the generic natural wetland for conditions in which Λ>1 and Λ<1.

The flow for the case in which bed drag is balancing the pressure forcing (Λ>>1) has no preferred path. Inflow to the wetland fills the entire wetland volume and velocities

(a)



(b)

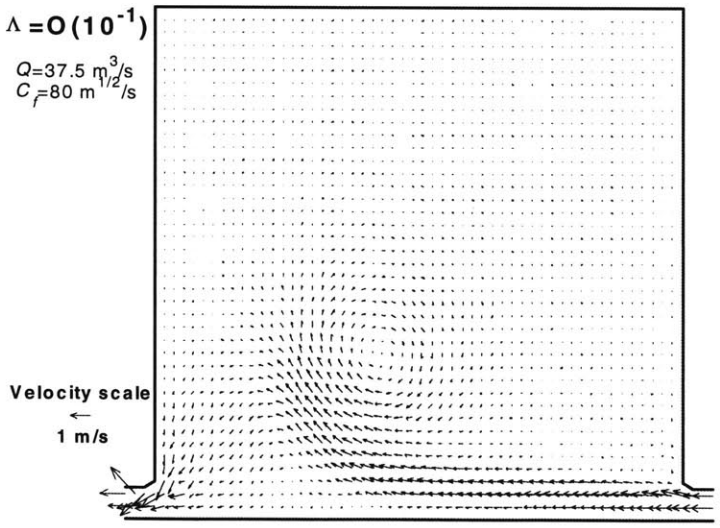


Figure 5-8. Bi-modal Flow Behavior: (a) drag-dominated ($\Lambda=O(10^2)$), and (b) inertia-dominated ($\Lambda=O(10^{-1})$). When bed drag is balancing the pressure forcing ($\Lambda>>1$), there appears to be no preferred path. Inflow to the wetland fills the entire wetland volume. The detention time under this flow regime is near the nominal detention time of the basin. When $\Lambda<<1$, the inertial forcing dominates the bed drag created by the bed and aquatic plants. Inflow momentum causes some flow to travel directly from the inlet to the outlet while other flow recirculates in the main basin. Identical vector scaling is used on both plots.

over the entire basin are the same order of magnitude. The detention time under this flow regime is near the nominal detention time of the basin. When $\Lambda<<1$, the inertial forcing dominates the bed drag created by the bed and aquatic plants. Inflow acts like a jet entering a basin. A coherent vortex is created due to shear instability in the flow between the direct pathway of the inlet and outlet and the main basin (Kimura & Hosoda 1997). Velocity magnitudes are much greater for this case than in the drag-dominated case. This flow regime creates short-circuiting. Some flow travels directly from the inlet to the outlet while other flow recirculates in the main basin.

Figure 5-8a is a simulation for Q=37.5 m^3/s (flowrate) and C_f=4.25 $m^{1/2}/s$ (Eq. 3.9: uniform moderate vegetation). Figure 5-8b is a simulation for the identical conditions except the Chezy coefficient, C_f, is artificially increased to 80 $m^{1/2}/s$ (concrete channel; not a wetland). Both plots have the identical vector scaling.

Numerical simulations in this study were performed on a grid (41 x 60) with grid spaces 10 m by 10 m. A timestep of 0.1 s was used. Flowrate per unit width, q [m^2/s], was assigned as the inlet boundary condition. The outlet boundary condition was a constant water level of h=1.5 m. The characteristics of the numerical experiments in this study of flow through natural wetlands are compiled in Table 5-1.

The viscous scaling factor in Table 5-1 is defined as

$$\frac{2\upsilon_t}{\overline{u}_o L} \tag{5.7}$$

since both non-dimensional scaling factors in the viscous term are the same order of magnitude. The pressure scaling factor is represented by

$$\frac{gh_{avg}}{\overline{u}_o^2}. \tag{5.8}$$

The minimum detention time, t_{min}, of each experiment is defined as the shortest time for flow to travel from the inlet to the outlet of the natural wetland. It is calculated from the velocity

information in the second row from the bottom solid boundary (channel) using the following equation:

TABLE 5-1: NUMERICAL EXPERIMENTS OF FLOW THROUGH NATURAL WETLANDS							
Figure	Description	Q (m^3/s)	C_f ($m^{1/2}/s$)	Λ (-)	viscous scaling factor (-)	pressure scaling factor (-)	t_{min} (hr)
5-8 a	drag-dominated	37.5	4.25	$O(10^2)$	$O(10^{-4})$	$O(10)$	1.8
5-8 b	inertia-dominated	37.5	80	$O(10^{-1})$	$O(10^{-4})$	$O(10)$	0.6
5-9 a	UFB (non-storm)	0.12	4.25	$O(10^2)$	$O(10^{-1})$	$O(10^6)$	180
5-10 a	UFB (storm)	6.16	4.25	$O(10^2)$	$O(10^{-3})$	$O(10^2)$	7.5
5-11 a	UFB (storm)	6.16	30	$O(1)$	$O(10^{-3})$	$O(10^2)$	4.0
5-11 b	UFB (storm)	6.16	60	$O(1)$	$O(10^{-3})$	$O(10^2)$	2.6

$$t_{\min} = \sum_{i=1}^{60} \left(\frac{\Delta x}{\overline{u}_i} \right). \tag{5.9}$$

where

$\overline{u}_i$ is the component of velocity in the x-direction at each grid space in the row.

5.2.2 Field observations

To confirm the validity of the bi-modal flow theory depicted in Figure 5-8, I compared results from model simulations with those from two field studies which showed characteristics of the two regimes. Both studies were conducted on the Upper Forebay of the Mystic Lakes. One study was conducted under non-storm conditions and the other under storm conditions. During the storm conditions, the flowrate to the Upper Forebay increased by an order of magnitude causing a shift from the drag-dominated flow regime to the inertia-dominated flow regime.

Andradottir (1997) performed a dye study in the Upper Forebay under non-storm conditions (Q=0.12 m^3/s). Since 30% of the Upper Forebay is covered by moderately dense submerged and emergent plant species, C_f=4.25 $m^{1/2}/s$ was used in the simulation. Figure 5-9 contains the results from the non-storm dye study conducted on August 29-30, 1996 along with a numerical simulation under these conditions.

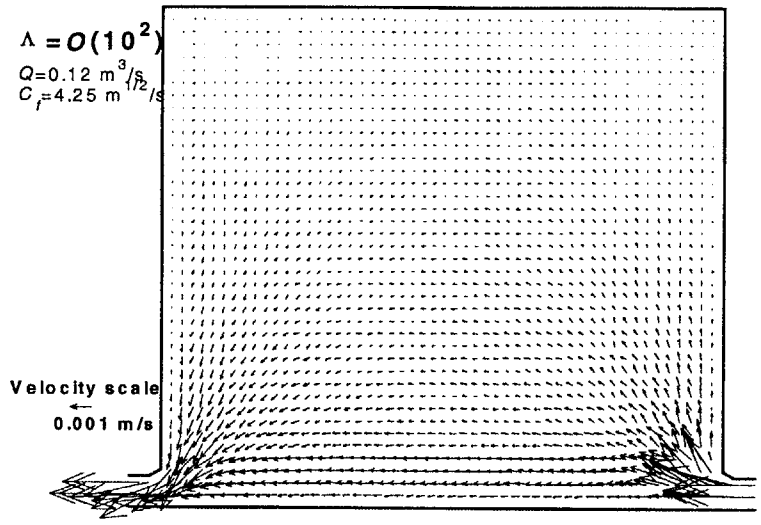
Under these non-storm conditions, $\Lambda=O(10^2)$. As a result, the simulated flow structure possesses the characteristics of drag-dominated flow. Flow fills the entire basin with no preferred pathway. The dye study provides support for the model results; a very similar flow signature is evident. The minimum detention time, t_{min}, from the numerical simulation is 7.4 days (180 hrs). The nominal detention time, calculated with Eqn. 2.2, is 13.5 days (UFB volume=140,000 m^3).

During a storm event, temperature data was collected in the Upper Forebay (November 1-3, 1997). Measurements were taken at the inlet, middle of the channel (midway between the inlet and outlet), and the center of the main basin. Temperature probes were placed 10 to 20 cm from the water surface and 10 to 20 cm from the bed at the channel and main basin locations. Only one probe was placed in the inlet because it is shallow (10 to 20 cm from bed). The flowrate during this storm was 6.16 m^3/s. The temperature data and numerical simulation of this storm are in Figure 5-10.

Only flowrate has varied from the non-storm to the storm conditions. Since Λ is not a function of Q, Λ is still on the order of 100. Given the parameter analysis, this implies that bed drag is still dominant and flow should behave in the same manner as it did for the non-storm case. The numerical simulation shows this result. However, the field results show a very different flow structure.

From Figure 5-10b, one can see that as the flowrate into the system increases due to the storm, a pulse of warmer water comes through the inlet and enters the forebay. This warm water serves as a tracer for flow through the system. The warm water pulse arrives at the middle of the channel approximately 1 to 4 hours after entering the system. On the other hand, there is a lag, approximately 1 day, and gradual increase of temperature observed in the center of the main basin. This implies that flow from the inlet travels very quickly from the inlet toward the outlet (short-circuiting) and only enters the main basin after a much longer time period and after some mixing. Using field observations and a transport model, Andradottir estimates that the effective detention time during this storm due to short-circuiting is 3 hrs. The nominal detention time (Eqn 2.2) for this storm flow is approximately 10 hrs. The t_{min} simulated by the model is 7.5 hrs.

(a)



(b)

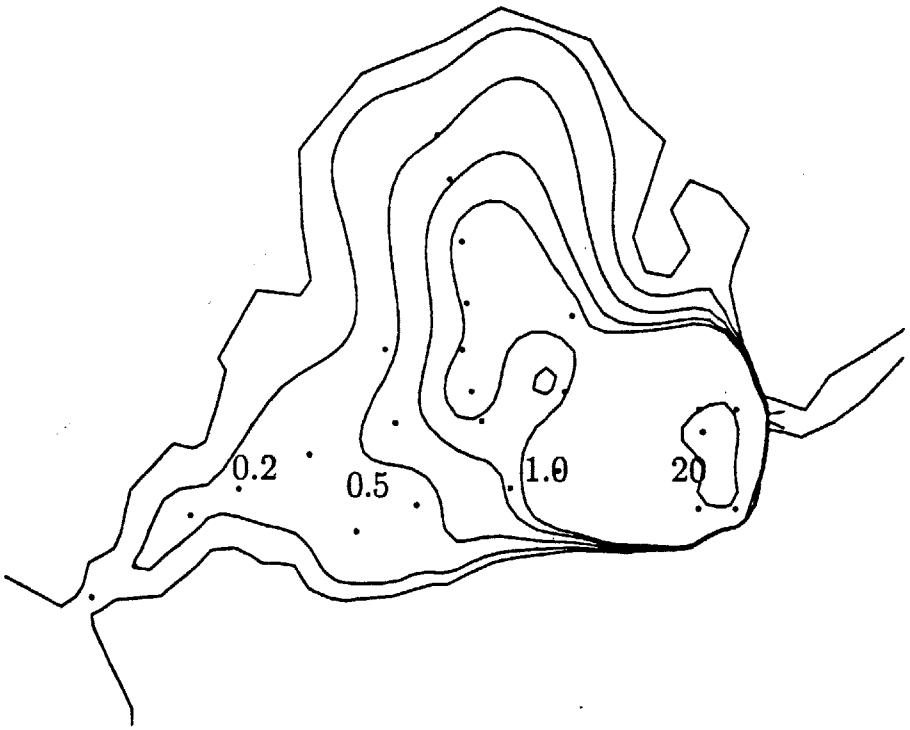
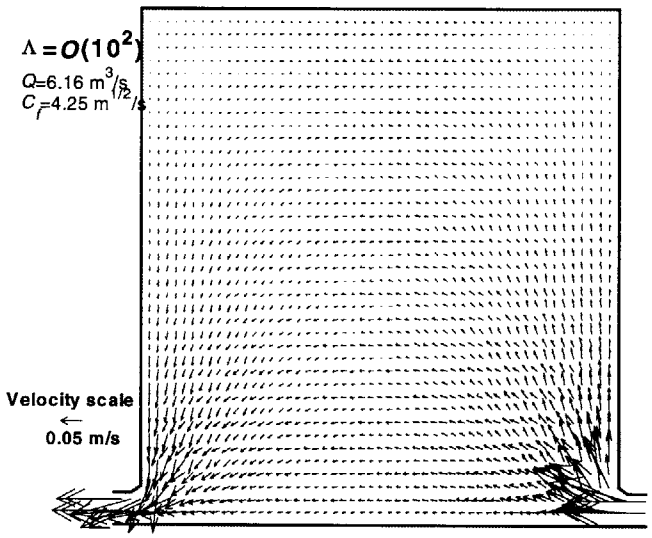


Figure 5-9. Non-storm Conditions in the Upper Forebay: (a) numerical simulation, and (b) field observations (dye study). $\Lambda=O(10^2)$ for these conditions. As a result, the flow structure possesses the characteristics of drag-dominated flow as seen in Figure 5-8a. Flow fills the entire basin. The dye study results show a similar flow signature. Contours depict dye concentrations in μg/L and dots indicate observation locations. (Figure b: Andradottir 1997)

(a)



(b)

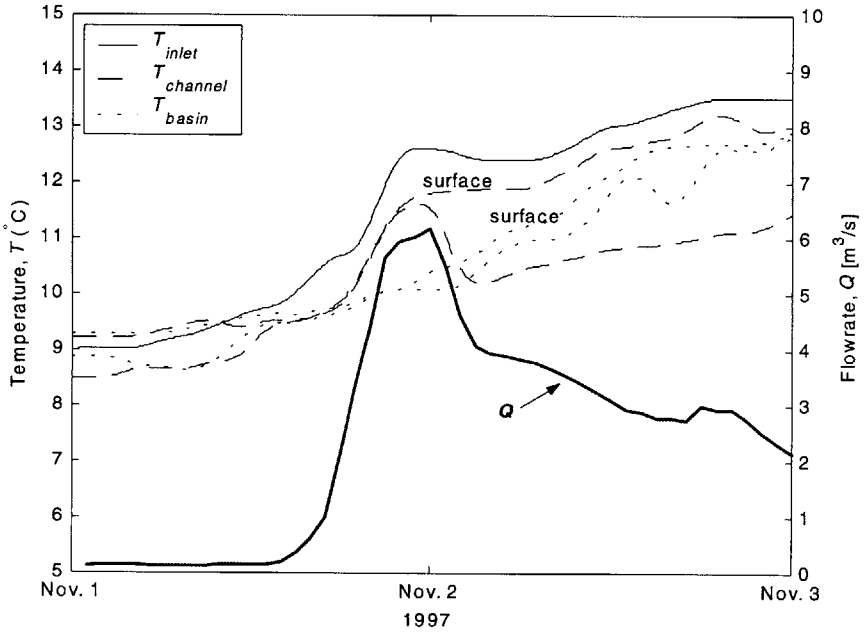


Figure 5-10. Storm Conditions in the Upper Forebay: (a) numerical simulation, and (b) field observations (temperature data). The numerical model simulated a drag-dominated flow structure. The field study conflicts with these results. As the flowrate to the UFB increases, it carries warmer water which acts as a tracer of flow. Flow in the channel (observation midway between the inlet and outlet) receives flow from the storm after 1 to 4 hrs (short-circuiting). Observations in the main basin show a lag, 1 day, and mixing before the warm water from the storm reaches the main basin. (Figure b: unpublished data Andradottir).

The observations from the Upper Forebay during the storm clearly imply an inertia-dominated flow structure such as that in Figure 5-8b though the numerical simulation does not. The parameter Λ suggests that the bi-modal behavior depicted in Figure 5-8 is only a function of C_f. All other variables in the parameter are given by the geometry of the wetland. The field results presented suggest that the bi-modal behavior is also a function of $\overline{u}_o$ though it is not included in the non-dimensional parameter, Λ; a change in flowrate was observed to change the flow structure.

5.2.3 Discussion

The numerical model was not able to capture the short-circuiting behavior induced by the large flowrate of a storm. It depicted the flow through the wetland as drag-dominated since the Chezy coefficient remained constant. It should be made clear that this is not a problem with the physics of the numerical model, but rather the assumption that C_f is not a function of velocity. Remember Λ is a function of C_f and basin geometry only.

In laboratory experiments of flow over flexible strips which emulate aquatic vegetation, three plant behaviors are observed: erect, waving, and prone vegetation. Most natural vegetation is flexible and will deform, vibrate, and sway coherently in the flow of water. In flume experiments by Dunn, Lopez, and Garcia (1996), flexible vegetation deflected from the vertical position (erect condition) with increasing velocity. This is also supported by the work of Vivoni-Gallart (1998). This deflection resulted in an altered plant height and caused a decrease in the Manning coefficient. The mean velocity range in these experiments ($\overline{u}$) was 0.18 to 0.734 m/s. Over this range of velocities, a minimum Manning coefficient of 0.02 $m^{1/6}$ (maximum velocity) and maximum of 0.061 $m^{1/6}$ (minimum velocity) were reported. This corresponds to a 67% decrease in Manning coefficient for an increase in mean velocity from 0.18 to 0.734 m/s or a nearly 200% increase in Chezy coefficient from 13.24 to 39.0 $m^{1/2}/s$.

In a similar study, a decrease in the value of the Manning coefficient was observed as the water depth (h)-to-plant height (h_p) ratio increased (Vivoni-Gallart 1998). This

trend is expected since the vegetation encompasses a smaller portion of the water column and therefore provides less resistance to flow.

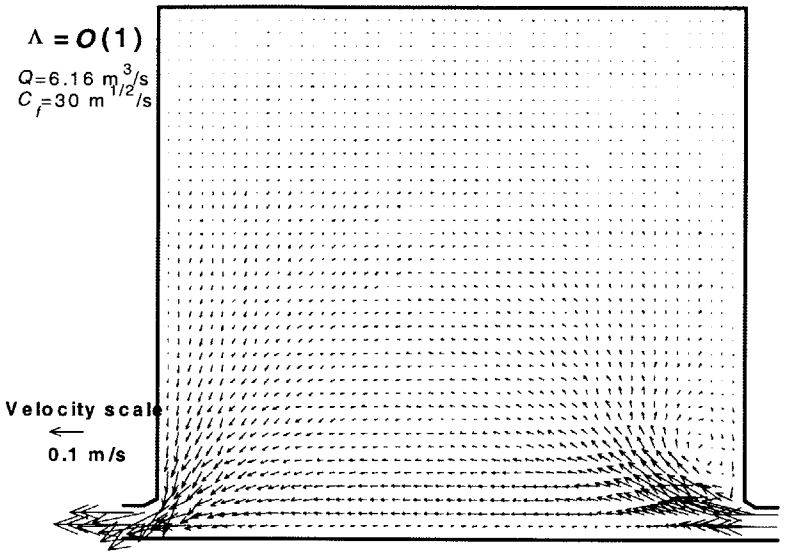
These studies show that both increased water depth and velocity which occur during storm events will contribute to decreased drag. If the UFB plants were deflected, the drag associated with the vegetation will have decreased, an increase in C_f.

The results expected for the UFB storm event are more accurately simulated by increasing C_f, compensating for the potential effects of water depth and velocity on vegetation drag. In Figure 5-11, the storm conditions are simulated with Chezy coefficient values of 30 and 60 $m^{1/2}/s$. The parameter Λ is near its critical value for shifting to inertia-dominated flow. One can see that with C_f=30 $m^{1/2}/s$ the flow is almost short-circuiting and with C_f =60 $m^{1/2}/s$ short-circuiting is achieved (Λ has been lowered). Though a preferred path is evident, a large gyre is not formed because there is not enough inertia in the system. However, circulation patterns can be seen in both simulations at the inlet corner.

From Table 5-1, one can see that as C_f increased, the t_{min} of the natural wetland decreased from 7.5 hrs to 2.6 hrs. Andradottir estimated the storm detention time to be 3 hrs from field observations. This suggests that the increased water depth and velocity as a result of the storm may be responsible for an increase of C_f from 4.25 to 60 $m^{1/2}/s$. However, this cannot be shown conclusively without observations of the aquatic vegetation behavior in the UFB. The increased Chezy coefficient in the work by Dunn, Lope, and Garcia (1996) and Vivoni-Gallart (1998) suggests that such an increase in C_f due to increased water depth and velocity is not unreasonable.

Without a more reliable way of estimating C_f for use in numerical modelling, it will be difficult to have confidence in modelling results. Calibrating a model for a specific site is not enough; under different flowrates the calibrations will be in error. Additional research on accurately characterizing vegetation drag as a function of not only density and water depth, but also velocity is crucial to accurately employing hydrodynamic models to constructed and natural wetland problems.

(a)



(b)

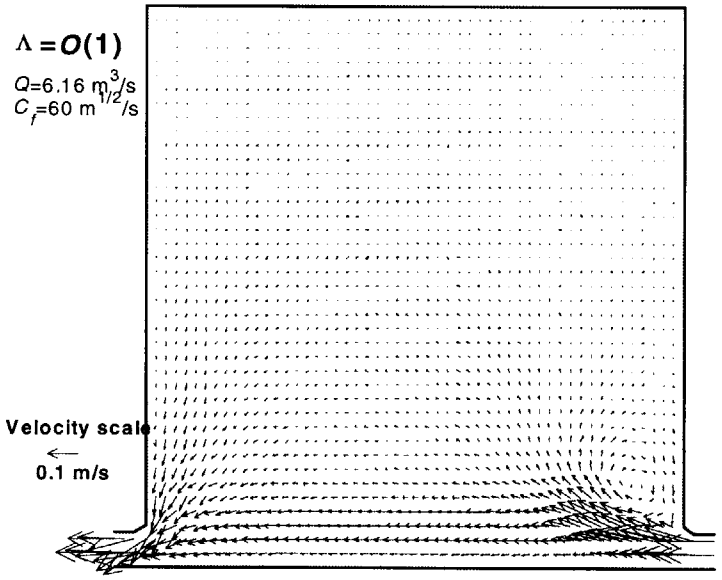


Figure 5-11. Simulated Storm Flow with Modified C_f: (a) C_f=30 m$^{1/2}$/s, and (b)C_f=60 m$^{1/2}$/s. The non-dimensional parameter, Λ, is near its critical value where drag-dominated flow becomes inertia-dominated. For larger Chezy coefficient, Λ decreases and the flow structure becomes fully inertia-dominated. The inertia in this system is not large enough to create the large gyre seen in Figure 5-8. Identical velocity scales are used in both figures.

CHAPTER 6: CONCLUSION

A numerical model was developed to study wetland hydrodynamics. Despite the fact that understanding the hydrodynamics of wetlands is crucial to harnessing their water quality enhancement potential, hydrodynamic modelling of wetlands is only beginning to gain momentum. The model for this study was tested for accuracy with the analytical solution to uni-directional flow in a sloped rectangular channel when the Chezy coefficient was set to zero. It tested well, with a root mean square error of zero to 14 significant digits (processor precision).

The model was applied to both constructed and natural wetlands to showcase its versatility and gain insight into the following topics:

- flow through non-uniform aquatic vegetation in constructed wetlands,
- open-water or deep zones in constructed wetlands, and
- the bi-modal flow structure of natural wetlands.

Conclusions from each topic of study will be addressed in turn.

Using a typical constructed wetland design, a sloped (0.05%) rectangular channel with an aspect ratio of 3:1, water depth of 0.3 m, and υ_t=0.1 m^2/s, flow through non-uniform vegetation was investigated. To begin, uniform moderately dense vegetation (C_f =0.25 $m^{1/2}/s$) was tested to verify the accuracy of the model. Plug-flow was observed, but as vegetation cover decreased to 80% moderately dense vegetation (C_f=0.25 $m^{1/2}/s$) and 20% sparse (C_f=1.5 $m^{1/2}/s$), deviations from uniform flow became significant. The heterogeneity of the plant cover was applied by varying C_f at grid cells both randomly and as channels. Channels are often created due to an increasing percentage of sparse zones causing connectivity amongst them, or human and animal paths.

