Quasi-3D Nearshore Circulation Model SHORECIRC

Version 2.0

Ib A. Svendsen, Kevin Haas, and Qun Zhao

RESEARCH REPORT NO. CACR-02-01
May 2002

Center for Applied Coastal Research
Ocean Engineering Laboratory
University of Delaware
Newark, DE 19716

Contents

1 ACKNOWLEDGMENTS

The development of the SHORECIRC model has been a longterm effort which started in 1992. The present version of the model is therefore the result of the joint efforts of many people whose contributions we gratefully acknowledge. The first - very preliminary - version was developed by Uday Putrevu in 1992-93 when he worked as a post doctoral research assistant here at the Center for Applied Coastal Research after finishing his PhD degree here. A much improved version was developed by Ap Van Dongeren and almost simultaneously further improvements were introduced by Francisco Sancho, as part of their PhD theses (1997). Since then the present team has introduced many further improvements and also tested the model further with continued valuable input from Uday Putrevu, Ap Van Dongeren, and Francisco Sancho. During this process James Kaihatu's extensive work with the model has lead to many improvements and clarifications. Wenkai Qin and Junwoo Choi have meticulously scrutinized both manual and code for typos, inconsistencies and bugs, and Jeremy Kalmbacher has assisted in analysing new elements for the code. We want to thank them and we also want to thank Bradley Johnson for pointing out a weak point in the code, and Fengyan Shi for his contributions linked to his work with the curvilinear version of the code.

Funding for the work has come from many different sources. A continuous source of funding from the very beginning has been the Delaware Sea Grant program which through a series of two year projects 1991-2001 has provided the backbone of the environment that has made this possible. Further funding has been provided by the Army Research Office (University Research Initiative, contract DAAL 03-92-G-0016), the Office of Naval Research (contracts N00014-95-0011 (UP), and N00014-95-C-0075, N00014-99-1-0075).

2 INTRODUCTION

2.1 Introductory information

About this manual

This manual is intended to serve two purposes. One is to make it possible to use the model without a detailed insight into the theoretical background for the model, the other to make it possible to use the model as a research tool.

All users should find the description of the capabilities and limitations of the model in Chapter 4 and the description in Chapter 5 of how to operate the model particularly useful. These two chapters are meant to contain the information needed for users who mainly want to use the model for obtaining information about nearshore waves and currents.

However, as backup for any practical use of the model, and to help users who want to apply the model as a research tool and maybe develop it further we have also provided a brief outline of the scientific basis for the model including the underlying equations solved or used to compute many of the parameters used in the model. This is primarily done in Chapter 3. We have also given a rather extensive list of references to sources where more information can be obtained.

In order to facilitate the overview the code itself is presented separately at the end in Chapter 8

About the computer code

Traditionally nearshore circulation models have been 2-D horizontal models that assume depth uniform currents. The present model can of course be operated in a 2-D horizontal mode - and new users may want to start in that way.

However, the nearshore wave generated currents are generally not depth uniform and the vertical variation is not just a curiosity that marks improved detailing. The vertical current variation plays an integral role in the way in which the currents are distributed in the horizontal plane because they contribute decisively to the horizontal exchange of momentum known as "lateral mixing".

The numerical model SHORECIRC is a quasi-3D model, which combines the effects of vertical structure of the currents with the simplicity of a 2D-horizontal model for nearshore circulation. This is done by using an analytical solution for the 3D current profiles in combination with a numerical solution for the depth-integrated 2D horizontal equations.

The theoretical background for SHORECIRC is described in Putrevu and Svendsen(1999) which is an extension of Svendsen and Putrevu (1994), and the first version of the model was described by Van Dongeren et al. (1994). The derivation of the model equations is omitted here, but the resulting governing equations used in the model and information about model elements such as the vertical flow structure, boundary conditions, bottom friction, turbulence representation, etc. are given in Chapter 3. This

also includes a brief outline of the numerical solution method. As mentioned the capabilities and limitations of the model are discussed in Chapter 4. User information and examples of input files are shown in Chapter 5, which also includes some test cases that can be used as benchmarks for checking the correct function of the model. Chapter 5.7 gives instructions in how to operate the model.

The SHORECIRC model system essentially consists of two parts:
- A depth integrated, short wave averaged component that determines the nearshore currents and infragravity wave motion, including the vertical variation over the depth of the currents and particle motions in the IG waves.

- A short wave model - called the wave driver - which, in addition to wave heights and direction, determines the short wave forcing for the time and space varying currents and infragravity wave motions. This forcing consists of the distribution of the short wave generated radiation stresses and mass fluxes.

The model predicts the motion in the time domain and hence is in principle capable of describing the effect of random wave motions. See also Chapter 4 for a discussion of model capabilities and limitations

The SHORECIRC has been verified by comparison to data from detailed laboratory experiments (Haas et al., 1998, Haas and Svendsen 2000a,b, 2002) and from the DELILAH field experiment (Svendsen et al, 1997, Van Dongeren et al., 2002), and it has successfully been applied to a wide variety of nearshore phenomena, such as surfbeats (Van Dongeren *et al.*, 1995), longshore currents (Sancho et al., 1995), infragravity waves (Van Dongeren *et al.*1996, 1998), shear waves (Sancho and Svendsen, 1998, Zhao et al., 2002), flows around detached breakwaters (Sancho et al., 1999, Drei et al., 2000) and rip currents (Haas and Svendsen, 1998, 2000, Svendsen and Haas, 1999, 2000, Haas et al., 2002).

3 THEORETICAL BACKGROUND

3.1 Governing equations

In the SHORECIRC, the instantaneous total fluid velocity $u_\alpha(x, y, z, t)$ is split into three components

$$u_\alpha = u'_\alpha + u_{w\alpha} + V_\alpha \tag{3.1}$$

where u'_α is the turbulent velocity component, $u_{w\alpha}$ is the wave component defined so that $\bar{u}_w = 0$ below through level, and V_α is the current velocity, which in general is varying over depth. The overbar denotes short wave averaging and the subscripts α and β denote the directions in a horizontal Cartesian coordinate system.

Fig 1 shows the definitions of the coordinate system and components of velocities used in the following. Thus $\bar{\zeta}$ represents the mean surface elevation and h_o is the still water depth. The local water depth i then determined as

$$h = h_0 + \overline{\zeta} \tag{3.2}$$

Q_α represents the total volume flux which is defined by

$$Q_\alpha = \overline{\int_{-h_o}^{\zeta} u_\alpha \, dz} \tag{3.3}$$

and $Q_{w\alpha}$ is the volume flux due to the short wave motion defined by

$$Q_{w\alpha} = \overline{\int_{-h_o}^{\zeta} u_{w\alpha} \, dz} = \overline{\int_{\zeta_t}^{\zeta} u_{w\alpha} \, dz} \tag{3.4}$$

where ζ_t is the elevation of the wave trough.

Hence we have

$$Q_\alpha = \int_{-h_o}^{\bar{\zeta}} V_\alpha \, dz + Q_{w\alpha} \tag{3.5}$$

The current velocity V_α is divided into a depth uniform part $V_{m\alpha}$ and a depth-varying part $V_{d\alpha}$ by

$$V_\alpha = V_{m\alpha} + V_{d\alpha} \tag{3.6}$$

where $V_{m\alpha}$ is defined by

$$V_{m\alpha} = \frac{Q_\alpha - Q_{w\alpha}}{h_o + \bar{\zeta}} \tag{3.7}$$

This implies that

$$\int_{h_o}^{\bar{\zeta}} V_{d\alpha} \, dz = 0 \tag{3.8}$$

The current V_α is the current that would be measured by a current meter placed below trough level. Notice that the definitions of $V_{m\alpha}$ and $V_{d\alpha}$ differ slightly from the definitions of $\tilde{V}_\alpha$ and $V_{1\alpha}$ used in Putrevu and Svendsen and in some of the earlier publications on the model. This only influences the form of the dispersive mixing coefficients described below, but has been found to have a favorable influence on the robustness of the model.

In the equations below $\tau_{\alpha\beta}$ is the Reynolds stress, while τ_β^S and τ_β^B represent the surface and the bottom shear stress, respectively.

The SHORECIRC model is based on the depth-integrated, time-averaged equations which in complete form and in tensor notation read:

$$\frac{\partial \bar{\zeta}}{\partial t} + \frac{\partial Q_\alpha}{\partial x_\alpha} = 0 \tag{3.9}$$

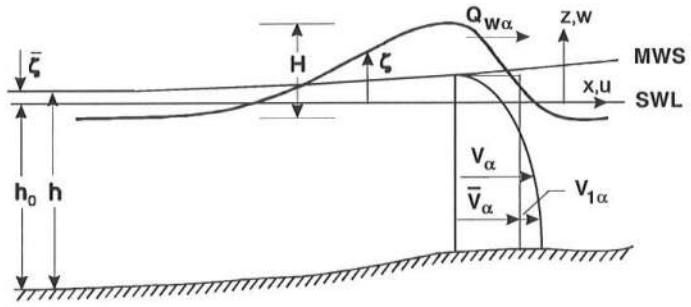

Figure 1: Definition sketch.

$$\frac{\partial Q_\beta}{\partial t} + \frac{\partial}{\partial x_\alpha}\left(\frac{Q_\alpha Q_\beta}{h}\right) + \frac{\partial}{\partial x_\alpha}\int_{-h_0}^{\bar{\zeta}} V_{d\alpha}V_{d\beta}\,dz + \frac{\partial}{\partial x_\alpha}\overline{\int_{\zeta_t}^{\zeta} u_{w\alpha}V_{d\beta} + u_{w\beta}V_{d\alpha}\,dz}$$
$$+ g\,(h_0+\bar{\zeta})\frac{\partial \bar{\zeta}}{\partial x_\beta} - \frac{\tau_\beta^S}{\rho} + \frac{\tau_\beta^B}{\rho} + \frac{1}{\rho}\frac{\partial}{\partial x_\alpha}\left(S_{\alpha\beta} - \overline{\int_{-h_0}^{\zeta}\tau_{\alpha\beta}\,dz}\right) = 0 \qquad (3.10)$$

In this formulation the radiation stress $S_{\alpha\beta}$ is defined as

$$S_{\alpha\beta} \equiv \overline{\int_{-h_o}^{\zeta}(p\,\delta_{\alpha\beta} + \rho\,u_{w\alpha}\,u_{w\beta})\,dz} - \delta_{\alpha\beta}\frac{1}{2}\rho\,g\,h^2 - \rho\,\frac{Q_{w\alpha}Q_{w\beta}}{h} \qquad (3.11)$$

where $u_{w\alpha}$ is the horizontal shortwave-induced velocity. The definition (3.11) is corresponds to the definition for $S_{\alpha\beta}$ used by Phillips (1966, 1977).

3.1.1 Current profiles

In the current-current and current-wave interaction terms, the third and fourth terms, in Eq.(3.10) the current/IG wave velocities are functions of depth and need to be expressed in terms of the depth-averaged quantities in order to calculate the integrals in the quasi 3D equations. In order to do that we need to determine the vertical profiles of the current/IG-wave velocities.

For the derivation of these profiles we use the local (non depth-integrated) time-averaged horizontal momentum equations and introduce the separation (3.7) between the depth uniform current $V_{m\alpha}$, the depth-varying current $V_{d\alpha}$, and the short wave velocities.

In this context $V_{d\alpha}$ is divided into two parts by the definition

$$V_{d\alpha} = V_{d\alpha}^{(0)} + V_{d\alpha}^{(1)} \qquad (3.12)$$

The derivation is too lengthy for this manual. A general version is in Putrevu and Svendsen (1999). The slightly simplified form used in the present version of the model can be found in Van Dongeren and Svendsen (1997a). However, as can be seen from the coefficients in section 3.1.2 below only the $V_{d\alpha}^{(0)}$-component of the velocity is required to account for the values of the current-current and current-wave interaction terms. The vertical variation of this velocity is then given by

$$V_{d\alpha}^{(0)} = d_{1\alpha}\xi^2 + e_{1\alpha}\xi + f_{1\alpha} + f_{2\alpha} \tag{3.13}$$

where

$$\xi = z + h_0 \tag{3.14}$$

and

$$d_{1\alpha} = -\frac{F_\alpha}{2\nu_t} \tag{3.15}$$

$$e_{1\alpha} = \frac{\tau_\alpha^B}{\rho\nu_t} \tag{3.16}$$

$$f_{1\alpha} = -\frac{h}{2}\frac{\tau_\alpha^B}{\rho(\nu_t)} \tag{3.17}$$

$$f_{2\alpha} = \frac{h^2 F_\alpha}{6(\nu_t)} \tag{3.18}$$

$$\tag{3.19}$$

with F_α given by

$$F_\alpha = \left\{ \frac{1}{\rho h}\frac{\partial S'_{\alpha\beta}}{\partial x_\beta} + \frac{\tau_\alpha^B}{\rho h} - f_\alpha \right\}. \tag{3.20}$$

where

$$f_\alpha = \frac{\partial}{\partial x_\alpha}(\overline{u_{w\alpha}u_{w\beta}}) + \frac{\partial}{\partial z}(\overline{u_{w\alpha}w_w}). \tag{3.21}$$

In addition, $V_{d\alpha}^{(1)}$ is given by

$$V_{d\alpha}^{(1)} = V_{d\alpha}^{(1,a)} + V_{d\alpha}^{(1,b)} + V_{d\alpha}^{(1,w)} + V_{d\alpha}^{(1,c)} \tag{3.22}$$

with

$$V_{d\alpha}^{(1,a)} = V_{d\alpha}^{(1,a,4)}\xi^4 + V_{d\alpha}^{(1,a,3)}\xi^3 + V_{d\alpha}^{(1,a,2)}\xi^2 \tag{3.23}$$

$$V_{d\alpha}^{(1,b)} = V_{d\alpha}^{(1,b,4)}\xi^4 + V_{d\alpha}^{(1,b,3)}\xi^3 + V_{d\alpha}^{(1,b,2)}\xi^2 \tag{3.24}$$

$$V_{d\alpha}^{(1,w)} = V_{d\alpha}^{(1,w,4)}\xi^4 + V_{d\alpha}^{(1,w,3)}\xi^3 + V_{d\alpha}^{(1,w,2)}\xi^2 \tag{3.25}$$

and

$$V_{d\alpha}^{(1,c)} = \left[V_{d\alpha}^{(1,a,4)} + V_{d\alpha}^{(1,b,4)} + V_{d\alpha}^{(1,w,4)}\right]\frac{h^4}{5} \left[V_{d\alpha}^{(1,a,3)} + V_{d\alpha}^{(1,b,3)} + V_{d\alpha}^{(1,w,3)}\right]\frac{h^3}{4} \left[V_{d\alpha}^{(1,a,2)} + V_{d\alpha}^{(1,b,2)} + V_{d\alpha}^{(1,w,2)}\right]\frac{h^2}{3}. \tag{3.26}$$

Further details about $V_{d\alpha}^{(1)}$ can be found in Haas and Svendsen (2000).

3.1.2 Dispersive mixing coefficients (3-D version)

From the current/IG-wave velocity profiles the current-current and current-wave interaction terms in (3.10) can be rewritten in terms of a set of coefficients $M_{\alpha\beta}$, $D_{\alpha\gamma}$, $B_{\alpha\beta}$, and $A_{\alpha\beta\gamma}$ which are the 3D dispersion coefficients. As shown by Svendsen and Putrevu (1994) the dispersive mixing represented by these coefficients can account for a major part of the lateral mixing in the nearshore region. These coefficients can all be calculated from the depth varying part $V_{d\alpha}$ of the current velocity profiles. The general definitions of the coefficients are given in Putrevu and Svendsen (1999). The version used here corresponds to the modified version version used by Haas and Svendsen (2000a) which differs in the way the current velocity is split in mean and depth varying parts as shown in (3.6). The 3D-dispersion coefficients are defined by

$$\begin{aligned} A_{\alpha\beta\gamma} &= \\ &-\left\{\int_{-h_o}^{\overline{\zeta}} \frac{1}{(\nu_t)}\left(\int_{-h_o}^{z}\frac{\partial V_{d\alpha}^{(0)}}{\partial x_\gamma} - \frac{\partial \frac{Q_{w\alpha}}{h}}{\partial x_\gamma} - \frac{\partial h_o}{\partial x_\gamma}\frac{\partial V_{d\alpha}^{(0)}}{\partial z}\right)\left(\int_{-h_o}^{z} V_{d\beta}^{(0)} - \frac{Q_{w\beta}}{h}dz'\right)dz \right. \\ &\left. + \int_{-h_o}^{\overline{\zeta}} \frac{1}{(\nu_t)}\left(\int_{-h_o}^{z}\frac{\partial V_{d\beta}^{(0)}}{\partial x_\gamma} - \frac{\partial \frac{Q_{w\beta}}{h}}{\partial x_\gamma} - \frac{\partial h_o}{\partial x_\gamma}\frac{\partial V_{d\beta}^{(0)}}{\partial z}\right)\left(\int_{-h_o}^{z} V_{d\alpha}^{(0)} - \frac{Q_{w\alpha}}{h}dz'\right)dz\right\} \end{aligned} \tag{3.27}$$

$$\begin{aligned} B_{\alpha\beta} &= -\frac{1}{h}\left\{\int_{-h_o}^{\overline{\zeta}} \frac{1}{(\nu_t)}\left(\int_{-h_o}^{z}(h_o+z')\frac{\partial V_{d\alpha}^{(0)}}{\partial z}dz'\right)\left(\int_{-h_o}^{z} V_{d\beta}^{(0)} - \frac{Q_{w\beta}}{h}dz'\right)dz + \right. \\ &\left. \int_{-h_o}^{\overline{\zeta}} \frac{1}{(\nu_t)}\left(\int_{-h_o}^{z}(h_o+z')\frac{\partial V_{d\beta}^{(0)}}{\partial z}dz'\right)\left(\int_{-h_o}^{z} V_{d\alpha}^{(0)} - \frac{Q_{w\alpha}}{h}dz'\right)dz\right\} \end{aligned} \tag{3.28}$$

$$D_{\alpha\beta} = \frac{1}{h}\int_{-h_o}^{\overline{\zeta}} \frac{1}{(\nu_t)}\left(\int_{-h_o}^{z} V_{d\alpha}^{(0)} - \frac{Q_{w\alpha}}{h}dz'\right)\left(\int_{-h_o}^{z} V_{d\beta}^{(0)} - \frac{Q_{w\beta}}{h}dz'\right)dz \tag{3.29}$$

$$M_{\alpha\beta} = \int_{-h_o}^{\overline{\zeta}} V_{d\alpha}^{(0)}V_{d\beta}^{(0)}dz + V_{d\beta}^{(0)}(\overline{\zeta})Q_{w\alpha} + V_{d\alpha}^{(0)}(\overline{\zeta})Q_{w\beta} \tag{3.30}$$

The values of the 3D-dispersion coefficients implemented in the present form of the SHORECIRC have been determined for the simplified case of a depth uniform eddy viscosity ν_t and quasi steady flow. [1]

In the program there are options to use the 3-D dispersion or to do computations in the 2-D horizontal mode with depth uniform currents. Operation with depth uniform currents implies that the dispersive mixing is turned off. The only lateral mixing is then provided by ν_t in (3.49). This will usually provide too little lateral mixing. Depending on the problem under study the values of ν_t in (3.49) may have to be increased substantially (typically by a factor of 10 - 20) to compensate for the missing 3-D mixing. For further discussion see section 3.3.2.

3.1.3 Final form of the basic equations

Using these coefficients the equations can then be written in terms of surface elevations $\overline{\zeta}$ and the total volume flux components Q_x and Q_y as dependent unknowns. The result is

$$\frac{\partial \overline{\zeta}}{\partial t} + \frac{\partial Q_x}{\partial x} + \frac{\partial Q_y}{\partial y} = 0 \tag{3.31}$$

and

$$\begin{aligned}
&\frac{\partial Q_x}{\partial t} + \frac{\partial}{\partial x}\left(\frac{Q_x^2}{h} + M_{xx}\right) + \frac{\partial}{\partial y}\left(\frac{Q_x Q_y}{h} + M_{xy}\right) \\
&-\frac{\partial}{\partial x} h\left[(2D_{xx} + B_{xx})\frac{\partial}{\partial x}\left(\frac{Q_x}{h}\right) + 2D_{xy}\frac{\partial}{\partial y}\left(\frac{Q_x}{h}\right) + B_{xx}\frac{\partial}{\partial y}\left(\frac{Q_y}{h}\right)\right] \\
&-\frac{\partial}{\partial y} h\left[(D_{xy} + B_{xy})\frac{\partial}{\partial x}\left(\frac{Q_x}{h}\right) + D_{yy}\frac{\partial}{\partial y}\left(\frac{Q_x}{h}\right) + D_{xx}\frac{\partial}{\partial x}\left(\frac{Q_y}{h}\right) + (D_{xy} + B_{xy})\frac{\partial}{\partial y}\left(\frac{Q_y}{h}\right)\right] \\
&+\frac{\partial}{\partial x}\left[A_{xxx}\frac{Q_x}{h} + A_{xxy}\frac{Q_y}{h}\right] + \frac{\partial}{\partial y}\left[A_{xyx}\frac{Q_x}{h} + A_{xyy}\frac{Q_y}{h}\right] \\
&= -gh\frac{\partial \overline{\zeta}}{\partial x} - \frac{1}{\rho}\left(\frac{\partial S_{xx}}{\partial x} + \frac{\partial S_{xy}}{\partial y}\right) + \frac{1}{\rho}\left(\frac{\partial}{\partial x}\overline{\int_{-h_0}^{\zeta} \tau_{xx} dz} + \frac{\partial}{\partial y}\overline{\int_{-h_0}^{\zeta} \tau_{xy} dz}\right) \\
&+\frac{\tau_x^S - \tau_x^B}{\rho}
\end{aligned} \tag{3.32}$$

[1] It is emphasized here that although these formulas only contain $V_{d\alpha}^{(0)}$ the full effect of $V_{d\alpha}^{(1)}$ in (3.12) is included because in the above expressions for the dispersive mixing coefficients the contributions involving $V_{d\alpha}^{(1)}$ have been expressed analytically in terms of $V_{d\alpha}^{(0)}$. See Putrevu and Svendsen (1999), Appendix B.

$$\frac{\partial Q_y}{\partial t} + \frac{\partial}{\partial x}(\frac{Q_x Q_y}{h} + M_{xy}) + \frac{\partial}{\partial y}(\frac{Q_y^2}{h} + M_{yy})$$
$$-\frac{\partial}{\partial x}h[(D_{xy} + B_{xy})\frac{\partial}{\partial x}(\frac{Q_x}{h}) + D_{yy}\frac{\partial}{\partial y}(\frac{Q_x}{h}) + D_{xx}\frac{\partial}{\partial x}(\frac{Q_y}{h}) + (D_{xy} + B_{xy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]$$
$$-\frac{\partial}{\partial y}h[B_{yy}\frac{\partial}{\partial x}(\frac{Q_x}{h}) + 2D_{xy}\frac{\partial}{\partial x}(\frac{Q_y}{h}) + (2D_{yy} + B_{yy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]$$
$$+\frac{\partial}{\partial x}[A_{xyx}\frac{Q_x}{h} + A_{xyy}\frac{Q_y}{h}] + \frac{\partial}{\partial y}[A_{yyx}\frac{Q_x}{h} + A_{yyy}\frac{Q_y}{h}]$$
$$= -gh\frac{\partial \bar{\zeta}}{\partial y} - \frac{1}{\rho}(\frac{\partial S_{xy}}{\partial x} + \frac{\partial S_{yy}}{\partial y}) + \frac{1}{\rho}(\frac{\partial}{\partial x}\overline{\int_{-h_0}^{\zeta} \tau_{xy} dz} + \frac{\partial}{\partial y}\overline{\int_{-h_0}^{\zeta} \tau_{yy} dz})$$
$$+\frac{\tau_y^S - \tau_y^B}{\rho} \quad (3.33)$$

In the above, $\tau_\alpha^B, \tau_\alpha$, are the bottom shear stresses, the surface (wind) shear stresses, and $S_{\alpha\beta}, \tau_{\alpha\beta}$ are the radiation stresses and the turbulent Reynold's stresses, respectively.

The equations (3.31), (3.32) and (3.33) are the equations solved by the SHORECIRC model.

As mentioned the values of ν_t and τ_β^B used in the model are specified in sections 3.5 and 3.3, respectively.

3.2 Short wave quantities

3.2.1 The wave driver

The short wave driver determines the short wave motion and calculates the short averaged forcing that drives the currents and IG waves. There are two options for modeling the short waves: using the built in wave driver, or using a separate wave driver and specifying the wave input via data files. Additional information about what data files to include is given in section 5.4

In the present version of the SHORECIRC model, an improved version of the REF/DIF1 (Kirby and Dalrymple, 1994) is incorporated as the short wave driver. The REF/DIF package is included as a subroutine within the code. However, it only provides the wave heights, wave angles, wave celerities and group velocities. The associated short wave forcing is calculated in a separate subroutine *shortwave_driver* which also calls the REF/DIF.

The REF/DIF wave driver is based on the parabolic approximation of the mild-slope equation, and accounts for refraction, diffraction, shoaling and breaking phenomena. The present version uses a single offshore wave height and direction, although it can represent the peak in a spectrum.

Wave/current interaction is important for many applications with strong currents such as rip currents. It is modelled by periodically recalculating the wave forcing including the effects of currents and setup. The intervals between each recalculation of the wave conditions can be chosen in the input for the calculation.

The wave-current interaction is included as an option in the code when the build-in wave driver is used. When using a separate wave driver via the data files wave-current interaction is only possible by doing a hot start with the newly calculated waves each time the wave conditions have been updated.

3.2.2 The short wave forcing

As mentioned the short wave forcing represents the time-averaged contributions to the mass and momentum equations generated by the short wave motion. These contributions are the short wave mass flux $Q_{w\alpha}$ and the radiation stresses $S_{\alpha\beta}$.

Radiation Stress

Using the results of wave heights and wave angles calculated by the short-wave-driver the radiation stresses are calculated. The generalized radiation stresses tensor is given by

$$S_{\alpha\beta} = e_{\alpha\beta} S_m + \delta_{\alpha\beta} S_p \tag{3.34}$$

where

$$e_{\alpha\beta} = \begin{bmatrix} \cos^2 \alpha_w & \sin \alpha_w \cos \alpha_w \\ \sin \alpha_w \cos \alpha_w & \sin^2 \alpha_w \end{bmatrix} \tag{3.35}$$

and α_w is the wave angle relative to the positive x axis. The scalars S_m and S_p are defined as

$$S_m = \overline{\int_{-h_0}^{\zeta} \rho {u_w}^2 dz} \tag{3.36}$$

$$S_p = \overline{-\int_{-h_0}^{\zeta} \rho {w_w}^2 dz} + \frac{1}{2} \rho g \overline{\eta^2} \tag{3.37}$$

where

$$u_w = \mid u_{w\alpha} \mid \tag{3.38}$$

is the wave particle velocity in the wave direction.

In the calculation of $S_{\alpha\beta}$ the model uses the local wave heights determined by the wave driver (presently REF/DIF1). Outside the surf zone the results for sinusoidal waves are used. For sinusoidal waves we have

$$S_m = \frac{1}{16} \rho \, g \, H^2 (1 + G) \tag{3.39}$$

$$S_p = \frac{1}{16} \rho \, g \, H^2 G \tag{3.40}$$

where G is

$$G = \frac{2kh}{\sinh 2kh} \tag{3.41}$$

Inside the surfzone those results are augmented with the contribution from the roller using the results from Svendsen (1984). Thus S_m and S_p are determined as

$$S_m = \rho g H^2 \frac{c^2}{gh}[B_0 + \frac{A}{HL}\frac{h}{H}] = \rho g H^2 \frac{c^2}{gh}[B_0 + \frac{A}{H^2}\frac{h}{L}] \tag{3.42}$$

and

$$S_p = \frac{1}{2}\rho g H^2 B_0 \tag{3.43}$$

where A is the roller area and B_0 is the wave shape parameter defined by

$$B_0 = \frac{1}{T}\int_0^T (\frac{\eta}{H})^2 \, dt \tag{3.44}$$

Thus $S_{\alpha\beta}$ is computed as

$$S_{\alpha\beta} = e_{\alpha\beta}\rho g H^2 \frac{c^2}{gh}[B_0 + \frac{A}{H^2}\frac{h}{L}] + \delta_{\alpha\beta}\frac{1}{2}\rho g H^2 B_0 \tag{3.45}$$

A transition is established between the two expressions for $S_{\alpha\beta}$ to simulate the growth of the roller when breaking starts.

Parameterizations for B_0 can be found in Hansen (1990). For sinusoidal long waves the value of B_0 is $\frac{1}{8}$. In the present version of the model this is used as the default value.

Notice that L is calculated as cT where c is the local value of the phase velocity.

Wave Volume Flux

Outside the surf zone, the wave volume flux is given by

$$Q_{w\alpha} = B_0 \frac{gH^2}{c}\frac{k_\alpha}{k} \tag{3.46}$$

where k_α is the wave number vector in direction x_α, $k_\alpha = k(cos\alpha_w, sin\alpha_w)$ and c is the value of the phase velocity.

Inside the surf zone, the wave volume flux is given by (Svendsen 1984)

$$Q_{w\alpha} = \frac{gH^2}{c}\frac{c^2}{gh}(B_0 + \frac{A}{HL}\frac{h}{H})\frac{k_\alpha}{k} = \frac{gH^2}{c}\frac{c^2}{gh}(B_0 + \frac{A}{H^2}\frac{h}{L})\frac{k_\alpha}{k} \tag{3.47}$$

As for the radiation stress a transition is established between the two expressions for $Q_{w\alpha}$ to simulate the growth of the roller when breaking starts.

In the expressions for both the radiation stress and the volume flux the roller area A may be approximated by either $0.9H^2$ (Svendsen, 1984) or by $0.06HL$ (Okayasu et al., 1986). The latter is used as default in the present version of the program.

3.3 Turbulence modelling

3.3.1 The quasi-3D version

In the quasi-3D version of the model the turbulent shear stresses are calculated by the eddy viscosity model, as

$$\tau_{\alpha\beta} = \rho\nu_t(\frac{\partial V_{m\alpha}}{\partial x_\beta} + \frac{\partial V_{m\beta}}{\partial x_\alpha}) \tag{3.48}$$

(3.48) is depth uniform. However, since the contribution from $\tau_{\alpha\beta}$ is generally small this it is a reasonable approximation.

The eddy viscosity formulation accounts for both wave-breaking and bottom generated turbulence. Outside the surf zone, the model is based on Svendsen and Putrevu (1994) and Coffey and Nielsen (1984). Inside the surf zone, a modified Battjes (1975) model is applied. The combined contributions to the eddy viscosity from these sources is calculated as

$$\nu_t = C_1\kappa\sqrt{\frac{f_w}{2}}u_0 h + Mh(\frac{D}{\rho})^{1/3} + \nu_{t,0} + \nu_s \tag{3.49}$$

where κ is the von Karman constant ($\kappa \simeq 0.4$), f_w is the wave related bottom friction coefficient (sect 3.5). u_0 is the short-wave particle velocity amplitude evaluated at the bottom, and D is the energy dissipation per unit area given by the formulation used in the REF/DIF1. The first term in equation (3.49) represents the bottom induced turbulence which is always present and the second term represents the wave breaking induced turbulence which is only present in the surf zone. By comparing the eddy viscosity estimates from this equation with the experimental results of Nadaoka and Kondoh(1982) and the values suggested by Svendsen(1987), $C_1 \simeq 0.2$. For M values between 0.05 wnd 0.1 are recommended. As a default the value $M = 0.08$ should be used. The constant $\nu_{t,0}$ is an empirical measure of the background eddy viscosity found offshore.

The subgrid stresses are modelled by using the Smagorinsky eddy viscosity model (Smagorinsky 1963). The Smagorinsky eddy viscosity models the turbulence generated by the shear in the flow and accounts for the dissipation by the eddies which are too small to be resolved by the grid resolution.

The following equation is used to calculate the Smagorinksy eddy viscosity

$$\nu_s = (C_s\Delta)^2\sqrt{2e_{\alpha\beta}e_{\alpha\beta}} \tag{3.50}$$

where

$$\Delta = \sqrt{\Delta x\Delta y} \tag{3.51}$$

$$E_{\alpha\beta} = \frac{1}{2}\left(\frac{\partial U_{m\alpha}}{\partial x_\beta} + \frac{\partial U_{m\beta}}{\partial x_\alpha}\right) \tag{3.52}$$

C_s is the Smagorinksy coefficient and has a typical value of $0.05 < C_s < 0.25$.