A random vegetation pattern with 80% plant coverage created a deviation from plug-flow resulting in a hydraulic efficiency of 81%. For comparison, the hydraulic efficiency for a wetland with uniform coverage is 100% (plug-flow conditions). As uniform vegetational cover becomes increasingly sparse in a random fashion, the bulk

Chezy coefficient for the wetland will increase proportionally and the hydraulic efficiency will decrease proportionally.

When channels of sparse vegetation exist, preferred flow paths in these channels form and the hydraulic efficiency of the constructed wetland plummets (efficiency of 15%). This is comparable to the 81% efficiency when the vegetation was randomly distributed.

Heterogeneity of vegetation cover in constructed wetlands is largely responsible for the non-uniform flow typical to them. Large amounts of time and energy are focused on achieving good aquatic plant coverage. This time and effort appears to be substantiated given its potential impact on hydraulic efficiency and thus water quality enhancement. Extreme care must be taken to avoid the creation of channels due to their more destructive impact on hydraulic efficiency.

Open-water zones or deep zones are recommended by designers as a method to control flow through wetlands and create a return to plug-flow. Deep zones, over one m in depth, are too deep for rooted plants (C_f =20 $m^{1/2}/s$) and are credited with redistributing flow to uniform flow, increasing hydraulic detention time, and increasing mixing.

Open-water zones were placed in the constructed wetland with channels of sparse vegetation. These zones, scaled on those used in practice, are 30 m by 200 m spanning the entire width of the wetland. Simulations of flow though the system with deep zones, resulted in a similar flow structure to the case without open-water zones. However, flow in the open-water zones recirculates. Results from this numerical experiment suggest that open-water zones increase detention time (increased volume in flow), and increase mixing (recirculation), but do not necessarily redistribute flow. A more detailed simulation of flow through open-water zones is necessary to provide conclusive evidence.

Lastly, a set of numerical experiments were performed on the generic geometry of a natural wetland. This generic wetland is characterized by a narrow inlet and outlet (25 m), and wide rectangular main basin (400 m) of length (590 m). This wetland is scaled on the Upper Forebay of the Mystic Lakes in Winchester, MA.

Two distinctly different flow structures exist for inertia-dominated and drag-dominated flow. The non-dimensional parameter Λ defines the shift from inertia- ($\Lambda<<1$) to drag-dominated ($\Lambda>>1$) flow. Drag-dominated flow enters and fills the entire wetland; the effective detention time during these conditions is very close to the nominal detention time of the wetland. Inertia-dominated flow short-circuits causing some flow to travel directly from the inlet to the outlet. A recirculating gyre may form if the momentum of the flow is large enough to sustain it. These two flow regimes may exist for a given basin alternating between the two regimes in time in response to changes in the variables that affect Λ.

To compare model results with observation, field studies conducted by Andradottir (1997 & unpublished) in the UFB were used. Both non-storm and storm conditions were studied. The non-storm field results show a flow structure similar to the solution predicted by the model ($\Lambda>>1$). The storm results from the field, on the other hand, show short-circuiting while the model predicts drag-dominated flow.

A closer examination of Λ shows a dependence only on wetland geometry and C_f. The bed stress formulation of the Navier Stokes equations assumed the Chezy coefficient was not a function of velocity. For this reason, an increase in velocity (storm) will not result in the shift to inertia-dominated flow in the model though it is seen in nature. Unless C_f is defined as a function of velocity, numerical modelling will not be able to capture the bi-modal flow structure seen in the field.

REFERENCES

Alizai, S.A.K., and J. McManus. 1980. The significance of reed beds on siltation in the Tay Estuary. *Proceedings of the Royal Society of Edinburgh* 78B: s1-s13.

Andradottir, H.O. 1997. *Circulation and mixing in the Upper Forebay of the Mystic Lake system, Winchester, Massachusetts.* M.S. Thesis. Department of Civil and Environmental Engineering. Massachusetts Institute of Technology.

Andreae, M.O., and D. Klumpp. 1979. Biosynthesis and release of organoarsenic compounds by marine algae. *Environmental Science and Technology* 13(6): 738-741.

Aurilio, A.C. et al. 1995. Sources and distribution of arsenic in the Aberjona Watershed, Eastern Massachusetts. *Water, Air, and Soil Pollution* 81: 265-282.

Aurillo, A.C., R.P. Mason, and H.F. Hemond. 1994. Speciation and fate of arsenic in three lakes of the Aberjona Watershed. *Environmental Science and Technology* 28(4): 577-585.

Azadpour, A., and J.E. Matthews. 1996. Remediation of metal-contaminated sites using plants. *Remediation* Summer: 1-18.

Barber, R.W. et al. 1997. Numerical modelling of wind-induced circulation in the Gulf of Thermaikos using a non-orthogonal boundary-fitted coordinate system. In *Computer modelling of seas and coastal regions III*, ed. J.R. Acinas, and C.A. Brebbia, 13-22. Boston: Computational Mechanics Inc.

Barrett, K.R. 1996. *Two-dimensional modeling of flow and transport in treatment wetlands: Development and testing of a new method for wetland design and analysis.* Ph.D. Thesis. Department of Civil Engineering. Northwestern University.

Baumert, H., and G. Stoyan. 1990. Operational forecasting of toxic waves in rivers. *Acta Hydrochimica et Hydrobiologica* 18(4): 449-458.

Bencala, K.E., and R.A. Walters. 1983. Simulation of solute transport in a mountain pool-and-riffle stream: A transient storage model. *Water Resources Research* 19(3): 718-724.

Blake, G. et al. 1987. Incorporation of cadmium by water hyacinth. *Water Science and Technology* 19(10): 123-128.

Buchberger, S.G., and G.B. Shaw. 1995. An approach toward rational design of constructed wetlands for wastewater treatment. *Ecological Engineering* 4: 249-275.

Capriulo, G.M. 1995. Marine microbial ecology. In *Encyclopedia of environmental biology*, ed. W.A. Nierenberg, 483-506. San Diego: Academic Press Inc.

Chigbo, F.E., R.W. Smith, and F.L. Shore. 1982. Uptake of arsenic, cadmium, lead, and mercury from polluted waters by the water hyacinth *Eichornia crassipes. Environmental Pollution* 27(Series A): 31-36.

Cooper, P.F., and B.C. Findlater. 1990. *Constructed wetlands in water pollution control.* New York: Pergamon Press, Inc.

Corbitt, R.A., and P.T. Bowen. 1994. Constructed wetlands for wastewater treatment. In *Applied wetlands science and technology*, ed. D.M. Kent, 221-241. Boca Raton, FL: CRC Press Inc.

Domenico, P.A., and F.W. Schwartz. 1998. *Physical and chemical hydrogeology.* New York: John Wiley & Sons, Inc.

Dunn, C., F. Lopez, and M. Garcia. 1996. Mean flow and turbulence in a laboratory channel with simulated vegetation. *Civil Engineering Studies: Hydraulic Engineering Series* 51.

Durant, J.L, J.J. Zemach, and H.F. Hemond. 1990. The history of leather industry waste contamination in the Aberjona Watershed: A mass balance approach. *Civil Engineering Practice* 5: 41-66.

Feng, K., and F.J. Molz. 1997. A 2-D, diffusion-based, wetland flow model. *Journal of Hydrology* 196: 230-250.

Fischer, H.B. et al. 1979. *Mixing in inland and coastal waters.* London: Academic Press.

Guardo, M., and R.S. Tomasello. 1995. Hydrodynamic simulations of a constructed wetland in South Florida. *Water Resources Bulletin* 31(4): 687-701.

Hammer, D.A. 1989. *Constructed wetlands for wastewater treatment.* Chelsea, MI: Lewis Publishers, Inc.

Hammer, D.A. 1995. Water quality improvement functions of wetlands. In *Encyclopedia of environmental biology*, ed. W.A. Nierenberg, 485-516. San Diego: Academic Press Inc.

Hammer, D.A. 1997. *Creating freshwater wetlands.* Boca Raton, FL: CRC Press Inc.

Hicks, B.B. 1975. A procedure for the formulation of bulk transfer coefficients over water. *Boundary-Layer Meteorology* 8: 515-524.

Hicks, B.B, R.L. Drinkrow, and G. Grauze. 1974. Drag and bulk transfer coefficients associated with shallow water surface. *Boundary-Layer Meteorology* 6: 287-297.

Hoffman, J.D. 1992. *Numerical methods for engineers and scientists.* New York: McGraw-Hill, Inc.

Hotchkiss, N. 1972. *Common marsh, underwater, and floating-leaved plants of the United States and Canada.* New York: Dover Publications, Inc.

Imberger, Jorg, and J.C. Patterson. 1990. Physical limnology. In *Advances in applied mechanics,* 303-455. San Diego: Academic Press Inc.

Jadhav, R.S., and S.G. Buchberger. 1995. Effects of vegetation on flow through free water surface wetlands. *Ecological Engineering* 5: 481-496.

Jordan, T.E., J.W. Pierce, and D.L. Correll. 1986. Flux of particulate matter in the tidal marshes and subtidal shallows of the Rhode River Estuary. *Estuaries* 9(4B): 310-319.

Kadlec, R.H. 1990. Overland flow in wetlands: Vegetation resistance. *Journal of Hydraulic Engineering* 116(5): 691-706.

Kadlec, R.H. 1994. Detention and mixing in free water wetlands. *Ecological Engineering* 3: 345-380.

Kadlec, R.H., and R.L. Knight. 1996. *Treatment wetlands.* Boca Raton, FL: CRC Press Inc.

Kadlec, R.H., W. Bastiaens, and D.T. Urban. 1993. Hydrological design of free water surface treatment wetlands. In *Constructed wetlands for water quality improvement,* ed. G.A. Moshiri, 77-86. Boca Raton, FL: CRC Press Inc.

Kaise, T. et al. 1997. Biomethylation of arsenic in an arsenic-rich freshwater environment. *Applied Organometallic Chemistry* 11(4): 297-304.

Kent, D.M. 1994. *Applied wetlands science and technology.* Boca Raton, FL: CRC Press Inc.

Kim, I. 1996. Harnessing the green clean. *Chemical Engineering* 103(12): 39-41.

Kimura, I., and T. Hosoda. 1997. Fundamental properties of flows in open channels with dead zone. *Journal of Hydraulic Engineering* 123(2): 98-107.

Knight, R.L., and M.E. Iverson. 1990. Design of Fort Deposit, Alabama, constructed wetlands treatment system. In *Constructed wetlands in water pollution control,* ed. Cooper, P.F., and B.C. Findlater, 521-524. New York: Pergamon Press, Inc.

Kraus, M.L. 1987. Wetlands: Toxicant sinks or reservoirs? In *National wetland symposium: Wetland hydrology,* ed. J.A. Kusler, and G. Brooks, 192-196.

Lee, C.K., K.S. Low, and N.S. Hew. 1991. Accumulation of arsenic by aquatic plants. *The Science of the Total Environment* 103: 215-227.

Leonard, L.A., A.C. Hine, and M.E. Luther. 1995. Surficial sediment transport and deposition processes in a *Juncus roemerianus* marsh, West-central Florida. *Journal of Coastal Research* 11(2): 322-336.

Liggett, J.A. Lake circulation. In *Unsteady flow in open channels*, ed. K. Mahmood, and V. Yevjevich, 879-923. Fort Collins, CO: Water Resources Publications.

Maeda, S. et al. 1987. Methylation of inorganic arsenic by arsenic-tolerant freshwater algae. *Applied Organometallic Chemistry* 1: 465-472.

Maeda, S. et al. 1990. Transformation of arsenic compounds in a freshwater food chain. *Applied Organometallic Chemistry* 4(3): 251-254.

Maeda, S. et al. 1992. Biomethylation of arsenic and its excretion by the alga *Chlorella vulgaris. Applied Organometallic Chemistry* 6(4): 407-413.

Maeda, S. et al. 1992. Uptake and excretion of total inorganic arsenic by the freshwater alga *Chlorella vulgaris. Applied Organometallic Chemistry* 6(4): 399-405.

Magee, D.M. 1981. *Freshwater wetlands: A guide to common indicator plants of the northeast.* MA: The University of Massachusetts Press.

Markert, B. 1993. Instrumental analysis of plants. In *Plants as biomonitors*, ed. B. Markert, 66-102. New York: VCH Publishers Inc.

Marske, D.M., and J.D. Boyle. 1973. Chlorine contact chamber design - A field evaluation. *Water and Sewage Works* 120(1): 70-77.

Mayes, R.A., A.W. McIntosh, and V.L. Anderson. 1977. Uptake of cadmium and lead by a rooted aquatic macrophyte (*Elodea canadensis*). *Ecology* 58: 1176-1180.

Mitsch, W.J., and J.G. Gosselink. 1993. *Wetlands.* New York: Van Nostrand Reinhold.

Molls, T., and M.H. Chaudry. 1995. Depth-averaged open-channel flow model. *Journal of Hydraulic Engineering* 121(6): 453-465.

Moshiri, G.A. 1993. *Constructed wetlands for water quality improvement.* Boca Raton, FL: CRC Press Inc.

Mudroch, A., and J.A. Capobianco. 1979. Effects of mine effluent on uptake of Co, Ni, Cu, As, Zn, Cd, Cr, and Pb by aquatic macrophytes. *Hydrobiologia* 64(3): 223-231.

Mudroch, A., and O. Mudroch. 1977. Analysis of plant material by x-ray fluorescence spectrometry. *X-ray Spectrometry* 6: 215-217.
Nepf, H.M. 1999. Drag, turbulence, and diffusion in flow through emergent vegetation. *Water Resources Research* 35(2): 479-489.

Parr, T.W. 1990. Factors affecting reed (*Phragmies australis*) growth in U.K. reed bed treatment systems. In *Constructed wetlands in water pollution control*, ed. P.F. Cooper, and B.C. Findlater, 67-76. New York: Pergamon Press, Inc.

Prescott, K. 1996. *The application of mass balance and hydrodynamic/pollutant transport models for wetland restoration.* Master of Engineering. Department of Civil Engineering. McMaster University.

Purnama, A. 1988. The effect of dead zones on longitudinal dispersion in streams. *Journal of Fluid Mechanics* 186: 351-377.

Reay, P.F. 1972. The accumulation of arsenic from arsenic-rich natural waters by aquatic plants. *Journal of Applied Ecology* 9: 557-565.

Reddy, K.R. and T.A. DeBusk. 1987. State-of-the-art utilization of aquatic plants in water pollution control. *Water Science and Technology* 19(10): 61-79.

Reed, S.C., and D.S. Brown. 1992. Constructed wetland design: The first generation. *Water Environment Research* 64(6): 776-781.

Reed, S.C., R.W. Crites, and E.J. Middlebrooks. 1995. *Natural systems for waste management and treatment.* New York: McGraw-Hill, Inc.

Sanders, J.G., G.F. Riedel, and R.W. Osman. 1994. Arsenic cycling and its impact in estuarine and coastal marine ecosystems. In *Arsenic and the environment, Part I: Cycling and characterization*, ed. J.O. Nriagu, 289-308. New York: John Wiley and Sons, Inc.

Seemann, H.B. 1996. *Temporal variability of arsenic transport into the Upper Mystic Lake.* M.S. Thesis. Department of Civil and Environmental Engineering. Massachusetts Institute of Technology.

Shih, S.F., and G.S. Rahi. 1982. Seasonal variations of Manning's roughness coefficient in a subtropical marsh. *Transactions of the ASAE* 25(1): 116-119.

Simpson, R.L. et al. 1983. The role of Delaware River freshwater tidal wetlands in the retention of nutrients and heavy metals. *Journal of Environmental Quality* 12(1): 41-48.

Solo-Gabriele, H. 1995. *Metal transport in the Aberjona River system: Monitoring, modeling, mechanisms.* Ph.D. Thesis. Department of Civil and Environmental Engineering. Massachusetts Institute of Technology.

Solo-Gabriele, H.M., and F.E. Perkins. 1997. Metal transport within a small urbanized watershed. *Journal of Irrigation and Drainage Engineering* 123(2): 114-122.

Solo-Gabriele, H.M., F.E. Perkins. 1997. Streamflow and suspended sediment transport in an urban environment. *Journal of Hydraulic Engineering* 123(9): 807-811.

Solo-Gabriele, H.M., and F.E. Perkins. 1997. Watershed-specific model for streamflow, sediment, and metal transport. *Journal of Environmental Engineering* 123(1): 61-70.

Steger, J.L., and R.L. Sorenson. 1979. Automatic mesh-point clustering near a boundary in grid generation with elliptic partial differential equations. *Journal of Computational Physics* 33: 405-410.

Steiner, G.R., and R.J. Freeman. 1989. Configuration and substrate design considerations for constructed wetlands wastewater treatment. In *Constructed wetlands for wastewater treatment*, ed. D.A. Hammer, 363-377. Chelsea, MI: Lewis Publishers, Inc.

Stumpf, R.P. 1983. The process of sedimentation on the surface of a salt marsh. *Estuarine, Coastal, and Shelf Science* 17: 495-508.

Tannehill, J.C., D.A. Anderson, and R.H. Pletcher. 1997. *Computational fluid mechanics and heat transfer*. Washington, D.C.: Taylor and Francis.

Tchobanoglous, G. 1993. Constructed wetlands and aquatic plant systems: Research, design, operational, and monitoring issues. In *Constructed wetlands for water quality improvement*, ed. G.A. Moshiri, 23-34. Boca Raton, FL: CRC Press Inc.

Thackston, E.L., F.D. Shields, and P.R. Schroeder. 1987. Residence time distributions of shallow basins. *Journal of Environmental Engineering* 113(6): 1319-1332.

Thompson, J.F. 1980. Numerical solution of flow problems using body-fitted coordinate system. In *Computational fluid dynamics*, ed. W. Kollmann, 1-98. Washington: Hemisphere Publishing Corporation.

Thompson, J.F., Z.U.A. Warsi, and C.W. Mastin. 1985. *Numerical grid generation: Foundations and applications*. New York: Elsevier Science Publishing Co., Inc.

Tsanis, I.K., and M. Blaisdell. 1991. An interactive graphic display for Eulerian and Lagrangian lake circulation modelling. *Advances in Water Resources* 14(3): 118-124.
Tsanis, I.K., H. Shen, and S. Venkatesh. 1996. Water currents in the St. Clair and Detroit Rivers. *Journal of Great Lakes Research* 22(2): 213-223.

Valentine, E.M., and I.R. Wood. 1977. Longitudinal dispersion with dead zones. *Journal of the Hydraulics Division* 103(HY9): 975-990.

Vivoni-Gallart, E.R. 1998. *Turbulence structure of a model seagrass meadow.* M.S. Thesis. Department of Civil and Environmental Engineering. Massachusetts Institute of Technology.

Vreugdenhil, C.B. 1994. *Numerical methods for shallow-water flow.* Norwell, MA: Kluwer Academic Publishers.

Vymazal, J. 1998. Introduction. In *Constructed wetlands for wastewater treatment in Europe*, eds. J. Vymazal et al., 1-15. Leiden, Netherlands: Backhuys Publishers.

Vymazal, J. et al. 1998. *Constructed wetlands for wastewater treatment in Europe.* Leiden, Netherlands: Backhuys Publishers.

Wade, M.J. et al. 1993. Environmental transformation of toxic metals. *Occupational Medicine* 8(3): 575-601.

Walker, D.J. 1998. Modelling residence time in stormwater ponds. *Ecological Engineering* 10: 247-262.

Watson, J.T., and J.A. Hobson. 1989. Hydraulic design considerations and control structures for constructed wetlands for wastewater treatment. In *Constructed wetlands for wastewater treatment*, ed. D.A. Hammer, 363-377. Chelsea, MI: Lewis Publishers, Inc.

Wetzel, R.G. 1993. Constructed wetlands: Scientific Foundations are critical. In *Constructed wetlands for water quality improvement*, ed. G.A. Moshiri, 3-7. Boca Raton, FL: CRC Press Inc.

Witthar, S.R. 1993. Wetland water treatment systems. In *Constructed wetlands for water quality improvement*, ed. G.A. Moshiri, 147-155. Boca Raton, FL: CRC Press Inc.

Wittig, R. 1993. General aspects of biomonitoring heavy metals by plants. In *Plants as biomonitors*, ed. B. Markert, 3-27. New York: VCH Publishers Inc.

Wood, A. 1990. Constructed Wetlands for wastewater treatment - Engineering and design considerations. In *Constructed wetlands in water pollution control*, ed. P.F. Cooper, and B.C. Findlater, 481-494. New York: Pergamon Press, Inc.

Wood, A. 1995. Constructed wetlands in water pollution control: Fundamentals to their understanding. *Water Science and Technology* 32(3): 21-29.

Wrench, J.J., and R.F. Addison. 1981. Reduction, methylation, and incorporation of arsenic into lipids by the marine phytoplankton *Dunaliella tertiolecta. Canadian Journal of Fisheries and Aquatic Sciences* 38: 518-523.

Wu, J., and I.K. Tsanis. 1994. Pollutant transport and residence time in a distorted scale model and a numerical model. *Journal of Hydraulic Research* 32(4): 583-598.

Ye, J., and J.A. McCorquodale. 1997. Depth-averaged hydrodynamic model in curvilinear collocated grid. *Journal of Hydraulic Engineering* 123(5): 380-388.

Zhang, T. et al. 1990. Metal uptake and associated pollution control by *Typha latifolia* in urban wetlands. In *Constructed wetlands in water pollution control*, ed. P.F. Cooper, and B.C. Findlater, 451-459. New York: Pergamon Press, Inc.

APPENDIX A: FIELD STUDY OF ARSENIC ACCUMULATION BY AQUATIC PLANTS

A.1 Site Description and Motivation

The Aberjona watershed (65 km^2), located in northern Massachusetts, is the site of over a century of industrial activity including leather and chemical manufacturing. This activity has resulted in the release of large amounts of toxic heavy metals such as arsenic (As) and chromium (Cr) into the Aberjona River, the main drainage river of the watershed. These metals are transported by the river and eventually deposited in the Forebays and Mystic Lakes located in Winchester, MA (Figure A-1). These wetlands, Upper and Lower Forebays, and lakes are used for recreational fishing, sailing, and swimming by many local residents. An understanding of the fate and transport of these toxic heavy metals is essential to protecting the environment and determining the contamination's impact on human health in this region. This field study focuses on the Upper Forebay (UFB), a free surface flow wetland which receives direct drainage from the Aberjona River.

A.2 Objective

Wetlands enhance water quality via physical, chemical, and biological processes. Three components are responsible for these processes: vegetation, microbial populations, and gas exchange between the atmosphere and the water column. The principle functions of vegetation in wetlands are to create environments for microbial populations, to obstruct flow thus enhancing sedimentation, and to assimilate nutrients and metals. The ability of plants to accumulate substances on their surface is rarely considered.

The objective of this study was to evaluate the significance of surficial accumulation of arsenic and chromium by the aquatic plants in the Upper Forebay of the Mystic Lake system. This work will provide us with a better understanding of the transport of heavy metals in the Upper Forebay as well as evaluate the importance of surficial plant accumulation of contaminants such as As and Cr in natural and constructed wetlands.

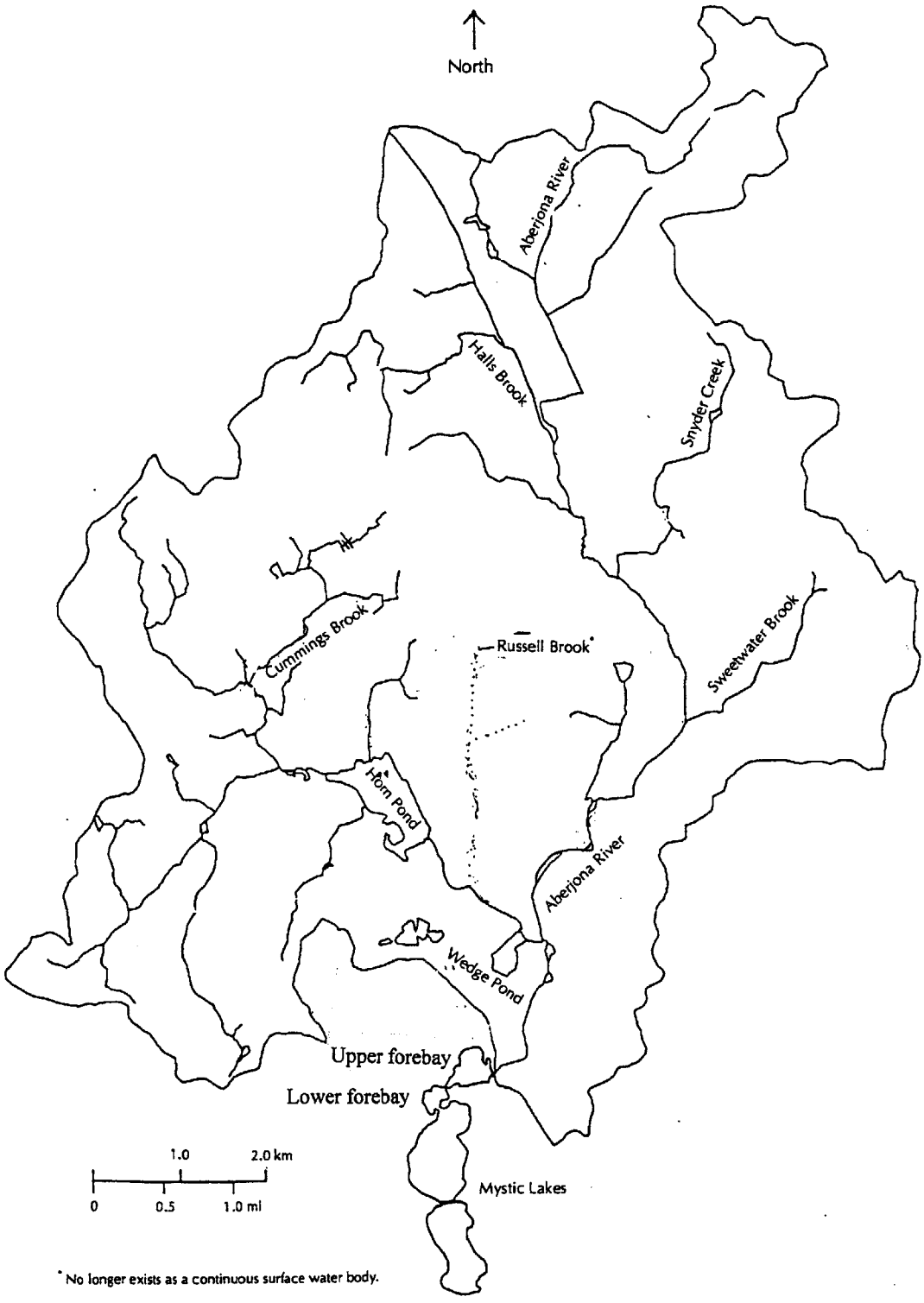


Figure A-1. Aberjona Watershed, MA. Toxic metals such as arsenic (As) and chromium (Cr) are transported from contamination sites in the Aberjona watershed downstream via the Aberjona River. These contaminants are ultimately deposited in the Forebays and Mystic Lakes in Winchester, MA. The Upper Forebay, a free surface wetland, serves as the field site for this investigation.

A.3 Background

Numerous studies have shown that aquatic plants accumulate toxic metals (Reay 1972; Mudroch & Capobianco 1979; Chigbo, Smith, & Shore 1982; Simpson et al. 1983; Blake et al. 1987; Kraus 1987; Lee, Low, & Hew 1991). The extent to which a plant accumulates metals has been shown to be species specific (Reay 1972; Mudroch & Capobianco 1979). Uptake of these metals has been assumed to be through the plant's root system. Evidence which appears to support this has revealed that the roots of an aquatic plant accumulate more metals than the stem of the same plant, and the stem accumulates more metals than the leaf (Kraus 1987; Zhang et al. 1990).

However, recent studies have suggested that accumulation of material on the outside of aquatic plants may play a major role in the overall accumulation of metals by plants (Reay 1972; Alizai & McManus 1980; Stumpf 1983; Jordan, Pierce, & Correll 1986; Leonard, Hine, & Luther 1995). Reay (1972) observed As concentrations as high as 650 ppm in the flocculant material attached to *Ceratophyllum demersum*, a submerged plant species. The mechanism by which material in the water column becomes attached to the outside of a plant will be defined as physical trapping. The effectiveness of physical trapping by a species is speculated to be a function of plant morphology. For, example, a plant species with greater surface area and non-smooth surface texture is believed to be more effective at removing material from the water column than one with lesser surface area and a smooth surface.

A.4 Methods

Samples of the two predominant plant species in the UFB, *Ceratophyllum demersum* and *Nymphaea odorata*, were collected October 9, 1997 from three sites. Figure A-2 depicts the location of the three sites from which the five plant samples were collected. Figure A-3 contains illustrations of the plant species collected. The two species are structurally very different.

Ceratophyllum demersum, or coontail, is a submerged aquatic plant with many branches. It is commonly found in ponds and sluggish streams. Its branches are long, slender, and flexible forming large entangled masses (Magee 1981). At each joint is a whorl of leaves (Hotchkiss 1972). *Nymphaea odorata* are commonly called white

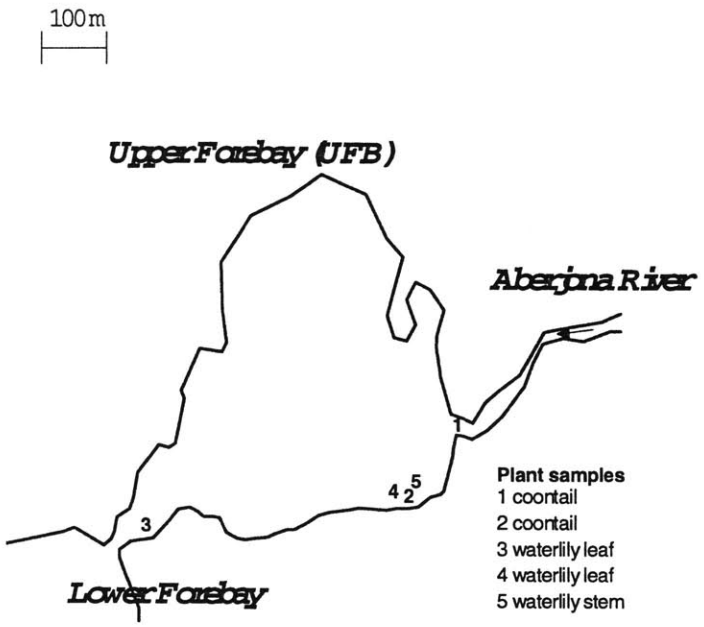


Figure A-2. Plant Samples from the Upper Forebay of the Mystic Lakes, Winchester MA. Samples of coontail (*Ceratophyllum demersum*) and waterlilies (*Nymphaea odorata*) were collected from the locations indicated above.

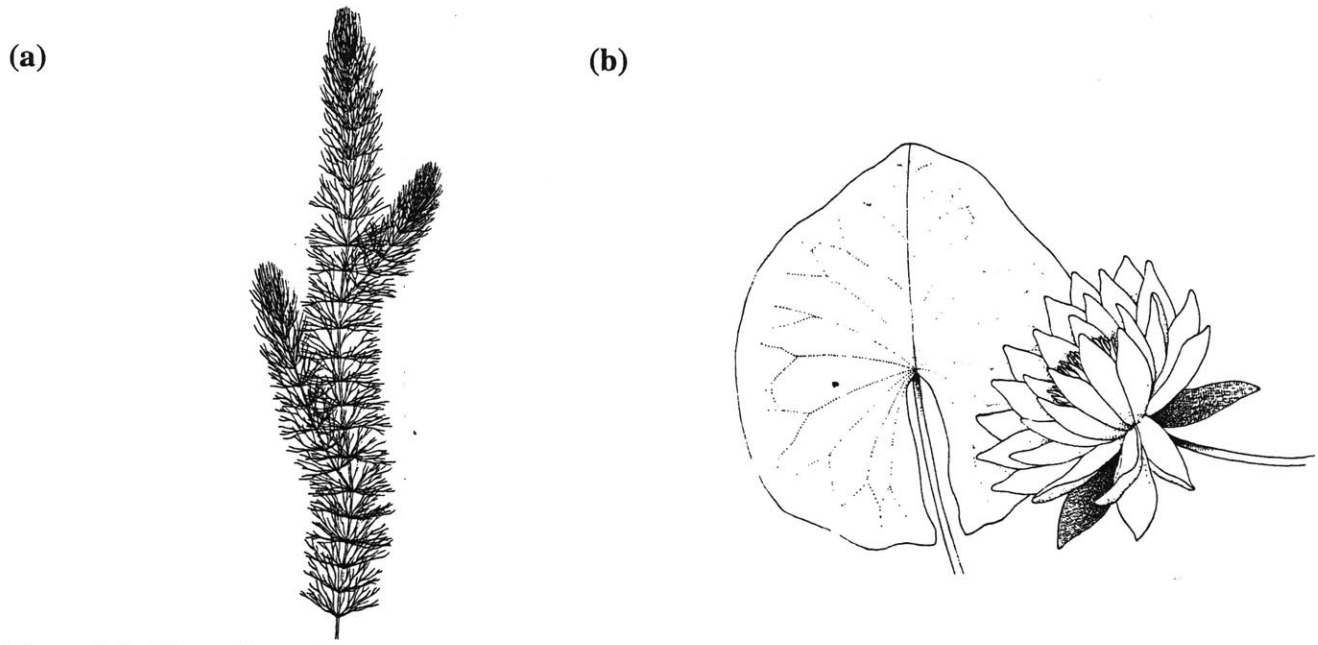


Figure A-3. Illustrations of Plants Species Sampled: (a) *Ceratophyllum demersum*, and (b) *Nymphaea odorata*. Note the large structural differences between the two plant species. Coontail is a delicate submerged species with slender, flexible, free-floating stems creating a large surface area. The white waterlily is an emergent species with a thick flexible underwater stalk and leaf that floats on the free water surface (Magee 1981).

waterlilies. They are typically found in ponds, lakes and pools in up to 1.2 m (4 ft) of water (Magee 1981). Upon maturation, the waterlily leaves are firm and usually float on the free water surface attached to a stem (Hotchkiss 1972). The leaf blade is round and up to 25 cm (10 inches) across with a notch at the base where the stalk is attached (Magee 1981).

During collection, care was taken neither to dredge up sediment nor lose material adhered to the outside of the plants. Samples were kept in sealed plastic bags and placed on ice until they were returned to the laboratory. Samples were kept in a freezer in the laboratory until analysis one week later. *N. odorata* material was separated into stems and leaves. The *C. demersum* material was not separated due to its delicate structure. Roots of the samples were not collected because I wanted to evaluate the physically trapped arsenic and chromium taken from the water column.

X-ray fluorescence spectrometry (XRF) was chosen as the method of analysis for this study because samples are not destroyed during analysis and numerous element concentrations in a sample can be determined simultaneously. XRF has been used for past 35 years as a method for analyzing various element concentrations in plants. Studies have shown that XRF readings correspond well with classic atomic absorption spectrometry (Mudroch & Mudroch 1977).

The plant samples were not washed to determine the element concentration solely on the plant because a reliable XRF protocol for the measurement of elements in filtered rinse water was not developed. However, comparing washed and unwashed samples would have been performed if this study showed plant accumulation was significant enough to warrant further investigation.

Using a standard XRF protocol, approximately 3.5 g of dried plant material (oven dried at 80° C for three days) was homogenized in a mixer mill. The homogenized plant sample was combined with a binder (copolywax powder) and pressed into a 32 mm diameter aluminum cup creating a sample pellet which was run on an XRF. To obtain 3.5 g of material, several plants had to be dried and combined to form one sample.

Since the material adhered to the outside of the plant samples was of primary interest, samples were rinsed with distilled water and the rinse water was observed under a microscope.

A.5 Results

A.5.1 X-ray Fluorescence Spectrometry (XRF)

Complete elemental analysis of the plant samples was performed. Included in this discussion are the results for arsenic (As), chromium (Cr), and silicon (Si). The XRF results are accurate to within 10% with a detection limit of 1 ppm. Arsenic, Cr, and Si concentrations for each plant sample are plotted in Figure A-4. There is a significant difference between the accumulation of As, Cr, and Si for the two plant species *C. demersum* and *N. odorata* at the same location.

C. demersum samples contain more than one order of magnitude greater As, Cr, and Si than *N. odorata* at the same location (samples 2, 4, & 5). *C. demersum* possessed As concentrations as high as 80 $\pm$ 10 ppm, Cr concentrations as high as 59 $\pm$ 6 ppm, and Si concentrations as high as 84.5 $\pm$ 8 ppt. In contrast, the maximum observed concentrations of As and Cr for *N. odorata* samples were below the detection limit of 1 ppm, and a maximum Si concentration of 5.3 $\pm$ 0.5 ppt was observed.

Samples 1 and 2 reveal the spatial variation of accumulation of As, Cr, and Si. Higher concentrations of the metals, As and Cr, are seen in the river vegetation (sample 1) than in vegetation downstream of the river (sample 2). Since the Aberjona River is the source of these metals, these results seem to be consistent. Silicon, however, shows the opposite spatial variation.

A.5.2 Microscopy

The source of the As and Cr to the UFB is the Aberjona River. The source of Si is unknown. Material adhered to the outside of the plant samples was observed under a microscope in an attempt to identify a potential silicon source. It was hypothesized that either the presence of diatoms or sediment could be responsible for the high silicon concentration seen in the plant samples.

Both sediment and diatoms were present in the surficial material of the plant samples. There were far greater numbers of diatoms than sediment particles (10:1). Observations under the microscope revealed that waterlily stems had approximately 10% of the amount of material adhered to them as the coontail, *C. demersum.* The waterlily

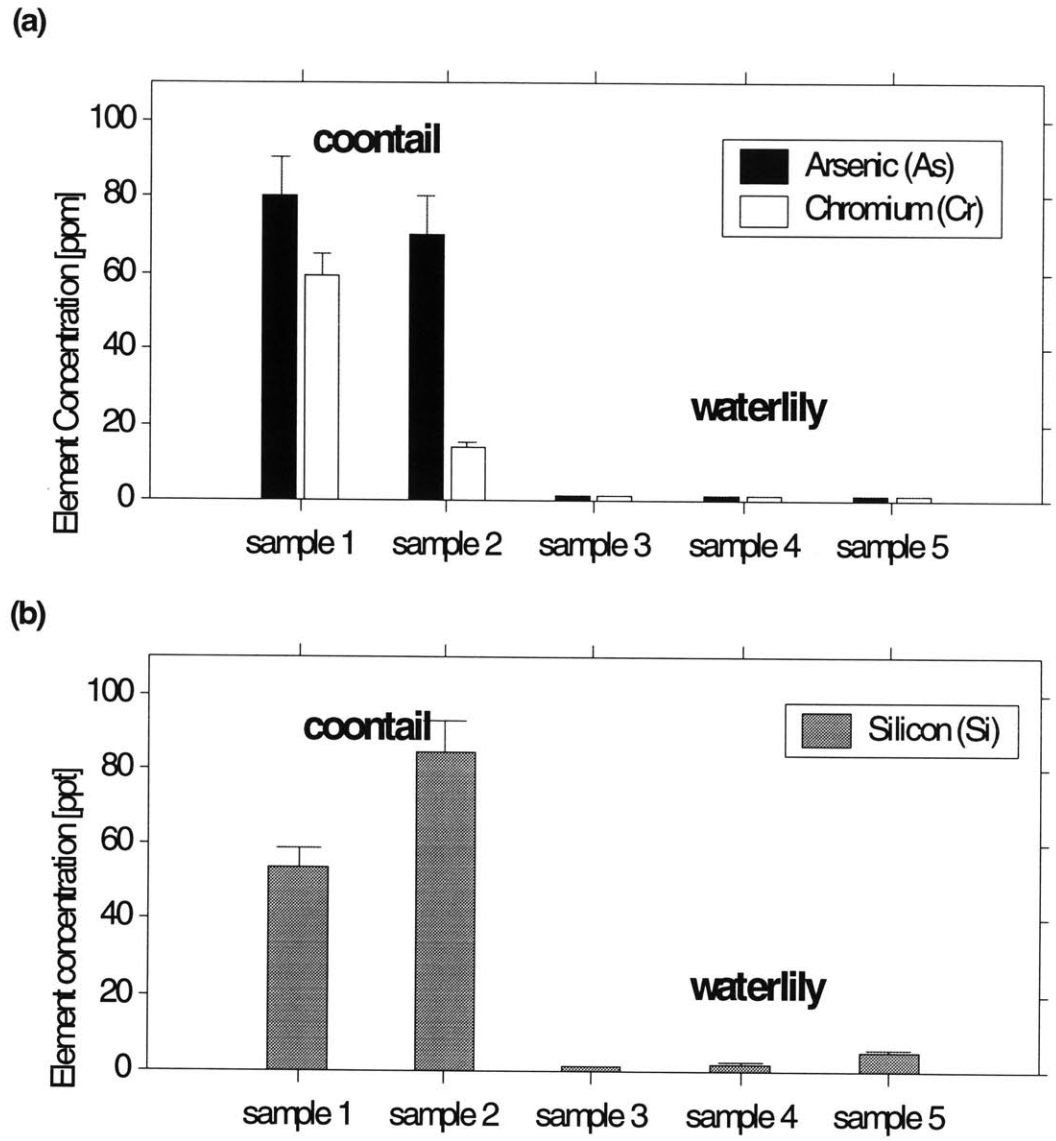


Figure A-4. Concentrations of As, Cr, and Si in Plant Samples of the UFB: (a) As and Cr, and (b) Si. *Ceratophyllum demersum* (coontail) accumulates significantly more As, Cr, and Si than *Nymphaea odorata* (white waterlilies).

leaves showed almost no surficial material. These microscopy observations are accurately reflected in the Si results in Figure A-4b.

A.6 Discussion

Most of the research on the accumulation of As by aquatic plants discover As concentrations less than 10 ppm. I observed seven to eight times that concentration in coontail samples. Reay (1972) looked at coontail as well and observed concentrations as

high as 650 ppm. The results from the UFB strengthen the argument that different species accumulate different concentrations of contaminants in the same system as seen in comparing coontail and waterlilies. Though the plant samples analyzed include both surficial and internal As and Cr of the plant, I speculate that the surficial accumulation of the elements is the most important mode of accumulation which explains the difference between the waterlily and coontail species (direct result of the coontail species' large surface area). The microscopy work supports this hypothesis; waterlily samples had approximately 10% the amount of material on their surface as coontail. In addition, the number of diatoms outnumbered the amount of sediment 10:1. Though one might think the metals are attached to particulates such as sediment, several freshwater algae including diatoms have been shown to accumulate and transform arsenic (Maeda et al. 1990; Wrench and Addison 1980; Andreae and Klumpp 1979; Maeda et al 1992; Kaise et al. 1997; Maeda et al. 1987; Maeda et al. 1992).

Assuming that the arsenic concentrations observed in the plant samples from the UFB are representative, I calculated the amount of surficial arsenic captured by all of the plant material associated with the two species, *C. demersum* and *N. odorata,* using the information from Table A-1. The As concentrations are from the XRF results, the wet-to-dry mass is the ratio of the non-dried plant material mass to the mass of the plant material after three days in an oven at 80° C. Harvest [kg/m^2] represents the mass of plant material in a 9.29 square decimeter area (square ft). Lastly, UFB coverage is the percentage of the UFB area (10.3×10^4 m^2) covered by a given aquatic plant species. *C. demersum* accumulates approximately 1 kg of As and *N. odorata* accumulates approximately 15 g.

TABLE A-1: AVERAGE CHARACTERISTICS OF VEGETATION IN UPPER FOREBAY, WINCHESTER, MA

Plant Species	[As] (mg/kg)	wet:dry mass (-)	Harvest (kg/m^2)	UFB Coverage
N. odorata	10	10	0.97	15%
C. demersum	75	12.3	6.0	30%

A.7 Conclusion

The greater accumulation of heavy metals by coontail than by waterlilies may be attributable to the larger surface area of the coontail which allows more material to be physically trapped. The XRF concentrations of As and Cr measured both the As and Cr in and on the plant. It is speculated that surficial material on the plant is primarily responsible for the elevated metal levels however. The microscopic analysis demonstrated that much more material was attached to the outside of the coontail than the waterlilies and diatoms have been known to accumulate and transform arsenic and other heavy metals. Since the estimated amount of As physically trapped by the aquatic vegetation in the UFB is the same order of magnitude as the amount trapped in the UFB in a typical storm, approximately 2 kg (Seemann 1996), the role of the plants may be important in enhancing water quality downstream during storm events. This would be an important mechanism since the largest contaminant loads are seen during and following storms.

APPENDIX B: MODEL SOURCE CODE

B.1 Overview

A hydrodynamic model of turbulent flow was developed in a boundary-fitted curvilinear coordinate system. The numerical model uses a finite difference approximation of the non-conservative depth-averaged momentum and continuity equations. The governing equations are solved on an unstaggered grid and discretized using a semi-implicit Euler scheme in time, $O(\Delta t)$, and central differencing in space, $O(\Delta x^2)$.

This chapter may be beneficial to those interested in using this model. It includes information regarding the preparation of input files, structure of the computer program, and the source code.

B.2 Preparation of Input Files

The computer program prompts the user to input (via the keyboard) the information necessary for proper execution of the code, i.e. grid size, timestep size, convergence criteria etc. In addition, since some of the input data is large in size, the computer program reads some input information from user-created data files rather than keyboard input. All user-created data files must be placed in the same folder as the executable code. Each file (*.dat) should contain information for a given variable at all grid nodes. The files should be tab-delimited in the form of a matrix of values in which each entry corresponds to a grid node. Each line of the file corresponds to a row of the grid. If the grid is non-rectangular, void spaces in the grid should be filled with dummy values (any numerical value) to create a rectangular grid. The number of data files needed by the program for proper execution differs for steady-state and transient scenarios; additional files are necessary for transient cases.

For steady-state scenarios, an individual data file must be created for each of the following:

- x coordinates of each grid node in physical space [m],
- y coordinates of each grid node in physical space [m],
- windspeed magnitude [m/s],

- wind direction (angle between wind direction and the positive x-axis) [°],
- Chezy coefficient [$m^{1/2}/s$],
- horizontal turbulent eddy viscosity [m^2/s],
- initial x-component of velocity [m/s],
- initial y-component of velocity [m/s],
- initial water level [m],
- boundary definition, and
- bathymetric information [m] (optional 11^{th} file).

All of the data files listed above, with the exception of the eleventh file, are required for proper execution of the program. The eleventh file may be used to define basin bathymetry rather than a constant bed slope. The user will be prompted with this option.

Example data files for a 100 m by 200 m basin with Δx=Δy=10 m are provided in Figure B-1. The only data file requiring a more detailed explanation should be the boundary definition file. Information in this file identifies grid nodes as solid boundaries, inlet, outlet, or otherwise. This file can be created using the following guide:

- '0' denotes solid boundaries,
- '1' denotes nodes one grid space from solid boundaries, the inlet, or outlet,
- '2' denotes inlet nodes ('21' for inlet nodes one grid space from solid boundary),
- '3' denotes outlet nodes ('31' for outlet nodes one grid space from solid boundary),
- '4' denotes solid boundary corners,
- '8' denotes active grid points not otherwise marked, and
- '13' denotes inactive grid nodes.

An example of a boundary definition file for the basin described above is provided in Figure B-1c. The physical representation of this basin is depicted in Figure B-2.

Lastly, a master file (user-specified name) must be created containing the names the user has assigned to each data file they have created (10 or 11 files). Each line of the master file should contain one data file name in the order presented above. Figure B-3 provides an example of a master file.