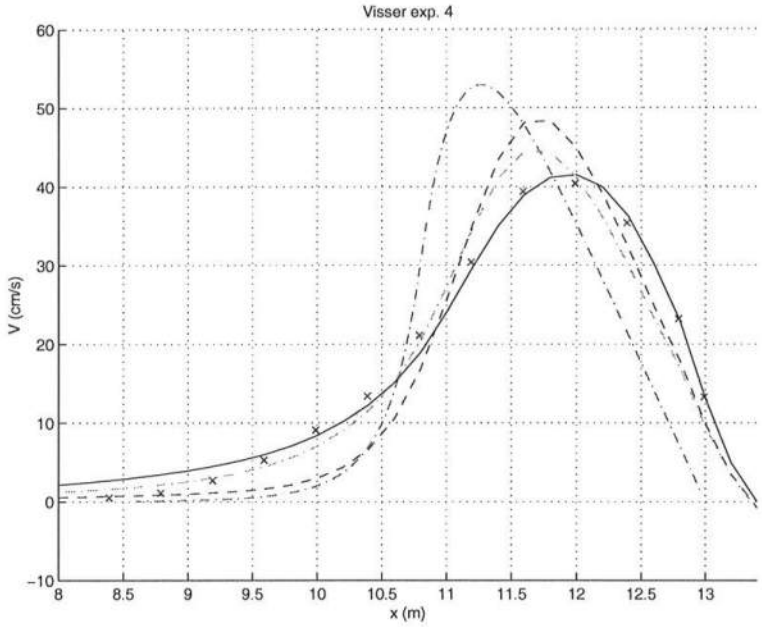

Figure 2: Cross-shore variation of the longshore current from Visser (1984): data (x), quasi-3D SHORECIRC (solid line), 2-D SHORECIRC (light dashed dot line), traditional 2-D model (dashed line) and Longuet-Higgins (1970) (dark dashed dot line). All models use $f_{cw} = 0.012$, $M = 0.07$ and $C_1 = 0.2$.

While ν_t is of relatively limited direct importance through (3.48) the major function of ν_t is through its influence on the vertical current profiles and thereby the expressions for the dispersive lateral mixing coefficients given by (3.27),(3.28),(3.29), and (3.30).

3.3.2 The 2-D model version

If the quasi-3D option is turned off the model will operate as a 2-D model. The 2-D version of the model is enhanced relative to traditional 2-D models because of the inclusion of $\frac{Q_{w\alpha}Q_{w\beta}}{h}$ in the modified radiation stress given by 3.11. This extra term enhances the lateral mixing of a traditional 2-D model. Figure (2) demonstrates the effect of the quasi-3D mixing, the enhanced 2-D contained in SHORECIRC, the traditional 2-D and the Longuet-Higgins (1970) analytical solution (hereafter referred to as L-H) by showing the cross-shore variation of the longshore current from the laboratory experiment by Visser (1984). Clearly the quasi-3D version fits the data the best. The 2-D versions have large peaks as well as the wrong shape for the cross-shore profile of the longshore current. However, the additional mixing contained by the 2-D SHORECIRC is evident by the change in the peak and width of the cross-shore profile of the longshore current relative to the traditional 2-D version. The difference between the L-H and traditional 2-D is due to the simple wave forcing based on a saturated breaker utilzed by L-H.

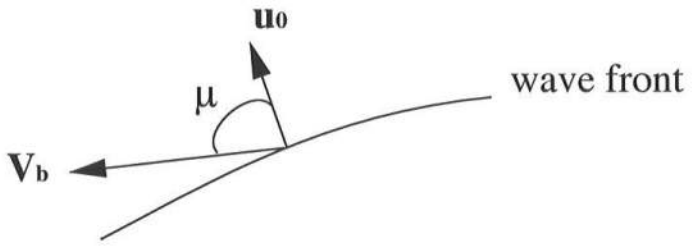

Figure 3: Definition sketch for calculation of bottom shear stress in combined wave-current motion.

For most practical applications, when only using the 2-D version of the model, the eddy viscosity needs to be enhanced substantially to compensate for the missing dispersive mixing in order to give reasonable results. In doing so it should be remembered that in nature the lateral mixing can vary substantially depending on the situation. While this will be (at least partly) accounted for automatically by the dispersive mixing it needs be included in the specified values for ν_t when running the model in the 2-D version.

For the simple case of a uniform longshore current on a long straight coast Svendsen and Putrevu (1994) found that the 2-D eddy viscosity should be of the order 20 times larger than the quasi-3D eddy viscosity relative to the L-H result to get the same cross-shore distribution of the longshore current. As shown in the example above the different wave forcing and different built in eddy viscosity in the present version of the SC will reduce the difference between 2-D and 3-D results and hence require a smaller increase in M in the simple case of longshore uniform longshore currents. For more general cases the effect could be both weaker and stronger. The reader is referred to the literature for suitable formulations of the eddy viscosity variation in 2-D computations.

In addition to affecting the mean currents, the dispersive mixing strongly affects the stability of the flow. Zhao *et al.* (2002) and Haas *et al.* (2002) found that the 3D dispersion reduces the instabilities (such as shear waves and fluctuations of rip currents) in the flow substantially . This is an effect that is completly missing in 2-D models.

3.4 Bottom topography

In the present version the SHORECIRC program reads the topography in subroutine *topography*. The details for this are in the User's Guide.

3.5 Bottom friction

The wave averaged bottom shear stress $\overline{\tau_\alpha^B}$ is evaluated under the assumption that the instantaneous bottom shear stress $\tau_\alpha^B(t)$ can be written

$$\tau_\alpha{}^B(t) = \frac{1}{2}\rho f_{cw}(u_{0,\alpha}(t) + V_{b,\alpha})|(u_{0,\alpha}(t) + V_{b,\alpha}| \tag{3.53}$$

where $u_{0,\alpha}(t)$ is the bottom velocity in the wave motion, V_b is the bottom velocity in the current motion ($= V_\alpha$ when depth uniform currents are used), f_{cw} is the bottom friction factor which can vary with space, but is considered constant in time. We also have $u_{0\alpha} = (u_{0x}, u_{0y})$, $V_{b\alpha} = (V_{bx}, V_{by})$.

After short wave averaging this expression for $\overline{\tau_\alpha^B}$ can be written (Svendsen and Putrevu, 1990),

$$\overline{\tau_\alpha^B} = \frac{1}{2}\rho f_{cw} u_0(\beta_1 V_{b\alpha} + \beta_2 u_{0\alpha}). \tag{3.54}$$

For sinusoidal waves, with

$$\mid u_{0,\alpha} \mid = u_0 \, cos\theta \tag{3.55}$$

with θ the phase angle in the wave motion, the weight factors for the current and wave motion β_1 and β_2 are, respectively.

$$\beta_1 = \frac{}{[(\frac{V_b}{u_0})^2 + 2\frac{V_b}{u_0}\cos\theta\cos\mu + \cos^2\theta]^{1/2}} \tag{3.56}$$

$$\beta_2 = \frac{}{\cos\theta[(\frac{V_b}{u_0})^2 + 2\frac{V_b}{u_0}\cos\theta\cos\mu + \cos^2\theta]^{1/2}} \tag{3.57}$$

In the above equations, θ is the short-wave phase angle, $\theta = \omega t - \int \vec{k}\cdot d\vec{x}$, and μ is the angle between the short-wave direction and the current velocity at the bottom. For definitions see figure 3. We have also defined

$$V_b = |V_{b,\alpha}| \tag{3.58}$$

The corresponding variations of β_1 and β_2 versus V_b/u_0 and μ are shown in figure 4 and figure 5.

In the code the values of β_1 and β_2 are approximated by simple curve fits to (3.57) and (3.57).

Steady-streaming

It has been found (see Putrevu and Svendsen, 1995) that particularly outside the surf zone the steady streaming induced in the oscillatory bottom boundary layer is important for the proper modelling of the undertow. In the momentum equation this is represented by the $\overline{u_w w_w}$-stress which is modelled here by the expression found by Longuet-Higgins, (1956).

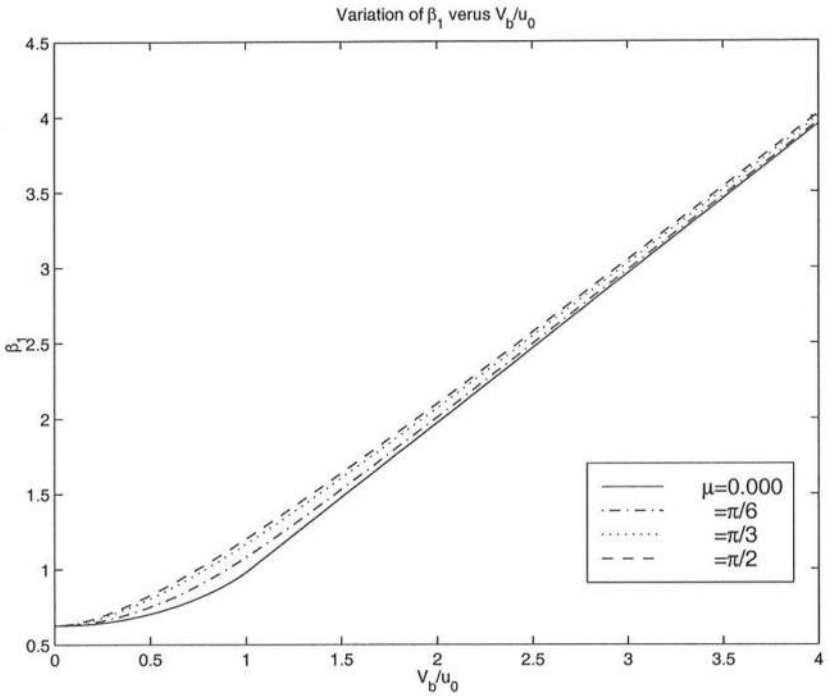

Figure 4: Variation of β_1 versus V_b/u_0.

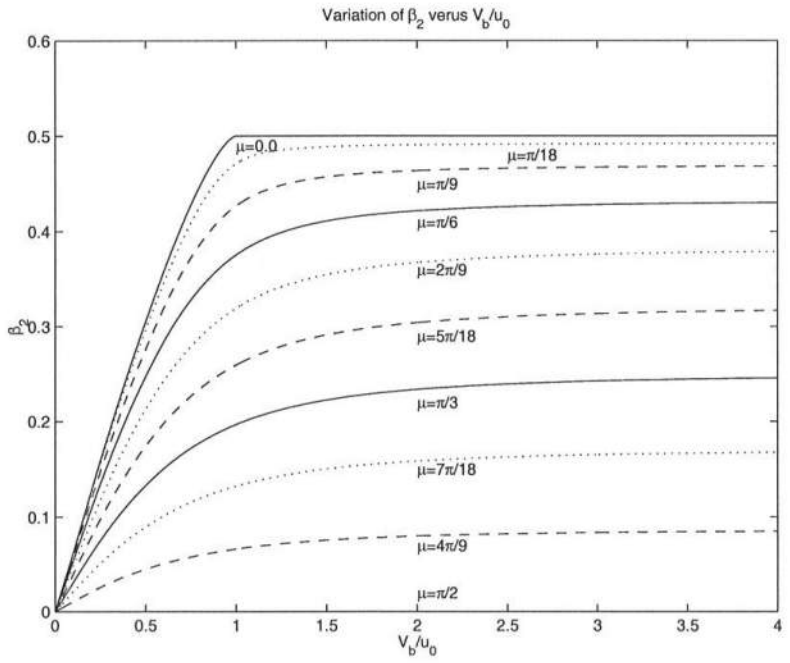

Figure 5: Variation of β_2 versus V_b/u_0.

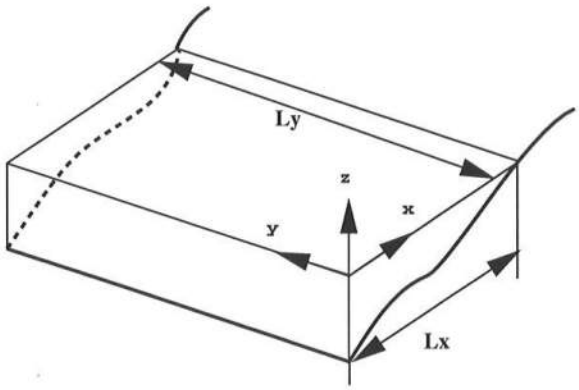

Figure 6: Typical model domain.

3.6 Boundary conditions

The equations are typically solved in a rectangular domain of the coastal region. See Fig. 6 for a sketch of a typical model domain. Boundary conditions therefore need to be specified along three different types of boundaries:

- Seaward boundaries.
- Cross-shore boundaries.
- Shoreline boundaries.

In the computations it is assumed that waves are incident through both the seaward and the cross-shore boundaries.

Type of boundary condition
In the model several types of boundary conditions can be chosen depending on the problem. The choice of boundary conditions is controlled by the value of the control parameters $ibc1, ibc2, ibc3$ and $ibc4$. For details see Section 5.5.1.

3.6.1 Offshore boundaries

At the seaward boundary, an open boundary condition needs to be specified that can generate incoming (long) waves and currents and at the same time allow the outgoing waves to leave the calculation domain with minimal reflection .

This kind of (absorbing-generating) boundary condition is available in SHORECIRC. The detailed description of this approach is shown in Van Dongeren and Svendsen (1997b). The problem is to establish a method for updating the total velocity and surface elevation at the boundary to the next time level when only the incoming (long) wave motion is specified at that level. For this purpose the governing equations are locally approximated by the shallow water equations and are resolved in terms of in-

and outgoing Riemann variables. Local superposition of incoming and outgoing (long) waves is assumed, and the relationship between the volume flux and the surface elevation of the (long) waves is used to determine the (outgoing) reflected waves at the next time step. This is then added to the (specified) incoming wave motion to update in time the total wave motion at the boundary.

3.6.2 Cross-shore boundaries

In the present version of the model, there are several ways of specifying the lateral boundary conditions which represent the conditions along the upstream and downstream cross-shore boundaries (in the sense of the dominating longshore current). The following options are available:

- A flux boundary condition can be specified . The volume flux Q_y will then need to be prescribed at all points of the cross-shore boundary.
- A wall along the boundary can be prescribed. In this case both Q_y and $\frac{\partial \overline{\zeta}}{\partial y}$ are automatically set to zero along the cross-shore boundary.
- A periodic boundary condition can be used. This means that the instantaneous flow at each point of one of the cross-shore boundaries is mirrored at the equivalent point of the other cross-shore boundary.

Notice that the periodicity condition only makes sense on a coast where the cross-shore profile is the same at the two cross-shore boundaries.

3.6.3 Shoreline boundary

There are presently two options for modeling the shoreline. One option is to use a no-flux condition which follows the still water shoreline position. The boundary location is determined within the program based on the initial conditions and is then held fixed throughout the entire computation. Both the cross-shore and longshore volume fluxes are set to zero at the shoreline.

The other option is to place a vertical wall at a very small depth (a few cm) along the shoreline. Only the cross-shore volume flux is set to zero, no constraint is required on Q_y and in general the model computes $Q_y \neq 0$ along the shoreline. Analysis shows that this usually does not influence the circulation pattern noticeably. However, the option of an artificial shelf at the shoreline is not recommended because this may disturb the nearshore flow significantly.

3.7 Wind surface stress

The wind-induced surface stress is computed as (Church and Thornton 1993, Smith *et al.*, 1993)

$$\tau_\alpha^S = C_D \rho_\alpha |W| W_\alpha \tag{3.59}$$

where C_D is the drag coefficient, ρ_α is the air density and W is the wind velocity at the standard 10m elevation. The wind drag coefficient C_D is calculated from the formula recommended by the WAMDI group (1988):

$$C_D \simeq \begin{cases} 1.2875 \times 10^{-3} & U < 7.5m/s \\ (0.8 + 0.065U) \times 10^{-3} & U \geq 7.5m/s \end{cases} \tag{3.60}$$

3.8 Numerical solution scheme

The system of governing equations (3.31)(3.32)(3.33) are solved using Predictor-Corrector scheme originally developed for Boussinesq equations by Wei and Kirby (1995). This is a central finite difference scheme on a fixed spatial grid which is implemented with an explicit third-order Adams-Bashforth predictor and a third-order Adams-Moulton corrector time-stepping scheme. In space the difference scheme is fourth order in the grid size, except for diffusion terms (terms with $\tau_{\alpha\beta}$ and the 3-D dispersive mixing terms) which are second order in space. For details see Sancho and Svendsen, (1997). In order to solve the system, Eqs.(3.31)(3.32) and (3.33) are rewritten so that only the local acceleration appears on the left-hand side

$$\frac{\partial \mathbf{E}}{\partial t} = \mathbf{F} \tag{3.61}$$

where $\mathbf{E}$ is the vector quantity given by $\mathbf{E}= [\bar{\zeta}, Q_x, Q_y]^T$, and $\mathbf{F}$ is the vector corresponding to the right hand side of the continuity and momentum equations (3.31), (3.32), and (3.33).

In the predictor step, Eqs.(3.61) are approximated by the Adams-Bashforth scheme

$$\mathbf{E^*_{i,j}} = \mathbf{E^n_{i,j}} + \mathbf{\Delta t}\alpha_0(\alpha_1 \mathbf{F^n_{i,j}} + \alpha_2 \mathbf{F^{n-1}_{i,j}} + \alpha_3 \mathbf{F^{n-2}_{i,j}}) + \mathbf{O(\Delta t^3)} \tag{3.62}$$

where

$$\alpha_0 = 1/12 \qquad \alpha_1 = 23 \qquad \alpha_2 = -16 \qquad \alpha_3 = 5 \tag{3.63}$$

At the corrector step, the Adams-Bashforth-Moulton method reads

$$\mathbf{E^{n+1}_{i,j}} = \mathbf{E^n_{i,j}} + \mathbf{\Delta t}\beta_0(\mathbf{a_1 F^*_{i,j}} + \mathbf{a_2 F^n_{i,j}} + \beta_3 \mathbf{F^{n-1}_{i,j}}) + \mathbf{O(\Delta t^3)} \tag{3.64}$$

and

$$\beta_0 = 1/12 \qquad a_1 = 5 \qquad a_2 = 8 \qquad \beta_3 = -1 \tag{3.65}$$

The grid sizes in the x and y-directions ($\Delta x, \Delta y$ respectively) can be different but are assumed to be constant over the computational region.

Filters

Filtering of the solution is used to avoid unreal high-frequency disturbances to develop. The filters implemented in the code are high order

filters which do not create noticeable numerical dissipation.

4 CAPABILITIES AND LIMITATIONS OF THE SHORECIRC

The circulation part

The circulation part of the SHORECIRC solves the depth integrated continuity and momentum equations, thereby providing information about the total depth integrated volume fluxes $\overline{Q_x}, \overline{Q_y}$ and the surface elevations $\overline{\zeta}$. The vertical variation of the current velocities (in magnitude and direction) are calculated as well in the process and the effect of this variation is accounted for in the determination of the solutions for $\overline{Q_x}, \overline{Q_y}$, and $\overline{\zeta}$ through the dispersive mixing coefficients.

The derivation of these equations have required only the following assumptions:

- it is assumed that the pressure in the current and infragravity wave motion is hydrostatic.
- it is assumed that it is possible to do averaging over a wave period, if needed as a moving average.

These restrictions are very mild and the basic circulation equations solved can therefore in general be considered very accurate. Inaccuracies are mainly associated with the way the individual terms are approximated.

The computations of the nearshore circulation takes place in the time domain and it often shows that, even when the forcing is constant or slowly varying, the flow patterns can be highly unsteady (shear waves, time-varying rip currents are examples).

Essential limitations on the model accuracy are linked to the approximations used for the turbulent stresses:

- The turbulence is represented only by a relatively simple model. As a consequence the eddy viscosity governing the vertical profiles in the wave averaged motion is constant over depth.
- The bottom friction is only represented by a simple friction factor model.
- The flow in the bottom boundary layer is not resolved. (For further description see 3.5).

The wave driver

The computations of the nearshore circulation is based on the distribution of the short wave generated mass flux and radiation stresses which are determined from the wave driver. At present the REF/DIFF1 is used as wave driver. This is imposes the following limitations on the model system:

- The short wave motion must have one well defined frequency and a wave height which is constant in time. However, these quantities can be the peak values of a spectrum.
- The short wave motion is supposed to have a dominating direction (parabolic approximation). However the parabolic model used in REF/DIF1 is a "wide angle

model" which allows substantial variations of the wave direction over the model domain.

- Reflection of the short wave motion from steep slopes and obstacles cannot be represented (mild slope assumption).

The numerical scheme

The numerical scheme is of high order which allows relatively large grid-spacings, and it is normally quite robust. Though numerical instabilities may occur, they are usually connected with unrealistically strong variations over the vertical for the velocity profiles. A possible remedy is to increase the eddy viscosity.

However, the present version of the model has several numerical constraints:

- It is based on a rectangular grid in the horizontal plane in the direction of the chosen coordinate directions.

- The grid size is constant in a coordinate direction (but can differ in the two directions). Too large grid sizes can limit the resolution near the shoreline.

the

Work is ongoing to lift some of these restrictions.

5 USER GUIDE

5.1 Introduction

This section will give a brief overview of the *SHORECIRC* model structure and introduce the procedure for using the model. The compiling instructions, input files and data files are all explained. Also, the operating instructions and model output are presented.

5.2 SHORECIRC revision history

5.2.1 Original version of the model

This was the version used in the analysis of Van Dongeren et al. (1994)

- Simplified dispersive mixing
- Analytical forcing
- Absorbing/generating boundary condition.
- Numerical scheme of $O(\Delta t^3, \Delta x^2)$

5.2.2 Major changes appearing in version 1.1

This was the version described in Van Dongeren & Svendsen (1997a)

- First version with quasi-3D effects included.

5.2.3 Major changes appearing in version 1.2

This was the version described in Sancho and Svendsen (1997)

- Numerical scheme of $O(\Delta t^3, \Delta x^4)$
- REF/DIF forcing
- Basis for first standardized version

5.2.4 Major changes appearing in version 1.3

This is the model version used till the end of 2001 and described in Van Dongeren and Svendsen (2000).

- Upgrade of quasi-3D terms to form described in this manual
- Wave/current interaction included.
- Created more user friendly environment with many optional features

5.2.5 Major changes appearing in version 2.0

This is essentially the version described in Haas and Svendsen (2000a).

- Changed definition of the depth averaged velocity from $\tilde{V}$ to V_m and V_1 to V_d.
- Equivalent changes in the dispersive mixing coefficients.
- Many additional small improvements and bug-removals
- Added no-flux boundary condition following the still water line

5.3 Overview of model and flow chart.

The Shorecirc package consists of the wave and circulation model as well as programs for creating the topography and the input control files. Notice that the wave driver included is based on the REF/DIF. The REF/DIF used is based on the the Modified REF/DIF, Version 2.5, which has been modified further to serve our needs.

The Shorecirc package is contained in the archive file *sc2_0.tar*,
In order to extract the files use the command

```
tar -xvf sc2_0.tar
```

This will extract 9 Fortran files, 3 Fortran "include" files and 1 compiling file.

Seven of the Fortran files and the three "include" files comprise the program code for the model:

winc_std_main.f,
winc_std_deriv.f,
winc_std_time.f,
winc_std_filter.f,
winc_std_sub1.f,
infile1.f,
refdif2v25a.f
as well as three included files
winc_std_common.inc
param.h.
pass.inc.

One of the two additional Fortran files is a program element

create_bath.f,

which is included for creating certain simple topographies. Running this file will interactively guide the user through the generation. This program element can create

- a plane beach of any slope,
- a beach with a plateau in front
- or an equilibrium type beach with a profile given by Ax^n.

It can add a bar, by specifying its height, width and position, and place any number of channels in the bar, specifying their width and location.

To input a measured topography users need to create their own files. Note that the topography data must provide information about the water depth at all grid points. The program will read columns corresponding to x, y, h values.

The other Fortran file,

create_input.f

is used to create the user controlled input (other than topography) for operating the model.

The compiling file

Makefile.

produces the compilation operation and linking of the executable files.

The main program and its subroutines.

The central element of the model is the main program called *main.* The function of this part is to call all the relevant subroutines. Most of these routines as well as the *main* itself are all included in the file called *winc_std_main.f.* These routines primarily establish the background for the computation and thus prepares for the integration in time that constitutes the major part of a model run.

The structure of this part of the program is given in Figure 7. The flowchart shows the subroutines called by *main* in the order that they are called as well as any other subroutines that are called by the subroutines.

The last routine which is called by *main* is the routine *time_model*, located in the file *winc_std_time.f*, which performs the time integration. During this process is also called a series of additional subroutines. These routines are contained in the other program files listed above. The structure of the subroutine *time_model* is shown in the flowchart Figure 8. The flowchart shows the order in which the calculations are performed.

Note that most pertinent results are calculated during the time integration. Therefore most of the output files are generated during the time computation within the *time_model* subroutine.

List of program subroutines called from *main.*

The following is a list of the subroutines called from *main.* It also identifies the files in which they are located.

- *main.* The main program which calls the following routines. Contained in *winc_std_main.f.*
- *constantvals.* This defines the physical and mathematical constants. Contained in *winc_std_main.f.*
- *inputvals.* The input file is read from this subroutine. Contained in *winc_std_main.f.*
- *topography.* This specifies the topography, either from a file or analytically. Contained in *winc_std_main.f.*
- *numerical.* This sets up the numerical parameters. Contained in *winc_std_main.f.*
- *init_output.* This initializes the output files. Contained in *winc_std_main.f.*
- *refract_longwaves.* Calculates the angle of incidence of long waves entering the computational domain from outside. Contained in *winc_std_main.f.*

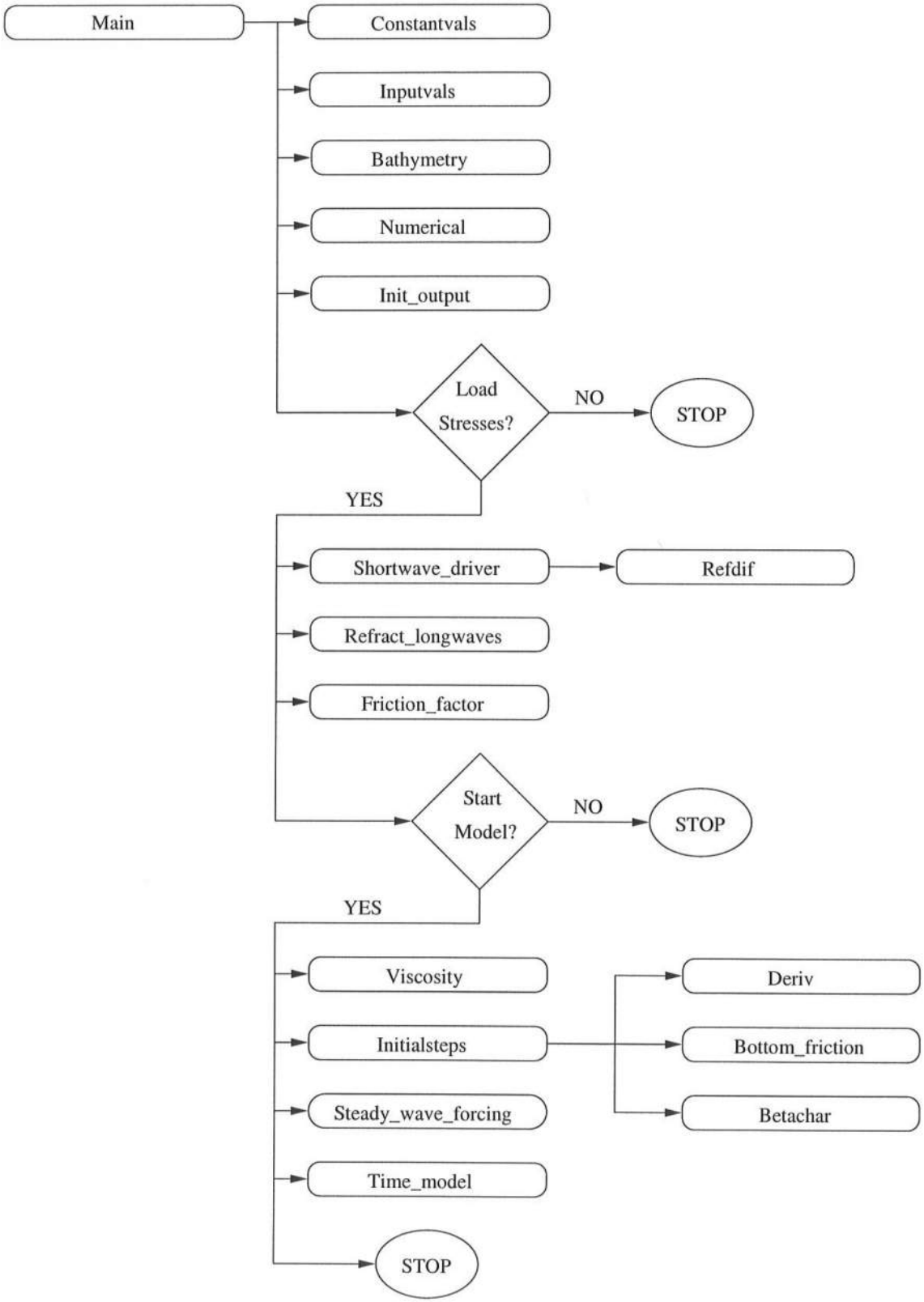

Figure 7: Flowchart of the program structure.

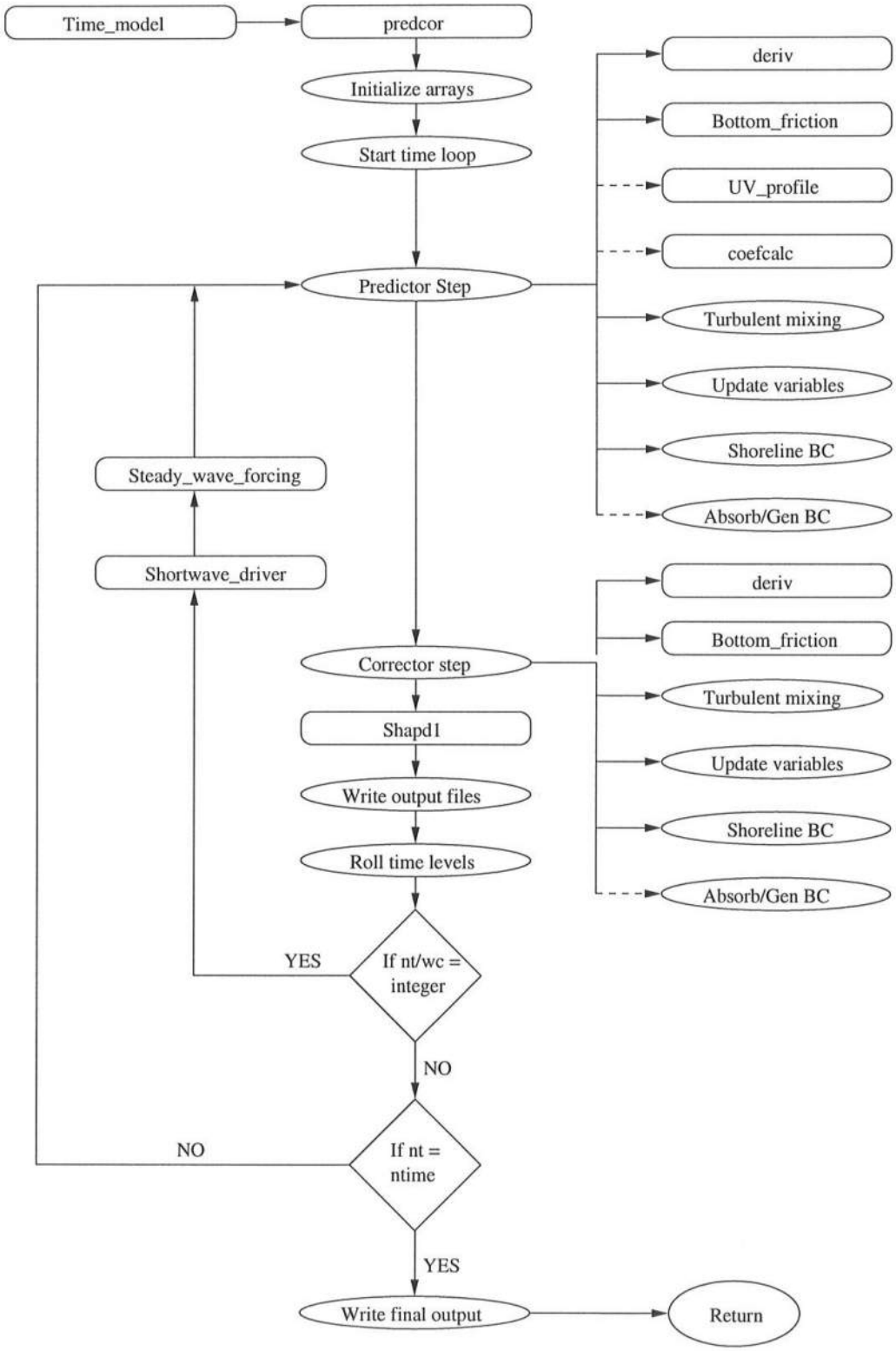

Figure 8: Flowchart of the subroutine *time_model*. Rectangles represent subroutines and ovals represent calculations within *time_model*. Dashed lines represent optional calculations.

- *shortwave_driver.* This routine calls the short wave driver (REF/DIF) and calculates the short-wave parameters. This includes the radiation stress $S_{\alpha,\beta}$ and the short wave induced mass flux, $Q_{w\alpha}$. Contained in *winc_std_main.f.*

- *friction_factor.* This calculates a spatially varying bottom friction factor. It can easily be substituted by a constant friction factor. Contained in *winc_std_main.f.*

- *viscosity.* This evaluates the turbulent eddy viscosity coefficient, ν_t. Contained in *winc_std_main.f.*

- *initialsteps.* In this subroutine the arrays for timesteps $t = -2\Delta t, -\Delta t, 0$ are filled. Contained in *winc_std_main.f.*

- *wave_forcing.* The subroutine which is used to calculate the terms for the velocity profile using the shortwave parameters. Contained in *winc_std_sub1.f.*

- *refdif.* The subroutine containing the REF/DIF model, called in the subroutine *shortwave_driver.* Contained in *refdif2v25a.f.*

The time-integration component and its subroutines

This section lists the routines used in the time integration and also identifies the files in which they can be found.