(a)

0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
88	88	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
88	88	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
88	88	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200

(b)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30
40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40
50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50	50
60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60
70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70
88	88	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80
88	88	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90
88	88	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100

(c)

4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
21	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	31
2	1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	1	3
2	1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	1	3
2	1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	1	3
2	1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	1	3
21	1	1	1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	1	3
4	0	4	1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	1	3
13	13	0	1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	1	3
13	13	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	31
13	13	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4

Figure B-1. Sample Steady-State Data Files: (a) x coordinates of grid nodes in physical space [m], (b) y coordinates of grid nodes in physical space [m], and (c) boundary definition file. These data files are for a basin of dimensions 100 m by 200 m with $\Delta x = \Delta y = 10$ m. The dummy value '88' is used to fill in the void spaces of the grid to make it rectangular. A data file for a spatially uniform horizontal eddy viscosity would have all of its entries identical. The boundary definition file defines the inlet, outlet, and solid boundary locations.

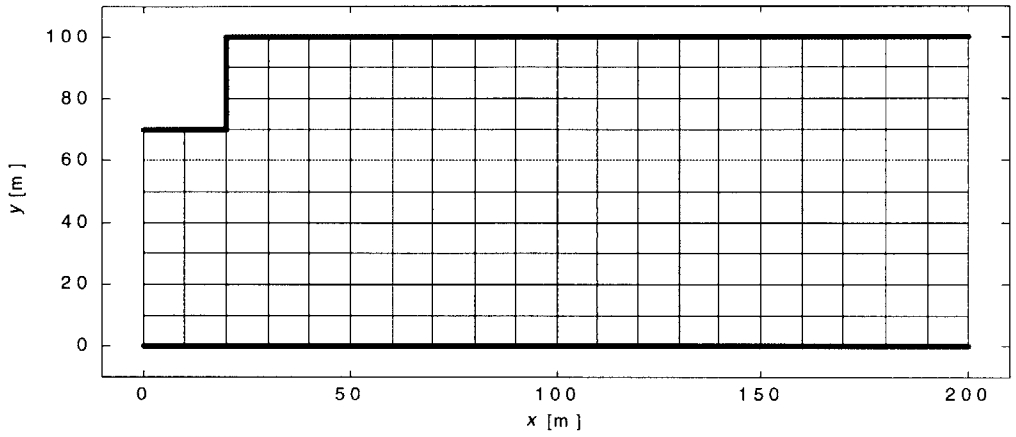


Figure B-2. Physical Representation of the Basin Grid. This grid corresponds to the example data files in Figure B-1. The bold lines indicate solid boundaries or walls of the basin.

x_locations.dat
y_locations.dat
windspeed.dat
wind_angle.dat
chezy.dat
turbulence.dat
initial_u.dat
initial_v.dat
initial_h.dat
boundary.dat

Figure B-3. Steady-state Master File. A master file contains the user-specified names (*.dat) of the data files in the order presented in the text. The program will prompt the user for the name of this master file. Note that no bathymetric file has been created so the user must define the bathymetry with a constant bed slope when prompted to do so. The data files in Figure B-1 would correspond to the following user-specified names: (a) x_locations.dat, (b) y_locations.dat, and (c) boundary.dat.

For transient scenarios, three additional data files and one additional master file must be created listing the names of the three transient data files to be accessed. The transient master file should list each data file by name on an individual line in the following order:

- inlet boundary condition (define $\partial\overline{u}/\partial x$ [1/s], $\partial h/\partial x$ [-], $\overline{u}$ [m/s], h [m], or q[m^2/s]),
- outlet boundary condition (define $\partial\overline{u}/\partial x$ [1/s], $\partial h/\partial x$ [-], $\overline{u}$ [m/s], h [m], or q[m^2/s]), and
- rainfall rate [m/s].

The program will prompt the user to indicate which inlet and outlet boundary conditions they have chosen to define. For the steady-state case, inlet and outlet conditions will be input via the keyboard. These files should be formatted as one column in which the lines of the data file represent values at consecutive timesteps. Figure B-4 is an example of a transient data file.

1
1.1
1.2
1.4
1.5
1.6
1.7
1.8
1.9
2.0

Figure B-4. Transient Data File: inlet boundary condition, $h(t)$ [m]. Each line in this file represents the water level at the inlet (all nodes labeled '2' or '21' in the boundary definition file) for a given timestep; the water level is increasing with time. The program will prompt the user for the corresponding timestep used in preparing transient files.

B.3 Structure of the Source Code

This section describes the basic structure of the hydrodynamic model source code. If more detail is needed, Section B.4 contains the commented FORTRAN 90 source code. The main program is entitled *Hydro*; it calls upon nine subroutines: *StaticDataInput*, *TransientDataInput*, *InitBoundary*, *Xmomentum*, *UnewBoundary*, *Ymomentum*, *VnewBoundary*, *Waterlevel*, and *HnewBoundary*. The bulk of the numerical work is

performed in these subroutines. The main program serves as a shell which holds the individual subroutines together, but it also obtains information from the user and produces the model output files.

Below is a brief synopsis of each subroutine:

StaticDataInput & *TransientDataInput*
Reads and stores information from user's data files.
InitBoundary
Assigns initial boundary conditions for the inlet, outlet , and solid boundaries (no-slip).
Xmomentum
Solves the discretized x-momentum equation for $\overline{u}$ [m/s].
UnewBoundary
Re-establishes boundary conditions that may have been affected by the *Xmomentum* subroutine.
Ymomentum
Solves the discretized y-momentum equation for $\overline{v}$ [m/s].
VnewBoundary
Re-establishes boundary conditions that may have been affected by the *Ymomentum* subroutine.
Waterlevel
Solves the discretized continuity equation for h [m].
HnewBoundary
Re-establishes boundary conditions that may have been affected by the *Waterlevel* subroutine.

The main program (*Hydro*) calls these subroutines in this order and repeats the subroutines from *Xmomentum* through *HnewBoundary* until the maximum number of iterations has been reached or the convergence criteria has been satisfied as defined by the user.

The model output is saved to the files outputu.dat, outputv.dat, and outputh.dat rather than to the screen or a printer. These three ASCII data files contain $\overline{u}$ [m/s], $\overline{v}$ [m/s], and h [m] information respectively. They contain tab-delimited values for each grid node in the form a matrix. Any graphing program can use these data files for visualization of the results.

B.4 Model Source Code

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! HYDRO.F90 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Depth-averaged Hydrodynamic Model.
! Using non-conservative forms of continuity and momentum equations.
! Unstaggered grid.
! Semi-implicit Euler time discretization.
! Central difference spatial discretization.
! Coded in Fortran 90.

! Author: Laura L. DePaoli, depaoli@alum.mit.edu
! Began Development 9/98
! Completed Development 2/99

!!!!!!!!!!!!!!!!!!!!!!!!!! MAIN PROGRAM !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PROGRAM Hydro

IMPLICIT NONE

INTEGER:: rowsize,columnsize,count,timerecordlength,row,allocatestatus_1,allocatestatus_2,openstatus,inletbc,outletbc
CHARACTER(15):: question,answer,transientfile
REAL(8):: slope,numsteps,timestep,convcrit
REAL(8), DIMENSION(:,:), ALLOCATABLE::
xcoord,ycoord,windspeed,winddir,chezycoeff,turbulence,initu,initv,inith,u,v,h,bedheight,unew,vnew,hnew,outputu,outputv,outputh,uerror,
verror,herror,bdrydef
REAL(8), DIMENSION(:), ALLOCATABLE:: levelin,levelout,uin,uout,ufluxin,ufluxout,hfluxin,hfluxout,qin,qout,rain

! Credits
PRINT *, "Depth-averaged Hydrodynamic Model"
PRINT *, "Author: Laura L. DePaoli (depaoli@alum.mit.edu)"
PRINT *, "Copyright. Massachusetts Institute of Technology 1999. All Rights Reserved."
PRINT *, "This model was developed in partial fulfillment of the requirements for the "
PRINT *, "degree of Master of Science in Civil & Environmental Engineering at M.I.T."
PRINT *, ""

        ! MAIN VARIABLES
        ! rowsize: maximum number of rows in grid
        ! columnsize: maximum number of columns in grid
        ! count: counter for iterations through u,v,h update loop
        ! timerecordlength: length of transient data files (one for steady-state cases)
        ! row: counter for row index (used in creating program output files)
        ! allocatestatus_1: zero if able to allocate memory for necessary variables
        ! allocatestatus_2: zero if able to allocate memory for necessary variables
        ! openstatus: zero if able to open program output files
        ! inletbc: user's choice of boundary condition at inlet
        ! outletbc: user's choice of boundary condition at outlet
        ! question: 'S' for uniform bed with slope & 'B' for complex bathymetry
        ! answer: 'T' for transient case & 'S' for steady-state case
        ! transientfile: name of file containing names of transient data files
        ! slope: bed slope from inlet to outlet
        ! numsteps: maximum number of timesteps to be run
        ! timestep: size of timestep [s]
        ! convcrit: convergence criteria (represents u,v,h update error that is acceptable)
        ! xcoord: x location of a computational node in physical space [m]
        ! ycoord: y location of a computational node in physical space [m]
        ! windspeed: windspeed magnitude [m/s]
        ! winddir: angle between wind direction and positive x-axis [degree]
        ! chezycoeff: chezy coefficient used to calculate bed stress [(m^.5)/s]
        ! turbulence: horizontal turbulent eddy viscosity [m^2/s]
        ! initu: initial x-velocity field [m/s]
        ! initv: initial y-velocity field [m/s]
        ! inith: initial water level field [m]
        ! u: x-velocity field [m/s]
        ! v: y-velocity field [m/s]
        ! h: water level field [m]
        ! bedheight: bed height for all gridpoints [m]
        ! unew: update of u (velocity component in x-direction) [m/s]
        ! vnew: update of v (velocity component in y-direction) [m/s]
        ! hnew: update of h (water depth) [m]
```

```
        ! outputu: u after convergence criteria or maximum number of iterations reached; saved as a file
        ! outputv: v after convergence criteria or maximum number of iterations reached; saved as a file
        ! outputh: h after convergence criteria or maximum number of iterations reached; saved as a file
        ! uerror: update error of u; difference between u from current & previous timestep
        ! verror: update error of v; difference between v from current & previous timestep
        ! herror: update error of h; difference between h from current & previous timestep
        ! bdrydef: defines the status of each gridpoint (i.e. inlet, outlet)
        ! levelin: water level at inlet boundary for a given timestep [m]
        ! levelout: water level at outlet boundary for a given timestep [m]
        ! uin: u at inlet boundary for a given timestep [m/s]
        ! uout: u at outlet boundary for a given timestep [m/s]
        ! ufluxin: du/dx at inlet boundary for a given timestep [1/s]
        ! ufluxout: du/dx at outlet boundary for a given timestep [1/s]
        ! hfluxin: dh/dx at inlet boundary for a given timestep [-]
        ! hfluxout: dh/dx at outlet boundary for a given timestep [-]
        ! qin: flowrate per unit width at inlet boundary for a given timestep [m^2/s]
        ! qout: flowrate per unit width at outlet boundary for a given timestep [m^2/s]
        ! rain: rainfall rate applied to grid [m/s]

! Prompts user for grid size and allocates space for variable matrices
PRINT *, "Enter the maximum number of rows in your grid: "
READ *, rowsize
PRINT *, "Enter the maximum number of columns in your grid: "
READ *, columnsize
ALLOCATE(xcoord(rowsize,columnsize),ycoord(rowsize,columnsize),windspeed(rowsize,columnsize),winddir(rowsize,columnsize),chez
        ycoeff(rowsize,columnsize),turbulence(rowsize,columnsize),initu(rowsize,columnsize),initv(rowsize,columnsize),inith(rowsize,
        columnsize),u(rowsize,columnsize),v(rowsize,columnsize),h(rowsize,columnsize),bedheight(rowsize,columnsize),unew(rowsiz
        e,columnsize),vnew(rowsize,columnsize),hnew(rowsize,columnsize),outputu(rowsize,columnsize),outputv(rowsize,columnsize)
        ,outputh(rowsize,columnsize),uerror(rowsize,columnsize),verror(rowsize,columnsize),herror(rowsize,columnsize),bdrydef(rows
        ize,columnsize),STAT=allocatestatus_1)
IF (Allocatestatus_1 /= 0)  STOP "** Not enough memory to allocate space for variables **"

! Prompts user for bed information
PRINT *, "Does your system have a uniform bed with a slope (S) or "
PRINT *, "  complex bathymetry (B)? S or B"
READ *, question

        IF ((question=='S').OR.(question=='s')) THEN
        ! Prompts user for bed slope from inlet to outlet
                PRINT *, "Enter the bed slope of your system (from inlet to outlet): "
                READ *, slope
        ELSE IF ((question=='B').OR.(question=='b')) THEN
                PRINT *, "The data file name containing bathymetric information for all grid points"
                PRINT *, "should be included at the end of your static data file list."
                PRINT *, ""
        ELSE
                PRINT *, "Not a Valid Entry."
                STOP
        END IF

!Prompts user for steady state or transient case
PRINT *, "Are you modeling transient or steady-state phenomena? T or S"
READ *, answer

! STEADY STATE CASE
IF ((answer=='S').OR.(answer=='s')) THEN
        timerecordlength=1

        ALLOCATE(levelin(timerecordlength),levelout(timerecordlength),uin(timerecordlength),uout(timerecordlength),hfluxin(timer
        ecordlength),hfluxout(timerecordlength),ufluxin(timerecordlength),ufluxout(timerecordlength),qin(timerecordlength),qout(time
        recordlength),STAT=allocatestatus_2)
        IF (allocatestatus_2 /=0) STOP "** Not enough memory to allocate space for time series variables **"

        uerror(1,1)=1
        verror(1,1)=1
        herror(1,1)=1
        count=1

        PRINT *, "Enter convergence criteria: "
        READ *, convcrit
        PRINT *, "Enter maximum number of timesteps:"
```

```
            READ *, numsteps
            PRINT *, "Enter size of timestep [s]:"
            READ *, timestep

            ! Reads user's static data files
            CALL
            StaticDataInput(xcoord,ycoord,windspeed,winddir,chezycoeff,turbulence,initu,initv,inith,bdrydef,rowsize,columnsi
            ze,bedheight,slope,question)

            ! Assign initial conditions
            u=initu
            v=initv
            h=inith

! TRANSIENT CASE
ELSE IF ((answer=='T').OR.(answer=='t')) THEN

            ! Prompts user for size of time records (Time record data should be arranged in one column)
            PRINT *, "Input filename containing the names of all transient data files: "
            READ *, transientfile
            PRINT *, "Enter the number of rows in your transient datafile:"
            READ *, timerecordlength
            PRINT *, "Enter size of the timestep between entries in the transient datafile [s]:"
            READ *, timestep

ALLOCATE(levelin(timerecordlength),levelout(timerecordlength),uin(timerecordlength),uout(timerecordlength),hfluxin(timer
ecordlength),hfluxout(timerecordlength),ufluxin(timerecordlength),ufluxout(timerecordlength),qin(timerecordlength),qout(time
recordlength),rain(timerecordlength),STAT=allocatestatus_2)
            IF (allocatestatus_2 /= 0) STOP "** Not enough memory to allocate space for time series variables**"

            ! Read in static data from user files
            CALL
            StaticDataInput(xcoord,ycoord,windspeed,winddir,chezycoeff,turbulence,initu,initv,inith,bdrydef,rowsize,columnsi
            ze,bedheight,slope,question)

            ! Assign initial conditions
            u=initu
            v=initv
            h=inith
            count=2
ELSE
PRINT *, "Not a Valid Entry."
STOP
END IF

! Prompts user for inlet & outlet boundary conditions
! STEADY-STATE & TRANSIENT CASE

! Inlet Boundary Conditions assignment
PRINT *, "INLET BOUNDARY CONDITIONS"
PRINT *, "Option (1): Define h."
PRINT *, "Option (2): Define dh/dx."
PRINT *, "Option (3): Define u."
PRINT *, "Option (4): Define du/dx."
PRINT *, "Option (5): Define q."
PRINT *, " Enter 1, 2, 3, 4, or 5."
READ *, inletbc

            IF (inletbc==1) THEN
                        IF ((answer=='S').OR.(answer=='s')) THEN
                                    PRINT *, "Enter water level at inlet [m]: "
                                    READ *, levelin(1)
                        ELSE IF ((answer=='T').OR.(answer=='t')) THEN
                                    PRINT *, "Read data from file."
                        END IF
            ELSE IF (inletbc==2) THEN
                        IF ((answer=='S').OR.(answer=='s')) THEN
                                    PRINT *, "Enter flux (dh/dx) at inlet: "
                                    READ *, hfluxin(1)
                        ELSE IF ((answer=='T').OR.(answer=='t')) THEN
                                    PRINT *, "Read data from file."
```

```
                END IF
        ELSE IF (inletbc==3) THEN
                IF ((answer=='S').OR.(answer=='s')) THEN
                        PRINT *, "Enter U at inlet [m/s]: "
                        READ *, uin(1)
                ELSE IF ((answer=='T').OR.(answer=='t')) THEN
                        PRINT *, "Read data from file."
                END IF
        ELSE IF (inletbc==4) THEN
                IF ((answer=='S').OR.(answer=='s')) THEN
                        PRINT *, "Enter flux (du/dx) at inlet: "
                        READ *, ufluxin(1)
                ELSE IF   ((answer=='T').OR.(answer=='t')) THEN
                        PRINT *, "Read data from file."
                END IF
        ELSE IF (inletbc==5) THEN
                IF ((answer=='S').OR.(answer=='s')) THEN
                        PRINT *, "Enter flowrate per unit width(q) at inlet [m^2/s]:"
                        READ *, qin(1)
                ELSE IF ((answer=='T').OR.(answer=='t')) THEN
                        PRINT *, "Read data from file."
                END IF
        ELSE
                PRINT *, "Not a Valid Entry."
                STOP
        END IF

! Outlet Boundary Condition assignments
PRINT *, ""
PRINT *, "OUTLET BOUNDARY CONDITIONS"
PRINT *, "Option (1): Define h."
PRINT *, "Option (2): Define dh/dx."
PRINT *, "Option (3): Define u."
PRINT *, "Option (4): Define du/dx."
PRINT *, "            Option (5): Define q."
PRINT *, " Enter 1, 2, 3, 4,  or 5."
READ *, outletbc

        IF (outletbc==1) THEN
                IF ((answer=='S').OR.(answer=='s')) THEN
                        PRINT *, "Enter water level at outlet [m]: "
                        READ *, levelout(1)
                ELSE IF ((answer=='T').OR.(answer=='t')) THEN
                        PRINT *, "Read data from file."
                END IF
        ELSE IF (outletbc==2) THEN
                IF ((answer=='S').OR.(answer=='s')) THEN
                        PRINT *, "Enter flux (dh/dx) at outlet: "
                        READ *, hfluxout(1)
                ELSE IF ((answer=='T').OR.(answer=='t')) THEN
                        PRINT *, "Read data from file."
                END IF
        ELSE IF (outletbc==3) THEN
                IF ((answer=='S').OR.(answer=='s')) THEN
                        PRINT *, "Enter U at outlet [m/s]: "
                        READ *, uout(1)
                ELSE IF ((answer=='T').OR.(answer=='t')) THEN
                        PRINT *, "Read data from file."
                END IF
        ELSE IF (outletbc==4) THEN
                IF ((answer=='S').OR.(answer=='s')) THEN
                        PRINT *, "Enter flux (du/dx) at outlet: "
                        READ *, ufluxout(1)
                ELSE IF ((answer=='T').OR.(answer=='t')) THEN
                        PRINT *, "Read data from file."
                END IF
        ELSE IF (outletbc==5) THEN
                IF ((answer=='S').OR.(answer=='s')) THEN
                        PRINT *, "Enter flowrate per unit width(q) at outlet [m^2/s]: "
                        READ *, qout(1)
                ELSE IF ((answer=='T').OR.(answer=='t')) THEN
```

```
                    PRINT *, "Read data from file."
                END IF
        ELSE
                PRINT *, "Not a Valid Entry."
                STOP
        END IF

! STEADY STATE CASE
IF ((answer=='S').OR.(answer=='s')) THEN

        ! Assigns initial boundary conditions
        CALL
        InitBoundary(rowsize,columnsize,u,v,h,levelin,levelout,bdrydef,timerecordlength,inletbc,outletbc,uin,uout,hfluxin,
        hfluxout,ufluxin,ufluxout,qin,qout,xcoord)
        PRINT *, "Iteration number :"

        ! Solves discretized X-momentum, Y-momentum, & Continuity equations for u,v,h
        DO WHILE
        ((count<=numsteps).AND.((maxval(uerror)>convcrit).OR.(maxval(verror)>convcrit).OR.(maxval(herror)>convcri
        t)))

                IF ((MOD(count,50)==0)) THEN
                print *, count
                END IF

                count=count+1

                CALL
                Xmomentum(xcoord,ycoord,rowsize,columnsize,u,v,h,unew,chezycoeff,windspeed,winddir,turbulence,b
                edheight,bdrydef,timestep)
                CALL
                Unewboundary(xcoord,rowsize,columnsize,unew,h,bdrydef,answer,inletbc,outletbc,uin,uout,ufluxin,ufl
                uxout,qin,qout,count,numsteps)
                CALL
                Ymomentum(xcoord,ycoord,rowsize,columnsize,unew,v,h,vnew,chezycoeff,windspeed,winddir,turbulen
                ce,bedheight,bdrydef,timestep)
                CALL Vnewboundary(xcoord,rowsize,columnsize,vnew,bdrydef)
                CALL
                Waterlevel(xcoord,ycoord,timerecordlength,rowsize,columnsize,unew,vnew,h,hnew,rain,bdrydef,timeste
                p,answer,count)
                CALL
                Hnewboundary(rowsize,columnsize,levelin,levelout,count,timerecordlength,unew,hnew,bdrydef,answer,
                inletbc,outletbc,hfluxin,hfluxout,qin,qout,xcoord,ycoord)

                uerror=abs(unew-u)
                verror=abs(vnew-v)
                herror=abs(hnew-h)

                ! Updates u,v,h after each iteration
                u=unew
                v=vnew
                h=hnew
        END DO

! TRANSIENT CASE
ELSE IF ((answer=='T').OR.(answer=='t')) THEN

        ! Reads user's time series data files
        CALL
        TransientDataInput(levelin,levelout,uin,uout,hfluxin,hfluxout,ufluxin,ufluxout,qin,qout,rain,timerecordlength,trans
        ientfile,inletbc,outletbc)

        ! Assigns initial boundary conditions
        CALL
        InitBoundary(rowsize,columnsize,u,v,h,levelin,levelout,bdrydef,timerecordlength,inletbc,outletbc,uin,uout,hfluxin,
        hfluxout,ufluxin,ufluxout,qin,qout,xcoord)
        PRINT *, "Time elapsed [s]: "

        ! Solves discretized X-momentum, Y-momentum, & Continuity equations for u,v,h
        DO WHILE (count<=timerecordlength)
        PRINT *, (count-1)*timestep
```

```
		CALL
		Xmomentum(xcoord,ycoord,rowsize,columnsize,u,v,h,unew,chezycoeff,windspeed,winddir,turbulence,bedheight,bd
		rydef,timestep)
		CALL
		Unewboundary(xcoord,rowsize,columnsize,unew,h,bdrydef,answer,inletbc,outletbc,uin,uout,ufluxin,ufluxout,qin,q
		out,count,numsteps)
		CALL
		Ymomentum(xcoord,ycoord,rowsize,columnsize,unew,v,h,vnew,chezycoeff,windspeed,winddir,turbulence,bedheigh
		t,bdrydef,timestep)
		CALL Vnewboundary(xcoord,rowsize,columnsize,vnew,bdrydef)
		CALL
		Waterlevel(xcoord,ycoord,timerecordlength,rowsize,columnsize,unew,vnew,h,hnew,rain,bdrydef,timestep,answer,c
		ount)
		CALL
		Hnewboundary(rowsize,columnsize,levelin,levelout,count,timerecordlength,unew,hnew,bdrydef,answer,inletbc,outl
		etbc,hfluxin,hfluxout,qin,qout,xcoord,ycoord)

		! Updates u,v,h after each iteration
		u=unew
		v=vnew
		h=hnew
		count=count+1
		END DO

	END IF

! Produces program output files in matrix format (outputu.dat,outputv.dat,outputh.dat)
OPEN (UNIT=69,FILE="outputu.dat",STATUS="REPLACE",ACTION="WRITE",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0) STOP "** Cannot open program output file **"
OPEN (UNIT=70,FILE= "outputv.dat",STATUS="REPLACE",ACTION="WRITE",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0) STOP "** Cannot open program output file **"
OPEN (UNIT=71, FILE= "outputh.dat",STATUS="REPLACE",ACTION="WRITE",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0) STOP "** Cannot open program output file **"

DO row=1,rowsize
WRITE (69,'(1X,61(ES15.5E5,3X))'),unew(row,1:columnsize)
WRITE (70,'(1X,61(ES15.5E5,3X))'),vnew(row,1:columnsize)
WRITE (71,'(1X,61(ES15.5E5,3X))'),hnew(row,1:columnsize)
END DO

END PROGRAM Hydro
!!!!!!!!!!!!!!!!!!!!!!!!! MAIN PROGRAM END !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!! STATIC DATA INPUT SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Reads in user's files containing static information
SUBROUTINE StaticDataInput(x,y,wind,theta,chezy,turb,initialu,initialv,initialh,boundarydef,rowmax,columnmax,zb,m,question)

INTEGER:: openstatus,row,column
CHARACTER(25):: xfile,yfile,windfile,winddirfile,chezyfile,turbfile,initufile,initvfile,inithfile,boundname,bedfile,masterfile
INTEGER, INTENT(IN):: rowmax, columnmax
REAL(8), INTENT(IN):: m
REAL(8), DIMENSION(rowmax,columnmax), INTENT(OUT):: x,y,wind,theta,chezy,turb,initialu,initialv,initialh,boundarydef,zb

	! STATIC_DATA_INPUT VARIABLES
	! openstatus: zero if files can be opened to read information
	! row: counter for row index
	! column: counter for column index
	! xfile: file containing x locations of grid nodes (physical space)
	! yfile: file containing y locations of grid nodes (physical space)
	! windfile: file containing windspeed information
	! winddirfile: file containing wind direction information
	! chezyfile: file containing chezy coefficient data
	! turbfile: file containing horizontal turbulent eddy viscosity
	! initufile: file containing intitial x-velocity information
	! initvfile: file containing initial y-velocity information
	! inithfile: file containing initial water level information
	! OPTION: bedfile: file containing bathymetric information
	! masterfile: file listing names of user's datafiles to be accessed
	! rowmax: maximum number of rows in grid
	! columnmax: maximum number of columns in grid
```

```
            ! OPTION: m: bed slope from inlet to outlet
            ! x: x location of computational nodes in physical space [m]
            ! y: y location of computational nodes in physical space [m]
            ! wind: windspeed magnitude [m/s]
            ! theta: wind direction angle counterclockwise from positive x axis [degree]
            ! chezy: chezy coefficient used to calculate bed stress [(m^.5)/s]
            ! turb: horizontal turbulent eddy viscosity used to calculate turbulence [m^2/s]
            ! initialu: initial x-velocity field [m/s]
            ! initialv: initial y-velocity field [m/s]
            ! initialh: initial water depth surface [m]
            ! boundarydef: defines status of each grid point (i.e. solid boundaries, inlet)
            ! zb: bed height [m]

! Reads masterfile which contains datafile names as described below
! Filename on Line 1: X coordinate of grid nodes [m](physical space)
! Filename on Line 2: Y coordinate of grid nodes [m](physical space)
! Filename on Line 3: Windspeed [m/s] data at each grid node
! Filename on Line 4: Wind direction [degree] data at each grid node
! Filename on Line 5: Chezy coefficient [(m^.5)/s] data at each grid node
! Filename on Line 6: Horizontal turbulent eddy viscosity [m^2/s] at each grid node
! Filename on Line 7: Initial x-velocity [m/s] at each grid node
! Filename on Line 8: Initial y-velocity [m/s] at each grid node
! Filename on Line 9: Initial water depth [m] at each grid node
! Filename on Line 10: Boundary definition grid
! OPTION: Line 11: Bathymetric information (bed height) at each grid node [m]

PRINT *, "Input filename containing names of all static data files:"
READ *, masterfile
OPEN (UNIT=88,FILE=masterfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open file **"
READ (88,*) xfile
READ (88,*) yfile
READ (88,*) windfile
READ (88,*) winddirfile
READ (88,*) chezyfile
READ (88,*) turbfile
READ (88,*) initufile
READ (88,*) initvfile
READ (88,*) inithfile
READ (88,*) boundname

IF ((question=='B').OR.(question=='b')) THEN
        READ (88,*) bedfile
END IF

! Reads grid coordinate information (physical space) [m]
! Input file: Each line should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=1,FILE=xfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open x file **"
DO Row=1,rowmax
        READ (1,*) (x(row,column),column=1,columnmax)
END DO

OPEN (UNIT=2,FILE=yfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open y file **"
DO Row=1,rowmax
        READ (2,*) (y(row,column),column=1,columnmax)
END DO

! Wind magnitude data used to calculate wind shear [m/s]
! Each row of datafile should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=3,FILE=windfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open wind file **"
DO Row=1,rowmax
        READ (3,*) (wind(row,column),column=1,columnmax)
END DO

! Wind direction data [degree]
! Angle between the wind direction and positive x-axis
```

```
! Each row of datafile should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=4,FILE=winddirfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0) STOP  "** Cannot open wind direction file **"
DO Row=1,rowmax
        READ (4,*) (theta(row,column),column=1,columnmax)
END DO

! Chezy coefficient data used to calculate bed stress [(m^.5)/s]
! Each row of datafile should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=13,FILE=chezyfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open chezy file **"
DO Row=1,rowmax
        READ (13,*) (chezy(row,column),column=1,columnmax)
END DO

! Horizontal turbulent eddy viscosity [m^2/s]
! Each row of datafile should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=14,FILE=turbfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open turbulence file **"
DO Row=1,rowmax
        READ (14,*) (turb(row,column),column=1,columnmax)
END DO

! Initial u (x-velocity) [m/s]
! Each row of datafile should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=15,FILE=initufile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open initial u file **"
DO Row=1,rowmax
        READ (15,*) (initialu(row,column),column=1,columnmax)
END DO

! Initial v (y-velocity) [m/s]
! Each row of datafile should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=16,FILE=initvfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open initial v file **"
DO Row=1,rowmax
        READ (16,*) (initialv(row,column),column=1,columnmax)
END DO

! Initial h (waterdepth) [m]
! Each row of datafile should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=17,FILE=inithfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open initial h file **"
DO Row=1,rowmax
        READ (17,*) (initialh(row,column),column=1,columnmax)
END DO

! Defines the status of each gridpoint as boundary, inlet etc.
! Enter 0 to denote solid boundaries
! Enter 1 to denote gridpoints one point from solid boundaries or inlet or outlet
! Enter 2 to denote inlet gridpoints (Enter 21 for inlet points one step from solid bdry)
! Enter 3 to denote outlet gridpoints (Enter 31 for outlet points one step from solid bdry)
! Enter 4 to denote solid boundary corners
! Enter 8 to denote active gridpoints in the physical space not otherwise marked
! Enter 13 to denote non-active cells
! Each row of datafile should contain all column info for a given row index of the grid
! Datafile should be of size rowmax*columnmax
OPEN (UNIT=18,FILE=boundname,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open definition file **"
DO Row=1,rowmax
        READ (18,*) (boundarydef(row,column),column=1,columnmax)
END DO

CLOSE(88)
CLOSE(1)
```

```
CLOSE(2)
CLOSE(3)
CLOSE(4)
CLOSE(13)
CLOSE(14)
CLOSE(15)
CLOSE(16)
CLOSE(17)
CLOSE(18)

IF ((question=='S').OR.(question=='s')) THEN
        ! Creates a full matrix of bed height values [m] from slope information
        ! Assumes outlet is at maximum grid column and is at a greater x distance
        ! than the inlet.
        ! The outlet is used as the zero datum
        DO Row=1,rowmax
                DO Column=1,columnmax
                zb(row,column)=-m*(x(1,columnmax)-x(row,column))
                END DO
        END DO
ELSE IF ((question=='B').OR.(question=='b')) THEN
        ! Bed height [m]
        ! Each row of datafile should contain all column info for a given row index of the grid
        ! Datafile should be of size rowmax*columnmax
        OPEN (UNIT=20,FILE=bedfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
        IF (openstatus > 0 )  STOP "** Cannot open bed file **"
        DO Row=1,rowmax
                READ (20,*) (zb(row,column),column=1,columnmax)
        END DO
        CLOSE(20)
END IF

END SUBROUTINE StaticDataInput
!!!!!!!!!!!!!!!!!!!!! END STATIC DATA INPUT SUBROUTINE!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!! TRANSIENT DATA INPUT SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Reads user's files containing transient information
SUBROUTINE
TransientDataInput(hin,hout,uin,uout,hfluxin,hfluxout,ufluxin,ufluxout,qin,qout,rainrate,numtimesteps,masterfile,inletbc,outletbc)

INTEGER:: openstatus,index
CHARACTER(20):: infile,outfile,rainfile
CHARACTER(15),INTENT(IN):: masterfile
INTEGER, INTENT(IN):: numtimesteps,inletbc,outletbc
REAL(8), DIMENSION(numtimesteps), INTENT(OUT):: hin,hout,uin,uout,hfluxin,hfluxout,ufluxin,ufluxout,qin,qout,rainrate

                ! TRANSIENT_DATA_INPUT VARIABLES
                ! openstatus: zero if files can be opened to read information
                ! index: counter for row index; used to read in files
                ! infile: file containing inlet boundary condition information
                ! outfile: file containing outlet boundary condition information
                ! rainfile: file containing transient rainrate information
                ! masterfile: file listing names of user's datafiles to be accessed
                ! numtimesteps: number of timesteps for which data is provided
                ! inletbc: user's choice of Neumann or Dirichlet boundary conditions at inlet
                ! outletbc: user's choice of Neumann or Dirichlet boundary conditions at outlet
                ! hin: water level at inlet boundary for all timesteps [m]
                ! hout: water level at outlet boundary for all timesteps [m]
                ! uin: u at inlet boundary for all timesteps [m/s]
                ! uout: u at outlet boundary for all timesteps [m/s]
                ! hfluxin: dh/dx at inlet boundary for all timesteps [-]
                ! hfluxout:  dh/dx at outlet boundary for all timesteps [-]
                ! ufluxin: du/dx at inlet boundary for all timesteps [1/s]
                ! ufluxout: du/dx at outlet boundary for all timesteps [1/s]
                ! qin: flowrate per unit width at inlet boundary for all timesteps [m^2/s]
                ! qout: flowrate per unti width at outlet boundary for all timesteps [m^2/s]
                ! rainrate: rate of ranfall on grid [m/s]

! Reads masterfile which contains data file names as described below
! Filename on Line 1: Inlet boundary condition information
! Filename on Line 2: Outlet boundary condition information
```

```
! Filename on Line 3: Rate of rainfall on grid [m/s]

OPEN (UNIT=77,FILE=masterfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "***Cannot open list of transient datafiles***"

READ (77,*) infile
READ (77,*) outfile
READ (77,*) rainfile

! Inlet Boundary Condition information
OPEN (UNIT=19,FILE=infile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open inlet b.c. file **"

        IF (inletbc==1) THEN
                ! Reads in times series of water levels at the inlet [m]
                ! Assumes same water level across entire inlet for each timestep
                ! Data should be formatted into one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (19,*) hin(index)
                END DO
        ELSE IF (inletbc==2) THEN
                ! Reads in times series of dh/dx at the inlet [-]
                ! Assumes same flux across entire inlet for each timestep
                ! Data should be formatted into one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (19,*) hfluxin(index)
                END DO
        ELSE IF (inletbc==3) THEN
                ! Reads in times series of x-velocity at the inlet [m/s]
                ! Assumes same u across entire inlet for each timestep
                ! Data should be formatted into one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (19,*) uin(index)
                END DO
        ELSE IF (inletbc==4) THEN
                ! Reads in times series of du/dx at the inlet [1/s]
                ! Assumes same du/dx across entire inlet for each timestep
                ! Data should be formatted into one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (19,*) ufluxin(index)
                END DO
        ELSE IF (inletbc==5) THEN
                ! Reads in time series of flowrate per unit width(q) at the inlet [m^2/s]
                ! Assumes uniform flow across inlet
                ! Data should be formatted into one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (19,*) qin(index)
                END DO
        END IF

! Outlet Boundary Condition information
OPEN (UNIT=20,FILE=outfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus > 0 )  STOP "** Cannot open outlet b.c. file **"

        IF (outletbc==1) THEN
                ! Reads in time series of water levels at outlet [m]
                ! Assumes same water level across entire outlet for each timestep
                ! Data should be formatted to one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (20,*), hout(index)
                END DO
        ELSE IF (outletbc==2) THEN
                ! Reads in time series of dh/dx at outlet [-]
                ! Assumes same flux across entire outlet for each timestep
                ! Data should be formatted to one column
```

```
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (20,*) hfluxout(index)
                END DO
        ELSE IF (outletbc==3) THEN
                ! Reads in time series of x-velocity at outlet [m/s]
                ! Assumes same u across entire outlet for each timestep
                ! Data should be formatted to one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (20,*), uout(index)
                END DO
        ELSE IF (outletbc==4) THEN
                ! Reads in time series of du/dx at outlet [1/s]
                ! Assumes same du/dx across entire outlet for each timestep
                ! Data should be formatted to one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (20,*) ufluxout(index)
                END DO
        ELSE IF (outletbc==5) THEN
                ! Reads in time series of flowrate per unit width(q) at outlet [m^2/s]
                ! Assumes uniform flow across outlet
                ! Data should be formatted to one column
                ! Each row in the datafile contains data at a new timestep
                DO index=1,numtimesteps
                READ (20,*) qout(index)
                END DO
        END IF

! Rate of rainfall over the grid [m/s]
! Data is formatted into one column
! Assumes rainfall uniform over entire grid
! Each row contains rainfall rate [m/s] for a time step
OPEN (UNIT=21,FILE=rainfile,STATUS="OLD",ACTION="READ",POSITION="REWIND",IOSTAT=openstatus)
IF (openstatus1 > 0 )  STOP "** Cannot open rain file **"
DO index=1,numtimesteps
        READ (21,*), rainrate(index)
END DO

CLOSE(77)
CLOSE(19)
CLOSE(20)
CLOSE(21)

END SUBROUTINE TransientDataInput
!!!!!!!!!!!!!!!!!!!! END TRANSIENT DATA INPUT SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!! INIT BOUNDARY SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Assigns initial boundary conditions for inlet, outlet, and solid boundaries
SUBROUTINE
InitBoundary(rowmax,columnmax,unew,vnew,hnew,hin,hout,definition,numtimesteps,inletbc,outletbc,uin,uout,hfluxin,hfluxout,ufluxin,uf
luxout,qin,qout,x)

IMPLICIT NONE
INTEGER, INTENT(IN):: rowmax,columnmax,numtimesteps,inletbc,outletbc
REAL(8), DIMENSION(rowmax,columnmax), INTENT(IN):: definition,x
REAL(8), DIMENSION(rowmax,columnmax), INTENT(INOUT):: unew,vnew,hnew
REAL(8), DIMENSION(numtimesteps), INTENT(IN):: hin,hout,uin,uout,ufluxin,ufluxout,hfluxin,hfluxout,qin,qout
INTEGER:: j,k

        ! INIT_BOUNDARY VARIABLES
        ! rowmax: maximum row number in grid
        ! columnmax: maximum column number in grid
        ! numtimesteps: number of timesteps at which transient data is provided
        ! inletbc: user's choice of boundary condition at inlet
        ! outletbc: user's choice of boundary condition at outlet
        ! definition: grid point status as a solid-boundary, inlet, etc.
        ! x: x location of computational nodes in physical space [m]
        ! unew: x-velocity field [m/s]
        ! vnew: y-velocity field [m/s]
```

```
            ! hnew: water depth surface [m]
            ! hin: water level at inlet [m]
            ! hout: water level at outlet [m]
            ! uin: u at inlet [m/s]
            ! uout: u at outlet [m/s]
            ! ufluxin: du/dx at inlet [1/s]
            ! ufluxout: du/dx at outlet [1/s]
            ! hfluxin: dh/dx at inlet [-]
            ! hfluxout: dh/dx at outlet [-]
            ! qin: flowrate per unit width(q) at inlet [m^2/s]
            ! qout: flowrate per unit width(q) at outlet [m^2/s]
            ! j: counter for column index
            ! k: counter for row index

! Sets initial boundary conditions at inlet and outlet
! Sets no-slip and no-flux boundary condition at solid boundaries
DO k=1,rowmax
        DO j=1,columnmax

                ! Inlet
                IF ((definition(k,j)==2).OR.(definition(k,j)==21)) THEN
                        IF (inletbc==1) THEN
                                hnew(k,j)=hin(1)
                        ELSE IF (inletbc==2) THEN
                                IF (definition(k,j+1)==1) THEN
                                hnew(k,j)=hfluxin(1)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*hnew(k,j+1)-(1/3)*hnew(k,j+2)
                                ELSE IF (definition(k,j-1)==1) THEN
                                hnew(k,j)=hfluxin(1)*(x(k,j)-x(k,j-2))/3+(4/3)*hnew(k,j-1)-(1/3)*hnew(k,j-2)
                                END IF
                        ELSE IF (inletbc==3) THEN
                                unew(k,j)=uin(1)
                        ELSE IF (inletbc==4) THEN
                                IF (definition(k,j+1)==1) THEN
                                unew(k,j)=ufluxin(1)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*unew(k,j+1)-(1/3)*unew(k,j+2)
                                ELSE IF (definition(k,j-1)==1) THEN
                                unew(k,j)=ufluxin(1)*(x(k,j)-x(k,j-2))/3+(4/3)*unew(k,j-1)-(1/3)*unew(k,j-2)
                                END IF
                        ELSE IF (inletbc==5) THEN
                                unew(k,j)=qin(1)/hnew(k,j)
                                vnew(k,j)=0
                        END IF

                ! Outlet
                ELSE IF ((definition(k,j)==3).OR.(definition(k,j)==31)) THEN
                        IF (outletbc==1) THEN
                                hnew(k,j)=hout(1)
                        ELSE IF (outletbc==2) THEN
                                IF (definition(k,j+1)==1) THEN
                                hnew(k,j)=hfluxout(1)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*hnew(k,j+1)-(1/3)*hnew(k,j+2)
                                ELSE IF (definition(k,j-1)==1) THEN
                                hnew(k,j)=hfluxout(1)*(x(k,j)-x(k,j-2))/3+(4/3)*hnew(k,j-1)-(1/3)*hnew(k,j-2)
                                END IF
                        ELSE IF (outletbc==3) THEN
                                unew(k,j)=uout(1)
                        ELSE IF (outletbc==4) THEN
                                IF (definition(k,j+1)==1) THEN
                                unew(k,j)=ufluxout(1)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*unew(k,j+1)-(1/3)*unew(k,j+2)
                                ELSE IF (definition(k,j-1)==1) THEN
                                unew(k,j)=ufluxout(1)*(x(k,j)-x(k,j-2))/3+(4/3)*unew(k,j-1)-(1/3)*unew(k,j-2)
                                END IF
                        ELSE IF (outletbc==5) THEN
                                unew(k,j)=qout(1)/hnew(k,j)
                                vnew(k,j)=0
                        END IF

                ! Solid boundaries (no-slip condition)
                ELSE IF ((definition(k,j)==0).OR.(definition(k,j)==4)) THEN
                        unew(k,j)=0
                        vnew(k,j)=0
                END IF
```

```
        END DO
END DO

END SUBROUTINE InitBoundary
!!!!!!!!!!!!!!!!!!!! END INIT BOUNDARY SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!! U NEW BOUNDARY SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Re-establishes u boundary conditions
SUBROUTINE
UnewBoundary(x,rowmax,columnmax,unew,hold,definition,answer,inletbc,outletbc,uin,uout,ufluxin,ufluxout,qin,qout,counter,numtimest
eps)

IMPLICIT NONE
INTEGER, INTENT(IN):: rowmax,columnmax,counter,numtimesteps,inletbc,outletbc
CHARACTER(5), INTENT(IN):: answer
REAL(8), DIMENSION(rowmax,columnmax), INTENT(IN):: x,definition,hold
REAL(8), DIMENSION(rowmax,columnmax), INTENT(INOUT):: unew
REAL(8), DIMENSION(numtimesteps), INTENT(IN):: uin,uout,ufluxin,ufluxout,qin,qout
INTEGER:: j,k

        ! UNEW_BOUNDARY VARIABLES
        ! rowmax: maximum row number in grid
        ! columnmax: maximum column number in grid
        ! counter: indicates current timestep
        ! numtimesteps: number of timesteps at which transient data is provided
        ! inletbc: user's choice of boundary condition at inlet
        ! outletbc: user's choice of boundary condition at outlet
        ! answer: 'T' for transient case & 'S' for steady-state case
        ! x: x location of computational nodes in physical space [m]
        ! definition: grid point status as a solid-boundary, inlet, etc.
        ! unew: updated x-velocity field [m/s]
        ! hold: water depth at the previous timestep [m]
        ! uin: u at inlet [m/s]
        ! uout: u at outlet [m/s]
        ! qin: flowrate per unit width(q) at inlet [m^2/s]
        ! qout: flowrate per unit width(q) at outlet [m^2/s]
        ! ufluxin: du/dx at inlet [1/s]
        ! ufluxout: du/dx at outlet [1/s]
        ! j: counter for column index
        ! k: counter for row index

! Set no-slip solid boundary condition with u=0
DO k=1,rowmax
        DO j=1,columnmax
                IF ((definition(k,j)==0).OR.(definition(k,j)==4)) THEN
                unew(k,j)=0
                END IF
        END DO
END DO

! Linear extrapolation: inlet/outlet
DO j=1,columnmax
        DO k=1,rowmax
                IF ((definition(k,j)==2).OR.(definition(k,j)==21).OR.(definition(k,j)==3).OR.(definition(k,j)==31)) THEN
                        IF ((definition(k,j+1)==1).AND.(inletbc/=5)) THEN
                        unew(k,j)=-((unew(k,j+2)-unew(k,j+1))/(x(k,j+2)-x(k,j+1)))*(x(k,j+1)-x(k,j))+unew(k,j+1)
                        ELSE IF ((definition(k,j-1)==1).AND.(outletbc/=5)) THEN
                        unew(k,j)=-((unew(k,j-2)-unew(k,j-1))/(x(k,j-2)-x(k,j-1)))*(x(k,j-1)-x(k,j))+unew(k,j-1)
                        END IF
                END IF
        END DO
END DO

! STEADY-STATE CASE: Reset steady-state boundary conditions at inlet and outlet
IF ((answer=='S').OR.(answer=='s')) THEN

        DO j=1,columnmax
                DO k=1,rowmax
                        ! Inlet
                        IF ((definition(k,j)==2).OR.(definition(k,j)==21)) THEN
                                IF (inletbc==3) THEN
```

```
                                        unew(k,j)=uin(1)
                                ELSE IF (inletbc==4) THEN
                                        IF (definition(k,j+1)==1) THEN
                                        unew(k,j)=ufluxin(1)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*unew(k,j+1)-
                                        (1/3)*unew(k,j+2)
                                        ELSE IF (definition(k,j-1)==1) THEN
                                        unew(k,j)=ufluxin(1)*(x(k,j)-x(k,j-2))/3+(4/3)*unew(k,j-1)-(1/3)*unew(k,j-2)
                                        END IF
                                ELSE IF (inletbc==5) THEN
                                        unew(k,j)=qin(1)/hold(k,j)
                                END IF
                        END IF

                        ! Outlet
                        IF ((definition(k,j)==3).OR.(definition(k,j)==31)) THEN
                                IF (outletbc==3) THEN
                                        unew(k,j)=uout(1)
                                ELSE IF (outletbc==4) THEN
                                        IF (definition(k,j+1)==1) THEN
                                        unew(k,j)=ufluxout(1)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*unew(k,j+1)-
                                        (1/3)*unew(k,j+2)
                                        ELSE IF (definition(k,j-1)==1) THEN
                                        unew(k,j)=ufluxout(1)*(x(k,j)-x(k,j-2))/3+(4/3)*unew(k,j-1)-(1/3)*unew(k,j-2)
                                        END IF
                                ELSE IF (outletbc==5) THEN
                                        unew(k,j)=qout(1)/hold(k,j)
                                END IF
                        END IF