- *time_model.* This is the bulk of the program where the time loop is performed. The rest of the subroutines are called by this subroutine. Contained in *winc_std_time.f.*

- *betachar.* Calculates the beta characteristics for the absorbing-generating boundary condition. Contained in *winc_std_main.f.*

- *bottom_friction.* Calculates the bottom friction for the momentum equations. Contained in *winc_std_main.f.*

- *average_3ptx.* Performs a 3-point average in the x-direction for every row in the y-direction. Contained in *winc_std_main.f.*

- *average_3pty.* Performs a 3-point average in the y-direction for every row in the x-direction. Contained in *winc_std_main.f.*

- *UV_profile.* This subroutine calculates the longshore and cross-shore vertical velocity profiles. Contained in *winc_std_sub1.f.*

- *coefcalc.* Calculates the 3-D dispersion coefficients. Contained in *winc_std_sub1.f.*

- *shapd1.* This subroutine is the Shapiro filter called from several of the subroutines. Contained in *winc_std_filter.f.*

- *predcor.* This subroutine calculates the coefficients to be used in the predictor, corrector numerical scheme. Contained in *winc_std_deriv.f.*
- *deriv.* This is the subroutine called when calculating the spatial derivatives. Contained in *winc_std_deriv.f.*
- *splinex.* This subroutine is called by deriv and actually calculates the 4th order x-direction spatial derivatives. Contained in *winc_std_deriv.f.*
- *spliney.* This subroutine is called by deriv and actually calculates the 4th order y-direction spatial derivatives. Contained in *winc_std_deriv.f.*

5.4 Interaction between short wave driver and current model

There are two options for incorporating the short-wave forcing.

Short-waves forcing with REF/DIF

The first option is to use the embedded wave driver REF/DIF. In the code the wave driver and the circulation part of the code are communicating via a common block contained in the file *pass.inc.* To function this way the wave driver must provide

- the wave height H,
- the wave direction angle α_w,
- the phase velocity c, and the group velocity c_g for the waves

The common block also includes the parameters required to describe the currents $V_{m\alpha}$ which are used by the short wave driver.

The shortwave_driver subroutine calculates the short wave forcing parameters from the data provided by the short wave driver. This includes

- the radiation stress $S_{\alpha,\beta}$,
- the short wave induced mass flux, $Q_{w\alpha}$,
- the bottom wave particle amplitude u_0,
- the bottom shear stress τ_B, and
- f_α used in the current velocity profiles.

Short-wave forcing from files

The second option is to have the model read data files created by a separate wave driver. The required files are

- *sxx.dat* contains S_{xx}
- *sxy.dat* contains S_{xy}
- *syy.dat* contains S_{yy}
- *qw.dat* contains the short-wave volume flux
- *height.dat* contains the wave height
- *angle.dat* contains the wave angle
- *diss.dat* contains the dissipation due to wave breaking
- *u0.dat* contains the amplitude of the bottom orbital velocity
- *k.dat* contains the wave number($\frac{2\pi}{L}$)
- *freq.dat* contains the frequency ($\frac{2\pi}{T}$)
- *fx.dat* contains f_x given by 3.21
- *fy.dat* contains f_y given by 3.21
- *tsx.dat* contains the cross-shore steady streaming stress
- *tsy.dat* contains the longshore steady streaming stress

The files must include a value for the wave height at each grid point with the following format

```
do i=1,nx
   read(8,*) (H(i,j),j=1,ny)
end do
```

5.5 Input files

Two types of input files are used in the Shorecirc modeling system: control files and data files. They are explained in the following two subsections.

5.5.1 The control input files

The control input files are the files that control the various modes of the model. They are contained in the *indat.dat* (for Ref/Dif wave driver part) and *input_winc.dat* (for the circulation part).

The REF/DIF1 control file *indat.dat*

The details for this file are in the Ref/Dif manual. We have not included a printout of indat.dat because the form of it can vary based on the choice of parameters. Users are referred to the manual for the REF/DIF1 model in Kirby and Dalrymple (1994), which is valid even though we use a modified version 2.5 of the model.

Make sure the grid parameters match between the circulation and short wave input control files, although when using *createdat* (see below) they will always match properly. The following list provides some guidelines for the main parameters used in *indat.dat.*

- icur = 1 (always)
- ibc: 0 for lateral wall boundaries, 1 for all others
- dxr = dx (this model-notation)
- dyr = dy (this model-notation)
- freqs = short wave period (seconds)
- amp = short wave amplitude (= $H/2$)
- dir = short wave direction in degrees counter clockwise from the x-axis

The parameter κ represents the wave height to water depth ratio H/h at the initiation of breaking. When H/h exceeds κ the waves start breaking. Similar when H/h decreases below γ the wave breaking stops. Default values for κ and γ are 0.78 and 0.4, respectively.

The circulation control file *input_winc.dat*

This file contains several lists of variables which the user can change *without having to recompile the program.* The file format is as follows (the terminology used in the file listing is described in more detail after the listing)

Control of model mode

```
\begin{itemize}
&BOUNDARY
 IBC1 = 2 or 4          2 Absorbing/generating, 4 wall
 IBC2 = 4 or 6          4 wall, 6 no-flux
 IBC3 = 1, 4 or 5       1 specify flux, 4 wall, 5 periodic
 IBC4 = 1, 4 or 5       1 specify flux, 4 wall, 5 periodic
&GRIDS
 DX =  (m)              cross-shore grid spacing
 DY =  (m)              longshore grid spacing
```

```
 NX =  #                  number of cross-shore grid points
 NY = #                   number of longshore grid points
 NXWALL = i               wall location
 HTTIDE =  (m)            tidal level
&SHORT_WAVE
 PERIOD =  (s)            short-wave period
 WC = (#)                 number of time steps between short-wave updates
 AMERGE =  real           Coefficient for roller transition
 FILES = 0 or 1           0 use R/D, 1 read files
&PHYSICS
 FCW =  real              friction factor
 VTSHEAR =  real          eddy viscosity coeff C1 k
 MDISS =  real            eddy viscosity coeff M
 CS =  real               Smagorinksy coeff
 WIND_VEL = (m/s)         wind velocity
 WIND_DIR = (degrees)     wind direction
 DISP3D = 1 or 0          1 3D, 0 2D
&CONTROL
 CR =  real               Courant number
 NTIME = #                number of time steps
 KOLD_START = 0 or 1      0 hot start, 1 cold start
 DEPTHMIN =  (m)          minimum depth
 DELAY =  (s)             time for ramping short-wave forcing.\\
\end{itemize}
```

Control of model output

```
&SENSOR
 JUMP = #                 interval for writing time series
 XSENSOR = i              i location of time series
 YSENSOR = j              j location of time series
 YPROFILE = j             j location for vertical profiles
&OUTPUT
 INTERVAL = #             interval for writing snapshots
 TAVGOUT = 1 or 0         1 writes, 0 doesn't write time averages
 MOMOUT = 1 or 0          1 writes, 0 doesn't write momentum balances
 DISPOUT = 1 or 0         1 writes, 0 doesn't write 3D coefficients\\
```

DESCRIPTION OF CONTROL INPUT FILES

Control of the model mode

The variables in the circulation control input file are defined below.

- **Field &BOUNDARY**
  - *ibc1*: boundary condition type on $x = 0$ (offshore). Takes values 2 or 4.
  - *ibc2*: boundary condition type on $x = L_x$ (shore). Takes values 4 or 6.
  - *ibc3*: boundary condition type on $y = 0$ (lateral). Takes values 1, 4 or 5.
  - *ibc4*: boundary condition type on $y = L_y$ (lateral). Takes values 1, 4 or 5.

  These numerical boundary condition parameters can have the following values:
  - 1: flux specified in files *qyb3.in* and *qyb4.in*.
  - 2: absorbing/generating boundary condition.
  - 4: no flux/ straight wall.
  - 5: periodicity.
  - 6: no flux following still water line. The program identifies the location of the shoreline by sweeping through each j and finding the last grid point where the depth exceeds the minimum depth specified by depthmin.

  For further details see also section 3.6

- **Field &GRID**
  - nx: number of grid points in x.
  - ny: number of grid points in y.
  - dx: grid spacing Δx.
  - dy: grid spacing Δy.
  - nxwall: Position of straight wall when ibc2=4 (i=?).
  - httide: Tidal water level added to the depth everywhere (m).

  Thus the size of the model domain is essentially specified through the specification of these six variables.

  The default maximum grid in the program is $200 * 200$. Changes in this default requires the following program changes:
  - in *winc_std_common.inc* change parameters nxm, nym and $nmax$ (= max(nxm,nym)) to desired values.
  - in *winc_std_filter* make similar changes.
  - in *param.h* change ixr, iyr to equal nxm, nym

Proper choices for the values of Δx and Δy are in the range of $1-2$ times the depth of breaking for the case you are computing. Because the discretization of the equations uses fourth order expressions for the space derivatives this will give a quite good representation of spacial variations with normal nearshore length scales. Smaller grid sizes can give better resolution but also increase the computation time.

- **Field &SHORT_WAVE**
  - period: the short wave period in seconds
  - wc: specifies the number of time steps between each recalculation of the forcing. A value of 0 turns off the wave/current interaction
  - amerge: Coefficient for transition length, typically around 5. Increasing it will increase the transition length.
  - files: A value of 0 uses Ref/Dif to calculate the wave forcing and a value of 1 reads the wave forcing from files.

A rule of thumb is to update the short wave forcing once every 1 to 5 short wave period which will be around 30 to 200 time steps.

- **Field &PHYSICS**
  - fcw: the bottom friction coefficient. Typical range $0.005 < f_{cw} < 0.03$
  - vtshear: the eddy viscosity coefficient $C_1\kappa$ in (3.49). Typically 0.08.
  - mdiss: the eddy viscosity coefficient M in (3.49). Typical range $0.05 < M < 0.1$. A value of $M = 0.08$ is recommended.
  - cs: the Smagorinksy eddy viscosity coefficient. Typical range $0.05 < C_s < 0.25$
  - wind_vel: the wind velocity W in (3.59) in m/s
  - wind_dir: the wind direction in degrees counter clockwise from the x-axis
  - disp3d: Value of 1 uses depth varying currents; 0 uses depth uniform currents.

    Notice that choosing the option 0 (depth uniform currents) implies that the dispersive mixing is turned off except for certain terms associated with Q_w. The only lateral mixing is then provided by ν_t in (3.49). As mentioned chapter 3.1.2 this will usually provide too little lateral mixing (see also Putrevu and Svendsen, 1999 for discussion).

- **Field &CONTROL**

  - cr: Courant number (maximum=0.5). Although some cases can use a Courant number up to 0.8. The cr is defined as

$$cr = \sqrt{gh_max}\Delta t/\Delta x \tag{5.1}$$

    where h_max is the largest depth in the computational domain.

  - ntime: number of time steps.

  - kold_start: *1* means the model is started from zero values, *0* indicates that a hot start will be used.

  - depthmin: minimum depth (positive small value). The depth uniform velocity is found by dividing the volume flux by the total depth and is used in many calculations. When the depth gets too small this can cause the velocity to get too large and blow the model up. Depthmin will be the minimum depth used in all calculations preventing this from occurring. Use values around 0.01 for field and 0.001 m for laboratory simulations.

  - delay: At cold starts the forcing for the flow is ramped in order to reduce initial oscillations and prevent instabilities. A $tanh^2(t/delay)$ ramping with a suitable time constant $delay(seconds)$ is used to phase in the forcing from zero to full value. This also controls the delay for computing the time-averaged quantities: time-averaging begins after delay*5.

    - Use values around 40-80 s for field conditions

    - and 10-20 s for laboratory simulations.

    Make sure for hot starts to set this parameter to a small value such as 0.01 s.
    Notice that when the absorbing generating boundary condition is used along some of the boundaries any oscillation initiated by the upstart of the motion will usually propagate out of the system quite quickly. However, if the all boundaries are reflecting walls (as in a closed basin) or boundaries where the volume flux is specified, then oscillations initiated at start may last very long. Such oscillations can be reduced significantly by choosing a value of *delay* which is reasonably large in comparison to the basic oscillation periods of the basin/computational domain. check your output for signs of basin oscillations and chose *delay* accordingly.

Control of output

- **Field &SENSOR**
  - jump: The interval (number of time steps) which the time series are written to files. Typically should be around 10.
  - xsensor: This field is a vector that contains the x-grid point numbers requested by the user for the output of time variation of $\bar{\zeta}, Q_\beta$. (A maximum of 20 locations can be specified. If there are less than 20 then the last one should be specified as 0). Numbers are in grid units, between 1 and nx. Each pair (xsensor(i),ysensor(i)) identifies a single point x and y-coordinates.
  - ysensor: vector containing the y-grid point numbers for output of time variation of $\bar{\zeta}, Q_\beta$. Numbers in grid units, between 1 and ny.
  - yprofile: vector containing the y-grid number for the location for the vertical current profiles. The variation of in the cross-shore direction of the vertical current profiles are written along 8 sections. Must have exactly 8 j coordinates. A value of 0 for the first one will suppress the file output

- **Field &OUTPUT**
  - interval: interval at which "snapshots" of the flow variables are taken for output files.
  - tavgout: Determines if the the time-averaged properties will be written to files (the fort.5xx series). A value of 0 turns it off and a value of 1 turns it on
  - momout: Determines if the instantaneous momentum balance will be written to files (the fort.6xx series). A value of 0 turns it off and a value of 1 turns it on
  - dispout: Determines if the 3D dispersive coefficients will be written to files (the fort.7xx series). A value of 0 turns it off and a value of 1 turns it on

5.5.2 Data input files

The **data files** contain the input of background information for the model computations: the bottom topography, data for hot start of the model, and data for the specified volume flux along cross-shore boundaries (when that option is used). The following data files are available:

- *bath.dat:* topography data in 3 columns: x, y, depth. The order of the points is arbitrary and the program will check to make sure a depth value is specified for every grid point and that there are no duplicates. For more information on topography options see under 5.7.

- *qyb3.in:* (optional) Volume flux specified at cross-shore boundary 3 in one column starting at $x = 0$.
- *qyb4.in:* (optional) Volume flux specified at cross-shore boundary 4 in one column starting at $x = 0$.
- *hot0.dat:* (optional) Data file for hot start $t = 0$. This file is automatically generated at the end of a (previous) model run.
- *hot1.dat:* (optional) Data file for hot start $t = -dt$. This file is automatically generated at the end of a (previous) model run.
- *hot2.dat:* (optional) Data file for hot start $t = -2dt$. This file is automatically generated at the end of a (previous) model run.

5.6 Compiling instructions.

Important!!! For reasons of efficiency the code is set up to write to many data files in binary form. As the first step when compiling on different types of platforms the record length for writing direct binary files must be checked and adjusted. Writing to binary files occurs in many locations within the two files *winc_std_main.f* and *winc_std_time.f*. Do a search-and-replace within these two files to fix this problem. The record length for some known platforms are listed as follows,

- CRAY : recl=1*ny
- SGI : recl=1*ny
- PC : recl=4*ny
- Solaris : recl=4*ny
- Digital : recl=1*ny

Compiling in Unix/Linux

The *Makefile* for compiling and linking is included in the code. In order to create an executable code three programs need to be compiled, all at one time, using the command

```
make shorecirc
```

The names of the resulting three executable files needed are

winc,
createbath,
createdat.

To **compile only one of the programs** use the following corresponding command listed below:

```
make winc
make createbath
make createdat
```

The *Makefile* will check to see if any component of the program has been changed, then compile that file and create the executable file.

Compiling in Windows

The program has been tested with Fortran Power Station in Windows. To create the executable program for generating the topography compile and build *create_bath.f* and *winc_std_common.inc*. To create the program for generating the input files compile and build *create_input.f*, *infile1.f*, *winc_std_common.inc* and *param.h*. To create the SHORECIRC executable program compile and build *winc_std_main.f*, *winc_std_deriv.f*, *winc_std_time.f*, *winc_std_filter.f*, *winc_std_sub1.f*, *infile1.f*, *refdif2v25a.f*, *winc_std_common.inc*, *param.h* and *pass.inc*.

C-preprocessors

A version of the model utilizing C-preprocessors has been tested. This version of the model was only marginally faster than the current version. Using C-preprocessors would require that the model be recompiled every time one of the parameters is changed. Because the current version allows most parameters to be changed without recompiling and is not much slower, we decided not to include the preprocessor option in this version of the model.

5.7 Operating procedure

This section will explain the procedure for running the model.

The first step is to create the topography file *bath.dat*.

If you want to use one of the simple topographies described in Section 5.3 this can be done using the program *createbath*. This program will interactively ask for a series of parameters and use them to create the simple topography of your choice. The program will end by creating the files *bath.dat* and *dim.dat*. The latter file is used by *createdat* when creating the input control files.

However, you can also create a *bath.dat* file using you own topography by following the format described in the section 5.5.2.

The next step is to use the program *createdat* to create the input control files *input_winc.dat* and *indat.dat*. This program asks interactively what value to use for all the parameters in the input files. The program will give you the option to use the dimensions from *createbath* in the file *dim.dat*. The details for all the parameters in *input_winc.dat* are listed in the next section. The details for *indat.dat* can be found in the REF/DIF manual although some guidelines have been provided in the next section.

The *createbath* and *createdat* programs are provided for easy means of creating the files initially. However, again the input files do not have to be created using those

programs. You can create them by hand using the guidelines in the next few sections. If you only need to change a few parameters this can be accomplished by simply editing the files directly.

The model can be operated with one of two different types of initial conditions: a **cold start** or a **hot start**.

Cold Start

In a cold start the initial conditions correspond to no flow or forcing at $t = 0$. At that time the forcing is started but in order to prevent unrealistic (and potentially unstable) oscillations to develop the forcing is ramped up from zero at $t = 0$ to its full value over a period or some time steps, based on the value of the *delay*-parameter chosen in the *input_winc.dat* input file.

When running the program there will be a prompt

```
Load stresses? (0=no, 1=yes) :
```

By typing in 0 the program will stop running. At this point the program will have created the topography and written it to a file called *fort.311* which can be checked to verify the model is using the correct topography.

The next step is to run the program again and calculate the forcing by hitting 1. The next prompt will be

```
Start current model? (0=no, 1=yes) :
```

It is usually best to hit 0 and stop the program again to check that the forcing is calculated correctly by checking the output files described in the next section.

If everything checks out then the program can be run by starting the program again and hitting 1 twice.

Since model runs can take a long time, a good idea is to run the program in the background by using the following command (or something similar),

```
winc < screenin > screenout &
```

The file *screenin* contains the following lines,

```
1
1
```

This file produces the two keystrokes required to run the model. The other file, *screenout* will be created during the computation with all of the screen output written to it.

Hot Start

The model also has the option of being operated from a hot start. A hot start requires input of a set of realistic values at all points for the dependent variables $\zeta, Q_x, and Q_y$ for three consecutive time steps. Usually this information is obtained by

storing the value of all variables from the last three time steps in a previous model run. Hence the hot start is particularly aimed at making it possible to restart (and hence extend) an already performed model run.

In the model this option is only provided when letting the computation run to the end of the time specified in the input. Before stopping the model will then automatically generate the 3 files, *hot0.dat*, *hot1.dat* and *hot2.dat*, which contain the required information from the last three time steps of the run. The model can then be restarted by changing the parameter *kold_start* to 0 and using those three files as the input for time steps $t = 0$, $t = -dt$ and $t = -2dt$ in the continuation of the previous run. When doing a hot start make sure the delay is set to some small small value such as 0.01 s. Otherwise, the operating procedure remains the same as the cold start.

5.8 Model output

This section will describe the model output, both on the screen and written to files.

5.8.1 Screen output

The screen output is fairly simple and is only really useful for debugging purposes. When the model is started it will write to the screen which subroutine it is working on. The program also writes several of the user defined parameters so the user can confirm that the model is running properly. Once the model gets to the time loop it will write the following,

```
Start Time Loop
 step          10
 step          20
 step          30
 ....
```

where the number is the current step. This is repeated every ten steps until the model run is done.

Whenever the model writes one of the "snapshots" the following is written to the screen,

```
      10001    316.1297809700816                120
```

where the first number is the step number, the second number is the actual time associated with that step number, and the last number is the file which is being created (see the listing of snapshot files below).

When the program is done it will write to the screen

```
end
```

signaling the completion of the model run.

5.8.2 Output to files

The program writes its output to files with names *fort.*xxx. The xxx represents numbers which differentiate the output files. The output files are listed as follows with both the theoretical and the Fortran variable names listed. The possible values of the indices (i,j) are indicated for each group of files. Where nothing else is indicated the files provide the parameter values at the last time step of the computation.

Printing out of input information

- The file *status.dat* contains the following information
  dx,dy,dt,cr
  nx,ny,ntime,interval
- The file numbers $1 \rightarrow 49$ are not used.

Snapshots of surface elevations, and depth averaged velocities

- Time Series of certain variables are written every *jump* time steps. The value for *jump* is specified in *input_winc.dat*
  - $51 \rightarrow 70$ These files will contain the time series of water surface and volume fluxes at the (i, j) chosen for the xsensor and ysensor locations (max 20). The output is $\overline{\zeta}$ $(zetan(i,j))$, Q_x $(qxn(i,j))$, and Q_y $(qyn(i,j))$. Each file corresponds to one sensor location.
- $100 \rightarrow 310$ These files contain the snapshots (up to a total of 211) which are taken every *interval* time steps of the water surface elevation $\overline{\zeta}$ $(zetan(i,j))$ and volume flux Q_x $(qxn(i,j))$, and Q_y $(qyn(i,j))$, respectively, at all grid points (i = 1:nx, j = 1:ny). The file created at $t = 0$ is numbered 100. As each new file is created at every interval, the file number is incremented up by 1. These files are written in binary format
- Printout of input and short wave forcing variables is made at all grid points $(i = 1 : nx, j = 1 : ny)$. The constant variables, such as topography, are only written once at the beginning. This applies to the short wave forcing when no wave-current interaction is invoked. However, when utilizing wave/current interaction variables such as wave height, change with some interval. The files containing those variables are then overwritten every *interval* time step. Hence these files always contain the values at the last time step of the *interval*-outputs. Again these are all written in binary format.

Topography information

- 311 topography h_o: $ht(i,j)$.

- 312 Cross-shore depth gradients $\frac{\partial h_o}{\partial x_\alpha}$: $dhodxn(i,j)$.
- 313 Longshore depth gradients $\frac{\partial h_o}{\partial y}$: $dhodyn(i,j)$.

Short wave data

- 314 Short wave height H: $H(i,j)$.
- 315 Short wave angle in degrees α: $theta(i,j)/\pi * 180$.
- 316 Short wave celerity c: $c\_sw(i,j)$.
- 317 Short wave breaking index $ibrk(i,j)$. Value 1 means breaking and 0 means non-breaking.
- 318 Radiation stress component, S_{xx}: $Sxx(i,j)$.
- 319 Radiation stress component, S_{xy}: $Sxy(i,j)$.
- 320 Radiation stress component, S_{yy}: $Syy(i,j)$.
- 321 $\frac{1}{\rho}\frac{\partial S_{xx}}{\partial x}$: $dSxxdxn((i,j)$.
- 322 $\frac{1}{\rho}\frac{\partial S_{xy}}{\partial x}$: $dSxydxn((i,j)$.
- 323 $\frac{1}{\rho}\frac{\partial S_{xy}}{\partial y}$: $dSxydyn((i,j)$.
- 324 $\frac{1}{\rho}\frac{\partial S_{yy}}{\partial y}$: $dSyydyn((i,j)$.
- 325 Short wave bottom particle velocity u_o: $u0(i,j)$.
- 326 Short wave cross-shore volume flux Q_{wx}: $qwx(i,j)$.
- 327 Short wave longshore volume flux Q_{wy}: $qwy(i,j)$.
- 328 Short wave dissipation D: $dissipation(i,j)$.
- 329 Smoothed breaking index $newibr(i,j)$ used when calculating ν_t.
- 330 Eddy viscosity ν_t: $v\_t(i,j)$.

Velocity profile information

- $401 \rightarrow 408$ Cross-shore sections of the coefficients for the vertical velocity profiles defined by equations (3.12), (3.13) and (3.22). These locations are specified by the input parameter yprofile in *input_winc.dat*. For one chosen j the variables are written out in the following order starting from the offshore boundary at $i = 1$ and going shoreward to $i = nx$:
d1x(i,j), 1x(i,j), f1x(i,j), f2x(i,j), d1y(i,j), e1y(i,j), f1y(i,j), f2y(i,j), ht(i,j), zetanp1(i,j), qxnp1(i,j), qynp1(i,j), uda4(i,j), uda3(i,j), uda2(i,j), vda4(i,j), vda3(i,j), vda2(i,j), udb4(i,j), udb3(i,j), udb2(i,j), vdb4(i,j), vdb3(i,j), vdb2(i,j), udw4(i,j), udw3(i,j), udw2(i,j), vdw4(i,j), vdw3(i,j), vdw2(i,j), udc(i,j), vdc(i,j), qwx(i,j), qwy(i,j) which provides 35 columns with nx rows. The files are written in ASCII format every *interval* time steps, every time overwriting the files from earlier times so that only the last values are kept.

Long term averaged results.

- If *tavgout* $= 1$, then long term averaged values of the short wave averaged variables are calculated from $t = 5 * delay$ till the end of the run. This implies that unless total running time specified for the model is larger than $5 * delay$ there will be no output for long term averaged values. If so, a warning is written to the screen of the computer.

  In the following "cross-shore" refers to x-components, "longshore" to the y-components of the equations.

  The results are written in binary format at the end of the model run and can be found in the output files listed below by their file number:

  - 501 $\overline{\frac{\partial Q_x}{\partial x}}$: $adqxdx(i,j) * inv\_average.$
  - 502 $\overline{\frac{\partial Q_y}{\partial y}}$: $adqydy(i,j) * inv\_average.$
  - 503 Cross-shore bottom friction component $\overline{\frac{\bar{\tau}_x^B}{\rho}}$:
    $africtx(i,j) * inv\_average.$
  - 504 Cross-shore turbulent mixing $\overline{\frac{1}{\rho}\frac{\partial}{\partial y}\overline{\int_{-h_0}^{\zeta}\tau_{xy}dz}}$):
    $adtaudy(i,j) * inv\_average.$
  - 505 Cross-shore pressure gradient $\overline{gh\frac{\partial \bar{\zeta}}{\partial x}}$:
    $adzdx(i,j) * inv\_average.$
  - 506 $\overline{\frac{\partial}{\partial x}(\frac{{Q_x}^2}{h})}$: $aqxqxdx(i,j) * inv\_average.$
  - 507 $\overline{\frac{\partial}{\partial y}(\frac{Q_xQ_y}{h})}$: $aqxqydy(i,j) * inv\_average.$
  - 508 Sum of all 3-D dispersion terms
    $\overline{-\frac{\partial}{\partial x}(M_{xy}) - \frac{\partial}{\partial y}(M_{yy})}$
    $+\frac{\partial}{\partial x}[(D_{xy}+B_{xy})\frac{\partial}{\partial x}(\frac{Q_x}{h}) + D_{yy}\frac{\partial}{\partial y}(\frac{Q_x}{h}) + D_{xx}\frac{\partial}{\partial x}(\frac{Q_y}{h}) + (D_{xy}+B_{xy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]$

$\overline{+\frac{\partial}{\partial y}[B_{yy}\frac{\partial}{\partial x}(\frac{Q_x}{h})+2D_{xy}\frac{\partial}{\partial x}(\frac{Q_y}{h})+(2D_{yy}+B_{xy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]}$
$\overline{-\frac{\partial}{\partial x}[A_{xyx}\frac{Q_x}{h}+A_{xyy}\frac{Q_y}{h}]-\frac{\partial}{\partial y}[A_{yyx}\frac{Q_x}{h}+A_{yyy}\frac{Q_y}{h}]}$:
$adispx(i,j)*inv\_average$.

- 509 Longshore bottom friction component. $\overline{\frac{\bar{\tau}_y^B}{\rho}}$:
  $africty(i,j)*inv\_average$.
- 510 Longshore turbulent mixing. $\overline{\frac{1}{\rho}\frac{\partial}{\partial x}\overline{\int_{-h_0}^{\zeta}\tau_{xy}dz}}$):
  $adtaudx(i,j)*inv\_average$.
- 511 Longshore pressure gradient. $\overline{gh\frac{\partial\bar{\zeta}}{\partial y}}$:
  $adzdy(i,j)*inv\_average$.
- 512 $\overline{\frac{\partial}{\partial y}(\frac{Q_y{}^2}{h})}$: $aqyqydy(i,j)*inv\_average$.
- 513 $\overline{\frac{\partial}{\partial x}(\frac{Q_xQ_y}{h})}$: $aqxqydx(i,j)*inv\_average$.
- 514 Time-averaged longshore 3-D dispersion
  $\overline{-\frac{\partial}{\partial x}(M_{xy})-\frac{\partial}{\partial y}(M_{yy})}$
  $\overline{+\frac{\partial}{\partial x}[(D_{xy}+B_{xy})\frac{\partial}{\partial x}(\frac{Q_x}{h})+D_{yy}\frac{\partial}{\partial y}(\frac{Q_x}{h})+D_{xx}\frac{\partial}{\partial x}(\frac{Q_y}{h})+(D_{xy}+B_{xy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]}$
  $\overline{+\frac{\partial}{\partial y}[B_{yy}\frac{\partial}{\partial x}(\frac{Q_x}{h})+2D_{xy}\frac{\partial}{\partial x}(\frac{Q_y}{h})+(2D_{yy}+B_{xy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]}$
  $\overline{-\frac{\partial}{\partial x}[A_{xyx}\frac{Q_x}{h}+A_{xyy}\frac{Q_y}{h}]-\frac{\partial}{\partial y}[A_{yyx}\frac{Q_x}{h}+A_{yyy}\frac{Q_y}{h}]}$:
  $adispy(i,j)*inv\_average$.
- 515 Water surface elevation $\bar{\bar{\zeta}}$: $azeta(i,j)$.
- 516 Cross-shore volume flux $\overline{Q_x}$: $aqxn(i,j)$.
- 517 Longshore volume flux $\overline{Q_y}$: $aqyn(i,j)$.
- 518 $\frac{\partial}{\partial x}(\frac{\overline{Q_xQ_y}}{h})$ calculated from time-averaged quantities.
- 519 $\frac{\partial}{\partial y}(\frac{\overline{Q_xQ_y}}{h})$ calculated from time-averaged quantities.
- 520 $\frac{\partial}{\partial x}(\frac{\overline{Q_x}^2}{h})$ calculated from time-averaged quantities.
- 521 $\frac{\partial}{\partial y}(\frac{\overline{Q_y}^2}{h})$ calculated from time-averaged quantities.
- 522 Cross-shore pressure gradient calculated from time-averaged quantities $gh\frac{\partial\bar{\bar{\zeta}}}{\partial x}$.
- 523 Longshore pressure gradient calculated from time-averaged quantities. $gh\frac{\partial\bar{\bar{\zeta}}}{\partial y}$.
- 524 Cross-shore bottom friction calculated from time-averaged quantities.
- 525 Longshore bottom friction calculated from time-averaged quantities

Terms in the instantaneous momentum balance

- If *momout* = 1, then terms in the instantaneous momentum balance are written every *interval* time steps. Written in binary format.