                END DO
        END DO
END IF

! TRANSIENT CASE: Reset transient boundary conditions at inlet and outlet
IF ((answer=='T').OR.(answer=='t')) THEN

        DO j=1,columnmax
                DO k=1,rowmax
                        ! Inlet
                        IF ((definition(k,j)==2).OR.(definition(k,j)==21)) THEN
                                IF (inletbc==3) THEN
                                        unew(k,j)=uin(counter)
                                ELSE IF (inletbc==4) THEN
                                        IF (definition(k,j+1)==1) THEN
                                        unew(k,j)=ufluxin(counter)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*unew(k,j+1)-
                                        (1/3)*unew(k,j+2)
                                        ELSE IF (definition(k,j-1)==1) THEN
                                        unew(k,j)=ufluxin(counter)*(x(k,j)-x(k,j-2))/3+(4/3)*unew(k,j-1)-
                                        (1/3)*unew(k,j-2)
                                        END IF
                                ELSE IF (inletbc==5) THEN
                                        unew(k,j)=qin(counter)/hold(k,j)
                                END IF
                        END IF

                        ! Outlet
                        IF ((definition(k,j)==3).OR.(definition(k,j)==31)) THEN
                                IF (outletbc==3) THEN
                                        unew(k,j)=uout(counter)
                                ELSE IF (outletbc==4) THEN
                                        IF (definition(k,j+1)==1) THEN
                                        unew(k,j)=ufluxout(counter)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*unew(k,j+1)-
                                        (1/3)*unew(k,j+2)
                                        ELSE IF (definition(k,j-1)==1) THEN
                                        unew(k,j)=ufluxout(counter)*(x(k,j)-x(k,j-2))/3+(4/3)*unew(k,j-1)-
                                        (1/3)*unew(k,j-2)
                                        END IF
                                ELSE IF (outletbc==5) THEN
                                        unew(k,j)=qout(counter)/hold(k,j)
                                END IF
                        END IF
```

```
            END DO
        END DO
END IF

END SUBROUTINE UnewBoundary
!!!!!!!!!!!!!!!!!!!! END U NEW BOUNDARY SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!! V NEW BOUNDARY SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Re-establishes v boundary conditions
SUBROUTINE VnewBoundary(x,rowmax,columnmax,vnew,definition)

IMPLICIT NONE
INTEGER, INTENT(IN):: rowmax,columnmax
REAL(8), DIMENSION(rowmax,columnmax), INTENT(IN):: x,definition
REAL(8), DIMENSION(rowmax,columnmax), INTENT(INOUT):: vnew
INTEGER:: j,k

        ! VNEW_BOUNDARY VARIABLES
        ! rowmax: maximum row number in grid
        ! columnmax: maximum column number in grid
        ! x: x location of computational nodes in physical space [m]
        ! definition: grid point status as a solid-boundary, inlet, etc.
        ! vnew: updated y-velocity field [m/s]
        ! j: counter for column index
        ! k: counter for row index

! Set no-slip solid boundary condition with v=0
DO k=1,rowmax
        DO j=1,columnmax
            IF
            ((definition(k,j)==0).OR.(definition(k,j)==4).OR.(definition(k,j)==2).OR.(definition(k,j)==3).OR.(definition(k,j)=
            =21).OR.(definition(k,j)==31)) THEN
            vnew(k,j)=0
            END IF
        END DO
END DO

END SUBROUTINE VnewBoundary
!!!!!!!!!!!!!!!!!!!! END V NEW BOUNDARY SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!! H NEW BOUNDARY SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Re-establishes h boundary conditions
SUBROUTINE
HnewBoundary(rowmax,columnmax,hin,hout,counter,numtimesteps,unew,hnew,definition,answer,inletbc,outletbc,hfluxin,hfluxout,qin,qo
ut,x,y)

IMPLICIT NONE
CHARACTER(5),INTENT(IN):: answer
INTEGER, INTENT(IN):: rowmax,columnmax,numtimesteps,counter,inletbc,outletbc
REAL(8), DIMENSION(rowmax,columnmax), INTENT(IN):: definition,x,y
REAL(8), DIMENSION(rowmax,columnmax), INTENT(INOUT):: hnew,unew
REAL(8), DIMENSION(numtimesteps), INTENT(IN):: hin,hout,hfluxin,hfluxout,qin,qout
INTEGER:: j,k

        ! HNEW_BOUNDARY VARIABLES
        ! answer: 'T' for transient case & 'S' for steady-state case
        ! rowmax: maximum row number in grid
        ! columnmax: maximum column number in grid
        ! numtimesteps: number of timesteps at which transient data is provided
        ! counter: indicates current timestep
        ! inletbc: user's choice of boundary condition at inlet
        ! outletbc: user's choice of boundary condition at outlet
        ! definition: grid point status as a solid-boundary, inlet, etc.
        ! x: x location of computational nodes in physical space [m]
        ! y: y location of computational nodes in physical space [m]
        ! hnew: update of water level information [m]
        ! unew: update of x-component of velocity [m/s]
        ! hin: water level at the inlet [m]
        ! hout: water level at the outlet [m]
        ! hfluxin: dh/dx at the inlet [-]
```

```
        ! hfluxout: dh/dx at the outlet [-]
        ! qin: flowrate per unit  width(q) at the inlet [m^2/s]
        ! qout: flowrate per unit width(q) at the outlet [m^2/s]
        ! j: counter for column index
        ! k: counter for row index

! STEADY-STATE CASE: Resets steady-state boundary conditions at inlet and outlet
IF ((answer=='S').OR.(answer=='s')) THEN

        DO j=1,columnmax
                DO k=1,rowmax

                        ! Inlet
                        IF ((definition(k,j)==2).OR.(definition(k,j)==21)) THEN
                                IF (inletbc==1) THEN
                                        hnew(k,j)=hin(1)
                                ELSE IF (inletbc==2) THEN
                                        IF (definition(k,j+1)==1) THEN
                                        hnew(k,j)=hfluxin(1)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*hnew(k,j+1)-
                                        (1/3)*hnew(k,j+2)
                                        ELSE IF (definition(k,j-1)==1) THEN
                                        hnew(k,j)=hfluxin(1)*(x(k,j)-x(k,j-2))/3+(4/3)*hnew(k,j-1)-(1/3)*hnew(k,j-2)
                                        END IF
                                ELSE IF (inletbc==5) THEN
                                        unew(k,j)=qin(1)/hnew(k,j)
                                END IF

                        ! Outlet
                        ELSE IF ((definition(k,j)==3).OR.(definition(k,j)==31)) THEN
                                IF (outletbc==1) THEN
                                        hnew(k,j)=hout(1)
                                ELSE IF (outletbc==2) THEN
                                        IF (definition(k,j+1)==1) THEN
                                        hnew(k,j)=hfluxout(1)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*hnew(k,j+1)-
                                        (1/3)*hnew(k,j+2)
                                        ELSE IF (definition(k,j-1)==1) THEN
                                        hnew(k,j)=hfluxout(1)*(x(k,j)-x(k,j-2))/3+(4/3)*hnew(k,j-1)-(1/3)*hnew(k,j-2)
                                        END IF
                                ELSE IF (outletbc==5) THEN
                                        unew(k,j)=qout(1)/hnew(k,j)
                                END IF
                        END IF

                END DO
        END  DO
END IF

! TRANSIENT CASE: Resets transient boundary conditions at inlet and outlet
IF ((answer=='T').OR.(answer=='t')) THEN

        DO j=1,columnmax
                DO k=1,rowmax

                        ! Inlet
                        IF ((definition(k,j)==2).OR.(definition(k,j)==21)) THEN
                                IF (inletbc==1) THEN
                                        hnew(k,j)=hin(counter)
                                ELSE IF (inletbc==2) THEN
                                        IF (definition(k,j+1)==1) THEN
                                        hnew(k,j)=hfluxin(counter)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*hnew(k,j+1)-
                                        (1/3)*hnew(k,j+2)
                                        ELSE IF (definition(k,j-1)==1) THEN
                                        hnew(k,j)=hfluxin(counter)*(x(k,j)-x(k,j-2))/3+(4/3)*hnew(k,j-1)-
                                        (1/3)*hnew(k,j-2)
                                        END IF
                                ELSE IF (inletbc==5) THEN
                                        unew(k,j)=qin(counter)/hnew(k,j)
                                END IF

                        ! Outlet
                        ELSE IF ((definition(k,j)==3).OR.(definition(k,j)==31)) THEN
```

```
                        IF (outletbc==1) THEN
                                hnew(k,j)=hout(counter)
                        ELSE IF (outletbc==2) THEN
                                IF (definition(k,j+1)==1) THEN
                                hnew(k,j)=hfluxout(counter)*(x(k,j+2)-x(k,j))/(-3)+(4/3)*hnew(k,j+1)-
                                (1/3)*hnew(k,j+2)
                                ELSE IF (definition(k,j-1)==1) THEN
                                hnew(k,j)=hfluxout(counter)*(x(k,j)-x(k,j-2))/3+(4/3)*hnew(k,j-1)-
                                (1/3)*hnew(k,j-2)
                                END IF
                        ELSE IF (outletbc==5) THEN
                                unew(k,j)=qout(counter)/hnew(k,j)
                        END IF
                END IF

        END DO
    END DO
END IF

! Linear extrapolation: grid corners
DO j=1,columnmax
    DO k=1,rowmax
        IF (definition(k,j)==4) THEN
                ! Bottom corners
                IF
                (((k+1)<=rowmax).AND.((definition(k+1,j)==1).OR.(definition(k+1,j)==21).OR.(definition(k+1,j)==
                31).OR.(definition(k+1,j)==0))) THEN
                hnew(k,j)=-((hnew(k+2,j)-hnew(k+1,j))/(y(k+2,j)-y(k+1,j)))*(y(k+1,j)-y(k,j))+hnew(k+1,j)
                ! Top corners
                ELSE IF (((k-1)>=1).AND.((definition(k-1,j)==1).OR.(definition(k-1,j)==21).OR.(definition(k-
                1,j)==31).OR.(definition(k-1,j)==0))) THEN
                hnew(k,j)=-((hnew(k-2,j)-hnew(k-1,j))/(y(k-2,j)-y(k-1,j)))*(y(k-1,j)-y(k,j))+hnew(k-1,j)
                END IF
        END IF
    END DO
END DO

END SUBROUTINE HnewBoundary
!!!!!!!!!!!!!!!!!!!! END H NEW BOUNDARY SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!! X MOMENTUM SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Solves the discretized X-momentum equation for u [m/s]
SUBROUTINE Xmomentum(x,y,rowmax,columnmax,uold,vold,hold,unew,Chezy,wind,theta,turb,zb,definition,timestep)

IMPLICIT NONE
INTEGER, INTENT(IN):: rowmax,columnmax
REAL(8), INTENT(IN):: timestep
REAL(8), DIMENSION(rowmax,columnmax), INTENT(IN):: x,y,chezy,wind,theta,turb,zb,definition
REAL(8), DIMENSION(rowmax,columnmax), INTENT(INOUT):: uold,vold,hold
REAL(8), DIMENSION(rowmax,columnmax), INTENT(OUT):: unew
REAL(8), DIMENSION(rowmax,columnmax)::
bigJ,xeps,xeta,yeps,yeta,bigU,bigV,h,term2,term3,term4,term5,term6,term7,term8,q11,q12,q22,q11eps,q12eps,ueps,uepseps,ueta,uetaeps,
q22eta,q12eta,uetaeta,T1

INTEGER:: deltaeps,deltaeta,j,k
REAL(8):: g,airdensity,waterdensity,Cw

deltaeps=1                              ! space between eps grid points in computational space
deltaeta=1                              ! space between eta grid points in computational space
g=9.81                                  ! gravity [m/s^2]
airdensity=1.2                          ! density of air [kg/m^3]
waterdensity=998                        ! density of water [kg/m^3]
Cw=.001                                 ! wind drag coefficient [-]

    ! X_MOMENTUM VARIABLES
    ! rowmax: maximum row number in grid
    ! columnmax: maximum column number in grid
    ! timestep: size of timestep [s]
    ! x: x location of a computational node in physical space [m]
    ! y: y location of a computational node in physical space [m]
    ! chezy: chezy coefficient used to calculate bed stress [(m^.5)/s]
```

```
            ! wind: windspeed magnitude [m/s]
            ! theta: wind direction angle counterclockwise from positive x axis [degree]
            ! turb: horizontal turbulent eddy viscosity used to calculate turbulence [m^2/s]
            ! zb: bed height [m]
            ! definition: defines status of each grid point (i.e. solid boundaries, inlet)
            ! uold: x-velocity field from previous timestep [m/s]
            ! vold: y-velocity field from previous timestep [m/s]
            ! hold: water level at previous timestep [m]
            ! unew: updated u calculated at the end of this subroutine
            ! j: counter for column index
            ! k: counter for row index
            ! Other variables have the same notation as that used in
            ! Ye & McCorquodale, "Depth-averaged hydrodynamic model in curvilinear collocated grid"
            ! Journal of Hydraulic Engineering, 123(5) 1997. Substituting eps for xi.

! Grid point has two points on either side of it
DO k=1,rowmax
        DO j=1,columnmax
                IF (definition(k,j)==8) THEN
                xeps(k,j)=(x(k,j+1)-x(k,j-1))/(2*deltaeps)
                xeps(k+1,j)=(x(k+1,j+1)-x(k+1,j-1))/(2*deltaeps)
                xeps(k-1,j)=(x(k-1,j+1)-x(k-1,j-1))/(2*deltaeps)
                xeps(k,j+1)=(x(k,j+2)-x(k,j))/(2*deltaeps)
                xeps(k,j-1)=(x(k,j)-x(k,j-2))/(2*deltaeps)
                yeps(k,j)=(y(k,j+1)-y(k,j-1))/(2*deltaeps)
                yeps(k+1,j)=(y(k+1,j+1)-y(k+1,j-1))/(2*deltaeps)
                yeps(k-1,j)=(y(k-1,j+1)-y(k-1,j-1))/(2*deltaeps)
                yeps(k,j+1)=(y(k,j+2)-y(k,j))/(2*deltaeps)
                yeps(k,j-1)=(y(k,j)-y(k,j-2))/(2*deltaeps)
                xeta(k,j)=(x(k+1,j)-x(k-1,j))/(2*deltaeta)
                xeta(k+1,j)=(x(k+2,j)-x(k,j))/(2*deltaeta)
                xeta(k-1,j)=(x(k,j)-x(k-2,j))/(2*deltaeta)
                xeta(k,j+1)=(x(k+1,j+1)-x(k-1,j+1))/(2*deltaeta)
                xeta(k,j-1)=(x(k+1,j-1)-x(k-1,j-1))/(2*deltaeta)
                yeta(k,j)=(y(k+1,j)-y(k-1,j))/(2*deltaeta)
                yeta(k+1,j)=(y(k+2,j)-y(k,j))/(2*deltaeta)
                yeta(k-1,j)=(y(k,j)-y(k-2,j))/(2*deltaeta)
                yeta(k,j+1)=(y(k+1,j+1)-y(k-1,j+1))/(2*deltaeta)
                yeta(k,j-1)=(y(k+1,j-1)-y(k-1,j-1))/(2*deltaeta)

                bigJ(k,j)=xeps(k,j)*yeta(k,j)-xeta(k,j)*yeps(k,j)
                bigJ(k+1,j)=xeps(k+1,j)*yeta(k+1,j)-xeta(k+1,j)*yeps(k+1,j)
                bigJ(k-1,j)=xeps(k-1,j)*yeta(k-1,j)-xeta(k-1,j)*yeps(k-1,j)
                bigJ(k,j+1)=xeps(k,j+1)*yeta(k,j+1)-xeta(k,j+1)*yeps(k,j+1)
                bigJ(k,j-1)=xeps(k,j-1)*yeta(k,j-1)-xeta(k,j-1)*yeps(k,j-1)

                bigU(k,j)=yeta(k,j)*uold(k,j)-xeta(k,j)*vold(k,j)
                bigU(k,j+1)=yeta(k,j+1)*uold(k,j+1)-xeta(k,j+1)*vold(k,j+1)
                bigU(k,j-1)=yeta(k,j-1)*uold(k,j-1)-xeta(k,j-1)*vold(k,j-1)
                bigV(k,j)=xeps(k,j)*vold(k,j)-yeps(k,j)*uold(k,j)
                bigV(k+1,j)=xeps(k+1,j)*vold(k+1,j)-yeps(k+1,j)*uold(k+1,j)
                bigV(k-1,j)=xeps(k-1,j)*vold(k-1,j)-yeps(k-1,j)*uold(k-1,j)

                q11(k,j)=xeta(k,j)**2+2*yeta(k,j)**2
                q11(k,j+1)=xeta(k,j+1)**2+2*yeta(k,j+1)**2
                q11(k,j-1)=xeta(k,j-1)**2+2*yeta(k,j-1)**2
                q12(k,j)=xeps(k,j)*xeta(k,j)+2*yeps(k,j)*yeta(k,j)
                q12(k+1,j)=xeps(k+1,j)*xeta(k+1,j)+2*yeps(k+1,j)*yeta(k+1,j)
                q12(k-1,j)=xeps(k-1,j)*xeta(k-1,j)+2*yeps(k-1,j)*yeta(k-1,j)
                q12(k,j+1)=xeps(k,j+1)*xeta(k,j+1)+2*yeps(k,j+1)*yeta(k,j+1)
                q12(k,j-1)=xeps(k,j-1)*xeta(k,j-1)+2*yeps(k,j-1)*yeta(k,j-1)
                q22(k,j)=xeps(k,j)**2+2*yeps(k,j)**2
                q22(k+1,j)=xeps(k+1,j)**2+2*yeps(k+1,j)**2
                q22(k-1,j)=xeps(k-1,j)**2+2*yeps(k-1,j)**2

                ueps(k,j)=(uold(k,j+1)-uold(k,j-1))/(2*deltaeps)
                uepseps(k,j)=(uold(k,j+1)-2*uold(k,j)+uold(k,j-1))/deltaeps**2
                ueta(k,j)=(uold(k+1,j)-uold(k-1,j))/(2*deltaeta)
                ueta(k,j+1)=(uold(k+1,j+1)-uold(k-1,j+1))/(2*deltaeta)
                ueta(k,j-1)=(uold(k+1,j-1)-uold(k-1,j-1))/(2*deltaeta)
                uetaeps(k,j)=(ueta(k,j+1)-ueta(k,j-1))/(2*deltaeps)
```

```
                    q11eps(k,j)=(q11(k,j+1)-q11(k,j-1))/(2*deltaeps)
                    q12eps(k,j)=(q12(k,j+1)-q12(k,j-1))/(2*deltaeps)
                    q22eta(k,j)=(q22(k+1,j)-q22(k-1,j))/(2*deltaeta)
                    q12eta(k,j)=(q12(k+1,j)-q12(k-1,j))/(2*deltaeta)
                    uetaeta(k,j)=(uold(k+1,j)-2*uold(k,j)+uold(k-1,j))/deltaeta**2

                    T1(k,j)=yeta(k,j)*(vold(k,j+1)-vold(k,j-1))/(2*deltaeps)-yeps(k,j)*(vold(k+1,j)-vold(k-1,j))/(2*deltaeta)
                    T1(k+1,j)=yeta(k+1,j)*(vold(k+1,j+1)-vold(k+1,j-1))/(2*deltaeps)-yeps(k+1,j)*(vold(k+2,j)-
                    vold(k,j))/(2*deltaeta)
                    T1(k-1,j)=yeta(k-1,j)*(vold(k-1,j+1)-vold(k-1,j-1))/(2*deltaeps)-yeps(k-1,j)*(vold(k,j)-vold(k-2,j))/(2*deltaeta)
                    T1(k,j+1)=yeta(k,j+1)*(vold(k,j+2)-vold(k,j))/(2*deltaeps)-yeps(k,j+1)*(vold(k+1,j+1)-vold(k-
                    1,j+1))/(2*deltaeta)
                    T1(k,j-1)=yeta(k,j-1)*(vold(k,j)-vold(k,j-2))/(2*deltaeps)-yeps(k,j-1)*(vold(k+1,j-1)-vold(k-1,j-1))/(2*deltaeta)

                    term2(k,j)=(1/bigJ(k,j))*(bigU(k,j)*(uold(k,j+1)-uold(k,j-1))/(2*deltaeps)+(bigU(k,j+1)-bigU(k,j-
                    1))/(2*deltaeps)*uold(k,j)+bigV(k,j)*(uold(k+1,j)-uold(k-1,j))/(2*deltaeta)+(bigV(k+1,j)-bigV(k-
                    1,j))/(2*deltaeta)*uold(k,j))
                    term3(k,j)=-(g/bigJ(k,j))*(yeta(k,j)*((zb(k,j+1)+hold(k,j+1))-(zb(k,j-1)+hold(k,j-1)))/(2*deltaeps)-
                    yeps(k,j)*((zb(k+1,j)+hold(k+1,j))-(zb(k-1,j)+hold(k-1,j)))/(2*deltaeta))
                    term4(k,j)=-(g/(Chezy(k,j)**2*hold(k,j)))*dsqrt(uold(k,j)**2+vold(k,j)**2)*uold(k,j)
                    !term4(k,j)=0
                    term5(k,j)=airdensity*Cw/waterdensity*wind(k,j)**2*dcosd(theta(k,j))/hold(k,j)
                    term6(k,j)=turb(k,j)/bigJ(k,j)**2*(q11eps(k,j)*ueps(k,j)+q11(k,j)*uepseps(k,j)-q12eps(k,j)*ueta(k,j)-
                    q12(k,j)*uetaeps(k,j))+turb(k,j)/bigJ(k,j)*((1/bigJ(k,j+1)-1/bigJ(k,j-1))/(2*deltaeps))*(q11(k,j)*ueps(k,j)-
                    q12(k,j)*ueta(k,j))
                    term7(k,j)=turb(k,j)/bigJ(k,j)**2*(q22eta(k,j)*ueta(k,j)+q22(k,j)*uetaeta(k,j)-q12(k,j)*uetaeps(k,j)-
                    q12eta(k,j)*ueps(k,j))+turb(k,j)/bigJ(k,j)*((1/bigJ(k+1,j)-1/bigJ(k-1,j))/(2*deltaeta))*(q22(k,j)*ueta(k,j)-
                    q12(k,j)*ueps(k,j))
                    term8(k,j)=turb(k,j)/bigJ(k,j)**2*(xeps(k,j)*(T1(k+1,j)-T1(k-1,j))/(2*deltaeta)-xeta(k,j)*(T1(k,j+1)-T1(k,j-
                    1))/(2*deltaeps))+turb(k,j)/bigJ(k,j)*(((1/bigJ(k+1,j)-1/bigJ(k-1,j))/(2*deltaeta))*xeps(k,j)*T1(k,j)-((1/bigJ(k,j+1)-
                    1/bigJ(k,j-1))/(2*deltaeps))*xeta(k,j)*T1(k,j))

                    unew(k,j)=(term3(k,j)+term4(k,j)+term5(k,j)+term6(k,j)+term7(k,j)+term8(k,j)-term2(k,j))*(timestep)+uold(k,j)