  - 601 $-\frac{1}{\rho}\frac{\partial S_{xx}}{\partial x}$: $-force\_xx(i,j,t)$.
  - 602 $-\frac{1}{\rho}\frac{\partial S_{xy}}{\partial y}$: $-force\_xy(i,j,t)$.
  - 603 Cross-shore bottom friction component $-\frac{\bar{\tau}_x^B}{\rho}$: $-frictx(i,j)$.
  - 604 Cross-shore turbulent mixing $\frac{1}{\rho}\frac{\partial}{\partial y}\overline{\int_{-h_0}^{\zeta}\tau_{xy}dz}$): $dtauxdy(i,j)$.
  - 605 Cross-shore pressure gradient $-gh\frac{\partial\bar{\zeta}}{\partial x}$: $-g*htt(i,j)*dzetadxstar(i,j)$.
  - 606 Cross-shore nonlinear convective terms $-\frac{\partial}{\partial x}(\frac{{Q_x}^2}{h}) - \frac{\partial}{\partial y}(\frac{Q_xQ_y}{h})$: $-qxqxdx(i,j) - qxqydy(i,j)$.
  - 607 $-\frac{1}{\rho}\frac{\partial S_{xy}}{\partial x}$: $-force\_xy(i,j,t)$.
  - 608 $-\frac{1}{\rho}\frac{\partial S_{yy}}{\partial y}$: $-force\_yy(i,j,t)$.
  - 609 Longshore bottom friction component. $-\frac{1}{\rho}\frac{\bar{\tau}_y^B}{\rho}$: $-fricty(i,j)$.
  - 610 Longshore turbulent mixing $\frac{1}{\rho}\frac{\partial}{\partial x}\overline{\int_{-h_0}^{\zeta}\tau_{xy}dz}$): $dtauxdx(i,j)$.
  - 611 Longshore pressure gradient $-gh\frac{\partial\bar{\zeta}}{\partial y}$: $-g*htt(i,j)*dzetadystar(i,j)$.
  - 612 Longshore nonlinear convective terms $-\frac{\partial}{\partial y}(\frac{{Q_y}^2}{h}) - \frac{\partial}{\partial x}(\frac{Q_xQ_y}{h})$: $-qxqydx(i,j) - qyqydy(i,j)$.
  - 613 Cross-shore 3-D dispersion
    $-\frac{\partial}{\partial x}(M_{xx}) - \frac{\partial}{\partial y}(M_{xy})$
    $+ \frac{\partial}{\partial x}[(2D_{xx}+B_{xx})\frac{\partial}{\partial x}(\frac{Q_x}{h}) + 2D_{xy}\frac{\partial}{\partial y}(\frac{Q_x}{h}) + B_{xx}\frac{\partial}{\partial y}(\frac{Q_y}{h})]$
    $+ \frac{\partial}{\partial y}[(D_{xy}+B_{xy})\frac{\partial}{\partial x}(\frac{Q_x}{h}) + D_{yy}\frac{\partial}{\partial y}(\frac{Q_x}{h}) + D_{xx}\frac{\partial}{\partial x}(\frac{Q_y}{h}) + (D_{xy}+B_{xy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]$
    $-\frac{\partial}{\partial x}[A_{xxx}\frac{Q_x}{h} + A_{xxy}\frac{Q_y}{h}] - \frac{\partial}{\partial y}[A_{xyx}\frac{Q_x}{h} + A_{xyy}\frac{Q_y}{h}]$:
    $dispx(i,j)$.
  - 614 Longshore 3-D dispersion
    $-\frac{\partial}{\partial x}(M_{xy}) - \frac{\partial}{\partial y}(M_{yy})$
    $+ \frac{\partial}{\partial x}[(D_{xy}+B_{xy})\frac{\partial}{\partial x}(\frac{Q_x}{h}) + D_{yy}\frac{\partial}{\partial y}(\frac{Q_x}{h}) + D_{xx}\frac{\partial}{\partial x}(\frac{Q_y}{h}) + (D_{xy}+B_{xy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]$
    $+ \frac{\partial}{\partial y}[B_{yy}\frac{\partial}{\partial x}(\frac{Q_x}{h}) + 2D_{xy}\frac{\partial}{\partial x}(\frac{Q_y}{h}) + (2D_{yy}+B_{xy})\frac{\partial}{\partial y}(\frac{Q_y}{h})]$
    $-\frac{\partial}{\partial x}[A_{xyx}\frac{Q_x}{h} + A_{xyy}\frac{Q_y}{h}] - \frac{\partial}{\partial y}[A_{yyx}\frac{Q_x}{h} + A_{yyy}\frac{Q_y}{h}]$:
    $dispy(i,j)$.
  - 615 Cross-shore local acceleration $-\frac{\partial Q_x}{\partial t}$: $-(qxnp1(i,j) - qxn(i,j))/dt$.
  - 616 Longshore local acceleration $-\frac{\partial Q_y}{\partial t}$: $-(qynp1(i,j) - qyn(i,j))/dt$.

- 617 $\frac{\partial}{\partial x}(\frac{{Q_x}^2}{h})$: $qxqxdx(i,j)$.
- 618 $\frac{\partial}{\partial y}(\frac{Q_x Q_y}{h})$: $qxqydy(i,j)$.
- 619 $\frac{\partial}{\partial y}(\frac{{Q_y}^2}{h})$: $qyqydy(i,j)$.
- 620 $\frac{\partial}{\partial x}(\frac{Q_x Q_y}{h})$: $qxqydx(i,j)$.
- 621 Cross-shore wind stress $\frac{\tau_x^S}{\rho}$: $wind_x(t)$.
- 622 Longshore wind stress $\frac{\tau_y^S}{\rho}$: $wind_y(t)$.

- 701 Position of the last wet point $nxa(i,j)$ written in ASCII every *interval* time steps.

The values of the dispersive mixing coefficients.

- When the model is run in quasi-3D mode and *dispout* = 1, then the dispersive mixing coefficients are written every *interval* time steps. These results are written in binary format on the following files (which are overwritten at each *interval* time steps):

  - 746 $B_{\alpha\beta}$: $Bxx(i,j)$,$Bxy(i,j)$,$Byy(i,j)$
  - 747 $D_{\alpha\beta}$: $Dxx(i,j)$,$Dxy(i,j)$,$Dyy(i,j)$
  - 748 $M_{\alpha\beta}$: $Mxx(i,j)$,$Mxy(i,j)$,$Myy(i,j)$
  - 749 $A_{\alpha\beta x}$: $Axxx(i,j)$,$Axyx(i,j)$,$Ayyx(i,j)$
  - 749 $A_{\alpha\beta y}$: $Axxy(i,j)$,$Axyy(i,j)$,$Ayyy(i,j)$

Data for next hot start

  - *hot0.dat*, *hot1.dat* and *hot2.dat* are created at the end of every model run and can be used for a hot start. They contain water levels and volume fluxes for the last three time steps.

5.9 Test examples

The following test results are provided to help users check the correct function of the model and show examples of the use of the model. Input files for the test examples are included with the code.

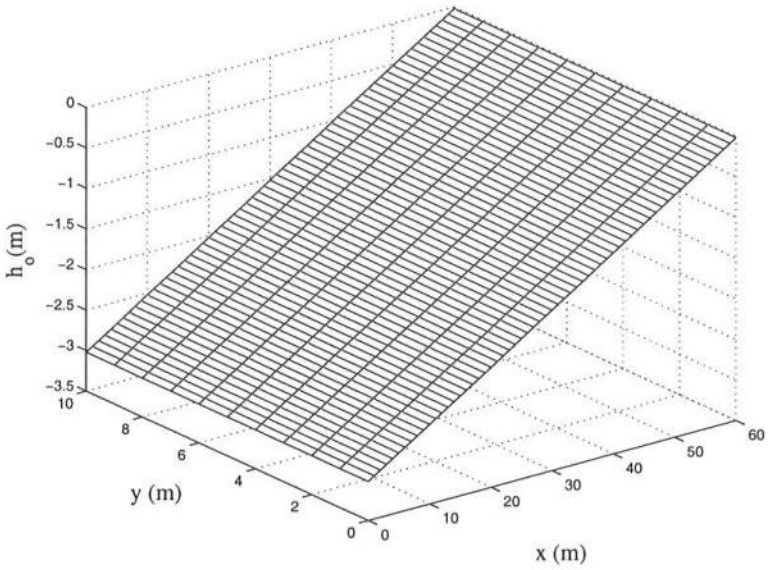

Figure 9: Topography used in the test case. Created by plotbath.m

5.9.1 Test Case: A stationary longshore current on a long plane beach

This test case describes the flow that you should get after sufficiently long time (of the order 5000 time steps) when running the model from cold start on a ("long") plane beach. The topography is shown in fig 9.

This figure also shows the computational domain. Formally the beach is "infinitely long". In the computations this is represented by applying periodic boundary conditions along the two cross-shore boundaries. To reduce the computational time the longshore length only needs to be long enough (= consist of sufficiently many grid points in the longshore direction) to ensure that the special version of the numerical scheme used at the grid points near one of the cross-shore boundaries does not overlap with the equivalent scheme used on the opposite cross-shore boundary. In the present case we have set the width of the computational domain to 10 grid points ($ny = 10$).

The still water depth at the outer boundary of the computational domain is $h_0 =$ 3.0 m. Regular waves are incident from the outer boundary at an angle $\alpha_w = 22.4°$. The incoming wave height is $H = 0.61m$ and the wave period is $T = 4.0s$. The waves start breaking at $x = 43m$ where the depth is $h_0 = 3.0m$. The short wave forcing is generated by the wave driver (REF/DIF1).

Input for the computations.

Note: The input files are included with the code.
Control files

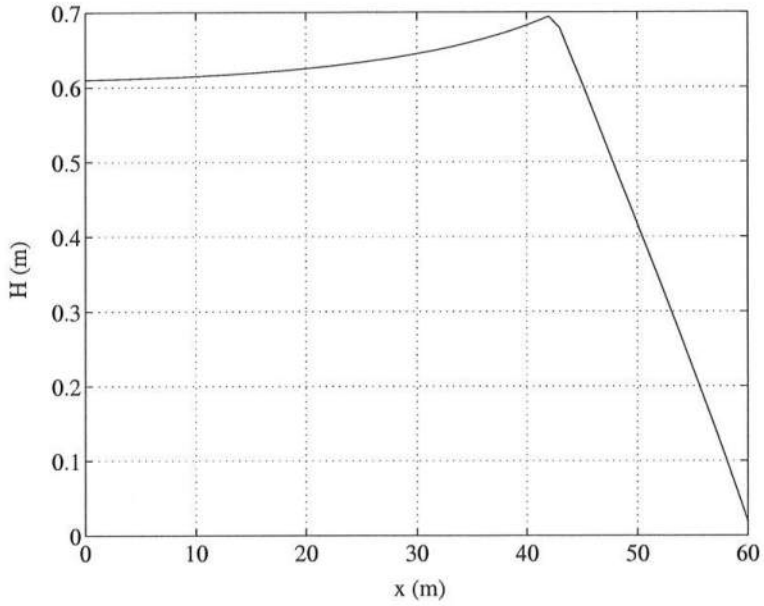

Figure 10: Wave height variation in the test case versus cross-shore distance from the outer boundary of the computational domain. The wave breaking starts approximately at $x = 43m$. Created by plotH.m

The following input data was used for the computations:

indat.dat: input data file for REF/DIF 1

```
&FNAMES
 FNAME1 = '''refdat.dat''',
 FNAME2 = '''outdat.dat''',
 FNAME3 = '''subdat.dat''',
 FNAME4 = '''wave.dat''',
 FNAME5 = '''owave.dat''',
 FNAME6 = '''surface.dat''',
 FNAME7 = '''bottomu.dat''',
 FNAME8 = '''angle.dat''',
 FNAME9 = ''' ''',
 FNAME10 = '''refdif1.log''',
 FNAME11 = '''height.dat''',
 FNAME12 = '''sxx.dat''',
 FNAME13 = '''sxy.dat''',
 FNAME14 = '''syy.dat''',
 FNAME15 = '''depth.dat'''/
&INGRID
 MR = 61,
```

```
 NR = 11,
 IU = 1,
 NTYPE = 0,
 ICUR = 1,
 IBC = 1,
 ISMOOTH = 0,
 DXR =  1.,
 DYR =  1.,
 DT =  0.005,
 ISPACE = 0,
 ND = 1,
 IFF = 1 0 0,
 ISP = 0,
 IINPUT = 1,
 IOUTPUT = 1/
&WAVES1A
 IWAVE = 1,
 NFREQS = 1,
 KAPP =  0.78,
 GAMM =  0.4/
&WAVES1B
 FREQS =  4.,
 TIDE =  0.,
 NWAVS = 1,
 AMP =  0.305,
 DIR =  22.4/
```

input_winc.dat: Input data file for circulation part of the model.

```
&BOUNDARY
 IBC1 = 2,
 IBC2 = 4,
 IBC3 = 5,
 IBC4 = 5/
&GRIDS
 DX =  1.,
 DY =  1.,
 NX = 61,
 NY = 11,
 NXWALL = 61,
 HTTIDE =  0./
&SHORT_WAVE
 PERIOD =  4.,
```

```
 WC = 0,
 AMERGE =  5.,
 FILES = 0/
&PHYSICS
 FCW =  0.012,
 VTSHEAR =  0.08,
 MDISS =  0.05,
 CS =  0.2,
 WIND_VEL =  0.,
 WIND_DIR =  0.,
 DISP3D = 1/
&CONTROL
 CR =  0.5,
 NTIME = 5001,
 KOLD_START = 1,
 DEPTHMIN =  0.002,
 DELAY =  40./
&SENSOR
 JUMP = 10,
 XSENSOR = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0,
 YSENSOR = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0,
 YPROFILE = 5 0 0 0 0 0 0 0/
&OUTPUT
 INTERVAL =100,
 TAVGOUT = 0,
 MOMOUT = 0,
 DISPOUT = 1/
```

Data files

The bottom topography is a plane beach with a slope of 1:20 which is generated by using the program *createbath* described in Sections 5.3 and 5.7. The shoreline is at x = 60 m.

Results.

The results from the computations are shown in the following figures. They are obtained after 5000 timesteps and represent the nearly steady flow conditions.

The wave height variation in the cross-shore direction is shown in fig 10. Similarly, Fig 11 shows the cross-shore distribution of the longshore currents $\tilde{V}$. The vertical variation of the velocities are a combination of the the depth varying undertow and the depth varying longshore currents. Fig 12 shows the variation of the cross- and longshore velocities over depth (left two panels) and the 3-D profiles of the total

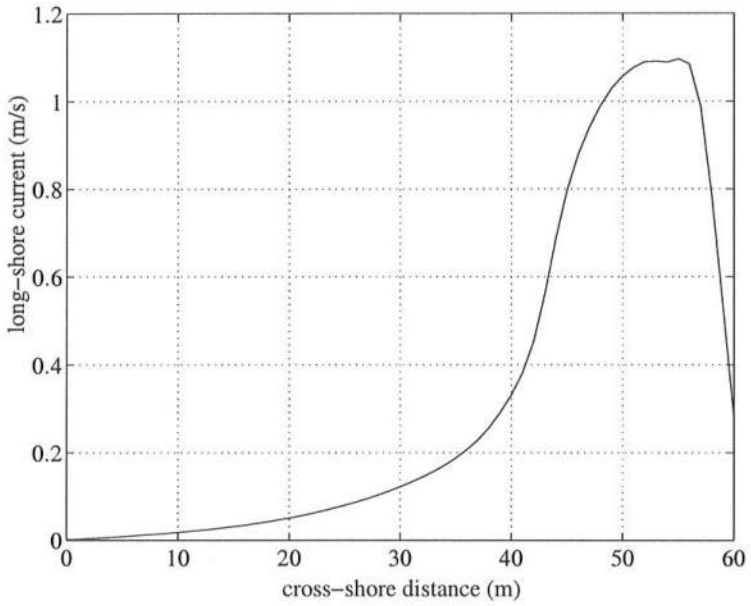

Figure 11: The longshore velocities $\tilde{V}$ for the test case versus cross-shore distance. Created by currentV.m

currents velocities at four different cross-shore positions (right panel). The top 2 rows are outside the surfzone and the bottom 2 rows are inside the surfzone. The same profiles are shown with the topography in Fig 13.

5.9.2 Example: Cold start of longshore current on a long plane beach

The purpose of this example is to demonstrate the capabilities of the model to handle the complexities of a seemingly simple case. The situation considered in this example is the cold start of a longshore current on a long straight coast. This is the same situation that was analysed by Van Dongeren et al (1994). However, because the model has developed considerably since then the actual results differ somewhat from the results of that publication.

This example is essentially the same flow situation that was shown in Example 1 above. However, in the present example we examine the flow in time as it develops from the cold start. The idea is to illustrate how complicated the details of the flow pattern are until it eventually reaches the relatively simple steady situation shown in the test case. Despite the complicated behavior, the flow remains longshore uniform throughout the development to the steady state.

This example demonstrates all parts of the model except the variations in the longshore direction and the wave current interaction.

Description of the flow

The (steady) short wave forcing for this example is generated by the wave driver

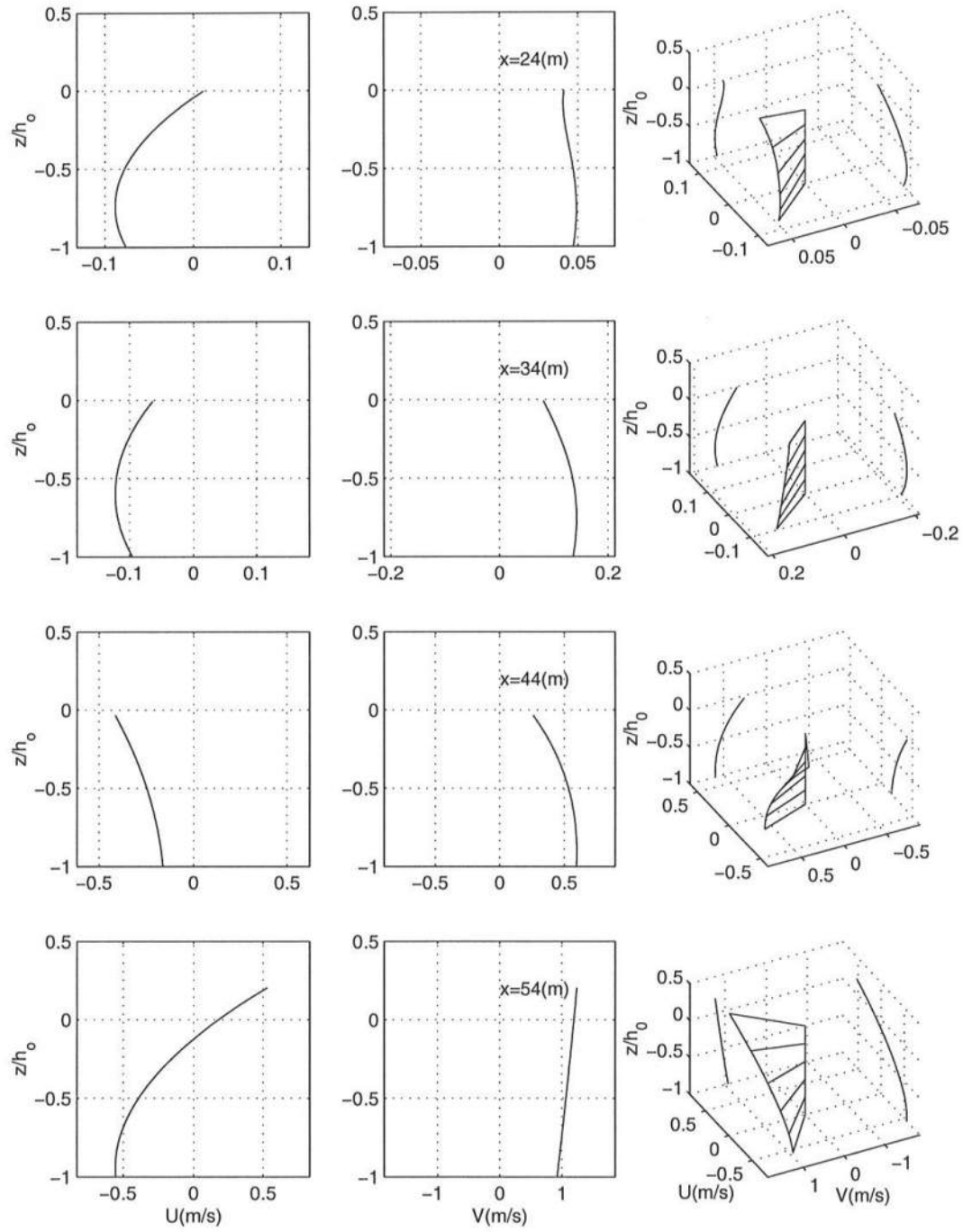

Figure 12: The vertical profiles of cross- and longshore current velocities, U and V respectively for the test case versus depth at four different cross-shore positions given by their x-values (two left panels). Also the equivalent 3-D profiles (right panel). Created by profstdvd.m

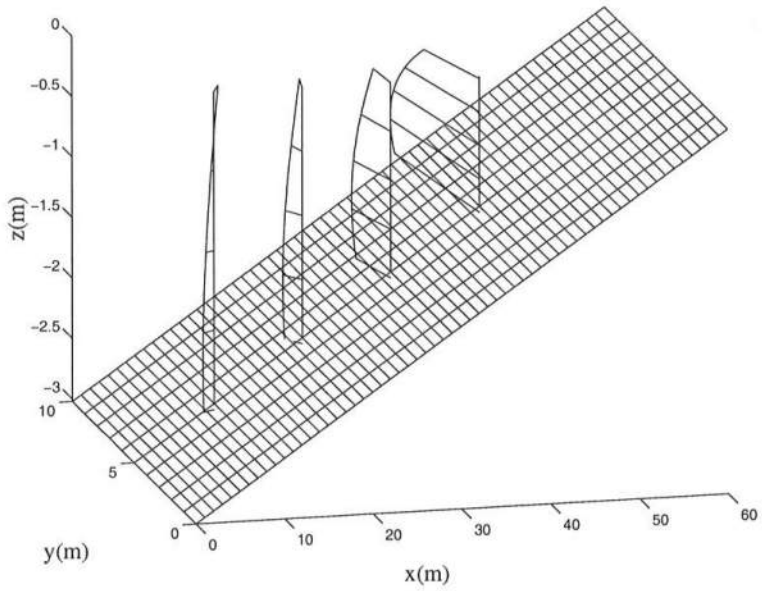

Figure 13: The 3-D velocity profiles for the test case shown in Fig 12. Created by profstdvd_bath.m

(REF/DIF1) for the wave motion on a long straight coast. The flow is generated as follows:

At $t = 0$ the short wave forcing is applied to the computational domain which initially is at rest. A $tanh^2$ ramping with a suitable time constant (delay = 10 s) is used to phase in the forcing from zero to its full value. The computations show that the cross-shore setup develops very quickly whereas the longshore current takes much longer time to develop to full value. During this period the vertical current velocity profiles change from almost entirely shore-normal velocities with a significant shoreward net flux in the early stage of the computations, to a longshore dominated flow as the longshore current gets developed. At that later stage the cross-shore flow shows little or no cross-shore net flux.

Input for the computations.

Control files

The following input data was used for the computations:

indat.dat: input file with data for REF/DIF 1

```
$fnames
&FNAMES
FNAME1 = '''refdat.dat''',
FNAME2 = '''outdat.dat''',
FNAME3 = '''subdat.dat''',
FNAME4 = '''wave.dat''',
FNAME5 = '''owave.dat''',
```

```
 FNAME6 = '''surface.dat''',
 FNAME7 = '''bottomu.dat''',
 FNAME8 = '''angle.dat''',
 FNAME9 = ''' ''',
 FNAME10 = '''refdif1.log''',
 FNAME11 = '''height.dat''',
 FNAME12 = '''sxx.dat''',
 FNAME13 = '''sxy.dat''',
 FNAME14 = '''syy.dat''',
 FNAME15 = '''depth.dat'''/
&INGRID
 MR = 61,
 NR = 11,
 IU = 1,
 NTYPE = 0,
 ICUR = 1,
 IBC = 1,
 ISMOOTH = 0,
 DXR =  1.,
 DYR =  1.,
 DT =  0.005,
 ISPACE = 0,
 ND = 1,
 IFF = 1 0 0,
 ISP = 0,
 IINPUT = 1,
 IOUTPUT = 1/
&WAVES1A
 IWAVE = 1,
 NFREQS = 1,
 KAPP =  0.78,
 GAMM =  0.4/
&WAVES1B
 FREQS =  4.,
 TIDE =  0.,
 NWAVS = 1,
 AMP =  0.305,
 DIR =  22.4/
```

input_winc.dat: Input data file for circulation part of the model.

```
&BOUNDARY
 IBC1 = 2,
```

```
 IBC2 = 4,
 IBC3 = 5,
 IBC4 = 5/
&GRIDS
 DX =  1.,
 DY =  1.,
 NX = 61,
 NY = 11,
 NXWALL = 61,
 HTTIDE =  0./
&SHORT_WAVE
 PERIOD =  4.,
 WC = 0,
 AMERGE =  5.,
 FILES = 0/
&PHYSICS
 FCW =  0.012,
 VTSHEAR =  0.08,
 MDISS =  0.05,
 CS =  0.2,
 WIND_VEL =  0.,
 WIND_DIR =  0.,
 DISP3D = 1/
&CONTROL
 CR =  0.5,
 NTIME = 5001,
 KOLD_START = 1,
 DEPTHMIN =  0.002,
 DELAY =  10./
&SENSOR
 JUMP = 10,
 XSENSOR = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0,
 YSENSOR = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0,
 YPROFILE = 5 0 0 0 0 0 0 0/
&OUTPUT
 INTERVAL =47,
 TAVGOUT = 0,
 MOMOUT = 0,
 DISPOUT = 1/
```

Results.

The following results are shown from the computations:

The time step is calculated by the program based on the chosen value of the Courant

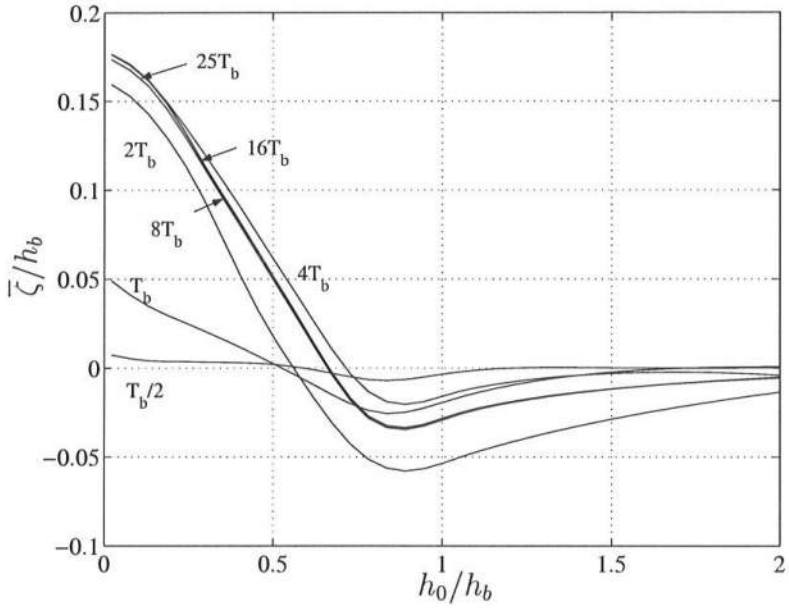

Figure 14: The variation of $\overline{\zeta}$ versus depth for the example at different times based on the parameter T_b defined by (5.2).

no and the max depth in the computational domain. It is printed to the screen. The value is $\Delta t = 0.065\ s$

The wave height variation in the cross-shore direction is shown in fig 10.

As the motion starts from rest the first major change is the generation of the setup in the surf zone. This is illustrated in fig 10 which shows the time variation of the mean water surface versus the depth relative to the breaker depth h_b at different values of the time after the start at $t = 0$. The timescale T_b used is defined as

$$T_b = L_b/\sqrt{gh_b} \tag{5.2}$$

where L_b is the surf zone width. Thus T_b is half the time it takes a long wave to propagate from the breaking point to the shoreline. For the experiment shown $T_b = 6.12s$. We see that the setup has been completely established already after approximately $5 - 7T_b$ (32-45 s).

Hence, with a value of the *delay*-parameter of $10s$ the first $1 - 2 * T_b$ of the startup of the cross-shore setup is flavored by the ramping up of the forcing. However, for $t > 2 - 3T_b$ (15-20 s) the forcing has developed completely and hence at that time the growth of the setup shows the fast response of the cross-shore variation to the steady forcing.

Fig 15 shows the equivalent cross-shore variation of the cross-shore depth averaged velocities ($\tilde{U}$) at different times. In accordance with the description above we see that the cross-shore velocities reach their maximum after just about $2T_b$ and quickly die down as the steady cross-shore balance develops.

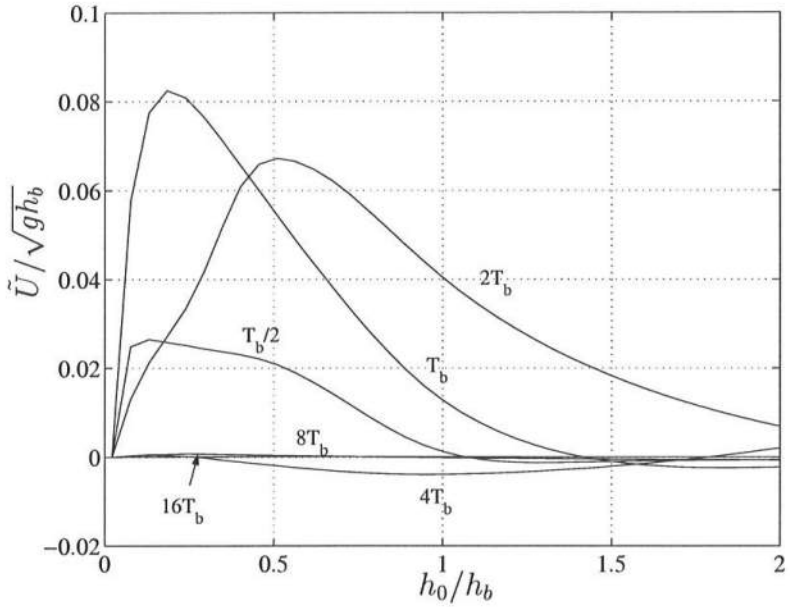

Figure 15: The cross-shore velocities $\tilde{U}/\sqrt{gh_b}$ versus depth for the example at different times based on the parameter T_b defined by (5.2).

In contrast to the cross-shore flow, the longshore flow develops quite slowly. Fig 16 shows the longshore current velocities. Here the timescale for the growth of the flow is clearly much longer. Even after $16T_b$ ($\sim 100s$) the velocity has only reached about 85% of the full value.

I may be worth mentioning that, because we use a periodic boundary condition in the longshore direction there are no longshore oscillations from the start up. If, however, a flux condition is specified at each of the cross-shore boundaries this would need to be ramped up at the same rate as the start-up rate for the longshore uniform longshore current in the middle of the computational domain in order to prevent longshore oscillations to develop. Such oscillations would typically have a time scale of $T_l = L_y/\sqrt{gh_b}$ which could be much longer than T_b. Furthermore, since cross-shore boundaries with a flux condition will fully reflect all (oscillatory) deviations from the specified flux, such oscillations will generally remain present for a long time and essentially only propagate out of the computational domain through the absorbing offshore boundary (as they turn into 2DH oscillations due to the cross-shore depth variation).

As mentioned the velocities shown in the previous figures are depth averaged velocities. However, the actual velocities vary over depth. Fig 17 shows the vertical profiles at three different positions (at $h/h_b = 1.5$, 1.0 and 0.23) at three different times ($t = 2T_b$, $6T_b$, and $20T_b$ respectively).

At $h/h_b = 1.5$, which is well outside the surf zone the longshore velocities are quite small and the cross-shore velocities dominate.

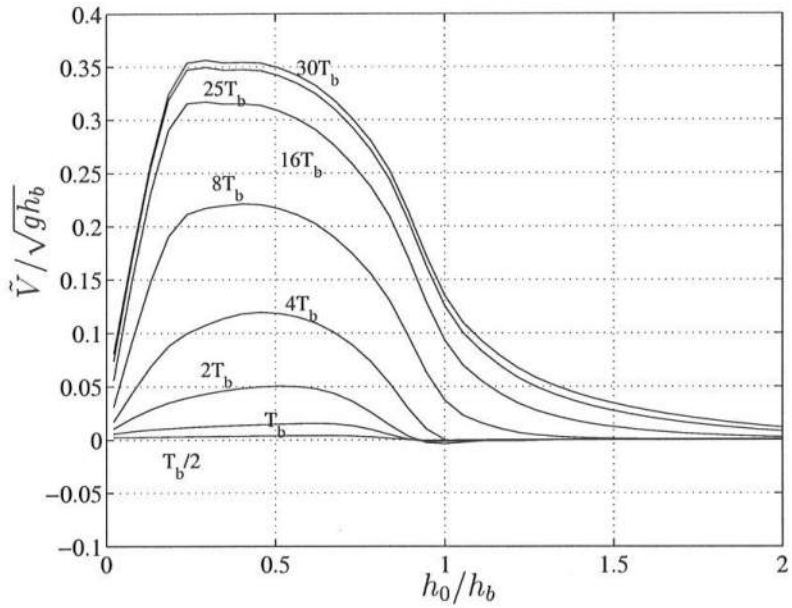

Figure 16: The longshore velocities $\tilde{V}/\sqrt{gh_b}$ for the example versus depth at different times based on the parameter T_b defined by (5.2).

Around the breaking point the two velocity components are of the same order but not evenly distributed over depth which cause the velocity profile to look like a spiral. Due to the difference in timescales the shape of the profile also changes with time.

Finally near the shore the longshore current grows to become the more dominant component after sufficiently long time, though the undertow at the bottom is still strong enough to significantly turn the velocity vector in an offshore direction.

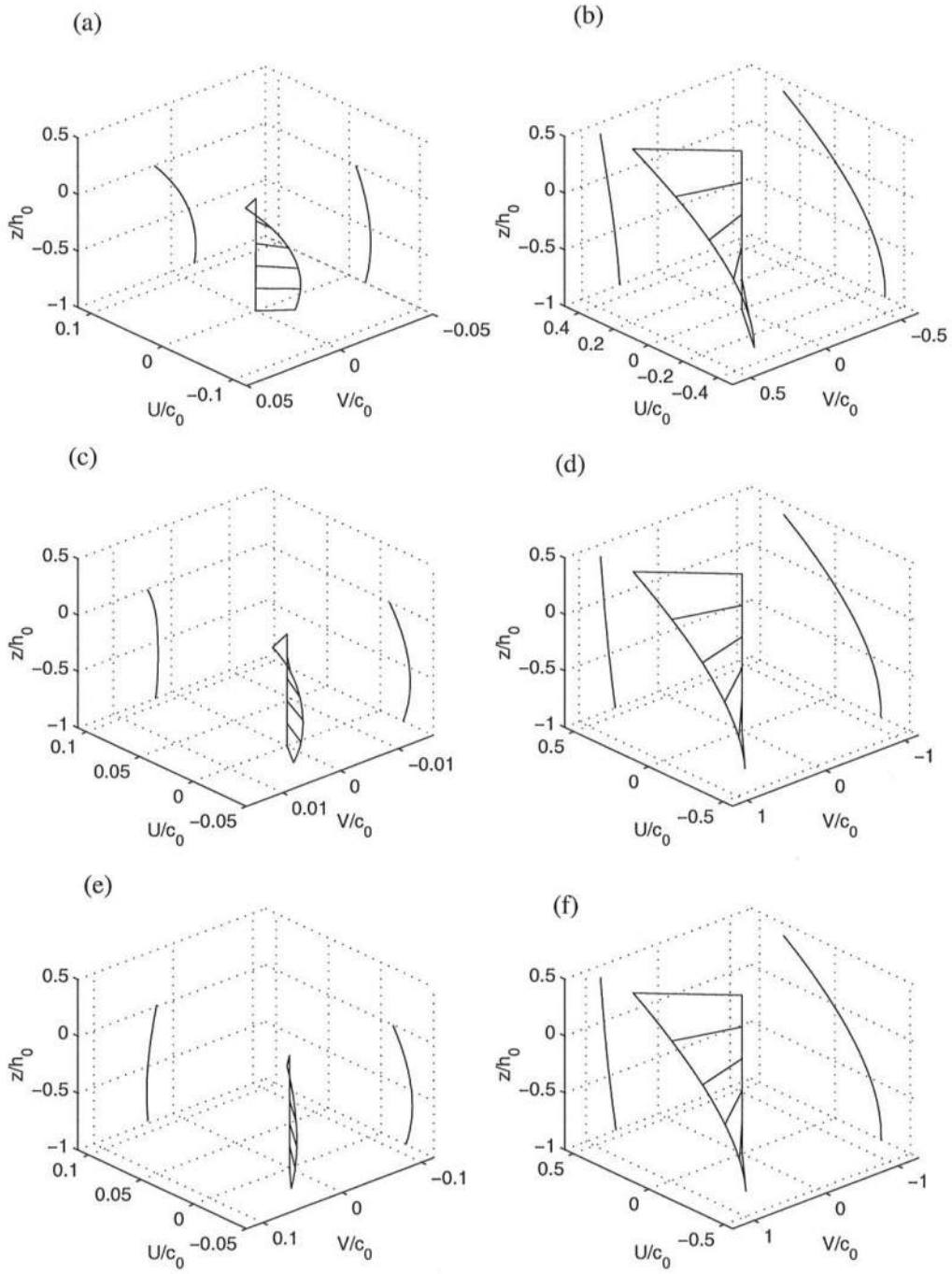

Figure 17: Development of vertical velocity profiles. Outside the surf zone (a,c,e) $h_0/h_b = 1.5$ and inside the surf zone (b,d,f) $h_0/h_b = 0.23$ at time $t = 6T_b$ (a,b), $t = 20T_b$(c,d) and $t = 48T_b$ (e,f)

6 List of symbols

u_α	m/s	total velocity in the α (= x or y) direction
$u_{w\alpha}$	m/s	wave particle velocity in the α direction
α	-	tensor index (= 1 or 2 for x or y direction)
V_α	m/s	$V_\alpha(x, y, z, t)$ total current velocity
Q_α	m^2/s	total wave averaged volume flux through vertical
$Q_{w\alpha}$	m^2/s	wave averaged volume flux due to short wave motion
$V_{m\alpha}$	m/s	depth uniform part of V_α
$V_{d\alpha}$	m/s	depth varying part of V_α
$\overline{\zeta}$	m	mean water level
x	m	horizontal coordinate (usually "cross-shore")
y	m	horizontal coordinate (usually "longshore")
t	s	time
z	m	vertical coordinate
h_0	m	local distance from $z = 0$ to bottom
h	m	local depth $= h_0 + \overline{\zeta}$
ζ	m	vertical distance from $z = 0$ to local water surface
$\overline{\zeta}$	m	distance from $z = 0$ to local mean water surface
ζ_t	m	distance from zero to short wave trough
p	N/m^2	pressure
$\delta_{\alpha\beta}$		Kronecker δ
ρ	t/m^3	density
$S_{\alpha B}$	N/m	radiation stress
$V_{d\alpha}^{(0)}$, $V_{d\alpha}^{(1)}$	m/s	components of $V_{d\alpha}$
ξ	m	distance from the bottom
$\tau_{\alpha x}^B$	N/m^2	bottom shear stress
τ_s	N/m^2	surface shear stress (wind stress)
$A_{\alpha\beta\gamma}$	m^2/s	3-D dispersive mixing coefficient
$B_{\alpha\beta}$	m^2/s	3-D dispersive mixing coefficient
$D_{\alpha\beta}$	m^2/s	3-D dispersive mixing coefficient
$M_{\alpha\beta}$	m^3/s^2	3-D dispersive mixing coefficient
S_m	N/m	momentum component of radiation stress
S_p	N/m	pressure component of radiation stress
$e_{\alpha\beta}$		directional tensor for radiation stress
G		$2kh/\sinh 2kh$
k	1/m	numerical value of wave number vector
k_{alpha}	1/m	wave number vector
B		$\overline{\eta^2}/H^2$
A	m^2	roller area
H	m	wave height
g	m/s^2	acceleration of gravity
L	m	short wave length

c	m/s	short wave phase velocity
$\tau_{\alpha\beta}$	N/m^2	horizontal turbulent stress on vertical internal surface
C_1		empirical constant for the eddy viscosity
κ		von Karman's constant
f_w		bottom friction factor
M		empirical constant for the eddy viscosity
ν_t	m^2/s	eddy viscosity
C_s		empirical coefficient for the Smagorinsky eddy viscosity
u_0	m/s	bottom particle velocity amplitude in short wave motion
θ		$\omega t - \int k_\alpha d_{x_\alpha}$ short wave phase
μ		angle between wave particle motion and current
$\beta_1,\ \beta_2$		bottom shear stress weighting coefficients defined by (3.57) and (3.57).
V_b	m/s	numerical value of current velocity V_{bd} at the bottom
C_D		empirical coefficient for wind
$W_\alpha,\ W$	m/s	stress, wind velocity vector, numerical value of w_d.

7 List of references

- Battjes, J. A. (1975): Modeling of turbulence in the surf zone. *ASCE Proc. Symp. on Modelling Techniques*, San Francisco, 1050-1061.
- Church, J.C. and E. B. Thornton (1993): Effects of breaking wave induced turbulence within a longshore current model. *Coastal Engineering* **20**, 1-28.
- Coffey, F.C. and P. Nielsen (1984): Aspects of wave current boundary layer flows. In *Proc., 19th Int. Conf. on Coast. Engrg.*, ASCE, 2232-2245.
- Drei, E. , A. Lamberti, and I. A. Svendsen (2000). Current analysis around a submerged breakwater. Proc. IVth Int. Conf. Hydrodyn. Yokohama, (eds Goda, Ikehata, Suzuki). 693 – 698
- Haas, K. and I.A. Svendsen and M.C. Haller (1998): Numerical modeling of nearshore circulation on a barred beach with rip channels. In *Proc., 26th Int. Conf. on Coast. Engrg.*, ASCE, 801-814.
- Haas, K. A. & I. A. Svendsen (2000a). Three-dimensional modelling of rip current systems. Rep. No. CACR-00-06, 250 pp.
- Haas, K. A. , I. A. Svendsen (2000b). 3-D modeling of rip currents. ICCE2000, Sydney, Australia,
- Haas, K. A. , I. A. Svendsen (2002). Laboratory measurements of the vertical structure of rip currents. To appear in JGR.
- Haas, K. A. , I. A. Svendsen, M. C. Haller and Q. Zhao (2002). Quasi 3-D modelling of rip currents: horizontal properties. Submitted for publication.
- Hansen, J. B. (1990) Periodic waves in the surf zone: Analysis of experimental data *Coastal Engineering.* **14**, 19–41.
- Kirby, J.T. and R.A. Dalrymple (1994): Combined refraction/diffraction model REF/DIF1, Version 2.5. *Res. Report CACR-94-22*, Center for Applied Coastal Research, Univ. of Delaware.
- Longuet-Higgins, M.S. (1956): The mechanics of the boundary-layer near the bottom in a progressive wave. *Proc. 6th Int. Conf. Coast. Engrg.*, ASCE, 184-193.
- Longuet-Higgins, M.S. (1970). Longshore currents generated by obliquely incident sea waves. Parts 1 and 2. *Journal of Geophysical Research, 75*, pp. 6778-6789 and pp. 6790-6801.
- Mei, C.C. (1983, 1989): The Applied Dynamics of Ocean Surface Waves. Singapore: *World Scientific.* PP. 740
- Nadaoka, K. and T. Kondoh (1982): Laboratory measurements of velocity field structure in surf zone by LDV. *Coastal Engineering in Japan* **25**, 125-145.
- Okayasu, A. ,T. Shibayama and N. Mimura (1986): Velocity field under plunging breakers. *Proc. 20th Int. Conf. Coast. Engrg.*, ASCE, 660-674.
- Putrevu, U. and I. A. Svendsen (1995): Vertical structure of the undertow outside the surf zone. *J. Geophys. Res.* **98**(C12), 22,707-22,716.
- Putrevu, U. and I. A. Svendsen (1999): Three-dimensional dispersion of momentum in wave-induced nearshore currents. *Eur. J. Mech. B/Fluids*, 83-101.
- Sancho, F. E. , I. A. Svendsen, A. R. Van Dongeren and U. Putrevu (1995): Longshore nonuniformities of nearshore currents. *Coastal Dynamics'95*, 425-436.

- Sancho, F. E. and I. A. Svendsen (1997): Unsteady nearshore currents on longshore varying topographies. *Res. Report CACR-97-10*, Center for Applied Coastal Research, University of Delaware.
- Sancho, F. E. and I. A. Svendsen (1998): Shear waves over longshore nonuniform barred beaches. In *Proc., 26th Int. Conf. on Coast. Engrg.*, ASCE, 230-243.
- Sancho, F. , C. J. Fortes, J. L. Fernandes and I. A. Svendsen (1999). On the wave field and wave-induced currents around a detached breakwater", ASCE Proc Coastal Structures Conf, Santander.
- Shapiro, R. (1970): Smoothing filtering and boundary effects. *Reviews of Geophysics and Space Physics* **8**(2), 359-387.
- Svendsen, I. A. (1984) Mass flux and undertow in a surf zone. *Coastal Engineering.* **8**, 347-365.
- Svendsen, I.A., H. A. Schäffer, and J. Buhr Hansen (1987): The interaction between undertow and the boundary layer flow on a beach. *J. Geophys. Res.* **92**(C11), 11,845-11,856.
- Svendsen, I. A. (1987): Analysis of surf zone turbulence. *J. Geophys. Res.* **92**(C5), 5115–5124,
- Svendsen, I. A. and U. Putrevu (1990): Nearshore circulation with 3-D profiles. *Proc 22th Int. Conf. Coastal Engrg.* , ASCE, 241-254.
- Svendsen, I. A. and U. Putrevu (1993): Surf-zone wave parameters from experimental data. *Coastal Engineering*, **19**, 282-310.
- Svendsen, I. A. and U. Putrevu (1994): Nearshore mixing and dispersion. *Proc. Roy. Soc. Lond* A **445**, 561-576.
- Svendsen, I. A., and U. Putrevu (1996): Surf Zone Hydrodynamics, Review paper to in *"Advances in Coastal and Ocean Engineering"*, vol 2, World Scientific Publ., 1 - 79.
- Svendsen, I. A. F. E. Sancho J. Oltman-Shay, & E. B. Thornton (1997). Modelling nearshore circulation under field conditions. Proceedings ASCE Waves'97 conference, Virginia Beach, p 765 – 776.
- Svendsen, I. A. and K. Haas (1999): Interaction of undertow and rip currents. In *Proc 5th Int. Conf. Coastal and Port Engrg. Developing Countries*, ASCE, 218-229.
- Svendsen, I. A. and K. A. Haas (2000). Analysis of rip current systems. ICCE2000, Sydney, Australia,
- Van Dongeren, A. R. , F. E. Sancho, I. A. Svendsen and U. Putrevu(1994): SHORECIRC: A quasi 3-D nearshore model. In *Proc., 24th Int. Conf. on Coast. Engrg.*, ASCE, 2741-2754.
- Van Dongeren, A. R. , I. A. Svendsen and F. E. Sancho (1995): Application of the Q-3D SHORECIRC model to surfbeat. *Coastal Dynamics'95*, 233-244.
- Van Dongeren, A. R. , I. A. Svendsen and F. E. Sancho (1996): Generation of infragravity waves. In *Proc., 25th Int. Conf. on Coast. Engrg.*, ASCE, 1335-1348.
- Van Dongeren, A. R. and I. A. Svendsen (1997a): Quasi 3-D modeling of nearshore hydrodynamics. Res. *Report CACR-97-04*, Center for Applied Coastal Research, University of Delaware.
- Van Dongeren, A. R. and I. A. Svendsen (1997b): An absorbing-generating boundary condition for shallow water models. *J. Waterway, Port, Coastal, and Ocean Eng.* **123**(6), 303-313.
- Van Dongeren, A. R. , I. A. Svendsen and U. Putrevu (1998): Quasi 3-D effects in infragravity waves. In *Proc., 26th Int. Conf. on Coast. Engrg.*, ASCE, 1323-1336.

- Van Dongeren, A. R. , I. A. Svendsen (2000). Nonlinear and quasi 3-D effects in leaky infragravity waves. Coastal Engineering vol. 41, 4, pp. 467-496.
- Visser, P.J. (1984). A mathematical model of uniform longshore currents and comparison with laboratory data. *Communications on Hydraulics. Report 84-2*, Department of Civil Engineering, Delft University of Technology, 151 pp.
- WAMDI Group (1988): The WAM model: A third generation ocean wave prediction model. *J. of Physical Oceanography* **18**, 1775-1810.
- Wei, G. and J. T. Kirby (1995). A time dependent numerical code for extended Boussinesq equations. *J. Waterway, Port, Coastal, and Ocean Eng.* **120**, 251-261.

  flows.
- Zhao, Q. , I. A. Svendsen, and K. A. Haas (2002). Three-dimensional analysis of shear waves. (submitted for publication).

8 The SHORECIRC code

This section contains a listing of the code for SHORECIRC. Each file is contained within its own subsection.

8.1 File winc_std_main.f

The file *winc_std_main.f* contains the main program as well as most subroutines called by the main program.

8.1.1 Program winc

This is the main program which calls all the subroutines

```
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
c-------------------------------------------------------------------------------|
c| WINC - Wave Induced Nearshore Circulation
c| Standard Version of SHORECIRC
c|
c| Version 2.0
c|
c| Ib A Svendsen, Kevin A Haas and Qun Zhao
c|
c| NOTICE:
c| The record_length in the direct write to files is as:
c| - CRAY     : recl=1*ny
c| - SGI      : recl=1*ny
c| - Linux/PC : recl=4*ny
c| - Sun      : recl=4*ny
c|
c| LAST MODIFICATION:
c| - May 6, 2002
c| Kevin A Haas
c-------------------------------------------------------------------------------|

c ##############################################################################
c MAIN PROGRAM
      program winc
c ##############################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i
```

```
c=====================================================================

      i=1
      call constantvals
      call inputvals
        depthmax=3.001d0
      call numerical
      call bathymetry
      call numerical
      call init_output
      write(*,901) 'Load stresses? (0=no, 1=yes) : '
      read(*,*) i
      if (i.eq.0) stop
      call refract_longwaves
      call friction_factor
      call shortwave_driver
      call wave_forcing
      write(*,901) 'Start current model? (0=no, 1=yes) : '
      read(*,*) i
      if (i.eq.0) stop
      call initialsteps
      call time_model
      write(*,902) 'end'

901   format(/A,$)
902   format(/A/)

      stop
      end
```

8.1.2 Subroutine constantvals

This subroutine sets the constants used throughout the program.

```
c ######################################################################
c CONSTANT VALUES
      subroutine constantvals
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'
```

```
c=====================================================================

      pi=4.d0*datan2(1.d0,1.d0)
      g=9.81d0
      rho=1030.d0
      rhod=1.d0/rho

      return
      end
```

8.1.3 Subroutine inputvals

This subroutine reads the input file *input_winc.dat.*

```
c ######################################################################
C INPUT VALUES
      subroutine inputvals
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      character*35 flestr

      namelist
     .        /boundary/ibc1,ibc2,ibc3,ibc4
     .        /grids/dx,dy,nx,ny,nxwall,httide
     .        /short_wave/period,wc,amerge,files
     .        /physics/fcw,vtshear,mdiss,Cs,wind_vel,wind_dir,disp3d
     .        /control/cr,ntime,kold_start,depthmin,delay
     .        /sensor/jump,xsensor, ysensor, yprofile
     .        /output/interval,tavgout,momout,dispout

c=====================================================================
c--- default initializations
      interval=400
      kold_start=1
      cr=0.5d0
      ntime=201
      fcw = 0.01d0
      vtshear = 0.05d0
      mdiss = 0.2d0
```

```
      jump=10
c--- read from namelist fields in file "input_winc.dat"
c ## 'period' is short wave period
c ##  koldstart : 1=yes 0=no
      flestr= 'input_winc.dat'
      open (unit=11, file=flestr, status='old')
      read(11,nml=boundary)
      read(11,nml=grids)
      read(11,nml=short_wave)
      read(11,nml=physics)
      read(11,nml=control)
      read(11,nml=sensor)
      read(11,nml=output)

      close(11)

c---  compute the inverse of the delay
      delay = 1.d0/delay


c--- boundary conditions
      iperx=0
      ipery=0
      if (ibc1.eq.5 .and. ibc2.eq.5) iperx=1
      if (ibc3.eq.5 .and. ibc4.eq.5) ipery=1

c--- long wave parameters
c ## alpha(1) corresponds to the angle of the incoming long-wave
c ## (if it's generated) and affects the CHARACTERISTICS @ x=0 bound.
      alpha(1)=0.d0
      alpha(1) = alpha(1)/180.d0*pi
      ampli=0.d0

c--- write control variables on screen
      write(6,*)'nx,ny,nxwall,httide:    ',nx,ny,nxwall,httide
      write(6,901) 'ibcs: ', ibc1,ibc2,ibc3,ibc4
      write(6,902) 'Periodicy in x & y :',iperx,ipery
      write(6,903) 'fcw, vtshear, mdiss, Cs :',fcw,vtshear,mdiss,Cs


901   format(1X,A,4I4)
902   format(1X,A,2I4)
903   format(1X,A,4F12.6)
```

```
      return
      end
```

8.1.4 Subroutine bathymetry

This subroutine reads the bathymetry from the file *bath.dat*.

```
c ######################################################################
c BATHYMETRY
      subroutine bathymetry
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i,j,ncode,nxflag,xcor,ycor,err
      real*8 fake(nxm,nym),fake1(nxm,nym),fake2(nxm,nym),
     . xc(nym),temp,temp1,temp2

c=====================================================================

      write(6,901) 'Make bathymetry'
c ## Initialize dummy variables <- Necessary b/c of call to DERIV !
      do j=1,ny
         do i=1,nx
    fake(i,j) = 0.d0
    fake1(i,j) = 0.d0
    fake2(i,j) = 0.d0
            ht(i,j)=9999
         end do
      end do

c ##  Read the bathymetry file bath.dat
c     Provides an error check
ckk 9/14/99

open(unit=19,file='bath.dat', status='old')
do j=1,ny
  do i=1,nx
    read(19,*)temp1,temp2,temp
    xcor=nint(temp1/dx+1)
```

```
            ycor=nint(temp2/dy+1)
            if (nint(10000*x(xcor)).ne.nint(10000*temp1).or.
     .           nint(10000*y(ycor)).ne.nint(10000*temp2)) then
               write(*,*)'x=',temp1,' y=',temp2,' not on grid'
            else if (ht(xcor,ycor).ne.9999) then
               write(*,*)'x=',temp1,' y=',temp2,' duplicated'
            else
               ht(xcor,ycor)=temp+httide
            end if
          end do
end do
        err=0
      do j=1,ny
         do i=1,nx
            if (ht(i,j).eq.9999) then
            write(*,*)'x=',x(i),' y=',y(j),' missing'
            err=1
            end if
         end do
      end do

      if (err.eq.1) stop

      if (ibc2.eq.4) then
         do j = 1,ny
            nxa(j)=nxwall
         end do
      end if

      if (ibc2.eq.6) then
         do j = 1,ny
         nxa(j)=nx
          do i = 1,nx
c-##---- shoreline position
            if ((ht(i,j)).le.depthmin) then
               if ((ht(i-1,j)).ge.depthmin) nxa(j)=i-1
            end if
          end do
         end do
      end if

      write(6,*) 'nxa(j)'
      write(6,*) (nxa(j),j=1,ny)
```

```
c<==================added sep.7,1999 qun ==============>
c hs:still water level, hx:bottom slope
c      hs=3.0
c      hx=1.0/20.
c      do j=1,ny
c        do i=1,nx
c        ht(i,j)=hs-float(i-1)*dx*hx
c        end do
c      end do

c<==================added sep.7,1999 qun ==============>

c ## Define bathymetry for a barred beach (such as Duck)

c--- bed profile
c      do i = 1,nx
c  if (x(i) .le. 365.) then
c            do j=1,ny
c        ht(i,j) = 0.55d0*(370.d0-x(i))**0.35d0
c            end do
c  else
c            do j=1,ny
c        ht(i,j) = 0.55d0*5.d0**0.35d0 -0.06d0*(x(i)-365)
c            end do
c  endif
c      end do

c--- crest position and 'bed profile'+'bar'
c      do j=1,ny
c         xc(j) = 0.65d0*400.d0
c      end do
c      do j=1,ny
c         do i=1,nx
c            ht(i,j) = ht(i,j) - 1.d0/150.d0*(xc(j)+250.d0-x(i))
c     . *dexp(-15.d0*(x(i)-xc(j))**2.d0/(xc(j)*xc(j)))
c         end do
c      end do



c ## find maximum water depth for evaluation of Courant no.
      depthmax=0.1d0
```

```
      do j=1,ny
         do i=1,nxa(j)
            hto(i,j) = ht(i,j)
            depthmax=DMAX1(depthmax,ht(i,j))
         if (ht(i,j).le.depthmin.and.ibc2.ne.6) ht(i,j)=depthmin
         end do
      end do

      write(*,902) ' depthmax =',depthmax



c ## Write bathymetry for REFDIF


      open(31,file='fort.311',access='direct',recl=4*ny)
      do i=1,nx
write(31,rec=i) (sngl(ht(i,j)),j=1,ny)
      end do

      close(31)

c ## Calculate depth gradients

      ncode = 3
      nxflag = 1
      call deriv(ht,fake,fake,dhodxn,dhodyn,fake1,fake2,
     .           ncode,nxflag)
      open(29,file='fort.312',access='direct',recl=4*ny)
      open(30,file='fort.313',access='direct',recl=4*ny)
      do i=1,nx
write(29,rec=i) (sngl(dhodxn(i,j)),j=1,ny)
write(30,rec=i) (sngl(dhodyn(i,j)),j=1,ny)
      end do
      close(29)
      close(30)


901   format(/A)
902   format(/A,F20.4)
      return
      end
```

8.1.5 Subroutine numerical

This subroutine sets the numerical grid.

```
c ########################################################################
c NUMERICAL SCHEME AND GRID DEFINITION
      subroutine numerical
c ########################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i,j
      real*8 step

c=======================================================================

      step = dx*dy / DSQRT(dx*dx+dy*dy)
      dt = cr*step/DSQRT(g*depthmax)


      x(1)=0.d0
      do i = 2,nx
        x(i) = x(1) + DBLE(i-1)*dx
      end do

      y(1)=0.d0
      do j = 2,ny
        y(j) = y(1) + DBLE(j-1)*dy
      end do


      return
      end
```

8.1.6 Subroutine refract_longwaves

This subroutine calculates the angle of incidence of long waves entering the domain.

```
c ######################################################################
c REFRACT LONG WAVES
      subroutine refract_longwaves
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i,j
      real*8 cp
c======================================================================

      wlec = 50.d0
      cp = DSQRT(g*depthmax)
      sigma = 2.d0*pi/wlec*cp
      j=ny/2
      do i=1,3
        alpha(i)=dasin(DSQRT(g*ht(i,j))*sin(alpha(1))/DSQRT(g*ht(1,j)))
        k(i)=sigma/DSQRT(g*ht(i,j))
      end do

      return
      end
```

8.1.7 Subroutine shortwave_driver

This subroutine controls the short-wave driver. It either calls Ref/Dif or reads the data from files and then calculates the short-wave parameters used in the model.

```
c ######################################################################
c SHORTWAVE DRIVER
      subroutine shortwave_driver
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'
      include 'pass.inc'

c--- Local variables ------------------------------------------------
      integer i, j, ncode, ii, nxflag, index,iflag,merge
      real*8 Sxx(nxm,nym), Sxy(nxm,nym), Syy(nxm,nym),
     . fake(nxm,nym),wavenumber,
     . capG,ursell,aux,
```

```
     . xfake(nxm),y2(nxm),qwrol(nxm),smrol(nxm),
     . A(nxm),Afake(nxm)
  real*8 dr(nxm,nym),ur(nxm,nym),avgdhdx,roloff,
     . vr(nxm,nym)
c=======================================================================

c read from files version
        if (files.eq.1) then
           open(unit=8, file='height.dat')
           open(unit=9, file='angle.dat')
           open(unit=1, file='sxx.dat')
           open(unit=2, file='sxy.dat')
           open(unit=3, file='syy.dat')
           open(unit=4, file='qw.dat')
           open(unit=10, file='diss.dat')
           open(unit=11, file='u0.dat')
           open(unit=12, file='k.dat')
           open(unit=13, file='freq.dat')

      do i=1,nx
read(8,*) (H(i,j),j=1,ny)
read(9,*) (theta(i,j),j=1,ny)
read(1,*) (Sxx(i,j),j=1,ny)
read(2,*) (Sxy(i,j),j=1,ny)
read(3,*) (Syy(i,j),j=1,ny)
read(4,*) (Qw(i,j),j=1,ny)
read(10,*) (dissipation(i,j),j=1,ny)
read(11,*) (u0(i,j),j=1,ny)
read(12,*) (wn(i,j),j=1,ny)
read(13,*) (freq(i,j),j=1,ny)
      end do


       close(1)
       close(2)
       close(3)
       close(4)
       close(8)
       close(9)
       close(10)
       close(11)
       close(12)
       close(13)
```

```
      do j=1,ny
do i=1,nx
          c_sw(i,j)=freq(i,j)/wn(i,j)
          sinth(i,j)= dsin(theta(i,j))
          costh(i,j)= dcos(theta(i,j))
          if (dissipation(i,j).gt.0) then
           ibrk(i,j)=1
          else
            ibrk(i,j)=0
          end if
end do
      end do

        call viscosity


c R/D version begins here
       else

c     create bathymetry and currents for REFDIF
do j=1,ny
c     Include 2 extra rows on offshore boundary
           do i=1,2
              dr(i,j)=(ht(1,j)+zetan(1,j))
              ur(i,j)=((qxn(1,j)-qwx(1,j))/(ht(1,j)+zetan(1,j)))
              vr(i,j)=((qyn(1,j)-qwy(1,j))/(ht(1,j)+zetan(1,j)))
           end do
      do i=1,nxa(j)
    dr(i+2,j)=(ht(i,j)+zetan(i,j))
    ur(i+2,j)=((qxn(i,j)-qwx(i,j))/(ht(i,j)+zetan(i,j)))
    vr(i+2,j)=((qyn(i,j)-qwy(i,j))/(ht(i,j)+zetan(i,j)))
  end do
          do i=nxa(j)+1,nx
    ur(i+2,j)=0
    vr(i+2,j)=0
          end do
end do


c      Call REFDIF as a subroutine
       call refdif(dr,ur,vr,nx+2,ny)
```

```
c     read wave input from common variables
c---when using REFDIF as a shortwave driver with nonzero initial angle
c!! Two aditional rows at the begining:

      do i=1,nx
        do j=1,ny
  H(i,j)=cwh(i+2,j)
  c_sw(i,j)=ccel(i+2,j)
  theta(i,j)=ctheta(i+2,j)*pi/180.d0
  ibrk(i,j)=cibr(i+2,j)
end do
      end do

c<==================added sep.7,1999 qun ==============>
c      output incident wave condition and calculation setup
       open(99,file='shorecirc.log')
       wavenumber = 2.d0*pi/period /c_sw(1,1)
       wavel=c_sw(1,1)*period
       ursell=H(1,1)*wavel*wavel/ht(1,1)**3
        do ii=1,2
        write(*,*)
        end do
        write(*,1)
        write(*,2)
        write(*,3)
        write(*,4)
        write(*,5)
        write(*,6)
        write(*,7)
        write(*,8)
        write(*,9)
       do ii=1,6
       write(*,10)
       end do
       write(*,11) sngl(ht(1,1))
       write(*,12) sngl(H(1,1))
c       write(*,13) sngl(hx)
       write(*,15) sngl(period)
       write(*,16) sngl(fake(1,1))
       write(*,17) sngl(H(1,1)/ht(1,1))
       write(*,18) sngl(ht(1,1)/wavel)
c       write(*,19) sngl(H(1,1)*(c_sw(1,1)*period)**2/(ht(1,1)**3))
       write(*,19) sngl(ursell)
```

```
c--- write control variables on screen
      write(*,20)  ibc1,ibc2,ibc3,ibc4
      write(*,21) iperx,ipery
      write(*,22) fcw, vtshear,mdiss
      write(*,23) nx,ny,ntime
      write(*,25) dx,dy,dt
       write(*,1)
 20   format(2x,9h#   ibcs:,21x,4I4,6x,1h#)
 21   format(2x,24h#   Periodicy in x & y :,15x,2I4,5x,1h#)
 22   format(2X,25h#   fcw, vtshear, mdiss :,3F8.3,3x,1h#)
 23   format(2x,19h#   nx , ny , ntime,3I8,9x,1h#)
 25   format(2x,16h#   dx , dy , dt:,3F10.3,6x,1h#)

      do ii=1,2
       write(*,*)
       end do
       write(99,1)
       write(99,2)
       write(99,3)
       write(99,4)
       write(99,5)
       write(99,6)
       write(99,7)
       write(99,8)
       write(99,9)
       do ii=1,6
       write(99,10)
       end do
       write(99,11) sngl(ht(1,1))
       write(99,12) sngl(H(1,1))
c       write(99,13) sngl(hx)
       write(99,15) sngl(period)
       write(99,16) sngl(fake(1,1))
       write(99,17) sngl(H(1,1)/ht(1,1))
       write(99,18) sngl(ht(1,1)/wavel)
c       write(99,19) sngl(H(1,1)*(c_sw(1,1)*period)**3/(ht(1,1)**3))
       write(99,19) sngl(ursell)
       write(99,1)

 1     format(2x,
     &    53h#######################################################)
 2      format(2x,
     &   53h#     Quasi-3D Nearshore Circulation Model          #)
```

```
 3      format(2x,
     &   53h#                    SHORECIRC                          #)
 4      format(2x,
     &   53h#                    IAS Group                          #)
 5      format(2x,
     &   53h#     Center for Applied Coastal Research               #)
 6      format(2x,
     &   53h#              University of Delaware                   #)
 7      format(2x,
     &   53h#               Newark, DE 19716                        #)
 8      format(2x,
     &   53h#                     U.S.A.                            #)
 9      format(2x,
     &   53h#                 September  1999                       #)

 10      format(2x,
     &   53h#                                                       #)
 11     format(2x,
     &   37h#   Offshore-shore still water level=,f10.3,6h         #)
 12     format(2x,
     &   37h#   Incident short  wave   height  H=,f10.3,6h         #)
 13     format(2x,
     &   37h#   Bottom        slope          hx=,f10.3,6h         #)
 15     format(2x,
     &   37h#   Short     wave   period        T=,f10.3,6h         #)
 16     format(2x,
     &   37h#   Incident  short  wave      angle=,f10.3,6h         #)
 17     format(2x,
     &    37h#   Relative  wave   height     H/ht=,f10.3,6h        #)
 18     format(2x,
     &    37h#   Relative   water   depth     h/L=,f10.3,6h        #)
 19     format(2x,
     &    37h#   Ursell      number      HL^2/h^3=,f10.3,6h        #)

c<==================added sep.7,1999 qun ==============>



c---Calculate sin(theta) and cos(theta)  to be used globally
      do j=1,ny
do i=1,nx
          sinth(i,j)= dsin(theta(i,j))
          costh(i,j)= dcos(theta(i,j))
```

```
end do
      end do



c ## Calculate short-wave particle velocity : u0     -----------------
c ## use "c" (read on unit 10) from refdif to get k

      do j=1,ny
        do i=1,nx
  wavenumber = 2.d0*pi/period /c_sw(i,j)
          u0(i,j) = H(i,j)*pi/period /dsinh(wavenumber*ht(i,j))
  if (ht(i,j).le.depthmin) then
     wavenumber = 2.d0*pi/period /dsqrt(g*depthmin)
             u0(i,j) = H(i,j)*pi/period /dsinh(wavenumber*depthmin)
  end if
end do
      end do

        call viscosity

c-- linear wave theory; long-crested waves outside surfzone
c      use a correction with roller inside surfzone

      write(6,*) 'Compute radiation stresses '

c --- look for 1st breakpoint along each j and compute Radiation Stresses
      do j=1,ny
c------ find breaking point: find first ibrk(i,j)=1
      index = 1
      iflag = 0
        do i=1,nxa(j)
   if (ibrk(i,j).eq.1 .and. iflag.eq.0) then
     index=i
             iflag=1
   end if
end do
        index=index-1

c     Calculate transition distance
        merge=dint(amerge*ht(index,j)/dx)
        merge=max(merge,3)
        merge=min(merge,7)
```

```
        roloff=1

c     Turn roller off if amerge=0
        if (amerge.eq.0) then
           merge=0
           roloff=0
        end if
c  Check to make sure merge does not extend past shoreline
        if ((index+merge).gt.nxa(j)) then
           merge=nxa(j)-index
        end if
c Outside the surfzone
 do i=1,index
            wavenumber = 2.d0*pi/period /c_sw(i,j)
    aux = 2*wavenumber*(ht(i,j)+zetan(i,j))
    capg = aux/SINH(aux)
    sm = 1.d0/16.d0*(1.d0+capg)
    aux = rho*H(i,j)*H(i,j)*g
            sm  = sm*aux
    sp = .5d0*rho*g*H(i,j)*H(i,j)*capg*.125d0
    sxx(i,j) = costh(i,j)*costh(i,j)*(sm) + sp
    sxy(i,j) = sinth(i,j)*costh(i,j)*(sm)
    syy(i,j) = sinth(i,j)*sinth(i,j)*(sm)  + sp
            Qw(i,j) =g*H(i,j)*H(i,j)/ c_sw(i,j)*Bo
            qwrol(i)=0.d0
            smrol(i)=0.d0
            A(i)=0.d0
            Afake(i)=0.d0
            xfake(i)=x(i)
         end do
c Roller, past transition
          do i=index+merge,nxa(j)
            A(i)=0.06d0/H(i,j)*c_sw(i,j)*period*roloff
            Afake(i-merge+1)=A(i)
            xfake(i-merge+1)=x(i)
 end do

c         write(*,*)Afake
c------ Do the cubic spline to for roller transition
        if (roloff.eq.1) then
         call spline(xfake,Afake,nxa(j)-merge,y2)
         call splint(xfake,Afake,y2,nxa(j)-merge,
     .        x,A,index,merge,j)
```

```
        end if

c------ Final value of Radiation stress and Volume flux in surfzone
          do i=index,nxa(j)
            wavenumber = 2.d0*pi/period /c_sw(i,j)
    aux = 2*wavenumber*(ht(i,j)+zetan(i,j))
    capg = aux/SINH(aux)
    sm = 1.d0/16.d0*(1.d0+capg)*rho*H(i,j)*H(i,j)*g
            qwrol(i)=A(i)/period*H(i,j)*H(i,j)
            smrol(i)=rho*c_sw(i,j)*A(i)/period*H(i,j)*H(i,j)
    sp = .5d0*rho*g*H(i,j)*H(i,j)*capg*.125d0
    sxx(i,j) = costh(i,j)*costh(i,j)*(sm+smrol(i)) + sp
    sxy(i,j) = sinth(i,j)*costh(i,j)*(sm+smrol(i))
    syy(i,j) = sinth(i,j)*sinth(i,j)*(sm+smrol(i))  + sp
            Qw(i,j) = Bo*g*H(i,j)*H(i,j)/c_sw(i,j)+qwrol(i)

 end do

c------ set wave flux at the last node to zero: Qw(nx,j)=0 ---
        Qw(nxa(j),j) = 0.d0

         do i=nxa(j)+1,nx
            sxx(i,j)=0d0
            sxy(i,j)=0d0
            syy(i,j)=0d0
            Qw(i,j)=0d0
         end do

         end do


c-- make sure forces are y-periodic
      if(ibc3.eq.5.and.ibc4.eq.5) then
      do i=1,nx
         sxx(i,1) = 0.25d0*sxx(i,ny-1)+0.25d0*sxx(i,ny)
     .              +0.25d0*sxx(i,1)+0.25d0*sxx(i,2)
         sxx(i,ny) = sxx(i,1)
         sxy(i,1) = 0.25d0*sxy(i,ny-1)+0.25d0*sxy(i,ny)
     .              +0.25d0*sxy(i,1)+0.25d0*sxy(i,2)
         sxy(i,ny) = sxy(i,1)
         syy(i,1) = 0.25d0*syy(i,ny-1)+0.25d0*syy(i,ny)
     .              +0.25d0*syy(i,1)+0.25d0*syy(i,2)
         syy(i,ny) = syy(i,1)
```

```
      end do
      end if

c-- 3-pt average along the x-dir (for each j)
      call average_3ptx(nx,ny,Qw)
      call average_3ptx(nx,ny,sxx)
      call average_3ptx(nx,ny,sxy)
      call average_3ptx(nx,ny,syy)
c-- 3-pt average along the y-dir (for each i)
      call average_3pty(nx,ny,sxx)
      call average_3pty(nx,ny,sxy)
      call average_3pty(nx,ny,syy)
      call average_3pty(nx,ny,Qw)

c     end of r/d version
      end if

c     All versions
c ## Calculate Qwx, Qwy ------------------------------------------
      do j=1,ny
        do i=1,nx
          qwx(i,j) = Qw(i,j)*costh(i,j)
          qwy(i,j) = Qw(i,j)*sinth(i,j)
        end do
      end do

c ## write out Qw values ------------------------------------------
      open(14,file='fort.326',access='direct',recl=4*ny)
      open(15,file='fort.327',access='direct',recl=4*ny)
      do i=1,nx
write(14,rec=i) (sngl(qwx(i,j)),j=1,ny)
write(15,rec=i) (sngl(qwy(i,j)),j=1,ny)
      end do
      close(14)
      close(15)


c ## compute radiation stresses' derivatives   ----------------------

      write(6,*) 'end input ; calculate forcing'
      ncode=3
      nxflag=1
      call deriv(Sxy,Sxx,Syy,dSxydxn,dSxydyn,dSxxdxn,dSyydyn,ncode,
```

```
     .  nxflag)
      do j=1,ny
        do i=1,nx
           dSxxdxn(i,j)=dSxxdxn(i,j)*rhod
           dSxydxn(i,j)=dSxydxn(i,j)*rhod
           dSxydyn(i,j)=dSxydyn(i,j)*rhod
           dSyydyn(i,j)=dSyydyn(i,j)*rhod
end do
      end do

c--- Smooth the forcing if necessary
      call average_3ptx(nx,ny,dSxxdxn)
      call average_3ptx(nx,ny,dSxydxn)
      call average_3ptx(nx,ny,dSxydyn)
      call average_3ptx(nx,ny,dSyydyn)
      call average_3pty(nx,ny,dSxxdxn)
      call average_3pty(nx,ny,dSxydxn)
      call average_3pty(nx,ny,dSxydyn)
      call average_3pty(nx,ny,dSyydyn)

c ## write output variables and continue

      write(6,*) 'Write out files with input values'
      open(14,file='fort.314',access='direct',recl=4*ny)
      open(15,file='fort.318',access='direct',recl=4*ny)
      open(16,file='fort.323',access='direct',recl=4*ny)
      open(17,file='fort.321',access='direct',recl=4*ny)
      open(18,file='fort.322',access='direct',recl=4*ny)
      open(19,file='fort.324',access='direct',recl=4*ny)
      open(25,file='fort.325',access='direct',recl=4*ny)
      open(26,file='fort.319',access='direct',recl=4*ny)
      open(27,file='fort.315',access='direct',recl=4*ny)
      open(28,file='fort.320',access='direct',recl=4*ny)
      open(29,file='fort.316',access='direct',recl=4*ny)
      open(20,file='fort.317',access='direct',recl=4*ny)

      do i=1,nx

 write(14,rec=i) (sngl(H(i,j)),j=1,ny)
 write(15,rec=i) (sngl(Sxx(i,j)),j=1,ny)
 write(16,rec=i) (sngl(dSxydyn(i,j)),j=1,ny)
 write(17,rec=i) (sngl(dSxxdxn(i,j)),j=1,ny)
 write(18,rec=i) (sngl(dSxydxn(i,j)),j=1,ny)
```

```
 write(19,rec=i) (sngl(dSyydyn(i,j)),j=1,ny)
 write(25,rec=i) (sngl(u0(i,j)),j=1,ny)
 write(26,rec=i) (sngl(Sxy(i,j)),j=1,ny)
 write(27,rec=i) (sngl(theta(i,j)/pi*180.d0),j=1,ny)
 write(28,rec=i) (sngl(Syy(i,j)),j=1,ny)
 write(29,rec=i) (sngl(c_sw(i,j)),j=1,ny)
 write(20,rec=i) (real(ibrk(i,j)),j=1,ny)
      end do

      close(14)
      close(15)
      close(16)
      close(17)
      close(18)
      close(19)
      close(25)
      close(26)
      close(27)
      close(28)
      close(29)
      close(20)



      return
      end

\subsubsection{Subroutine friction\_factor}

This subroutine sets the friction factor.

\begin{verbatim}

c ######################################################################
      subroutine friction_factor
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i,j
      real*8 a0, kn(nxm), lnfw, omega
```

```
c=====================================================================

c--- Can do either a constant or varying friction factor
c--- Currently setup to do a constant friction factor
c       omega=2.d0*pi/period
       do j=1,ny
         do i=1,nx
c     a0 = 0.5d0*period/pi*u0(i,j)
c     a0 = 0.5d0*g*H(i,j)/c_sw(i,j)/sigma
c     .           /DCOSH(omega/c_sw(i,j)*ht(i,j))
c     if (a0.le.0.10 .or. ht(i,j).le.depthmin) a0=0.1
c           lnfw = -5.977d0+5.213d0*(a0/kn(i))**(-0.194)
c           fcwij(i,j) = dexp(lnfw)
           fcwij(i,j) = fcw
         end do
       end do



       return
       end
```