                    END IF
          END DO
END DO

! Grid space is one space from solid boundary and not an inlet or outlet point
DO k=1,rowmax
          DO j=1,columnmax
                    IF (definition(k,j)==1) THEN
                    xeps(k,j)=(x(k,j+1)-x(k,j-1))/(2*deltaeps)
                    xeps(k+1,j)=(x(k+1,j+1)-x(k+1,j-1))/(2*deltaeps)
                    xeps(k-1,j)=(x(k-1,j+1)-x(k-1,j-1))/(2*deltaeps)
                    xeps(k,j+1)=(x(k,j+2)-x(k,j))/(2*deltaeps)
                    xeps(k,j-1)=(x(k,j)-x(k,j-2))/(2*deltaeps)
                    yeps(k,j)=(y(k,j+1)-y(k,j-1))/(2*deltaeps)
                    yeps(k+1,j)=(y(k+1,j+1)-y(k+1,j-1))/(2*deltaeps)
                    yeps(k-1,j)=(y(k-1,j+1)-y(k-1,j-1))/(2*deltaeps)
                    yeps(k,j+1)=(y(k,j+2)-y(k,j))/(2*deltaeps)
                    yeps(k,j-1)=(y(k,j)-y(k,j-2))/(2*deltaeps)
                    xeta(k,j)=(x(k+1,j)-x(k-1,j))/(2*deltaeta)
                    xeta(k+1,j)=(x(k+2,j)-x(k,j))/(2*deltaeta)
                    xeta(k-1,j)=(x(k,j)-x(k-2,j))/(2*deltaeta)
                    xeta(k,j+1)=(x(k+1,j+1)-x(k-1,j+1))/(2*deltaeta)
                    xeta(k,j-1)=(x(k+1,j-1)-x(k-1,j-1))/(2*deltaeta)
                    yeta(k,j)=(y(k+1,j)-y(k-1,j))/(2*deltaeta)
                    yeta(k+1,j)=(y(k+2,j)-y(k,j))/(2*deltaeta)
                    yeta(k-1,j)=(y(k,j)-y(k-2,j))/(2*deltaeta)
                    yeta(k,j+1)=(y(k+1,j+1)-y(k-1,j+1))/(2*deltaeta)
                    yeta(k,j-1)=(y(k+1,j-1)-y(k-1,j-1))/(2*deltaeta)
                              ! One step from lower boundary
                              IF ((definition(k-1,j)==0).OR.(definition(k-1,j)==2).OR.(definition(k-1,j)==3).OR.(definition(k-
                              1,j)==21).OR.(definition(k-1,j)==31).OR.(definition(k-1,j)==4)) THEN
                                        xeta(k-1,j)=(-3*x(k-1,j)+4*x(k,j)-1*x(k+1,j))/(2*deltaeta)
                                        yeta(k-1,j)=(-3*y(k-1,j)+4*y(k,j)-1*y(k+1,j))/(2*deltaeta)
                              END IF
                              ! One step away from upper boundary
```

```
            IF
            ((definition(k+1,j)==0).OR.(definition(k+1,j)==2).OR.(definition(k+1,j)==3).OR.(definition(k+1,j)==
            21).OR.(definition(k+1,j)==31).OR.(definition(k+1,j)==4)) THEN
                        xeta(k+1,j)=(3*x(k+1,j)-4*x(k,j)+1*x(k-1,j))/(2*deltaeta)
                        yeta(k+1,j)=(3*y(k+1,j)-4*y(k,j)+1*y(k-1,j))/(2*deltaeta)
            END IF
            ! One step away from left boundary
            IF ((definition(k,j-1)==0).OR.(definition(k,j-1)==2).OR.(definition(k,j-1)==3).OR.(definition(k,j-
            1)==21).OR.(definition(k,j-1)==31).OR.(definition(k,j-1)==4)) THEN
                        xeps(k,j-1)=(-3*x(k,j-1)+4*x(k,j)-1*x(k,j+1))/(2*deltaeps)
                        yeps(k,j-1)=(-3*y(k,j-1)+4*y(k,j)-1*y(k,j+1))/(2*deltaeps)
            END IF
            ! One step away from right boundary
            IF
            ((definition(k,j+1)==0).OR.(definition(k,j+1)==2).OR.(definition(k,j+1)==3).OR.(definition(k,j+1)==
            21).OR.(definition(k,j+1)==31).OR.(definition(k,j+1)==4)) THEN
                        xeps(k,j+1)=(3*x(k,j+1)-4*x(k,j)+1*x(k,j-1))/(2*deltaeps)
                        yeps(k,j+1)=(3*y(k,j+1)-4*y(k,j)+1*y(k,j-1))/(2*deltaeps)
            END IF

bigJ(k,j)=xeps(k,j)*yeta(k,j)-xeta(k,j)*yeps(k,j)
bigJ(k+1,j)=xeps(k+1,j)*yeta(k+1,j)-xeta(k+1,j)*yeps(k+1,j)
bigJ(k-1,j)=xeps(k-1,j)*yeta(k-1,j)-xeta(k-1,j)*yeps(k-1,j)
bigJ(k,j+1)=xeps(k,j+1)*yeta(k,j+1)-xeta(k,j+1)*yeps(k,j+1)
bigJ(k,j-1)=xeps(k,j-1)*yeta(k,j-1)-xeta(k,j-1)*yeps(k,j-1)

bigU(k,j)=yeta(k,j)*uold(k,j)-xeta(k,j)*vold(k,j)
bigU(k,j+1)=yeta(k,j+1)*uold(k,j+1)-xeta(k,j+1)*vold(k,j+1)
bigU(k,j-1)=yeta(k,j-1)*uold(k,j-1)-xeta(k,j-1)*vold(k,j-1)
bigV(k,j)=xeps(k,j)*vold(k,j)-yeps(k,j)*uold(k,j)
bigV(k+1,j)=xeps(k+1,j)*vold(k+1,j)-yeps(k+1,j)*uold(k+1,j)
bigV(k-1,j)=xeps(k-1,j)*vold(k-1,j)-yeps(k-1,j)*uold(k-1,j)

q11(k,j)=xeta(k,j)**2+2*yeta(k,j)**2
q11(k,j+1)=xeta(k,j+1)**2+2*yeta(k,j+1)**2
q11(k,j-1)=xeta(k,j-1)**2+2*yeta(k,j-1)**2
q12(k,j)=xeps(k,j)*xeta(k,j)+2*yeps(k,j)*yeta(k,j)
q12(k+1,j)=xeps(k+1,j)*xeta(k+1,j)+2*yeps(k+1,j)*yeta(k+1,j)
q12(k-1,j)=xeps(k-1,j)*xeta(k-1,j)+2*yeps(k-1,j)*yeta(k-1,j)
q12(k,j+1)=xeps(k,j+1)*xeta(k,j+1)+2*yeps(k,j+1)*yeta(k,j+1)
q12(k,j-1)=xeps(k,j-1)*xeta(k,j-1)+2*yeps(k,j-1)*yeta(k,j-1)
q22(k,j)=xeps(k,j)**2+2*yeps(k,j)**2
q22(k+1,j)=xeps(k+1,j)**2+2*yeps(k+1,j)**2
q22(k-1,j)=xeps(k-1,j)**2+2*yeps(k-1,j)**2

ueps(k,j)=(uold(k,j+1)-uold(k,j-1))/(2*deltaeps)
uepseps(k,j)=(uold(k,j+1)-2*uold(k,j)+uold(k,j-1))/deltaeps**2
ueta(k,j)=(uold(k+1,j)-uold(k-1,j))/(2*deltaeta)
ueta(k,j+1)=(uold(k+1,j+1)-uold(k-1,j+1))/(2*deltaeta)
ueta(k,j-1)=(uold(k+1,j-1)-uold(k-1,j-1))/(2*deltaeta)
uetaeps(k,j)=(ueta(k,j+1)-ueta(k,j-1))/(2*deltaeps)

q11eps(k,j)=(q11(k,j+1)-q11(k,j-1))/(2*deltaeps)
q12eps(k,j)=(q12(k,j+1)-q12(k,j-1))/(2*deltaeps)
q22eta(k,j)=(q22(k+1,j)-q22(k-1,j))/(2*deltaeta)
q12eta(k,j)=(q12(k+1,j)-q12(k-1,j))/(2*deltaeta)
uetaeta(k,j)=(uold(k+1,j)-2*uold(k,j)+uold(k-1,j))/deltaeta**2

T1(k,j)=yeta(k,j)*(vold(k,j+1)-vold(k,j-1))/(2*deltaeps)-yeps(k,j)*(vold(k+1,j)-vold(k-1,j))/(2*deltaeta)
T1(k+1,j)=yeta(k+1,j)*(vold(k+1,j+1)-vold(k+1,j-1))/(2*deltaeps)-yeps(k+1,j)*(3*vold(k+1,j)-
4*vold(k,j)+1*vold(k-1,j))/(2*deltaeta)
T1(k-1,j)=yeta(k-1,j)*(vold(k-1,j+1)-vold(k-1,j-1))/(2*deltaeps)-yeps(k-1,j)*(-3*vold(k-1,j)+4*vold(k,j)-
1*vold(k+1,j))/(2*deltaeta)
T1(k,j+1)=yeta(k,j+1)*(3*vold(k,j+1)-4*vold(k,j)+1*vold(k,j-1))/(2*deltaeps)-yeps(k,j+1)*(vold(k+1,j+1)-
vold(k-1,j+1))/(2*deltaeta)
T1(k,j-1)=yeta(k,j-1)*(-3*vold(k,j-1)+4*vold(k,j)-1*vold(k,j+1))/(2*deltaeps)-yeps(k,j-1)*(vold(k+1,j-1)-vold(k-
1,j-1))/(2*deltaeta)

term2(k,j)=(1/bigJ(k,j))*(bigU(k,j)*(uold(k,j+1)-uold(k,j-1))/(2*deltaeps)+(bigU(k,j+1)-bigU(k,j-
1))/(2*deltaeps)*uold(k,j)+bigV(k,j)*(uold(k+1,j)-uold(k-1,j))/(2*deltaeta)+(bigV(k+1,j)-bigV(k-
1,j))/(2*deltaeta)*uold(k,j))
```

```
                term3(k,j)=-(g/bigJ(k,j))*(yeta(k,j)*((zb(k,j+1)+hold(k,j+1))-(zb(k,j-1)+hold(k,j-1)))/(2*deltaeps)-
                yeps(k,j)*((zb(k+1,j)+hold(k+1,j))-(zb(k-1,j)+hold(k-1,j)))/(2*deltaeta))
                term4(k,j)=-(g/(Chezy(k,j)**2*hold(k,j)))*dsqrt(uold(k,j)**2+vold(k,j)**2)*uold(k,j)
                !term4(k,j)=0
                term5(k,j)=airdensity*Cw/waterdensity*wind(k,j)**2*dcosd(theta(k,j))/hold(k,j)
                term6(k,j)=turb(k,j)/bigJ(k,j)**2*(q11eps(k,j)*ueps(k,j)+q11(k,j)*uepseps(k,j)-q12eps(k,j)*ueta(k,j)-
                q12(k,j)*uetaeps(k,j))+turb(k,j)/bigJ(k,j)*((1/bigJ(k,j+1)-1/bigJ(k,j-1))/(2*deltaeps))*(q11(k,j)*ueps(k,j)-
                q12(k,j)*ueta(k,j))
                term7(k,j)=turb(k,j)/bigJ(k,j)**2*(q22eta(k,j)*ueta(k,j)+q22(k,j)*uetaeta(k,j)-q12(k,j)*uetaeps(k,j)-
                q12eta(k,j)*ueps(k,j))+turb(k,j)/bigJ(k,j)*((1/bigJ(k+1,j)-1/bigJ(k-1,j))/(2*deltaeta))*(q22(k,j)*ueta(k,j)-
                q12(k,j)*ueps(k,j))
                term8(k,j)=turb(k,j)/bigJ(k,j)**2*(xeps(k,j)*(T1(k+1,j)-T1(k-1,j))/(2*deltaeta)-xeta(k,j)*(T1(k,j+1)-T1(k,j-
                1))/(2*deltaeps))+turb(k,j)/bigJ(k,j)*(((1/bigJ(k+1,j)-1/bigJ(k-1,j))/(2*deltaeta))*xeps(k,j)*T1(k,j)-((1/bigJ(k,j+1)-
                1/bigJ(k,j-1))/(2*deltaeps))*xeta(k,j)*T1(k,j))

                unew(k,j)=(term3(k,j)+term4(k,j)+term5(k,j)+term6(k,j)+term7(k,j)+term8(k,j)-term2(k,j))*(timestep)+uold(k,j)

                END IF
        END DO
END DO

END SUBROUTINE Xmomentum
!!!!!!!!!!!!!!!!!!!! END X MOMENTUM SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!! Y MOMENTUM SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Solves the discretized Y-momentum equation for v [m/s]
SUBROUTINE Ymomentum(x,y,rowmax,columnmax,unew,vold,hold,vnew,Chezy,wind,theta,turb,zb,definition,timestep)

IMPLICIT NONE
INTEGER, INTENT(IN):: rowmax,columnmax
REAL(8), INTENT(IN):: timestep
REAL(8), DIMENSION(rowmax,columnmax), INTENT(IN):: x,y,chezy,wind,theta,turb,zb,definition
REAL(8), DIMENSION(rowmax,columnmax), INTENT(INOUT):: unew,vold,hold
REAL(8), DIMENSION(rowmax,columnmax), INTENT(OUT):: vnew
REAL(8), DIMENSION(rowmax,columnmax) ::
bigJ,xeps,xeta,yeps,yeta,bigU,bigV,term2,h,term3,term4,term5,q11,q12,q22,q11eps,q12eps,veps,vepseps,veta,vetaeps,term6,q22eta,q12et
a,vetaeta,term7,T2,term8

INTEGER:: deltaeps,deltaeta,j,k
REAL(8) :: g,airdensity,waterdensity,Cw

deltaeps=1                              ! space between eps grid points in computational space
deltaeta=1                              ! space between eta grid points in computational space
g=9.81                                  ! gravity [m/s^2]
airdensity=1.2                          ! density of air [kg/m^3]
waterdensity=998                        ! density of water [kg/m^3]
Cw=.001                                 ! wind drag coefficient [-]

        ! Y_MOMENTUM VARIABLES
        ! rowmax: maximum row number in grid
        ! columnmax: maximum column number in grid
        ! timestep: size of timestep [s]
        ! x: x location of a computational node in physical space [m]
        ! y: y location of a computational node in physical space [m]
        ! chezy: chezy coefficient used to calculate bed stress [(m^.5)/s]
        ! wind: windspeed magnitude [m/s]
        ! theta: wind direction angle counterclockwise from positive x axis [degree]
        ! turb: horizontal turbulent eddy viscosity used to calculate turbulence [m^2/s]
        ! zb: bed height [m]
        ! definition: defines status of each grid point (i.e. solid boundaries, inlet)
        ! unew: updated u field calculated at the end of X_momentum subroutine [m/s]
        ! vold: y-velocity field from previous timestep [m/s]
        ! hold: water level from previous timestep [m]
        ! vnew: update v field calculated at the end of this subroutine [m/s]
        ! j: counter for column index
        ! k: counter for row index
        ! Other variables have the same notation as that used in
        ! Ye & McCorquodale, "Depth-averaged hydrodynamic model in curvilinear collocated grid"
        ! Journal of Hydraulic Engineering, 123(5) 1997. Substituting eps for xi.

! Grid point has two points on either side of it
```

```
DO k=1,rowmax
        DO j=1,columnmax
                IF (definition(k,j)==8) THEN
                xeps(k,j)=(x(k,j+1)-x(k,j-1))/(2*deltaeps)
                xeps(k+1,j)=(x(k+1,j+1)-x(k+1,j-1))/(2*deltaeps)
                xeps(k-1,j)=(x(k-1,j+1)-x(k-1,j-1))/(2*deltaeps)
                xeps(k,j+1)=(x(k,j+2)-x(k,j))/(2*deltaeps)
                xeps(k,j-1)=(x(k,j)-x(k,j-2))/(2*deltaeps)
                yeps(k,j)=(y(k,j+1)-y(k,j-1))/(2*deltaeps)
                yeps(k+1,j)=(y(k+1,j+1)-y(k+1,j-1))/(2*deltaeps)
                yeps(k-1,j)=(y(k-1,j+1)-y(k-1,j-1))/(2*deltaeps)
                yeps(k,j+1)=(y(k,j+2)-y(k,j))/(2*deltaeps)
                yeps(k,j-1)=(y(k,j)-y(k,j-2))/(2*deltaeps)
                xeta(k,j)=(x(k+1,j)-x(k-1,j))/(2*deltaeta)
                xeta(k+1,j)=(x(k+2,j)-x(k,j))/(2*deltaeta)
                xeta(k-1,j)=(x(k,j)-x(k-2,j))/(2*deltaeta)
                xeta(k,j+1)=(x(k+1,j+1)-x(k-1,j+1))/(2*deltaeta)
                xeta(k,j-1)=(x(k+1,j-1)-x(k-1,j-1))/(2*deltaeta)
                yeta(k,j)=(y(k+1,j)-y(k-1,j))/(2*deltaeta)
                yeta(k+1,j)=(y(k+2,j)-y(k,j))/(2*deltaeta)
                yeta(k-1,j)=(y(k,j)-y(k-2,j))/(2*deltaeta)
                yeta(k,j+1)=(y(k+1,j+1)-y(k-1,j+1))/(2*deltaeta)
                yeta(k,j-1)=(y(k+1,j-1)-y(k-1,j-1))/(2*deltaeta)

                bigJ(k,j)=xeps(k,j)*yeta(k,j)-xeta(k,j)*yeps(k,j)
                bigJ(k+1,j)=xeps(k+1,j)*yeta(k+1,j)-xeta(k+1,j)*yeps(k+1,j)
                bigJ(k-1,j)=xeps(k-1,j)*yeta(k-1,j)-xeta(k-1,j)*yeps(k-1,j)
                bigJ(k,j+1)=xeps(k,j+1)*yeta(k,j+1)-xeta(k,j+1)*yeps(k,j+1)
                bigJ(k,j-1)=xeps(k,j-1)*yeta(k,j-1)-xeta(k,j-1)*yeps(k,j-1)

                bigU(k,j)=yeta(k,j)*unew(k,j)-xeta(k,j)*vold(k,j)
                bigU(k,j+1)=yeta(k,j+1)*unew(k,j+1)-xeta(k,j+1)*vold(k,j+1)
                bigU(k,j-1)=yeta(k,j-1)*unew(k,j-1)-xeta(k,j-1)*vold(k,j-1)
                bigV(k,j)=xeps(k,j)*vold(k,j)-yeps(k,j)*unew(k,j)
                bigV(k+1,j)=xeps(k+1,j)*vold(k+1,j)-yeps(k+1,j)*unew(k+1,j)
                bigV(k-1,j)=xeps(k-1,j)*vold(k-1,j)-yeps(k-1,j)*unew(k-1,j)

                q11(k,j)=2*xeta(k,j)**2+yeta(k,j)**2
                q11(k,j+1)=2*xeta(k,j+1)**2+yeta(k,j+1)**2
                q11(k,j-1)=2*xeta(k,j-1)**2+yeta(k,j-1)**2
                q12(k,j)=2*xeps(k,j)*xeta(k,j)+yeps(k,j)*yeta(k,j)
                q12(k+1,j)=2*xeps(k+1,j)*xeta(k+1,j)+yeps(k+1,j)*yeta(k+1,j)
                q12(k-1,j)=2*xeps(k-1,j)*xeta(k-1,j)+yeps(k-1,j)*yeta(k-1,j)
                q12(k,j+1)=2*xeps(k,j+1)*xeta(k,j+1)+yeps(k,j+1)*yeta(k,j+1)
                q12(k,j-1)=2*xeps(k,j-1)*xeta(k,j-1)+yeps(k,j-1)*yeta(k,j-1)
                q22(k,j)=2*xeps(k,j)**2+yeps(k,j)**2
                q22(k+1,j)=2*xeps(k+1,j)**2+yeps(k+1,j)**2
                q22(k-1,j)=2*xeps(k-1,j)**2+yeps(k-1,j)**2

                q11eps(k,j)=(q11(k,j+1)-q11(k,j-1))/(2*deltaeps)
                q12eps(k,j)=(q12(k,j+1)-q12(k,j-1))/(2*deltaeps)
                q22eta(k,j)=(q22(k+1,j)-q22(k-1,j))/(2*deltaeta)
                q12eta(k,j)=(q12(k+1,j)-q12(k-1,j))/(2*deltaeta)

                veps(k,j)=(vold(k,j+1)-vold(k,j-1))/(2*deltaeps)
                vepseps(k,j)=(vold(k,j+1)-2*vold(k,j)+vold(k,j-1))/deltaeps**2
                veta(k,j)=(vold(k+1,j)-vold(k-1,j))/(2*deltaeta)
                veta(k,j+1)=(vold(k+1,j+1)-vold(k-1,j+1))/(2*deltaeta)
                veta(k,j-1)=(vold(k+1,j-1)-vold(k-1,j-1))/(2*deltaeta)
                vetaeps(k,j)=(veta(k,j+1)-veta(k,j-1))/(2*deltaeps)
                vetaeta(k,j)=(vold(k+1,j)-2*vold(k,j)+vold(k-1,j))/deltaeta**2

                T2(k,j)=xeps(k,j)*(unew(k+1,j)-unew(k-1,j))/(2*deltaeta)-xeta(k,j)*(unew(k,j+1)-unew(k,j-1))/(2*deltaeps)
                T2(k+1,j)=xeps(k+1,j)*(unew(k+2,j)-unew(k,j))/(2*deltaeta)-xeta(k+1,j)*(unew(k+1,j+1)-unew(k+1,j-
                1))/(2*deltaeps)
                T2(k-1,j)=xeps(k-1,j)*(unew(k,j)-unew(k-2,j))/(2*deltaeta)-xeta(k-1,j)*(unew(k-1,j+1)-unew(k-1,j-
                1))/(2*deltaeps)
                T2(k,j+1)=xeps(k,j+1)*(unew(k+1,j+1)-unew(k-1,j+1))/(2*deltaeta)-xeta(k,j+1)*(unew(k,j+2)-
                unew(k,j))/(2*deltaeps)
                T2(k,j-1)=xeps(k,j-1)*(unew(k+1,j-1)-unew(k-1,j-1))/(2*deltaeta)-xeta(k,j-1)*(unew(k,j)-unew(k,j-
                2))/(2*deltaeps)
```

```
                term2(k,j)=(1/bigJ(k,j))*(bigU(k,j)*(vold(k,j+1)-vold(k,j-1))/(2*deltaeps)+(bigU(k,j+1)-bigU(k,j-
                1))/(2*deltaeps)*vold(k,j)+bigV(k,j)*(vold(k+1,j)-vold(k-1,j))/(2*deltaeta)+(bigV(k+1,j)-bigV(k-
                1,j))/(2*deltaeta)*vold(k,j))
                term3(k,j)=-(g/bigJ(k,j))*(xeps(k,j)*((zb(k+1,j)+hold(k+1,j))-(zb(k-1,j)+hold(k-1,j)))/(2*deltaeta)-
                xeta(k,j)*((zb(k,j+1)+hold(k,j+1))-(zb(k,j-1)+hold(k,j-1)))/(2*deltaeps))
                term4(k,j)=-(g/(Chezy(k,j)**2*hold(k,j)))*dsqrt(unew(k,j)**2+vold(k,j)**2)*vold(k,j)
                !term4(k,j)=0
                term5(k,j)=airdensity*Cw/waterdensity*wind(k,j)**2*dsind(theta(k,j))/hold(k,j)
                term6(k,j)=turb(k,j)/bigJ(k,j)**2*(q11eps(k,j)*veps(k,j)+q11(k,j)*vepseps(k,j)-q12eps(k,j)*veta(k,j)-
                q12(k,j)*vetaeps(k,j))+turb(k,j)/bigJ(k,j)*((1/bigJ(k,j+1)-1/bigJ(k,j-1))/(2*deltaeps))*(q11(k,j)*veps(k,j)-
                q12(k,j)*veta(k,j))
                term7(k,j)=turb(k,j)/bigJ(k,j)**2*(q22eta(k,j)*veta(k,j)+q22(k,j)*vetaeta(k,j)-q12(k,j)*vetaeps(k,j)-
                q12eta(k,j)*veps(k,j))+turb(k,j)/bigJ(k,j)*((1/bigJ(k+1,j)-1/bigJ(k-1,j))/(2*deltaeta))*(q22(k,j)*veta(k,j)-
                q12(k,j)*veps(k,j))
                term8(k,j)=turb(k,j)/bigJ(k,j)**2*(yeta(k,j)*(T2(k,j+1)-T2(k,j-1))/(2*deltaeps)-yeps(k,j)*(T2(k+1,j)-T2(k-
                1,j))/(2*deltaeta))+turb(k,j)/bigJ(k,j)*(((1/bigJ(k,j+1)-1/bigJ(k,j-1))/(2*deltaeps))*yeta(k,j)*T2(k,j)-
                ((1/bigJ(k+1,j)-1/bigJ(k-1,j))/(2*deltaeta))*yeps(k,j)*T2(k,j))

                vnew(k,j)=(term3(k,j)+term4(k,j)+term5(k,j)+term6(k,j)+term7(k,j)+term8(k,j)-term2(k,j))*(timestep)+vold(k,j)