8.1.8 Subroutine init_output

This subroutine initializes the output.

```
c ######################################################################
c INIT OUTPUT
      subroutine init_output
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i

c=====================================================================

c--- write file 'status.dat' basic values for MATLAB input
      open (unit=13,file='status.dat', status='unknown')
      write(13,903) dx, dy, dt, cr
      write(13,904) nx, ny, ntime, interval
```

```
      write(6,*) ' nx , ny , ntime',nx,ny,ntime
      write(6,*) ' dx , dy , dt',dx,dy,dt

      write(6,*) 'Open output files for model results'
        do i=1,nsensor
          if (xsensor(i).ne.0 .and. ysensor(i).ne.0) then
            write(i+50,997) xsensor(i), ysensor(i), jump
            write(13,904) i+50, xsensor(i), ysensor(i), jump
          endif
        end do

      close(13)

903   format(4F12.6)
904   format(4I12)
997   format( 2x, 3(I15,2x))

      return
      end
```

8.1.9 Subroutine initialsteps

This subroutine calculates the arrays for the initial timesteps.

```
c ######################################################################
c INITIAL TIME STEPS
      subroutine initialsteps
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local functions ------------------------------------------------
      real*8 cel,ux,uy,momx1,momx2,momy1,momy2,
     . force_xx, force_xy, force_yx, force_yy

c--- Local variables ------------------------------------------------

      integer i,j,ncode,nxflag

      real temp1(nxm,nym), temp2(nxm,nym), temp3(nxm,nym)
```

```
      real*8 h1, z1, t, qx1, qy1,
     . qxqy(nxm,nym), qxqx(nxm,nym), qyqy(nxm,nym),
     . qxqxdx(nxm,nym), qxqydx(nxm,nym),
     . qyqydy(nxm,nym), qxqydy(nxm,nym),
     . dqxdxn(nxm,nym),
     . dqydyn(nxm,nym),
     . Dxx(nxm,nym), Dxy(nxm,nym), Dyy(nxm,nym),dVo_dxdy(nxm,nym),
     . Mxx(nxm,nym), Mxy(nxm,nym), Myy(nxm,nym), dMxx_dx(nxm,nym),
     . dMxy_dx(nxm,nym), dMxy_dy(nxm,nym), dMyy_dy(nxm,nym),
     . disp1(nxm,nym), disp2(nxm,nym), disp3(nxm,nym),
     . disp1_dx(nxm,nym), disp2_dx(nxm,nym), disp2_dy(nxm,nym),
     . disp3_dy(nxm,nym), dispx(nxm,nym), dispy(nxm,nym),
     . Axxx(nxm,nym), Axxy(nxm,nym), Axyx(nxm,nym), Axyy(nxm,nym),
     . Ayyx(nxm,nym), Ayyy(nxm,nym), dVo_dx2(nxm,nym), dVo_dy2(nxm,nym),
     . Bxx(nxm,nym), Bxy(nxm,nym), Byy(nxm,nym),dUo_dx2(nxm,nym),
     . adv1(nxm,nym), adv2(nxm,nym), adv3(nxm,nym), adv3_dy(nxm,nym),
     . adv1_dx(nxm,nym), adv2_dx(nxm,nym), adv2_dy(nxm,nym),
     . qwxqwy(nxm,nym), qwxqwx(nxm,nym), qwyqwy(nxm,nym),
     . qwxqwxdx(nxm,nym), qwxqwydx(nxm,nym),
     . qwyqwydy(nxm,nym), qwxqwydy(nxm,nym),
     . dzetadxn(nxm,nym), dzetadyn(nxm,nym),
     . frictx(nxm,nym), fricty(nxm,nym),vs(nxm,nym),
     . momxnm1(nxm,nym), momxnm2(nxm,nym), dtauxxdx(nxm,nym),
     . momynm1(nxm,nym), momynm2(nxm,nym),tauyy(nxm,nym),
     . dtauxydx(nxm,nym),dtauxydy(nxm,nym),tauxx(nxm,nym),
     . factor,tauxy(nxm,nym),dtauyydy(nxm,nym), wind_angle,
     . Uo(nxm,nym), Vo(nxm,nym), dUo_dx(nxm,nym), dUo_dy(nxm,nym),
     . dVo_dx(nxm,nym), dVo_dy(nxm,nym),windx, windy
c=======================================================================
c ## define functions that will be used through out the routine -----
c
c ## WARNING: they look like variables but behave like functions,
c ## and must follow imediately the dimension statements
c-----------------------------------------------------------------------
      cel(h1,z1) = DSQRT(g*(h1+z1))
      ux(qx1,z1,h1) = qx1/(h1+z1)
      uy(qy1,z1,h1) = qy1/(h1+z1)

        force_xx(i,j,t)= dSxxdxn(i,j)*(1-kold_start)
        force_xy(i,j,t)= dSxydyn(i,j)*(1-kold_start)
        force_yx(i,j,t)= dSxydxn(i,j)*(1-kold_start)
        force_yy(i,j,t)= dSyydyn(i,j)*(1-kold_start)
```

```
      momx1(i,j,t) =
     .   - force_xx(i,j,t)
     .   - force_xy(i,j,t)
     .   - frictx(i,j)
     .   + dtauxydy(i,j)
     .   + dtauxxdx(i,j)
     .    + windx

      momx2(i,j) =
     .   -g*(ht(i,j)+zetan(i,j))*dzetadxn(i,j)
     .   - qxqxdx(i,j) -qxqydy(i,j)
     .    + (qwxqwxdx(i,j) +qwxqwydy(i,j))
     .    + dispx(i,j)

      momy1(i,j,t) =
     .   - force_yx(i,j,t)
     .   - force_yy(i,j,t)
     .   - fricty(i,j)
     .   + dtauxydx(i,j)
     .   + dtauyydy(i,j)
     .    + windy

      momy2(i,j) =
     .   -g*(ht(i,j)+zetan(i,j))*dzetadyn(i,j)
     .   - qyqydy(i,j) -qxqydx(i,j)
     .    + (qwxqwydx(i,j) +qwyqwydy(i,j))
     .    + dispy(i,j)

c-----------------------------------------------------------------------

      if(nx.gt.nxm) then
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'nx is larger than nxm'
         write(*,*)'Change nxm in winc_std_common.inc and
     .        winc_std_filter.f'
         stop
      end if
      if(ny.gt.nym) then
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
```

```
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'ny is larger than nym'
         write(*,*)'Change nym in winc_std_common.inc and
     .         winc_std_filter.f'
         stop
       end if
       if(nym.gt.nmax.or.nxm.gt.nmax) then
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'nmax should equal max(nxm,nym)'
         stop
       end if
       if(5/delay.gt.ntime*dt.and.tavgout.ne.0) then
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'WARNING WARNING WARNING WARNING WARNING WARNING'
         write(*,*)'Total run time less than 5*delay.'
         write(*,*)'Time-averages will not be computed.'
         write(*,*)'ntime*dt = ',ntime*dt
         write(*,*)'5*delay = ',5/delay
       end if

c-----------------------------------------------------------------------
c ## Calculate wind shear stress

       wind_angle = wind_dir*pi/180.d0
       cd = 1.3915d-3
       windx = 1.2/rho*cd*wind_vel*wind_vel*dcos(wind_angle)
       windy = -1.2/rho*cd*wind_vel*wind_vel*dsin(wind_angle)


c ## ---------------------------------------------------------------
c ## Calculating Quantities at t = -2*dt
c ## ---------------------------------------------------------------
       write(*,902)' Initial steps'
       write(6,*)'t=-2dt'
       t = -2.d0*dt

       if (kold_start.eq.1) then
```

```
c-##---- Cold start
         do j = 1,ny
          do i = 1,nx
            zetan(i,j) = 0.d0
            qxn(i,j)   = 0.d0
            qyn(i,j)   = 0.d0
          end do
         end do

      elseif (kold_start.eq.0) then
c-##---- Hot start
           open(unit=19,file='hot2.dat',access='direct',
     .          status='old',recl=4*ny)
           do i=1,nx
             read(19,rec=i) (temp1(i,j),j=1,ny)
             read(19,rec=i+nx) (temp2(i,j),j=1,ny)
             read(19,rec=i+2*nx) (temp3(i,j),j=1,ny)
           end do
   close(19)
   do i=1,nx
   do j=1,ny
    zetan(i,j)=dble(temp1(i,j))
    qxn(i,j)=dble(temp2(i,j))
  qyn(i,j)=dble(temp3(i,j))
 end do
 end do
       end if

c## Calculate the derivatives

      ncode =1
      nxflag = 1
      call deriv(zetan,qxn,qyn,dzetadxn,dzetadyn,dqxdxn,dqydyn,
     .           ncode,nxflag)

      do j=1,ny
        do i=1,nx
          qxqx(i,j) = qxn(i,j)*qxn(i,j)/(ht(i,j)+zetan(i,j))
          qxqy(i,j) = qxn(i,j)*qyn(i,j)/(ht(i,j)+zetan(i,j))
          qyqy(i,j) = qyn(i,j)*qyn(i,j)/(ht(i,j)+zetan(i,j))
          Uo(i,j) = ux(qxn(i,j),ht(i,j),zetan(i,j))
          Vo(i,j) = uy(qyn(i,j),ht(i,j),zetan(i,j))
  qwxqwy(i,j)=qwx(i,j)*qwy(i,j)/(ht(i,j)+zetan(i,j))
```

```
      qwyqwy(i,j)=qwy(i,j)*qwy(i,j)/(ht(i,j)+zetan(i,j))
      qwxqwx(i,j)=qwx(i,j)*qwx(i,j)/(ht(i,j)+zetan(i,j))
            end do
          end do

          ncode =1
          nxflag = 1
          call deriv(qxqy,qxqx,qyqy,qxqydx,qxqydy,qxqxdx,qyqydy,
         .  ncode,nxflag)
          ncode =3
          nxflag=1
          call deriv(qwxqwy,qwxqwx,qwyqwy,qwxqwydx,qwxqwydy,
         . qwxqwxdx,qwyqwydy,ncode,nxflag)

          call average_3ptx(nx,ny,qwxqwxdx)
          call average_3ptx(nx,ny,qwxqwydx)
          call average_3ptx(nx,ny,qwxqwydy)
          call average_3ptx(nx,ny,qwyqwydy)
          call average_3pty(nx,ny,qwxqwxdx)
          call average_3pty(nx,ny,qwxqwydx)
          call average_3pty(nx,ny,qwxqwydy)
          call average_3pty(nx,ny,qwyqwydy)

          call deriv(Uo, Vo, Vo, dUo_dx, dUo_dy, dVo_dx, dVo_dy,
         .           ncode,nxflag)
c ## eddy viscosity
          if (kold_start.eq.1) then
          do j=1,ny
 do i=1,nx
                dtauxydx(i,j) = 0.d0
                dtauxydy(i,j) = 0.d0
                dtauxxdx(i,j) = 0.d0
                dtauyydy(i,j) = 0.d0
 end do
          end do
          else
          do j=1,ny
 do i=1,nxa(j)-1
              vs(i,j)=Cs*Cs*dx*dy*sqrt(dUo_dx(i,j)*dUo_dx(i,j)*2.d0
         .           +dVo_dy(i,j)*dVo_dy(i,j)*2.d0
         .           +(dUo_dy(i,j)+dVo_dx(i,j))*(dUo_dy(i,j)+dVo_dx(i,j)))
        tauxy(i,j) = (v_t(i,j)+vs(i,j))*(ht(i,j)+zetan(i,j))*
         .  (dVo_dx(i,j)+dUo_dy(i,j))
```

```
            tauxx(i,j) = vs(i,j)*(ht(i,j)+zetan(i,j))*
     .                   (dUo_dx(i,j)*2.d0)
            tauyy(i,j) = vs(i,j)*(ht(i,j)+zetan(i,j))*
     .                   (dVo_dy(i,j)*2.d0)
 end do
 do i=nxa(j),nx
            tauxy(i,j) = 0.d0
            tauxx(i,j) = 0.d0
            tauyy(i,j) = 0.d0
            vs(i,j)=0.d0
 end do
      end do
      call deriv(tauxy,tauxx,tauyy,dtauxydx,dtauxydy,dtauxxdx,
     .           dtauyydy,ncode,nxflag)
      end if

c ## call bottom friction

      factor=1.d0
      call bottom_friction(zetan,qxn,qyn,frictx,fricty,factor,0)


c ## 3D dispersion, coded by KAH for Ap's reduced version ------------------

      if(disp3d.eq.0) then
         do j=1,ny
           do i=1,nx
             dispx(i,j)=0.d0
             dispy(i,j)=0.d0
           end do
         end do
         goto 554
      end if

        call UV_profile(zetan,frictx,fricty,t,windx,windy,vs)
        call coefcalc(Dxx,Dyy,Dxy,zetan,Mxx,Myy,Mxy,Bxx,Bxy,Byy,
     .       Axxx,Axxy,Axyx,Axyy,Ayyx,Ayyy,vs,Uo,Vo)

      do j=1,ny
 do i=1,nxa(j)
           htot = ht(i,j) + zetan(i,j)
           ht2 = htot*htot
           ht3 = ht2*htot
```

```
            ht4 = ht2*ht2
            adv1(i,j) = Axxx(i,j)*Uo(i,j)+Axxy(i,j)*Vo(i,j)
            adv2(i,j) = Axyx(i,j)*Uo(i,j)+Axyy(i,j)*Vo(i,j)
            adv3(i,j) = Ayyx(i,j)*Uo(i,j)+Ayyy(i,j)*Vo(i,j)
            disp1(i,j) = ( (2.d0*Dxx(i,j)+Bxx(i,j))*dUo_dx(i,j)
     .                    +2.d0*Dxy(i,j)*dUo_dy(i,j)
     .                    +Bxx(i,j)*dVo_dy(i,j))*htot
            disp2(i,j) = ( Dxx(i,j)*dVo_dx(i,j)
     .                    +(Dxy(i,j)+Bxy(i,j))*dVo_dy(i,j)
     .                    +(Dxy(i,j)+Bxy(i,j))*dUo_dx(i,j)
     .                    + Dyy(i,j)*dUo_dy(i,j)
     .                    )*htot
            disp3(i,j) = ( 2.d0*Dxy(i,j)*dVo_dx(i,j)
     .                    +(2.d0*Dyy(i,j)+Byy(i,j))*dVo_dy(i,j)
     .                    +(Byy(i,j))*dUo_dx(i,j)
     .                    )*htot
 end do
 do i=nxa(j)+1,nx
            disp1(i,j) = 0.d0
            disp2(i,j) = 0.d0
            disp3(i,j) = 0.d0
 end do
      end do
      ncode =3
      nxflag=1
      call deriv(disp2, disp1, disp3, disp2_dx, disp2_dy, disp1_dx,
     .  disp3_dy, ncode, nxflag)
      call deriv(Mxy, Mxx, Myy, dMxy_dx, dMxy_dy, dMxx_dx,
     .  dMyy_dy, ncode, nxflag)
      call deriv(adv2, adv1, adv3, adv2_dx, adv2_dy, adv1_dx,
     .           adv3_dy, ncode, nxflag)
c ## do 3pt-average of terms in dispx ------------------
      call average_3ptx(nx,ny,disp1_dx)
      call average_3ptx(nx,ny,disp2_dy)

c ## do 3pt-average of terms in dispy ------------------
      call average_3ptx(nx,ny,disp2_dx)
      call average_3ptx(nx,ny,disp3_dy)

c ## assemble the dispersion terms -----------
      do j=1,ny
 do i=1,3
            dispx(i,j) = 0.d0
```

```
            dispy(i,j) = 0.d0
            qwxqwxdx(i,j) = 0.d0
            qwxqwydy(i,j) = 0.d0
            qwxqwydx(i,j) = 0.d0
            qwyqwydy(i,j) = 0.d0
 end do
 do i=4,nxa(j)
            dispx(i,j) = disp1_dx(i,j) + disp2_dy(i,j)
     .            -dMxx_dx(i,j)-dMxy_dy(i,j)
     .            -adv1_dx(i,j)-adv2_dy(i,j)
            dispy(i,j) = disp2_dx(i,j) + disp3_dy(i,j)
     .           -dMyy_dy(i,j)-dMxy_dx(i,j)
     .           -adv2_dx(i,j)-adv3_dy(i,j)
 end do
 do i=nxa(j)+1,nx
            dispx(i,j) = 0.d0
            dispy(i,j) = 0.d0
            qwxqwxdx(i,j) = 0.d0
            qwxqwydy(i,j) = 0.d0
            qwxqwydx(i,j) = 0.d0
            qwyqwydy(i,j) = 0.d0
 end do
      end do
c ## do 3pt-average of dispx and dispy -----------------
      call average_3ptx(nx,ny,dispx)
      call average_3pty(nx,ny,dispx)
      call average_3ptx(nx,ny,dispy)
      call average_3pty(nx,ny,dispy)
ckk 12/16/99
      do j=1,ny
 do i=1,nx
           qwxqwxdx(i,j)=qwxqwxdx(i,j)
     .           *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
           qwxqwydy(i,j)=qwxqwydy(i,j)
     .           *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
           qwxqwydx(i,j)=qwxqwydx(i,j)
     .           *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
           qwyqwydy(i,j)=qwyqwydy(i,j)
     .           *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
           dispx(i,j)=dispx(i,j)*dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
           dispy(i,j)=dispy(i,j)*dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
 end do
      end do
```

```
ckk 12/16/99
554   continue


c ## calculate RHS of continuity & momentum eqs' ---------------------

      do j = 1,ny
        do i = 1,nx

          cnm2(i,j) = - (dqxdxn(i,j) + dqydyn(i,j))

          momxnm2(i,j) = momx1(i,j,t)
          mxnm2(i,j) = momx1(i,j,t) + momx2(i,j)

          momynm2(i,j) = momy1(i,j,t)
          mynm2(i,j) = momy1(i,j,t) + momy2(i,j)

       end do
      end do

c ## Moving shoreline bound. condition
c !! set the forcing beyond any initially dry point to 'Zero'
         do j = 1,ny
            do i=nxa(j)+1,nx
              cnm2(i,j)=0.d0
              momxnm2(i,j)=0.d0
              mxnm2(i,j)=0.d0
              momynm2(i,j)=0.d0
              mynm2(i,j)=0.d0
            end do
         end do



c## CHARACTERISTICS

      if ((ibc1.eq.3).or.(ibc1.eq.2)) then
        do j = 1,ny
          betan(1,j) = (ux(qxn(1,j),zetan(1,j),ht(1,j)) - 2.0d0*
     .              cel(zetan(1,j),ht(1,j)))
          betan(2,j) = (ux(qxn(2,j),zetan(2,j),ht(2,j)) - 2.0d0*
     .              cel(zetan(2,j),ht(2,j)))
          betan(3,j) = (ux(qxn(3,j),zetan(3,j),ht(3,j)) - 2.0d0*
```

```
     .              cel(zetan(3,j),ht(3,j)))
        end do

        call betachar(bnm2,betan,zetan,qxn,qyn,dqydyn,dzetadyn,
     .              momxnm2)
       end if



c ## ----------------------------------------------------------------
c ## Calculating Quantities at t = -dt
c ## ----------------------------------------------------------------
       write(6,*)'t=-dt'
       t = -dt

c ## Initial conditions

       if (kold_start.eq.1) then
c-##---- Cold start
          do j = 1,ny
           do i = 1,nx
             zetan(i,j) = 0.d0
             qxn(i,j)   = 0.d0
             qyn(i,j)   = 0.d0
           end do
          end do
       elseif (kold_start.eq.0) then
c-##---- Hot start
           open(unit=19,file='hot1.dat',access='direct',
     .          status='old',recl=4*ny)
            do i=1,nx
              read(19,rec=i) (temp1(i,j),j=1,ny)
              read(19,rec=i+nx) (temp2(i,j),j=1,ny)
              read(19,rec=i+2*nx) (temp3(i,j),j=1,ny)
            end do
   close(19)
   do i=1,nx
   do j=1,ny
    zetan(i,j)=dble(temp1(i,j))
    qxn(i,j)=dble(temp2(i,j))
  qyn(i,j)=dble(temp3(i,j))
 end do
 end do
```

```
      end if

c## Calculate the derivatives

      ncode =1
      nxflag = 1
      call deriv(zetan,qxn,qyn,dzetadxn,dzetadyn,dqxdxn,dqydyn,
     .           ncode,nxflag)

      do j=1,ny
        do i=1,nx
          qxqx(i,j) = qxn(i,j)*qxn(i,j)/(ht(i,j)+zetan(i,j))
          qxqy(i,j) = qxn(i,j)*qyn(i,j)/(ht(i,j)+zetan(i,j))
          qyqy(i,j) = qyn(i,j)*qyn(i,j)/(ht(i,j)+zetan(i,j))
          Uo(i,j) = ux(qxn(i,j),ht(i,j),zetan(i,j))
          Vo(i,j) = uy(qyn(i,j),ht(i,j),zetan(i,j))
  qwxqwy(i,j)=qwx(i,j)*qwy(i,j)/(ht(i,j)+zetan(i,j))
  qwyqwy(i,j)=qwy(i,j)*qwy(i,j)/(ht(i,j)+zetan(i,j))
  qwxqwx(i,j)=qwx(i,j)*qwx(i,j)/(ht(i,j)+zetan(i,j))
        end do
      end do

      ncode =1
      nxflag=1
      call deriv(qxqy,qxqx,qyqy,qxqydx,qxqydy,qxqxdx,qyqydy,
     .  ncode,nxflag)
      ncode =3
      nxflag=1
      call deriv(qwxqwy,qwxqwx,qwyqwy,qwxqwydx,qwxqwydy,
     . qwxqwxdx,qwyqwydy,ncode,nxflag)

      call average_3ptx(nx,ny,qwxqwxdx)
      call average_3ptx(nx,ny,qwxqwydx)
      call average_3ptx(nx,ny,qwxqwydy)
      call average_3ptx(nx,ny,qwyqwydy)
      call average_3pty(nx,ny,qwxqwxdx)
      call average_3pty(nx,ny,qwxqwydx)
      call average_3pty(nx,ny,qwxqwydy)
      call average_3pty(nx,ny,qwyqwydy)

      call deriv(Uo, Vo, Vo, dUo_dx, dUo_dy, dVo_dx, dVo_dy,
     .           ncode,nxflag)
```

```
c ## eddy viscosity
      if (kold_start.eq.1) then
      do j=1,ny
 do i=1,nx
            dtauxydx(i,j) = 0.d0
            dtauxydy(i,j) = 0.d0
            dtauxxdx(i,j) = 0.d0
            dtauyydy(i,j) = 0.d0
 end do
      end do
      else

      do j=1,ny
 do i=1,nxa(j)-1
            vs(i,j)=Cs*Cs*dx*dy*sqrt(dUo_dx(i,j)*dUo_dx(i,j)*2.d0
     .           +dVo_dy(i,j)*dVo_dy(i,j)*2.d0
     .           +(dUo_dy(i,j)+dVo_dx(i,j))*(dUo_dy(i,j)+dVo_dx(i,j)))
    tauxy(i,j) = (v_t(i,j)+vs(i,j))*(ht(i,j)+zetan(i,j))*
     .  (dVo_dx(i,j)+dUo_dy(i,j))
            tauxx(i,j) = vs(i,j)*(ht(i,j)+zetan(i,j))*
     .                   (dUo_dx(i,j)*2.d0)
            tauyy(i,j) = vs(i,j)*(ht(i,j)+zetan(i,j))*
     .                   (dVo_dy(i,j)*2.d0)
 end do
 do i=nxa(j),nx
            tauxy(i,j) = 0.d0
            tauxx(i,j) = 0.d0
            tauyy(i,j) = 0.d0
            vs(i,j)=0.d0
 end do
      end do
      call deriv(tauxy,tauxx,tauyy,dtauxydx,dtauxydy,dtauxxdx,
     .           dtauyydy,ncode,nxflag)
      end if

c ## call bottom friction

      factor=1.d0
      call bottom_friction(zetan,qxn,qyn,frictx,fricty,factor,0)
c ## 3D dispersion, coded by KAH for Ap's reduced version ----------------

      if(disp3d.eq.0) then
         do j=1,ny
```

```
            do i=1,nx
              dispx(i,j)=0.d0
              dispy(i,j)=0.d0
            end do
          end do
          goto 555
        end if

          call UV_profile(zetan,frictx,fricty,t,windx,windy,vs)
          call coefcalc(Dxx,Dyy,Dxy,zetan,Mxx,Myy,Mxy,Bxx,Bxy,Byy,
     .         Axxx,Axxy,Axyx,Axyy,Ayyx,Ayyy,vs,Uo,Vo)

        do j=1,ny
do i=1,nxa(j)
            htot = ht(i,j) + zetan(i,j)
            ht2 = htot*htot
            ht3 = ht2*htot
            ht4 = ht2*ht2
            adv1(i,j) = Axxx(i,j)*Uo(i,j)+Axxy(i,j)*Vo(i,j)
            adv2(i,j) = Axyx(i,j)*Uo(i,j)+Axyy(i,j)*Vo(i,j)
            adv3(i,j) = Ayyx(i,j)*Uo(i,j)+Ayyy(i,j)*Vo(i,j)
            disp1(i,j) = ( (2.d0*Dxx(i,j)+Bxx(i,j))*dUo_dx(i,j)
     .                    +2.d0*Dxy(i,j)*dUo_dy(i,j)
     .                    +Bxx(i,j)*dVo_dy(i,j))*htot
            disp2(i,j) = ( Dxx(i,j)*dVo_dx(i,j)
     .                    +(Dxy(i,j)+Bxy(i,j))*dVo_dy(i,j)
     .                    +(Dxy(i,j)+Bxy(i,j))*dUo_dx(i,j)
     .                    + Dyy(i,j)*dUo_dy(i,j)
     .                    )*htot
            disp3(i,j) = ( 2.d0*Dxy(i,j)*dVo_dx(i,j)
     .                    +(2.d0*Dyy(i,j)+Byy(i,j))*dVo_dy(i,j)
     .                    +(Byy(i,j))*dUo_dx(i,j)
     .                    )*htot
end do
do i=nxa(j)+1,nx
            disp1(i,j) = 0.d0
            disp2(i,j) = 0.d0
            disp3(i,j) = 0.d0
end do
        end do
        ncode =3
        nxflag=1
        call deriv(disp2, disp1, disp3, disp2_dx, disp2_dy, disp1_dx,
```

```
     .  disp3_dy, ncode, nxflag)
      call deriv(Mxy, Mxx, Myy, dMxy_dx, dMxy_dy, dMxx_dx,
     .  dMyy_dy, ncode, nxflag)
      call deriv(adv2, adv1, adv3, adv2_dx, adv2_dy, adv1_dx,
     .           adv3_dy, ncode, nxflag)
c ## do 3pt-average of terms in dispx ------------------
      call average_3ptx(nx,ny,disp1_dx)
      call average_3ptx(nx,ny,disp2_dy)

c ## do 3pt-average of terms in dispy ------------------
      call average_3ptx(nx,ny,disp2_dx)
      call average_3ptx(nx,ny,disp3_dy)

c ## assemble the dispersion terms -----------
      do j=1,ny
 do i=1,3
           dispx(i,j) = 0.d0
           dispy(i,j) = 0.d0
           qwxqwxdx(i,j) = 0.d0
           qwxqwydy(i,j) = 0.d0
           qwxqwydx(i,j) = 0.d0
           qwyqwydy(i,j) = 0.d0
 end do
 do i=4,nxa(j)
           dispx(i,j) = disp1_dx(i,j) + disp2_dy(i,j)
     .           -dMxx_dx(i,j)-dMxy_dy(i,j)
     .           -adv1_dx(i,j)-adv2_dy(i,j)
           dispy(i,j) = disp2_dx(i,j) + disp3_dy(i,j)
     .          -dMyy_dy(i,j)-dMxy_dx(i,j)
     .          -adv2_dx(i,j)-adv3_dy(i,j)
 end do
 do i=nxa(j)+1,nx
           dispx(i,j) = 0.d0
           dispy(i,j) = 0.d0
           qwxqwxdx(i,j) = 0.d0
           qwxqwydy(i,j) = 0.d0
           qwxqwydx(i,j) = 0.d0
           qwyqwydy(i,j) = 0.d0
 end do
      end do
c ## do 3pt-average of dispx and dispy ------------------
      call average_3ptx(nx,ny,dispx)
      call average_3pty(nx,ny,dispx)
```

```
      call average_3ptx(nx,ny,dispy)
      call average_3pty(nx,ny,dispy)
ckk 12/16/99
      do j=1,ny
 do i=1,nx
          qwxqwxdx(i,j)=qwxqwxdx(i,j)
     .           *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
          qwxqwydy(i,j)=qwxqwydy(i,j)
     .           *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
          qwxqwydx(i,j)=qwxqwydx(i,j)
     .           *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
          qwyqwydy(i,j)=qwyqwydy(i,j)
     .           *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
          dispx(i,j)=dispx(i,j)*dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
          dispy(i,j)=dispy(i,j)*dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
 end do
      end do
ckk 12/16/99
 555  continue

c ## calculate RHS of continuity & momentum eqs'

      do j = 1,ny
        do i = 1,nx

          cnm1(i,j) = - (dqxdxn(i,j) + dqydyn(i,j))

          momxnm1(i,j) = momx1(i,j,t)
          mxnm1(i,j) = momx1(i,j,t) + momx2(i,j)

          momynm1(i,j) = momy1(i,j,t)
          mynm1(i,j) = momy1(i,j,t) + momy2(i,j)

        end do
      end do

c ## Moving shoreline bound. condition
      if (ibc2.eq.6) then
         do j = 1,ny
            do i=nxa(j)+1,nx
              cnm1(i,j)=0.d0
              momxnm1(i,j)=0.d0
              mxnm1(i,j)=0.d0
```

```
              momynm1(i,j)=0.d0
              mynm1(i,j)=0.d0
            end do
          end do
        end if

c## CHARACTERISTICS

        if ((ibc1.eq.2).or.(ibc1.eq.3)) then
          do j = 1,ny
            betan(1,j) = (ux(qxn(1,j),zetan(1,j),ht(1,j)) - 2.0d0*
     .                  cel(zetan(1,j),ht(1,j)))
            betan(2,j) = (ux(qxn(2,j),zetan(2,j),ht(2,j)) - 2.0d0*
     .                  cel(zetan(2,j),ht(2,j)))
            betan(3,j) = (ux(qxn(3,j),zetan(3,j),ht(3,j)) - 2.0d0*
     .                  cel(zetan(3,j),ht(3,j)))
          end do
        end if


        call betachar(bnm1,betan,zetan,qxn,qyn,dqydyn,dzetadyn,
     .                  momxnm1)



c ##-----------------------------------------------------------------
c ## Calculating initial conditions at t=0
c ##-----------------------------------------------------------------
        write(6,*)'t=0'
        t = 0.d0

c ## Initial conditions

        if (kold_start.eq.1) then
c-##---- Cold start
          do j = 1,ny
           do i = 1,nx
             zetan(i,j) = 0.d0
             qxn(i,j)   = 0.d0
             qyn(i,j)   = 0.d0
           end do
          end do
        elseif (kold_start.eq.0) then
```

```
c-##---- Hot start
          open(unit=19,file='hot0.dat',access='direct',
     .          status='old',recl=4*ny)
           do i=1,nx
             read(19,rec=i) (temp1(i,j),j=1,ny)
             read(19,rec=i+nx) (temp2(i,j),j=1,ny)
             read(19,rec=i+2*nx) (temp3(i,j),j=1,ny)
           end do
   close(19)
   do i=1,nx
   do j=1,ny
    zetan(i,j)=dble(temp1(i,j))
    qxn(i,j)=dble(temp2(i,j))
  qyn(i,j)=dble(temp3(i,j))
 end do
 end do
         if (wc.ne.0) then
          call shortwave_driver
          call wave_forcing
end if
      end if

c ## Moving shoreline bound. condition
      if (ibc2.eq.6) then
         do j = 1,ny
            do i=nxa(j)+1,nx
              qxn(i,j)=0.d0
              qyn(i,j)=0.d0
              zetan(i,j)=-ht(i,j)+0.0001d0
            end do
         end do
      end if

c  Characteristix


      if ((ibc1.eq.2).or.(ibc1.eq.3)) then
        do j = 1,ny
          betan(1,j) = (ux(qxn(1,j),zetan(1,j),ht(1,j)) - 2.0d0*
     .               cel(zetan(1,j),ht(1,j)))
          betan(2,j) = (ux(qxn(2,j),zetan(2,j),ht(2,j)) - 2.0d0*
     .               cel(zetan(2,j),ht(2,j)))
          betan(3,j) = (ux(qxn(3,j),zetan(3,j),ht(3,j)) - 2.0d0*
```

```
     .                cel(zetan(3,j),ht(3,j)))
        end do
       end if

902    format(/A)

       return
       end
```