                END IF
        END DO
END DO

! Grid space is one space from solid boundary and not an inlet or outlet point
DO k=1,rowmax
        DO j=1,columnmax
                IF (definition(k,j)==1) THEN
                xeps(k,j)=(x(k,j+1)-x(k,j-1))/(2*deltaeps)
                xeps(k+1,j)=(x(k+1,j+1)-x(k+1,j-1))/(2*deltaeps)
                xeps(k-1,j)=(x(k-1,j+1)-x(k-1,j-1))/(2*deltaeps)
                xeps(k,j+1)=(x(k,j+2)-x(k,j))/(2*deltaeps)
                xeps(k,j-1)=(x(k,j)-x(k,j-2))/(2*deltaeps)
                yeps(k,j)=(y(k,j+1)-y(k,j-1))/(2*deltaeps)
                yeps(k+1,j)=(y(k+1,j+1)-y(k+1,j-1))/(2*deltaeps)
                yeps(k-1,j)=(y(k-1,j+1)-y(k-1,j-1))/(2*deltaeps)
                yeps(k,j+1)=(y(k,j+2)-y(k,j))/(2*deltaeps)
                yeps(k,j-1)=(y(k,j)-y(k,j-2))/(2*deltaeps)
                xeta(k,j)=(x(k+1,j)-x(k-1,j))/(2*deltaeta)
                xeta(k+1,j)=(x(k+2,j)-x(k,j))/(2*deltaeta)
                xeta(k-1,j)=(x(k,j)-x(k-2,j))/(2*deltaeta)
                xeta(k,j+1)=(x(k+1,j+1)-x(k-1,j+1))/(2*deltaeta)
                xeta(k,j-1)=(x(k+1,j-1)-x(k-1,j-1))/(2*deltaeta)
                yeta(k,j)=(y(k+1,j)-y(k-1,j))/(2*deltaeta)
                yeta(k+1,j)=(y(k+2,j)-y(k,j))/(2*deltaeta)
                yeta(k-1,j)=(y(k,j)-y(k-2,j))/(2*deltaeta)
                yeta(k,j+1)=(y(k+1,j+1)-y(k-1,j+1))/(2*deltaeta)
                yeta(k,j-1)=(y(k+1,j-1)-y(k-1,j-1))/(2*deltaeta)
                        ! One step from lower boundary
                        IF ((definition(k-1,j)==0).OR.(definition(k-1,j)==2).OR.(definition(k-1,j)==3).OR.(definition(k-
                        1,j)==21).OR.(definition(k-1,j)==31).OR.(definition(k-1,j)==4)) THEN
                                xeta(k-1,j)=(-3*x(k-1,j)+4*x(k,j)-1*x(k+1,j))/(2*deltaeta)
                                yeta(k-1,j)=(-3*y(k-1,j)+4*y(k,j)-1*y(k+1,j))/(2*deltaeta)
                        END IF
                        ! One step away from upper boundary
                        IF
                        ((definition(k+1,j)==0).OR.(definition(k+1,j)==2).OR.(definition(k+1,j)==3).OR.(definition(k+1,j)==
                        21).OR.(definition(k+1,j)==31).OR.(definition(k+1,j)==4)) THEN
                                xeta(k+1,j)=(3*x(k+1,j)-4*x(k,j)+1*x(k-1,j))/(2*deltaeta)
                                yeta(k+1,j)=(3*y(k+1,j)-4*y(k,j)+1*y(k-1,j))/(2*deltaeta)
                        END IF
                        ! One step away from left boundary
                        IF ((definition(k,j-1)==0).OR.(definition(k,j-1)==2).OR.(definition(k,j-1)==3).OR.(definition(k,j-
                        1)==21).OR.(definition(k,j-1)==31).OR.(definition(k,j-1)==4)) THEN
                                xeps(k,j-1)=(-3*x(k,j-1)+4*x(k,j)-1*x(k,j+1))/(2*deltaeps)
                                yeps(k,j-1)=(-3*y(k,j-1)+4*y(k,j)-1*y(k,j+1))/(2*deltaeps)
                        END IF
                        ! One step away from right boundary
```

```
            IF
            ((definition(k,j+1)==0).OR.(definition(k,j+1)==2).OR.(definition(k,j+1)==3).OR.(definition(k,j+1)==
            21).OR.(definition(k,j+1)==31).OR.(definition(k,j+1)==4)) THEN
                        xeps(k,j+1)=(3*x(k,j+1)-4*x(k,j)+1*x(k,j-1))/(2*deltaeps)
                        yeps(k,j+1)=(3*y(k,j+1)-4*y(k,j)+1*y(k,j-1))/(2*deltaeps)
            END IF

bigJ(k,j)=xeps(k,j)*yeta(k,j)-xeta(k,j)*yeps(k,j)
bigJ(k+1,j)=xeps(k+1,j)*yeta(k+1,j)-xeta(k+1,j)*yeps(k+1,j)
bigJ(k-1,j)=xeps(k-1,j)*yeta(k-1,j)-xeta(k-1,j)*yeps(k-1,j)
bigJ(k,j+1)=xeps(k,j+1)*yeta(k,j+1)-xeta(k,j+1)*yeps(k,j+1)
bigJ(k,j-1)=xeps(k,j-1)*yeta(k,j-1)-xeta(k,j-1)*yeps(k,j-1)

bigU(k,j)=yeta(k,j)*unew(k,j)-xeta(k,j)*vold(k,j)
bigU(k,j+1)=yeta(k,j+1)*unew(k,j+1)-xeta(k,j+1)*vold(k,j+1)
bigU(k,j-1)=yeta(k,j-1)*unew(k,j-1)-xeta(k,j-1)*vold(k,j-1)
bigV(k,j)=xeps(k,j)*vold(k,j)-yeps(k,j)*unew(k,j)
bigV(k+1,j)=xeps(k+1,j)*vold(k+1,j)-yeps(k+1,j)*unew(k+1,j)
bigV(k-1,j)=xeps(k-1,j)*vold(k-1,j)-yeps(k-1,j)*unew(k-1,j)

q11(k,j)=2*xeta(k,j)**2+yeta(k,j)**2
q11(k,j+1)=2*xeta(k,j+1)**2+yeta(k,j+1)**2
q11(k,j-1)=2*xeta(k,j-1)**2+yeta(k,j-1)**2
q12(k,j)=2*xeps(k,j)*xeta(k,j)+yeps(k,j)*yeta(k,j)
q12(k+1,j)=2*xeps(k+1,j)*xeta(k+1,j)+yeps(k+1,j)*yeta(k+1,j)
q12(k-1,j)=2*xeps(k-1,j)*xeta(k-1,j)+yeps(k-1,j)*yeta(k-1,j)
q12(k,j+1)=2*xeps(k,j+1)*xeta(k,j+1)+yeps(k,j+1)*yeta(k,j+1)
q12(k,j-1)=2*xeps(k,j-1)*xeta(k,j-1)+yeps(k,j-1)*yeta(k,j-1)
q22(k,j)=2*xeps(k,j)**2+yeps(k,j)**2
q22(k+1,j)=2*xeps(k+1,j)**2+yeps(k+1,j)**2
q22(k-1,j)=2*xeps(k-1,j)**2+yeps(k-1,j)**2

veps(k,j)=(vold(k,j+1)-vold(k,j-1))/(2*deltaeps)
vepseps(k,j)=(vold(k,j+1)-2*vold(k,j)+vold(k,j-1))/deltaeps**2
veta(k,j)=(vold(k+1,j)-vold(k-1,j))/(2*deltaeta)
veta(k,j+1)=(vold(k+1,j+1)-vold(k-1,j+1))/(2*deltaeta)
veta(k,j-1)=(vold(k+1,j-1)-vold(k-1,j-1))/(2*deltaeta)
vetaeps(k,j)=(veta(k,j+1)-veta(k,j-1))/(2*deltaeps)
vetaeta(k,j)=(vold(k+1,j)-2*vold(k,j)+vold(k-1,j))/deltaeta**2

q11eps(k,j)=(q11(k,j+1)-q11(k,j-1))/(2*deltaeps)
q12eps(k,j)=(q12(k,j+1)-q12(k,j-1))/(2*deltaeps)
q22eta(k,j)=(q22(k+1,j)-q22(k-1,j))/(2*deltaeta)
q12eta(k,j)=(q12(k+1,j)-q12(k-1,j))/(2*deltaeta)

T2(k,j)=xeps(k,j)*(unew(k+1,j)-unew(k-1,j))/(2*deltaeta)-xeta(k,j)*(unew(k,j+1)-unew(k,j-1))/(2*deltaeps)
T2(k+1,j)=xeps(k+1,j)*(3*unew(k+1,j)-4*unew(k,j)+1*unew(k-1,j))/(2*deltaeta)-xeta(k+1,j)*(unew(k+1,j+1)-
unew(k+1,j-1))/(2*deltaeps)
T2(k-1,j)=xeps(k-1,j)*(-3*unew(k-1,j)+4*unew(k,j)-1*unew(k+1,j))/(2*deltaeta)-xeta(k-1,j)*(unew(k-1,j+1)-
unew(k-1,j-1))/(2*deltaeps)
T2(k,j+1)=xeps(k,j+1)*(unew(k+1,j+1)-unew(k-1,j+1))/(2*deltaeta)-xeta(k,j+1)*(3*unew(k,j+1)-
4*unew(k,j)+1*unew(k,j-1))/(2*deltaeps)
T2(k,j-1)=xeps(k,j-1)*(unew(k+1,j-1)-unew(k-1,j-1))/(2*deltaeta)-xeta(k,j-1)*(-3*unew(k,j-1)+4*unew(k,j)-
1*unew(k,j+1))/(2*deltaeps)

term2(k,j)=(1/bigJ(k,j))*(bigU(k,j)*(vold(k,j+1)-vold(k,j-1))/(2*deltaeps)+(bigU(k,j+1)-bigU(k,j-
1))/(2*deltaeps)*vold(k,j)+bigV(k,j)*(vold(k+1,j)-vold(k-1,j))/(2*deltaeta)+(bigV(k+1,j)-bigV(k-
1,j))/(2*deltaeta)*vold(k,j))
term3(k,j)=-(g/bigJ(k,j))*(xeps(k,j)*((zb(k+1,j)+hold(k+1,j))-(zb(k-1,j)+hold(k-1,j)))/(2*deltaeta)-
xeta(k,j)*((zb(k,j+1)+hold(k,j+1))-(zb(k,j-1)+hold(k,j-1)))/(2*deltaeps))
term4(k,j)=-(g/(Chezy(k,j)**2*hold(k,j)))*dsqrt(unew(k,j)**2+vold(k,j)**2)*vold(k,j)
!term4(k,j)=0
term5(k,j)=airdensity*Cw/waterdensity*wind(k,j)**2*dsind(theta(k,j))/hold(k,j)
term6(k,j)=turb(k,j)/bigJ(k,j)**2*(q11eps(k,j)*veps(k,j)+q11(k,j)*vepseps(k,j)-q12eps(k,j)*veta(k,j)-
q12(k,j)*vetaeps(k,j))+turb(k,j)/bigJ(k,j)*((1/bigJ(k,j+1)-1/bigJ(k,j-1))/(2*deltaeps))*(q11(k,j)*veps(k,j)-
q12(k,j)*veta(k,j))
term7(k,j)=turb(k,j)/bigJ(k,j)**2*(q22eta(k,j)*veta(k,j)+q22(k,j)*vetaeta(k,j)-q12(k,j)*vetaeps(k,j)-
q12eta(k,j)*veps(k,j))+turb(k,j)/bigJ(k,j)*((1/bigJ(k+1,j)-1/bigJ(k-1,j))/(2*deltaeta))*(q22(k,j)*veta(k,j)-
q12(k,j)*veps(k,j))
```

```
                term8(k,j)=turb(k,j)/bigJ(k,j)**2*(yeta(k,j)*(T2(k,j+1)-T2(k,j-1))/(2*deltaeps)-yeps(k,j)*(T2(k+1,j)-T2(k-
                1,j))/(2*deltaeta))+turb(k,j)/bigJ(k,j)*(((1/bigJ(k,j+1)-1/bigJ(k,j-1))/(2*deltaeps))*yeta(k,j)*T2(k,j)-
                ((1/bigJ(k+1,j)-1/bigJ(k-1,j))/(2*deltaeta))*yeps(k,j)*T2(k,j))

                vnew(k,j)=(term3(k,j)+term4(k,j)+term5(k,j)+term6(k,j)+term7(k,j)+term8(k,j)-term2(k,j))*(timestep)+vold(k,j)

                END IF
        END DO
END DO

END SUBROUTINE Ymomentum
!!!!!!!!!!!!!!!!!!!! END Y MOMENTUM SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!! WATER LEVEL SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Solves discretized continuity equation for water level [m]
SUBROUTINE Waterlevel(x,y,numtimesteps,rowmax,columnmax,unew,vnew,hold,hnew,rainrate,definition,timestep,answer,counter)

IMPLICIT NONE
CHARACTER(5), INTENT(IN)::answer
INTEGER, INTENT(IN):: rowmax,columnmax,numtimesteps,counter
REAL(8), INTENT(IN):: timestep
REAL(8), DIMENSION(rowmax,columnmax), INTENT(IN):: x,y,definition
REAL(8), DIMENSION(rowmax,columnmax), INTENT(INOUT):: unew,vnew,hold
REAL(8), DIMENSION(rowmax,columnmax), INTENT(OUT):: hnew
REAL(8), DIMENSION(rowmax,columnmax):: bigJ,xeps,xeta,yeps,yeta,term1,term2,term3,term4,ueps,ueta,veps,veta,heps,heta
REAL(8), DIMENSION(numtimesteps), INTENT(IN):: rainrate

INTEGER:: deltaeps,deltaeta,j,k

deltaeps=1                      ! space between eps grid points in computational space
deltaeta=1                      ! space between eta grid points in computational space

        ! WATER_LEVEL VARIABLES
        ! answer: 'T' for transient case & 'S' for steady-state case
        ! rowmax: maximum row number in grid
        ! columnmax: maximum column number in grid
        ! numtimesteps: number of timesteps at which transient data is provided
        ! counter: current timestep
        ! timestep: size of timestep [s]
        ! x: x location of a computational node in physical space [m]
        ! y: y location of a computational node in physical space [m]
        ! definition: defines status of each grid point (i.e. solid boundaries, inlet)
        ! unew: updated u field calculated at the end of X_momentum subroutine [m/s]
        ! vnew: updated v field calculated at the end of Y_momentum subroutine [m/s]
        ! hold: water level from previous timestep [m]
        ! hnew: water level calculated at the end of this subroutine [m]
        ! j: counter for column index
        ! k: counter for row index
        ! Other variables have the same notation as that used in
        ! Ye & McCorquodale, "Depth-averaged hydrodynamic model in curvilinear collocated grid"
        ! Journal of Hydraulic Engineering, 123(5) 1997. Substituting eps for xi.

! Calculates h at new timestep grid points that are not solid boundary, or inlet, or outlet
DO k=1,rowmax
        DO j=1,columnmax
                IF ((definition(k,j)==8) .OR. (definition(k,j)==1)) THEN
                xeps(k,j)=(x(k,j+1)-x(k,j-1))/(2*deltaeps)
                xeta(k,j)=(x(k+1,j)-x(k-1,j))/(2*deltaeta)
                yeps(k,j)=(y(k,j+1)-y(k,j-1))/(2*deltaeps)
                yeta(k,j)=(y(k+1,j)-y(k-1,j))/(2*deltaeta)
                bigJ(k,j)=xeps(k,j)*yeta(k,j)-xeta(k,j)*yeps(k,j)

                ueps(k,j)=(unew(k,j+1)-unew(k,j-1))/(2*deltaeps)
                ueta(k,j)=(unew(k+1,j)-unew(k-1,j))/(2*deltaeta)
                veps(k,j)=(vnew(k,j+1)-vnew(k,j-1))/(2*deltaeps)
                veta(k,j)=(vnew(k+1,j)-vnew(k-1,j))/(2*deltaeta)

                heps(k,j)=(hold(k,j+1)-hold(k,j-1))/(2*deltaeps)
                heta(k,j)=(hold(k+1,j)-hold(k-1,j))/(2*deltaeta)

                term1(k,j)=hold(k,j)*(ueps(k,j)*yeta(k,j)-veps(k,j)*xeta(k,j)-ueta(k,j)*yeps(k,j)+veta(k,j)*xeps(k,j))
```

```
                term2(k,j)=unew(k,j)*(heps(k,j)*yeta(k,j)-heta(k,j)*yeps(k,j))
                term3(k,j)=vnew(k,j)*(-heps(k,j)*xeta(k,j)+heta(k,j)*xeps(k,j))
                        ! Include rainrate if transient case
                        IF ((answer=='T').OR.(answer=='t')) THEN
                        term4(k,j)=-rainrate(counter)
                        ELSE
                        term4(k,j)=0
                        END IF

                hnew(k,j)=-((1/bigJ(k,j))*(term1(k,j)+term2(k,j)+term3(k,j))+term4(k,j))*(timestep)+hold(k,j)
                END IF
        END DO
END DO

! Calculates h for the new time step at solid boundary gridpoints
DO k=1,rowmax
        DO j=1,columnmax
                IF
                ((definition(k,j)==0).OR.(definition(k,j)==2).OR.(definition(k,j)==3).OR.(definition(k,j)==21).OR.(definition(k,j)
                ==31)) THEN

                        xeps(k,j)=(x(k,j+1)-x(k,j-1))/(2*deltaeps)
                        xeta(k,j)=(x(k+1,j)-x(k-1,j))/(2*deltaeta)
                        yeps(k,j)=(y(k,j+1)-y(k,j-1))/(2*deltaeps)
                        yeta(k,j)=(y(k+1,j)-y(k-1,j))/(2*deltaeta)

                        ! Bottom boundary
                        IF ((definition(k+1,j)==1).OR.(definition(k+1,j)==21).OR.(definition(k+1,j)==31)) THEN
                                xeta(k,j)=(-3*x(k,j)+4*x(k+1,j)-1*x(k+2,j))/(2*deltaeta)
                                yeta(k,j)=(-3*y(k,j)+4*y(k+1,j)-1*y(k+2,j))/(2*deltaeta)
                        END IF
                        ! Top boundary
                        IF ((definition(k-1,j)==1).OR.(definition(k-1,j)==21).OR.(definition(k-1,j)==31)) THEN
                                xeta(k,j)=(3*x(k,j)-4*x(k-1,j)+1*x(k-2,j))/(2*deltaeta)
                                yeta(k,j)=(3*y(k,j)-4*y(k-1,j)+1*y(k-2,j))/(2*deltaeta)
                        END IF
                        ! Right boundary
                        IF ((definition(k,j-1)==1).OR.(definition(k,j-1)==21).OR.(definition(k,j-1)==31)) THEN
                                xeps(k,j)=(3*x(k,j)-4*x(k,j-1)+1*x(k,j-2))/(2*deltaeps)
                                yeps(k,j)=(3*y(k,j)-4*y(k,j-1)+1*y(k,j-2))/(2*deltaeps)
                        END IF
                        ! Left boundary
                        IF ((definition(k,j+1)==1).OR.(definition(k,j+1)==21).OR.(definition(k,j+1)==31)) THEN
                                xeps(k,j)=(-3*x(k,j)+4*x(k,j+1)-1*x(k,j+2))/(2*deltaeps)
                                yeps(k,j)=(-3*y(k,j)+4*y(k,j+1)-1*y(k,j+2))/(2*deltaeps)
                        END IF

                bigJ(k,j)=xeps(k,j)*yeta(k,j)-xeta(k,j)*yeps(k,j)

                ueps(k,j)=(unew(k,j+1)-unew(k,j-1))/(2*deltaeps)
                ueta(k,j)=(unew(k+1,j)-unew(k-1,j))/(2*deltaeta)
                veps(k,j)=(vnew(k,j+1)-vnew(k,j-1))/(2*deltaeps)
                veta(k,j)=(vnew(k+1,j)-vnew(k-1,j))/(2*deltaeta)
                heps(k,j)=(hold(k,j+1)-hold(k,j-1))/(2*deltaeps)
                heta(k,j)=(hold(k+1,j)-hold(k-1,j))/(2*deltaeta)

                        ! Bottom boundary
                        IF ((definition(k+1,j)==1).OR.(definition(k+1,j)==21).OR.(definition(k+1,j)==31)) THEN
                                ueta(k,j)=(-3*unew(k,j)+4*unew(k+1,j)-1*unew(k+2,j))/(2*deltaeta)
                                veta(k,j)=(-3*vnew(k,j)+4*vnew(k+1,j)-1*vnew(k+2,j))/(2*deltaeta)
                                heta(k,j)=(-3*hold(k,j)+4*hold(k+1,j)-1*hold(k+2,j))/(2*deltaeta)
                        END IF
                        ! Top boundary
                        IF ((definition(k-1,j)==1).OR.(definition(k-1,j)==21).OR.(definition(k-1,j)==31)) THEN
                                ueta(k,j)=(3*unew(k,j)-4*unew(k-1,j)+1*unew(k-2,j))/(2*deltaeta)
                                veta(k,j)=(3*vnew(k,j)-4*vnew(k-1,j)+1*vnew(k-2,j))/(2*deltaeta)
                                heta(k,j)=(3*hold(k,j)-4*hold(k-1,j)+1*hold(k-2,j))/(2*deltaeta)
                        END IF
                        ! Right boundary
                        IF ((definition(k,j-1)==1).OR.(definition(k,j-1)==21).OR.(definition(k,j-1)==31)) THEN
                                ueps(k,j)=(3*unew(k,j)-4*unew(k,j-1)+1*unew(k,j-2))/(2*deltaeps)
```

```
                    veps(k,j)=(3*vnew(k,j)-4*vnew(k,j-1)+1*vnew(k,j-2))/(2*deltaeps)
                    heps(k,j)=(3*hold(k,j)-4*hold(k,j-1)+1*hold(k,j-2))/(2*deltaeps)
               END IF
               ! Left boundary
               IF ((definition(k,j+1)==1).OR.(definition(k,j+1)==21).OR.(definition(k,j+1)==31)) THEN
                    ueps(k,j)=(-3*unew(k,j)+4*unew(k,j+1)-unew(k,j+2))/(2*deltaeps)
                    veps(k,j)=(-3*vnew(k,j)+4*vnew(k,j+1)-vnew(k,j+2))/(2*deltaeps)
                    heps(k,j)=(-3*hold(k,j)+4*hold(k,j+1)-hold(k,j+2))/(2*deltaeps)
               END IF

          term1(k,j)=hold(k,j)*(ueps(k,j)*yeta(k,j)-veps(k,j)*xeta(k,j)-ueta(k,j)*yeps(k,j)+veta(k,j)*xeps(k,j))
          term2(k,j)=unew(k,j)*(heps(k,j)*yeta(k,j)-heta(k,j)*yeps(k,j))
          term3(k,j)=vnew(k,j)*(-heps(k,j)*xeta(k,j)+heta(k,j)*xeps(k,j))

               ! Include rainrate if transient case
               IF ((answer=='T').OR.(answer=='t')) THEN
               term4(k,j)=-rainrate(counter)
               ELSE
               term4(k,j)=0
               END IF

          hnew(k,j)=-((1/bigJ(k,j))*(term1(k,j)+term2(k,j)+term3(k,j))+term4(k,j))*(timestep)+hold(k,j)

          END IF
     END DO
END DO

END SUBROUTINE WaterLevel
!!!!!!!!!!!!!!!!!!!!! END WATER LEVEL SUBROUTINE !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```