8.1.10 Subroutine betachar

This subroutine calculates the beta characteristics for the absorbing-generating boundary condition.

```
c ######################################################################
      subroutine betachar(b_RHS,beta,zeta,qx,qy,dqydy,dzetady,
     .                    momx)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local functions ------------------------------------------------
      real*8 cel

c--- Local variables ------------------------------------------------
      integer j
      real*8 b_RHS(nym), momx(nxm,nym), betatemp(nym), dbetady(nym),
     . beta(3,nym), zeta(nxm,nym), qx(nxm,nym), qy(nxm,nym),
     . dqydy(nxm,nym), dzetady(nxm,nym),
     . h1, z1,dbetadx, c, inv_ht(nym), ux, uy

c=====================================================================
c---- statement functions--------------------------------------------
      cel(h1,z1) = DSQRT(g*(h1+z1))
c---------------------------------------------------------------------

      do j=1,ny
        betatemp(j)=beta(1,j)
       end do

      call spliney(ny,betatemp,dbetady)
```

```
      do j=1,ny

        dbetadx=(-3.d0*beta(1,j)+4.d0*beta(2,j)-beta(3,j))/(2.d0*dx)
        c = cel(zeta(1,j),ht(1,j))

        inv_ht(j) = 1.d0/(zeta(1,j)+ht(1,j))
        ux = qx(1,j)*inv_ht(j)
        uy = qy(1,j)*inv_ht(j)
        b_RHS(j)= -(ux-c)*dbetadx -uy*dbetady(j)
     .       +c*( (ht(1,j)+zeta(1,j))*dqydy(1,j) - qy(1,j)*
     .            dzetady(1,j) ) *inv_ht(j)**2.d0
     .       +momx(1,j)*inv_ht(j)
     .       +g*dhodxn(1,j)

      end do

      return
      end
```

8.1.11 Subroutine viscosity

This subroutine calculates the eddy viscosity.

```
c ######################################################################
      subroutine viscosity

c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'
      include 'pass.inc'

c--- Local variables ------------------------------------------------
      integer i, j, index, iflag, ii, index1(10,nym)
      real*8 htot,
     .       cg(nxm,nym), gam,
     .       newibr(nxm,nym)
c======================================================================
      write(6,*) 'Compute v_t'
      write(6,903) 'vtshear :', vtshear
      Bo=0.125d0
```

```
c---- calculate wave energy dissipation from REFDIF's formulation ----
c     'gam' is the parameter in REFDIF for the stable wave energy
c     default:  gam=0.4
c---- Check if dissipation<0 !!!
c Only if R/D version is used
      if (files.eq.0) then
      do i=1,nx
         do j=1,ny
    cg(i,j)=ccg(i+2,j)
         end do
      end do
      open(31,file='fort.331',access='direct',recl=4*ny)
      do i=1,nx
write(31,rec=i) (sngl(cg(i,j)),j=1,ny)
      end do
      close(31)
      open(32,file='fort.328',access='direct',recl=4*ny)
      gam = 0.4d0
      do i=1,nx
        do j=1,ny
           dissipation(i,j) = g*0.125d0*H(i,j)*H(i,j)
     .      *0.15d0*cg(i,j)/ht(i,j)*(1.d0-(gam*ht(i,j)/H(i,j))**2.)
     .      *ibrk(i,j)
           if (dissipation(i,j) .lt. 0.d0) dissipation(i,j)=0.d0
end do
      end do
         call average_3ptx(nx,ny,dissipation)
      do i=1,nx
write(32,rec=i) (sngl(dissipation(i,j)),j=1,ny)
      end do
      close(32)

      end if

c Both versions
c Initialize the breaking index
do ii =1,4
do j =1,ny
  index1(ii,j)=0
end do
end do

c-- make a continuous approximation to the step function 'ibr'
```

```
      do j=1,ny
c------ find breaking point: find first ibrk(i,j)=1
        iflag=0
        ii=1
        do i=1,nx-1
   if (ibrk(i,j).eq.0 .and. ibrk(i+1,j).eq.1) then
     index1(ii,j)=i+1
             ii=ii+1
   elseif (ibrk(i,j).eq.1 .and. ibrk(i+1,j).eq.0) then
     index1(ii,j)=i
             ii=ii+1
   end if
end do
        if (ii.eq.2) index1(ii,j)=nx
        if (ii.eq.4) index1(ii,j)=nx
        if (ii.eq.6) index1(ii,j)=nx
        do i=1,index1(1,j)
           newibr(i,j)= 1.d0-DTANH(DBLE(index1(1,j)-i)/5.d0)
end do
        do i=index1(1,j),index1(2,j)
           newibr(i,j)= 1.d0
end do
        do i=index1(2,j)+1,index1(3,j)
           newibr(i,j)= 1.d0+0.5*DTANH(DBLE(index1(2,j)+1-i)/5.d0)
     .                   -0.5*DTANH(DBLE(index1(3,j)-i)/5.d0)
end do
        do i=index1(3,j),index1(4,j)
           newibr(i,j)= 1.d0
end do
      end do

      open(41,file='fort.329',access='direct',recl=4*ny)

      do i=1,nx
 write(41,rec=i) (sngl(newibr(i,j)),j=1,ny)
      end do
      close(41)



c------ calculate eddy viscosity  ----------------------------
c  Smooth tranistion to breaking waves by multiplying newibr
      do j=1,ny
```

```
c   Outside surfzone match dissipation to dissipation at breaker line
        do i=1,index1(1,j)
           htot = ht(i,j)+zetan(i,j)
  if (ht(i,j).le.depthmin) then
    v_t(i,j) = vtshear*DSQRT(0.5d0*fcwij(i,j))*depthmin*u0(i,j)
     .         +Mdiss*(dissipation(index1(1,j),j))**(1.0/3.0)*depthmin
     .         *newibr(i,j)
          else
    v_t(i,j) = vtshear*DSQRT(0.5d0*fcwij(i,j))*htot*u0(i,j)
     .         +Mdiss*(dissipation(index1(1,j),j))**(1.0/3.0)*htot
     .         *newibr(i,j)
  end if
         end do
c     Inside surfzone use actual dissipation
         do i=index1(1,j),nxa(j)
           htot = ht(i,j)+zetan(i,j)
  if (ht(i,j).le.depthmin) then
    v_t(i,j) = vtshear*DSQRT(0.5d0*fcwij(i,j))*depthmin*u0(i,j)
     .         +Mdiss*(dissipation(i,j))**(1.0/3.0)*depthmin
     .         *newibr(i,j)
          else
    v_t(i,j) = vtshear*DSQRT(0.5d0*fcwij(i,j))*htot*u0(i,j)
     .         +Mdiss*(dissipation(i,j))**(1.0/3.0)*htot
     .         *newibr(i,j)
          end if
        end do
       end do

c ## 3point average filter for v_t ---------------------------
         call average_3ptx(nx,ny,v_t)
         call average_3pty(nx,ny,v_t)

c ## cut-off upper limit on v_t from Svendsen's estimates
      do j=1,ny
        do i=1,nxa(j)
   if (v_t(i,j).gt.0.04*ht(i,j)*DSQRT(g*ht(i,j))) then
      v_t(i,j) = 0.04*ht(i,j)*DSQRT(g*ht(i,j))
           end if
ck 9/22/99     add ambient turbulence
           v_t(i,j)=v_t(i,j)+.0002d0
        end do
      end do
```

```
c ## write out vt ----------------------------------------
      open(16,file='fort.330',access='direct',recl=4*ny)
      do i=1,nx
write(16,rec=i) (sngl(v_t(i,j)),j=1,ny)
      end do
      close(16)

903   format(1X,A,F12.6)


      return
      end
```

8.1.12 Subroutine bottom_friction

This subroutine calculates the bottom shear stress.

```
c ######################################################################
      subroutine bottom_friction(zeta,xflux,yflux,frict_x,frict_y,
     .  factor,istep)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, j, istep, kkk
      real*8 zeta(nxm,nym),xflux(nxm,nym),yflux(nxm,nym),aux3,aux4,
     . frict_x(nxm,nym),frict_y(nxm,nym), Vby, Vbx,uoVb,
     . factor, Vk, aux1, aux2, miu, costheta, costheta2, Vbuo, Vbuoc,
     . beta1(21), beta2(21), intbeta1(nxm,nym), intbeta2(nxm,nym), Vb

c=====================================================================

      aux1 = fcw/pi
      aux3=(2/pi)**2
      aux4=1/pi
           do j=1,ny
         do i=1,nxa(j)
    Vby = (yflux(i,j)-qwy(i,j)*factor)/(ht(i,j)+zeta(i,j))
    Vbx = (xflux(i,j)-qwx(i,j)*factor)/(ht(i,j)+zeta(i,j))
```

```
c -------- Non-linear friction, Svendsen & Putrevu (1990)

            if(Vby.ne.0.and.Vbx.ne.0) then
               miu = datan2(Vby,Vbx)-theta(i,j)
            else
               miu = -theta(i,j)
            end if
      Vb = dsqrt(Vbx**2+Vby**2)
      Vk = Vb *dcos(miu)
      intbeta1(i,j)=0.d0
      intbeta2(i,j)=0.d0
      Vbuo = Vb*Vb/(u0(i,j)*u0(i,j))
      Vbuoc = 2.d0*Vk/u0(i,j)

c Numerical integral for calculating beta1 and beta2

c     beta1(1) = DSQRT(1.d0+Vbuo+Vbuoc)
c     beta2(1) = beta1(1)
c            do kkk=2,21
c               costheta = dcos(DBLE(kkk-1)*0.1d0*pi)
c               costheta2 = costheta*costheta
c        Vbuoc = 2.d0*Vk/u0(i,j)*costheta
c        beta1(kkk) = DSQRT(costheta2+Vbuo+Vbuoc)
c        beta2(kkk) = beta1(kkk)*costheta
c        intbeta1(i,j) = 0.025d0*(beta1(kkk)+beta1(kkk-1))
c     . +intbeta1(i,j)
c        intbeta2(i,j) = 0.025d0*(beta2(kkk)+beta2(kkk-1))
c     . +intbeta2(i,j)
c            end do

c Ib's approximation for beta1 and beta2

           uoVb = Vb/u0(i,j)
            intbeta1(i,j)= dsqrt(4*aux4*aux4+uoVb*uoVb)
            intbeta2(i,j)=0.5d0*dcos(miu)
     .                    *DTANH(4.d0*aux4*uoVb)



            aux2 = 0.5d0*fcwij(i,j)*u0(i,j)
      frict_y(i,j) = factor*aux2*( Vby*intbeta1(i,j)
       .         + intbeta2(i,j)*u0(i,j)*dsin(theta(i,j)))
      frict_x(i,j) = factor*aux2*( Vbx*intbeta1(i,j)
```

```
     .         + intbeta2(i,j)*u0(i,j)*dcos(theta(i,j)))
          end do

          do i=nxa(j)+1,nx
        frict_y(i,j) = 0.d0
        frict_x(i,j) = 0.d0
          end do
        end do


        return
        end
```

8.1.13 Subroutines average_3ptx and average_3pty

These subroutines do 3 point averages in the x and y direction.

```
c ######################################################################
c ######################################################################
      subroutine average_3ptx(npx,npy,afdata)
c
c     This subroutine performs a 3-point averaging filter along the
c     x-direction for every y=const row.
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, j, npx, npy
      real*8 fdata(nxm,nym), afdata(nxm,nym)
c======================================================================

      do j=1,npy
         do i=1,npx
            fdata(i,j)=afdata(i,j)
         end do
         do i=2, npx-1
            afdata(i,j) = 0.25d0*(fdata(i-1,j)+fdata(i+1,j))
     .                    +0.5d0*fdata(i,j)
         end do
         afdata(1,j)=fdata(1,j)
         afdata(npx,j)=fdata(npx,j)
```

```
      end do

      return
      end



c ######################################################################
      subroutine average_3pty(npx,npy,afdata)
c
c     This subroutine performs a 3-point averaging filter along the
c     y-direction for every x=const row.
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, j, npx, npy
      real*8 fdata(nxm,nym), afdata(nxm,nym)
c======================================================================

      do i=1,npx
         do j=1,npy
           fdata(i,j)=afdata(i,j)
         end do
         do j=2,npy-1
            afdata(i,j) = 0.25d0*(fdata(i,j-1)+fdata(i,j+1))
     .                    +0.5d0*fdata(i,j)
         end do
         afdata(i,1)=fdata(i,1)
         afdata(i,npy)=fdata(i,npy)
      end do

      return
      end
c ######################################################################
c ######################################################################
```

8.1.14 Subroutines spline and splint

These subroutines do a cubic spline.

```
c ######################################################################
      subroutine spline(xin,yin,n,y2)
c
c    This subroutine will compute the cubic spline for an array.
c
c ######################################################################
      include 'winc_std_common.inc'

      integer n,i,kk
      real*8 xin(nxm),yin(nxm),y2(nxm),p,qn,sig,un,u(nxm)

      y2(1)=0.d0
      u(1)=0.d0
      do i=2,n-1
         sig=(xin(i)-xin(i-1))/(xin(i+1)-xin(i-1))
         p=sig*y2(i-1)+2.d0
         y2(i)=(sig-1.d0)/p
         u(i)=(6.d0*((yin(i+1)-yin(i))/(xin(i+1)-xin(i))
     .         -(yin(i)-yin(i-1))
     .         /(xin(i)-xin(i-1)))/(xin(i+1)-xin(i-1))-sig*u(i-1))/p
      end do
      qn=0.d0
      un=0.d0
      y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.d0)

      do kk=n-1,1,-1
         y2(kk)=y2(kk)*y2(kk+1)+u(kk)
      end do
      return
      end

c ######################################################################
      SUBROUTINE SPLINT(xin,yin,y2,n,xout,yout,index,merge,j)
c
c    This subroutine will compute values from the cubic spline
c
c ######################################################################
      include 'winc_std_common.inc'

      integer n,i,kk,khi,klo,index,j
      real*8 xin(nxm),yin(nxm),y2(nxm),xout(nxm),yout(nxm),aa,bb,hh

      do i=index,index+merge
```

```
      KLO=1
      KHI=N
1     IF (KHI-KLO.GT.1) THEN
        kk=(KHI+KLO)/2
        IF(xin(kk).GT.xout(i))THEN
          KHI=kk
        ELSE
          KLO=kk
        ENDIF
      GOTO 1
      ENDIF
      hh=xin(KHI)-xin(KLO)
      IF (hh.EQ.0.) PAUSE 'Bad xin input.'
      aa=(xin(KHI)-xout(i))/hh
      bb=(xout(i)-xin(KLO))/hh
      yout(i)=aa*yin(KLO)+bb*yin(KHI)+
     *      ((aa**3-aa)*y2(KLO)+(bb**3-bb)*y2(KHI))*(hh**2)/6.

      end do
      RETURN
      END
```

8.2 File *winc_time.f*

This file contains the subroutine time_model which peforms the time integration. This is the bulk of the model.

```
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
c ******************************************************************************
c TIME MODEL
      subroutine time_model
c     includes 3-D dispersion coded by KAH
c LAST MODIFICATION:
c - May 6, 2002
c Version 2.0
c ******************************************************************************
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local functions ------------------------------------------------
      real*8 cel, ux, uy, force_xx, force_xy,
     . force_yx, force_yy, wind_x, wind_y
```

```
c--- Local variables ------------------------------------------------
      integer i, j, ncode, iqxfilter, iqyfilter, nxorder, nyorder,
     . nfilter1, nfilter2, nord, kounter, itstep, jj,
     . nxflag, iunit,
     . izxfilter, izyfilter

      character*3 name1,name2(100),name3(100)

      real*8 h1, z1, t, qx1, qy1,
     . qxqy(nxm,nym), qxqx(nxm,nym), qyqy(nxm,nym),
     . qxqxdx(nxm,nym), qxqydx(nxm,nym),
     . qyqydy(nxm,nym), qxqydy(nxm,nym),
     . qwxqwy(nxm,nym), qwxqwx(nxm,nym), qwyqwy(nxm,nym),
     . qwxqwxdx(nxm,nym), qwxqwydx(nxm,nym),
     . qwyqwydy(nxm,nym), qwxqwydy(nxm,nym),
     . dqxdxn(nxm,nym),vtsum(nxm,nym),dvtsumdx(nxm,nym),
     . dqydyn(nxm,nym),  dvtsumdy(nxm,nym),
     . dzetadxn(nxm,nym), dzetadyn(nxm,nym),
     . frictx(nxm,nym), fricty(nxm,nym),
     . coeff1(10), coeff2(10), denom1, denom2,
     . betanp1(3,nym), bn(nym), bstar(nym),
     . zetaup, qxup, qyup, bnup, vert,
     . zetanp1(nxm,nym), qxnp1(nxm,nym), qynp1(nxm,nym),
     . Qi, qxr, alphanew, alpha2(nym),
     . dzetadxstar(nxm,nym), dzetadystar(nxm,nym),
     . dqxdxstar(nxm,nym), dqydystar(nxm,nym),
     . momxstar(nxm,nym), momystar(nxm,nym),
     . momxn(nxm,nym), momyn(nxm,nym),dtauyydy(nxm,nym),
     . cstar(nxm,nym), mxstar(nxm,nym), mystar(nxm,nym),
     . dtauxydx(nxm,nym),tauxx(nxm,nym),tauyy(nxm,nym),
     . dtauxydy(nxm,nym), dtauxxdx(nxm,nym),
     . aux(nmax+1), aux2(nmax+1), aux3(nmax+1), aux4(nmax+1),
     . aux5(nmax+1), aux6(nmax+1), aux1, mass0, energy0,
     . mass, energy, qxtotal, qytotal, xfluxin, yfluxin,
     . dx3(nym),factor,
     . wind_angle, windx, windy, cd,
     . Dxx(nxm,nym), Dxy(nxm,nym), Dyy(nxm,nym),dVo_dxdy(nxm,nym),
     . Mxx(nxm,nym), Mxy(nxm,nym), Myy(nxm,nym), dMxx_dx(nxm,nym),
     . dMxy_dx(nxm,nym), dMxy_dy(nxm,nym), dMyy_dy(nxm,nym),
     . disp1(nxm,nym), disp2(nxm,nym), disp3(nxm,nym),
     . disp1_dx(nxm,nym), disp2_dx(nxm,nym), disp2_dy(nxm,nym),
     . disp3_dy(nxm,nym), dispx(nxm,nym), dispy(nxm,nym),
     . htot, ht2, ht3, ht4,  dUo_dy2(nxm,nym),dUo_dxdy(nxm,nym),
```

```
     . Uo(nxm,nym), Vo(nxm,nym), dUo_dx(nxm,nym), dUo_dy(nxm,nym),
     . dVo_dx(nxm,nym), dVo_dy(nxm,nym), qyb3(nxm), qyb4(nxm),
     . Axxx(nxm,nym), Axxy(nxm,nym), Axyx(nxm,nym), Axyy(nxm,nym),
     . Ayyx(nxm,nym), Ayyy(nxm,nym), dVo_dx2(nxm,nym), dVo_dy2(nxm,nym),
     . Bxx(nxm,nym), Bxy(nxm,nym), Byy(nxm,nym),dUo_dx2(nxm,nym),
     . adv1(nxm,nym), adv2(nxm,nym), adv3(nxm,nym), adv3_dy(nxm,nym),
     . adv1_dx(nxm,nym), adv2_dx(nxm,nym), adv2_dy(nxm,nym),
     . average, adqxdx(nxm,nym), adqydy(nxm,nym), africtx(nxm,nym),
     . adtaudy(nxm,nym), adzdx(nxm,nym), aqxqxdx(nxm,nym),
     . aqxqydy(nxm,nym), adispx(nxm,nym), africty(nxm,nym),
     . adtaudx(nxm,nym), adzdy(nxm,nym), aqyqydy(nxm,nym),
     . aqxqydx(nxm,nym), adispy(nxm,nym),vs(nxm,nym),
     . azeta(nxm,nym), aqxn(nxm,nym), aqyn(nxm,nym),
     . inv_ht, htt(nxm,nym), inv_denom1, inv_denom2, inv_average


c=====================================================================
c ## define functions that will be used through out the routine -----
c
c ## WARNING: they look like variables but behave like functions,
c ## and must follow imediately the dimension statements
c---------------------------------------------------------------------

      cel(h1,z1) = DSQRT(g*(h1+z1))
      ux(qx1,z1,h1) = qx1/(h1+z1)
      uy(qy1,z1,h1) = qy1/(h1+z1)
      force_xx(i,j,t)= DTANH(t*delay)*dSxxdxn(i,j)
      force_xy(i,j,t)= DTANH(t*delay)*dSxydyn(i,j)
      force_yx(i,j,t)= DTANH(t*delay)*dSxydxn(i,j)
      force_yy(i,j,t)= DTANH(t*delay)*dSyydyn(i,j)
      wind_x(t) = DTANH(t*delay)*windx
      wind_y(t) = DTANH(t*delay)*windy

c---------------------------------------------------------------------
      write(*,903)' Start time model'

c ## default filter parameters
      if (ipery.eq.1.) then
        iqyfilter=5
        izyfilter=5
      else
        iqyfilter=6
        izyfilter=2
```

```
      end if
      izxfilter=6
      iqxfilter=6
      nxorder=16
      nyorder=16
      nfilter1 = 1
      nfilter2 = 1
c-----------------------------------------------------------------------
c ## get coefficients for scheme

      nord = 2
      call predcor(nord,coeff1,coeff2,denom1,denom2)
      inv_denom1 = 1.d0/denom1*dt
      inv_denom2 = 1.d0/denom2*dt

c-----------------------------------------------------------------------
c ## initialize array and read Longshore velocities at boundaries 3&4
      do j=1,ny
        dx3(j) = dx
      end do

      if (ibc3.eq.1) then
         open(unit=14, file='qyb3.in', status='old')
         do i=1,nx
           read(14,*) qyb3(i)
         end do
         write(6,905) (qyb3(i),i=1,nx)
      end if
      if (ibc4.eq.1) then
         open(unit=15, file='qyb4.in', status='old')
         do i=1,nx
           read(15,*)qyb4(i)
         end do
         write(6,905) (qyb4(i),i=1,nx)
      end if

c-----------------------------------------------------------------------
c ## Calculate initial mass, energy
      mass=0.d0
      energy=0.d0
      do j=1,ny
        do i=1,nx
  energy = 0.5d0*( zetan(i,j)*zetan(i,j)-ht(i,j)*ht(i,j)
```

```
     . +(qxn(i,j)*qxn(i,j)+qyn(i,j)*qyn(i,j))
     . /(g*(ht(i,j)+zetan(i,j))) ) + energy
        end do
       end do
c
c calculate mass based on the trapezoidal rule
c
mass=.25d0*(zetan(1,1)+zetan(1,ny)+zetan(nx,1)+zetan(nx,ny)
     .  +ht(1,1)+ht(1,ny)+ht(nx,1)+ht(nx,ny))
do j=2,ny-1
  mass=mass+(zetan(1,j)+zetan(nx,j)+ht(1,j)+ht(nx,j))*.5d0
  do i=2,nx-1
    mass =mass+zetan(i,j)+ht(i,j)
  end do
end do

do i=2,nx-1
  mass=mass+(zetan(i,1)+zetan(i,ny)+ht(i,1)+ht(i,ny))*.5d0
end do
mass=mass*dx*dy

      mass0=mass
      energy0=energy

c-----------------------------------------------------------------------
c ## Calculate wind shear stress

      wind_angle = wind_dir*pi/180.d0
      cd = 1.3915d-3
      windx = 1.2/rho*cd*wind_vel*wind_vel*dcos(wind_angle)
      windy = -1.2/rho*cd*wind_vel*wind_vel*dsin(wind_angle)

c-----------------------------------------------------------------------
c ## set averages to zero
        do j = 1,ny
          do i=1,nx
            adqxdx(i,j) = 0.d0
            adqydy(i,j) = 0.d0
            africtx(i,j) = 0.d0
            adtaudy(i,j) = 0.d0
            adzdx(i,j) = 0.d0
            aqxqxdx(i,j) = 0.d0
            aqxqydy(i,j) = 0.d0
```

```
            adispx(i,j) = 0.d0
            africty(i,j) = 0.d0
            adtaudx(i,j) = 0.d0
            adzdy(i,j) = 0.d0
            aqyqydy(i,j) = 0.d0
            aqxqydx(i,j) = 0.d0
            adispy(i,j) = 0.d0
            azeta(i,j) = 0.d0
            aqxn(i,j) = 0.d0
            aqyn(i,j) = 0.d0
          end do
        end do
c-----------------------------------------------------------------------
c ## Start Time Loop
c-----------------------------------------------------------------------
      write(6,*) 'Start Time Loop'
      kounter=0
      t = 0.d0
      average = 0.d0
      do itstep = 1,ntime
      if (mod(itstep,10).eq.0) write(6,*) ' step', itstep
c  Recalculate the forcing every wc time steps
        if(wc.eq.0) goto 5
c        if (wc.ne.0.and.mod(itstep,wc).eq.0) then
         if (mod(itstep,wc).eq.0) then
  write(*,*)'recalculating forcing'
  call shortwave_driver
          call wave_forcing
end if
 5      continue
c-----------------------------------------------------------------------
c ## Compute RHS at time at predictor-level
c-----------------------------------------------------------------------

      ncode =1
      nxflag=1
      call deriv(zetan,qxn,qyn,dzetadxn,dzetadyn,dqxdxn,dqydyn,
     .           ncode,nxflag)

      if ((ibc1.eq.4)) then
         do j = 1,ny
  dzetadxn(1,j)=0.d0
        end do
```

```
      end if
      if ((ibc2.eq.4.or.ibc2.eq.6)) then
         do j = 1,ny
dzetadxn(nxa(j),j)=0.d0
         end do
      end if
      if ((ibc3.eq.4)) then
         do i = 1,nx
dzetadyn(i,1)=0.d0
         end do
      end if
      if ((ibc4.eq.4)) then
         do i = 1,nx
dzetadyn(i,ny)=0.d0
         end do
      end if

      do j=1,ny
        do i=1,nx
          htot = ht(i,j) + zetan(i,j)
          htt(i,j) = htot
          inv_ht = 1.d0/htot
          qxqx(i,j) = qxn(i,j)*qxn(i,j)*inv_ht
          qxqy(i,j) = qxn(i,j)*qyn(i,j)*inv_ht
          qyqy(i,j) = qyn(i,j)*qyn(i,j)*inv_ht
          Uo(i,j) = ux(qxn(i,j),ht(i,j),zetan(i,j))
          Vo(i,j) = uy(qyn(i,j),ht(i,j),zetan(i,j))
qwxqwy(i,j)=qwx(i,j)*qwy(i,j)*inv_ht
qwyqwy(i,j)=qwy(i,j)*qwy(i,j)*inv_ht
qwxqwx(i,j)=qwx(i,j)*qwx(i,j)*inv_ht
        end do
      end do

      ncode =1
      nxflag=1
      call deriv(qxqy,qxqx,qyqy,qxqydx,qxqydy,qxqxdx,qyqydy,
     .  ncode,nxflag)

      ncode =3
      nxflag=1
      call deriv(qwxqwy,qwxqwx,qwyqwy,qwxqwydx,qwxqwydy,
     . qwxqwxdx,qwyqwydy,ncode,nxflag)
```

```
      call average_3ptx(nx,ny,qwxqwxdx)
      call average_3ptx(nx,ny,qwxqwydx)
      call average_3ptx(nx,ny,qwxqwydy)
      call average_3ptx(nx,ny,qwyqwydy)
      call average_3pty(nx,ny,qwxqwxdx)
      call average_3pty(nx,ny,qwxqwydx)
      call average_3pty(nx,ny,qwxqwydy)
      call average_3pty(nx,ny,qwyqwydy)

c ## call bottom friction ----------------------------------------------

      factor=DTANH(t*delay)
      call bottom_friction(zetan,qxn,qyn,frictx,fricty,
     .                     factor,itstep)

      ncode =3
      nxflag=1
      call deriv(Uo, Vo, Vo, dUo_dx, dUo_dy, dVo_dx, dVo_dy,
     .           ncode,nxflag)


c ## eddy viscosity -----------------------------------------------------
cksm     include the effect of SGS Turbulence using the
c     Smagorinsky eddy viscosity model

      do j=1,ny
 do i=1,nxa(j)
            vs(i,j)=Cs*Cs*dx*dy*sqrt(dUo_dx(i,j)*dUo_dx(i,j)*2.d0
     .           +dVo_dy(i,j)*dVo_dy(i,j)*2.d0
     .           +(dUo_dy(i,j)+dVo_dx(i,j))*(dUo_dy(i,j)+dVo_dx(i,j)))
   vtsum(i,j) = (v_t(i,j)+vs(i,j))*(ht(i,j)+zetan(i,j))
 end do
 do i=nxa(j)+1,nx
            vs(i,j)=0.d0
 end do
      end do

      call deriv(vtsum,tauxx,tauyy,dvtsumdx,dvtsumdy,dtauxxdx,
     .           dtauyydy,3,nxflag)

      call deriv(Uo, Vo, Vo, dUo_dx2, dUo_dy2, dVo_dx2, dVo_dy2,
     .           4,nxflag)
```

```
      call dmixed(Uo, Vo, dUo_dxdy, dVo_dxdy,
     .           nxflag)

       do j=1,ny
     do i=1,nxa(j)

            dtauxxdx(i,j)=dvtsumdx(i,j)*
     .  (dUo_dx(i,j)+dUo_dx(i,j))
     .          +(v_t(i,j)+vs(i,j))*(ht(i,j)+zetanp1(i,j))
     .           *(dUo_dx2(i,j)+dUo_dx2(i,j))

            dtauyydy(i,j)=dvtsumdy(i,j)*
     .  (dVo_dy(i,j)+dVo_dy(i,j))
     .          +(v_t(i,j)+vs(i,j))*(ht(i,j)+zetanp1(i,j))
     .           *(dVo_dy2(i,j)+dVo_dy2(i,j))

            dtauxydx(i,j)=dvtsumdx(i,j)*
     .  (dVo_dx(i,j)+dUo_dy(i,j))
     .          +(v_t(i,j)+vs(i,j))*(ht(i,j)+zetan(i,j))*
     .           (dVo_dx2(i,j)+dUo_dxdy(i,j))

            dtauxydy(i,j)=dvtsumdy(i,j)*
     .  (dVo_dx(i,j)+dUo_dy(i,j))
     .          +(v_t(i,j)+vs(i,j))*(ht(i,j+zetan(i,j)))
     .            *(dUo_dy2(i,j)+dVo_dxdy(i,j))
      end do
     do i=nxa(j)+1,nx
            dtauxxdx(i,j)=0.d0
            dtauyydy(i,j)=0.d0
            dtauxydx(i,j)=0.d0
            dtauxydy(i,j)=0.d0
      end do
        end do

c ## 3D dispersion, coded by KAH for Ap's reduced version ----------------

      if(disp3d.eq.0) then
         do j=1,ny
           do i=1,nx
             dispx(i,j)=0.d0
             dispy(i,j)=0.d0
           end do
         end do
```

```
         goto 554
      end if

        call UV_profile(zetan,frictx,fricty,t,windx,windy,vs)
        call coefcalc(Dxx,Dyy,Dxy,zetan,Mxx,Myy,Mxy,Bxx,Bxy,Byy,
     .       Axxx,Axxy,Axyx,Axyy,Ayyx,Ayyy,vs,Uo,Vo)

      do j=1,ny
do i=1,nxa(j)
           htot = ht(i,j) + zetan(i,j)
           ht2 = htot*htot
           ht3 = ht2*htot
           ht4 = ht2*ht2
           adv1(i,j) = Axxx(i,j)*Uo(i,j)+Axxy(i,j)*Vo(i,j)
           adv2(i,j) = Axyx(i,j)*Uo(i,j)+Axyy(i,j)*Vo(i,j)
           adv3(i,j) = Ayyx(i,j)*Uo(i,j)+Ayyy(i,j)*Vo(i,j)
           disp1(i,j) = ( (2.d0*Dxx(i,j)+Bxx(i,j))*dUo_dx(i,j)
     .                  +2.d0*Dxy(i,j)*dUo_dy(i,j)
     .                  +Bxx(i,j)*dVo_dy(i,j))*htot
           disp2(i,j) = ( Dxx(i,j)*dVo_dx(i,j)
     .                  +(Dxy(i,j)+Bxy(i,j))*dVo_dy(i,j)
     .                  +(Dxy(i,j)+Bxy(i,j))*dUo_dx(i,j)
     .                  + Dyy(i,j)*dUo_dy(i,j)
     .                  )*htot
           disp3(i,j) = ( 2.d0*Dxy(i,j)*dVo_dx(i,j)
     .                  +(2.d0*Dyy(i,j)+Byy(i,j))*dVo_dy(i,j)
     .                  +(Byy(i,j))*dUo_dx(i,j)
     .                  )*htot
end do
do i=nxa(j)+1,nx
           disp1(i,j) = 0.d0
           disp2(i,j) = 0.d0
           disp3(i,j) = 0.d0
end do
      end do
      ncode =3
      nxflag=1
      call deriv(disp2, disp1, disp3, disp2_dx, disp2_dy, disp1_dx,
     .  disp3_dy, ncode, nxflag)
      call deriv(Mxy, Mxx, Myy, dMxy_dx, dMxy_dy, dMxx_dx,
     .  dMyy_dy, ncode, nxflag)
      call deriv(adv2, adv1, adv3, adv2_dx, adv2_dy, adv1_dx,
     .           adv3_dy, ncode, nxflag)
```

```
c ## do 3pt-average of terms in dispx ------------------
      call average_3ptx(nx,ny,disp1_dx)
      call average_3ptx(nx,ny,disp2_dy)

c ## do 3pt-average of terms in dispy ------------------
      call average_3ptx(nx,ny,disp2_dx)
      call average_3ptx(nx,ny,disp3_dy)

c ## assemble the dispersion terms -----------
      do j=1,ny
 do i=1,3
           dispx(i,j) = 0.d0
           dispy(i,j) = 0.d0
           qwxqwxdx(i,j) = 0.d0
           qwxqwydy(i,j) = 0.d0
           qwxqwydx(i,j) = 0.d0
           qwyqwydy(i,j) = 0.d0
 end do
 do i=4,nxa(j)
           dispx(i,j) = disp1_dx(i,j) + disp2_dy(i,j)
     .           -dMxx_dx(i,j)-dMxy_dy(i,j)
     .           -adv1_dx(i,j)-adv2_dy(i,j)
           dispy(i,j) = disp2_dx(i,j) + disp3_dy(i,j)
     .          -dMyy_dy(i,j)-dMxy_dx(i,j)
     .          -adv2_dx(i,j)-adv3_dy(i,j)
 end do
 do i=nxa(j)+1,nx
           dispx(i,j) = 0.d0
           dispy(i,j) = 0.d0
           qwxqwxdx(i,j) = 0.d0
           qwxqwydy(i,j) = 0.d0
           qwxqwydx(i,j) = 0.d0
           qwyqwydy(i,j) = 0.d0
 end do
      end do
c ## do 3pt-average of dispx and dispy ------------------
      call average_3ptx(nx,ny,dispx)
      call average_3pty(nx,ny,dispx)
      call average_3ptx(nx,ny,dispy)
      call average_3pty(nx,ny,dispy)
ckk 12/16/99
      do j=1,ny
 do i=1,nx
```

```
            qwxqwxdx(i,j)=qwxqwxdx(i,j)
     .             *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
            qwxqwydy(i,j)=qwxqwydy(i,j)
     .             *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
            qwxqwydx(i,j)=qwxqwydx(i,j)
     .             *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
            qwyqwydy(i,j)=qwyqwydy(i,j)
     .             *dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
            dispx(i,j)=dispx(i,j)*dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
            dispy(i,j)=dispy(i,j)*dtanh((dble((nx-i)/dble(nx))/.02d0)**4)
 end do
      end do
ckk 12/16/99
554   continue

c ## calculate RHS of continuity & momentum eqs' ----------------------
c ## predictor-level
      factor=DTANH(t*delay*0.5d0)

      do j = 1,ny
        do i = 1,nxa(j)

          cn(i,j) = - (dqxdxn(i,j) + dqydyn(i,j))

          momxn(i,j) = 0.d0
     .    - force_xx(i,j,t)
     .    - force_xy(i,j,t)
     .    - frictx(i,j)
     .    + dtauxydy(i,j)
     .    + dtauxxdx(i,j)
     .    + wind_x(t)

          mxn(i,j) = momxn(i,j)
     .    - g*htt(i,j)*dzetadxn(i,j)
     .    - qxqxdx(i,j) -qxqydy(i,j)
     .    + dispx(i,j)*factor
c     .     + (qwxqwxdx(i,j))*factor
     .    + (qwxqwxdx(i,j) +qwxqwydy(i,j))*factor

          momyn(i,j) =  0.d0
     .    - force_yx(i,j,t)
     .    - force_yy(i,j,t)
     .    - fricty(i,j)
```

```
     .     + dtauxydx(i,j)
     .     + dtauyydy(i,j)
     .     + wind_y(t)

           myn(i,j) = momyn(i,j)
     .     - g*htt(i,j)*dzetadyn(i,j)
     .     - qyqydy(i,j) -qxqydx(i,j)
     .     + dispy(i,j)*factor
c    .      + (qwyqwydy(i,j))*factor
     .     + (qwxqwydx(i,j) +qwyqwydy(i,j))*factor

         end do
       end do


         do j = 1,ny
            do i=nxa(j)+1,nx
              cn(i,j)=0.d0
              momxn(i,j)=0.d0
              mxn(i,j)=0.d0
              momyn(i,j)=0.d0
              myn(i,j)=0.d0
            end do
         end do


c-----------------------------------------------------------------------
c ## Characteristix on x=0

       if ((ibc1.eq.2).or.(ibc1.eq.3))
      . call betachar(bn,betan,zetan,qxn,qyn,dqydyn,dzetadyn,momxn)

c-----------------------------------------------------------------------
c ## The Predictor Step
c-----------------------------------------------------------------------
         do i = 1,nx
            do j = 1,ny

              zetaup = coeff1(1)*cn(i,j) + coeff1(2)*cnm1(i,j) +
     .                 coeff1(3)*cnm2(i,j)
              qxup = coeff1(1)*mxn(i,j) + coeff1(2)*mxnm1(i,j) +
     .                 coeff1(3)*mxnm2(i,j)
              qyup = coeff1(1)*myn(i,j) + coeff1(2)*mynm1(i,j) +
```

```
                    coeff1(3)*mynm2(i,j)

              zetanp1(i,j) = zetan(i,j) + zetaup*inv_denom1
              qxnp1(i,j)   = qxn(i,j)   + qxup*inv_denom1
              qynp1(i,j)   = qyn(i,j)   + qyup*inv_denom1

            end do
          end do


c ##   Check for negative water depths
       do j = 1,ny
        do i = nxa(j)+1,nx
           qxnp1(i,j)=0.d0
           qynp1(i,j)=0.d0
           zetanp1(i,j)=-ht(i,j)+0.0001d0
        end do
       end do


c-----------------------------------------------------------------------
c ## Flux Conditions


      if ((ibc1.eq.2)) then
         do j = 1,ny
              qxnp1(1,j) = 0.d0
         end do
      end if

      if (ibc1.eq.4) then
         do j=1,ny
              qxnp1(1,j) = 0.d0
         end do
      end if

      if (ibc2.eq.4) then
         do j = 1,ny
              qxnp1(nxa(j),j) = 0.d0
         end do
      end if

      if (ibc2.eq.6) then
```

```
         do j = 1,ny
              qxnp1(nxa(j),j) = 0.d0
              qynp1(nxa(j),j) = 0.d0
         end do
      end if

      if (ibc3.eq.1) then
         do i=1,nx
              qynp1(i,1) = qyb3(i)*factor
         end do
      end if

      if (ibc4.eq.1) then
         do i=1,nx
              qynp1(i,ny) = qyb4(i)*factor
         end do
      end if

      if (ibc3.eq.4) then
         do i=1,nx
              qynp1(i,1) = 0.d0
         end do
      end if

      if (ibc4.eq.4) then
         do i=1,nx
              qynp1(i,ny) = 0.d0
         end do
      end if
      if (ibc3.eq.5.and.ibc4.eq.5) then
         do i=1,nx
              qynp1(i,1) = qynp1(i,ny)
              qxnp1(i,1) = qxnp1(i,ny)
              zetanp1(i,1) = zetanp1(i,ny)
         end do
      end if

c-----------------------------------------------------------------------
c ## Absorb/generating cond. at the boundary x=0


      if ((ibc1.eq.2).or.(ibc1.eq.3)) then
```

```
        do j=1,ny

          bnup = coeff1(1)*bn(j) + coeff1(2)*bnm1(j) +
     .  coeff1(3)*bnm2(j)
          betanp1(1,j) = betan(1,j)+ bnup*inv_denom1
          alpha2(j)=-alpha(1)
c!!! "vert" is Qyr, the vol. flux of the reflected wave
c!!! "aqy" is the incoming y-volume flux and Qi=aqy !!
c          vert = (qynp1(1,j)-aqy(x(1),y(j),t+dt,1))
c          Qi = ampli*dsin(k(1)*dcos(alpha(1))*x(1)
c     .          + k(1)*dsin(alpha(1))*y(j) - sigma*(t+dt))
c!!! For the case of no incoming waves I simplify "vert, Qi" as:
          vert = qynp1(1,j)
          Qi = 0.d0

          aux1 = Qi**2*(.75d0-dcos(alpha(1))) + Qi*DSQRT(g*ht(1,j))*
     .  ht(1,j)*(dcos(alpha(1))-1.d0)

          do jj=1,50
c---------- Higher order bound. cond. ---
c            qxr = dcos(alpha2(j))*(DSQRT(g*ht(1,j))*ht(1,j)
c     .   *(dcos(alpha2(j))+1.d0)+Qi*(dcos(alpha(1))
c     . -dcos(alpha2(j))-1.5d0))/(2.d0*dcos(alpha2(j))+1.5d0)
c     .   *(-1.d0+DSQRT(1.d0-( (4.d0*dcos(alpha2(j))+3.0d0)
c     .   *(aux1 - (DSQRT(g*ht(1,j))*ht(1,j))**2
c     .   *(betanp1(1,j)/DSQRT(g*ht(1,j))+2.d0))
c     .   /(DSQRT(g*ht(1,j))*ht(1,j)*(dcos(alpha2(j))+1.d0)+Qi
c     .   *(dcos(alpha(1))-dcos(alpha2(j))-1.5d0))**2)))
c---------- Lower order bound. cond. ---
           qxr = dcos(alpha2(j))/(dcos(alpha2(j))+1.d0)
     .    *(ht(1,j)*(betanp1(1,j)+2.d0*DSQRT(g*ht(1,j)))
     .          -Qi*(dcos(alpha(1))-1.d0))

            if (vert.eq.0.d0 .and. qxr.eq.0.d0) then
               alphanew = 0.d0
            else
               alphanew = datan2(vert,qxr)
            end if
           if (alphanew .gt. (pi*0.5d0)) alphanew=alphanew-pi
           if (alphanew .le. (-pi*0.5d0)) alphanew=alphanew+pi

c---------- Ap's criterion:
c            if (dabs(alphanew-alpha2(j)).le.0.0000001) goto 1000
```

```
            if(dabs(alphanew-alpha2(j)).lt.0.001) goto 1000
            alpha2(j) = alphanew

          end do
1000      continue

          qxnp1(1,j) = Qi*dcos(alpha(1)) + qxr

        end do

      end if


c-----------------------------------------------------------------------
c ## Calculate RHS at corrector level
c-----------------------------------------------------------------------

      ncode =1
      nxflag=1
      call deriv(zetanp1,qxnp1,qynp1,dzetadxstar,dzetadystar,
     .           dqxdxstar,dqydystar,ncode,nxflag)

      if ((ibc1.eq.4)) then
         do j = 1,ny
  dzetadxstar(1,j)=0.d0
         end do
      end if
      if ((ibc2.eq.4.or.ibc2.eq.6)) then
         do j = 1,ny
  dzetadxstar(nxa(j),j)=0.d0
         end do
      end if
      if ((ibc3.eq.4)) then
         do i = 1,nx
  dzetadystar(i,1)=0.d0
         end do
      end if
      if ((ibc4.eq.4)) then
         do i = 1,nx
  dzetadystar(i,ny)=0.d0
         end do
      end if
```

```
      do j=1,ny
        do i=1,nx
          htot = ht(i,j) + zetanp1(i,j)
          htt(i,j) = htot
          inv_ht = 1.d0/htot
          qxqx(i,j) = qxnp1(i,j)*qxnp1(i,j)*inv_ht
          qxqy(i,j) = qxnp1(i,j)*qynp1(i,j)*inv_ht
          qyqy(i,j) = qynp1(i,j)*qynp1(i,j)*inv_ht
          Uo(i,j) = ux(qxnp1(i,j),ht(i,j),zetanp1(i,j))
          Vo(i,j) = uy(qynp1(i,j),ht(i,j),zetanp1(i,j))
        end do
      end do

      ncode =1
      nxflag=1
      call deriv(qxqy,qxqx,qyqy,qxqydx,qxqydy,qxqxdx,qyqydy,
     .  ncode,nxflag)


c ## call bottom friction  ----------------------------------------

ckk       factor=0.d0
      factor=DTANH(t*delay)
      call bottom_friction(zetanp1,qxnp1,qynp1,frictx,fricty,
     .    factor,itstep)
      ncode =3
      nxflag=1
      call deriv(Uo, Vo, Vo, dUo_dx, dUo_dy, dVo_dx, dVo_dy,
     .           ncode,nxflag)

c ## eddy viscosity  ------------------------------------------------
c
c     Inculudes the Smagorinsky eddy viscosity (vs)
c

      do j=1,ny
 do i=1,nxa(j)
            vs(i,j)=Cs*Cs*dx*dy*sqrt(dUo_dx(i,j)*dUo_dx(i,j)*2.d0
     .           +dVo_dy(i,j)*dVo_dy(i,j)*2.d0
     .           +(dUo_dy(i,j)+dVo_dx(i,j))*(dUo_dy(i,j)+dVo_dx(i,j)))

    vtsum(i,j) = (v_t(i,j)+vs(i,j))*(ht(i,j)+zetanp1(i,j))
         end do
```

```
do i=nxa(j)+1,nx
           vs(i,j)=0.d0
end do
     end do

     call deriv(vtsum,tauxx,tauyy,dvtsumdx, dvtsumdy,dtauxxdx,
    .           dtauyydy,3,nxflag)
     call deriv(Uo, Vo, Vo, dUo_dx2, dUo_dy2, dVo_dx2, dVo_dy2,
    .           4,nxflag)
     call dmixed(Uo, Vo, dUo_dxdy, dVo_dxdy,
    .           nxflag)

      do j=1,ny
  do i=1,nxa(j)
           dtauxxdx(i,j)=dvtsumdx(i,j)*
    .  (dUo_dx(i,j)+dUo_dx(i,j))
    .          +(v_t(i,j)+vs(i,j))*(ht(i,j)+zetanp1(i,j))
    .           *(dUo_dx2(i,j)+dUo_dx2(i,j))

           dtauyydy(i,j)=dvtsumdy(i,j)*
    .  (dVo_dy(i,j)+dVo_dy(i,j))
    .          +(v_t(i,j)+vs(i,j))*(ht(i,j)+zetanp1(i,j))
    .           *(dVo_dy2(i,j)+dVo_dy2(i,j))

           dtauxydx(i,j)=dvtsumdx(i,j)*
    .  (dVo_dx(i,j)+dUo_dy(i,j))
    .          +(v_t(i,j)+vs(i,j))*(ht(i,j)+zetanp1(i,j))
    .           *(dVo_dx2(i,j)+dUo_dxdy(i,j))

           dtauxydy(i,j)=dvtsumdy(i,j)*
    .  (dVo_dx(i,j)+dUo_dy(i,j))
    .          +(v_t(i,j)+vs(i,j))*(ht(i,j)+zetanp1(i,j))
    .            *(dUo_dy2(i,j)+dVo_dxdy(i,j))
   end do
  do i=nxa(j)+1,nx
           dtauxxdx(i,j)=0.d0
           dtauyydy(i,j)=0.d0
           dtauxydx(i,j)=0.d0
           dtauxydy(i,j)=0.d0
   end do
      end do

c ## calculate RHS of continuity & momentum eqs' ----------------------
```

```
c ## corrector-level

      factor=DTANH(t*delay*0.5d0)

      do j = 1,ny
        do i = 1,nxa(j)

           cstar(i,j)  = -(dqxdxstar(i,j) + dqydystar(i,j))

           momxstar(i,j) = 0.d0
     .     - force_xx(i,j,t)
     .     - force_xy(i,j,t)
     .     - frictx(i,j)
     .     + dtauxydy(i,j)
     .     + dtauxxdx(i,j)
     .     + wind_x(t)

           mxstar(i,j) = momxstar(i,j)
     .     -g*htt(i,j)*dzetadxstar(i,j)
     .     - qxqxdx(i,j) -qxqydy(i,j)
     .     + dispx(i,j)*factor
c    .      + (qwxqwxdx(i,j))*factor
     .     + (qwxqwxdx(i,j) +qwxqwydy(i,j))*factor

           momystar(i,j) = 0.d0
     .     - force_yx(i,j,t)
     .     - force_yy(i,j,t)
     .     - fricty(i,j)
     .     + dtauxydx(i,j)
     .     + dtauyydy(i,j)
     .     + wind_y(t)

           mystar(i,j) = momystar(i,j)
     .     - g*htt(i,j)*dzetadystar(i,j)
     .     - qyqydy(i,j) -qxqydx(i,j)
     .     + dispy(i,j)*factor
c    .      + (qwyqwydy(i,j))*factor
     .     + (qwxqwydx(i,j)+qwyqwydy(i,j))*factor

        end do
      end do
```

```
          do j=1,ny
            do i=nxa(j)+1,nx
              cstar(i,j)=0.d0
              momxstar(i,j)=0.d0
              mxstar(i,j)=0.d0
              momystar(i,j)=0.d0
              mystar(i,j)=0.d0
              cn(i,j)=0.d0
              mxn(i,j)=0.d0
              myn(i,j)=0.d0
            end do
          end do



c ## Characteristix on x = 0  ---------------------------------------


       if ((ibc1.eq.2).or.(ibc1.eq.3)) then
         do j=1,ny
           betanp1(2,j) = (ux(qxnp1(2,j),zetanp1(2,j),ht(2,j)) - 2.d0*
     .                  cel(zetanp1(2,j),ht(2,j)))
           betanp1(3,j) = (ux(qxnp1(3,j),zetanp1(3,j),ht(3,j)) - 2.d0*
     .                  cel(zetanp1(3,j),ht(3,j)))
         end do
         call betachar(bstar,betanp1,zetanp1,qxnp1,qynp1,
     .                dqydystar,dzetadystar,momxstar)
       end if


c-----------------------------------------------------------------------
c ## The Corrector Step
c-----------------------------------------------------------------------
          do j = 1,ny
             do i = 1,nx

               zetaup = coeff2(1)*cstar(i,j) + coeff2(2)*cn(i,j) +
     .                  coeff2(3)*cnm1(i,j) + coeff2(4)*cnm2(i,j)
               qxup = coeff2(1)*mxstar(i,j) + coeff2(2)*mxn(i,j) +
     .                coeff2(3)*mxnm1(i,j) + coeff2(4)*mxnm2(i,j)
               qyup = coeff2(1)*mystar(i,j) + coeff2(2)*myn(i,j) +
     .                coeff2(3)*mynm1(i,j) + coeff2(4)*mynm2(i,j)
```

```
              zetanp1(i,j) = zetan(i,j) + zetaup*inv_denom2
              qxnp1(i,j)   = qxn(i,j) + qxup*inv_denom2
              qynp1(i,j)   = qyn(i,j) + qyup*inv_denom2

              end do
           end do


c ##   Check for negative water depths
       do j = 1,ny
         do i = nxa(j)+1,nx
            qxnp1(i,j)=0.d0
            qynp1(i,j)=0.d0
            zetanp1(i,j)=-ht(i,j)+0.0001d0
            cn(i,j)=0.d0
            mxn(i,j)=0.d0
            myn(i,j)=0.d0
          end do
        end do

c-----------------------------------------------------------------------
c ## Flux Conditions (again)


      if ((ibc1.eq.2)) then
         do j = 1,ny
              qxnp1(1,j) = 0.d0
         end do
      end if

      if (ibc1.eq.4) then
         do j=1,ny
              qxnp1(1,j) = 0.d0
         end do
      end if


      if (ibc2.eq.4) then
         do j = 1,ny
             qxnp1(nxa(j),j) = 0.d0
         end do
      end if
```

```
      if (ibc2.eq.6) then
         do j = 1,ny
             qxnp1(nxa(j),j) = 0.d0
             qynp1(nxa(j),j) = 0.d0
         end do
      end if

      if (ibc3.eq.1) then
         do i=1,nx
              qynp1(i,1) = qyb3(i)*factor
         end do
      end if

      if (ibc4.eq.1) then
         do i=1,nx
              qynp1(i,ny) = qyb4(i)*factor
         end do
      end if

      if (ibc3.eq.4) then
         do i=1,nx
              qynp1(i,1) = 0.d0
         end do
      end if

      if (ibc4.eq.4) then
         do i=1,nx
              qynp1(i,ny) = 0.d0
         end do
      end if

      if (ibc3.eq.5.and.ibc4.eq.5) then
         do i=1,nx
              qynp1(i,1) = qynp1(i,ny)
              qxnp1(i,1) = qxnp1(i,ny)
              zetanp1(i,1) = zetanp1(i,ny)
         end do
      end if
c-----------------------------------------------------------------------
c ## Absorb/generating cond. at the boundary x=0


      if ((ibc1.eq.2).or.(ibc1.eq.3)) then
```

```
        do j=1,ny

          bnup = coeff2(1)*bstar(j)+ coeff2(2)*bn(j) +
     .           coeff2(3)*bnm1(j) + coeff2(4)*bnm2(j)
          betanp1(1,j) = betan(1,j)+ bnup*inv_denom2
          alpha2(j)=-alpha(1)
c!!! "aqy" is the incoming y-volume flux and Qi=aqy !!
c          vert = (qynp1(1,j)-aqy(x(1),y(j),t+dt,1))
c          Qi = ampli*dsin(k(1)*dcos(alpha(1))*x(1)
c     .          + k(1)*dsin(alpha(1))*y(j) - sigma*(t+dt))
c!!! For the case of no incoming waves I simplify "vert, Qi" as:
          vert = qynp1(1,j)
          Qi = 0.d0

          aux1 = Qi**2*(.75d0-dcos(alpha(1))) + Qi*DSQRT(g*ht(1,j))*
     .  ht(1,j)*(dcos(alpha(1))-1.d0)

          do jj=1,50
c---------- Higher order bound. cond. ---
c            qxr = dcos(alpha2(j))*(DSQRT(g*ht(1,j))*ht(1,j)
c     .   *(dcos(alpha2(j))+1.d0)+Qi*(dcos(alpha(1))
c     . -dcos(alpha2(j))-1.5d0))/(2.d0*dcos(alpha2(j))+1.5d0)
c     .   *(-1.d0+DSQRT(1.d0-( (4.d0*dcos(alpha2(j))+3.0d0)
c     .   *(aux1 - (DSQRT(g*ht(1,j))*ht(1,j))**2
c     .   *(betanp1(1,j)/DSQRT(g*ht(1,j))+2.d0))
c     .   /(DSQRT(g*ht(1,j))*ht(1,j)*(dcos(alpha2(j))+1.d0)+Qi
c     .   *(dcos(alpha(1))-dcos(alpha2(j))-1.5d0))**2)))
c---------- Lower order bound. cond. ---
           qxr = dcos(alpha2(j))/(dcos(alpha2(j))+1.d0)
     .    *(ht(1,j)*(betanp1(1,j)+2.d0*DSQRT(g*ht(1,j)))
     .          -Qi*(dcos(alpha(1))-1.d0))

            if (vert.eq.0.d0 .and. qxr.eq.0.d0) then
               alphanew = 0.d0
            else
               alphanew = datan2(vert,qxr)
            end if
            if(alphanew .gt. (pi*0.5d0)) alphanew=alphanew-pi
            if(alphanew .le. (-pi*0.5d0)) alphanew=alphanew+pi

c---------- Ap's criterion:
c            if (dabs(alphanew-alpha2(j)).le.0.0000001) goto 3000
```

```
            if(dabs(alphanew-alpha2(j)).lt.0.001) goto 3000
            alpha2(j) = alphanew

          end do
 3000     continue

          qxnp1(1,j) = Qi*dcos(alpha(1)) + qxr

        end do

      end if

c-----------------------------------------------------------------------
c ## Apply Shapiro (1970) averaging filter every "nfilter" times
c-----------------------------------------------------------------------

c--- filter the volume fluxes
        if ((mod((itstep-1),nfilter1).eq.0)) then
c ## in the x-direction for every j-point
         do j = 1,ny
            do i = 1,nxa(j)
               aux3(i) = qxnp1(i,j)
               aux4(i) = 0.d0
       aux5(i) = qynp1(i,j)
               aux6(i) = 0.d0
            end do
            call shapd1(aux3,aux4,nxa(j),nxorder,iqxfilter)
            call shapd1(aux5,aux6,nxa(j),nxorder,iqxfilter)
            do i = 1,nxa(j)
               qxnp1(i,j)= qxnp1(i,j) - aux4(i)
               qynp1(i,j)= qynp1(i,j) - aux6(i)
            end do
         end do


c ## in the y-direction for every i-point
         do i = 1,nx
            do j = 1,ny
               aux3(j) = qxnp1(i,j)
               aux4(j) = 0.d0
       aux5(j) = qynp1(i,j)
               aux6(j) = 0.d0
            end do
```

```
    if(iqyfilter.eq.5) then
                aux3(ny+1) = qxnp1(i,2)
                aux4(ny+1) = 0.d0
                aux5(ny+1) = qynp1(i,2)
                aux6(ny+1) = 0.d0
                call shapd1(aux3,aux4,ny+1,nyorder,iqyfilter)
                call shapd1(aux5,aux6,ny+1,nyorder,iqyfilter)
    else
                call shapd1(aux3,aux4,ny,nyorder,iqyfilter)
                call shapd1(aux5,aux6,ny,nyorder,iqyfilter)
    endif
            do j = 1,ny
               qxnp1(i,j)= qxnp1(i,j) - aux4(j)
               qynp1(i,j)= qynp1(i,j) - aux6(j)
            end do
         end do
         end if


c--- filter the surface elevation
        if ((mod((itstep-1),nfilter2).eq.0)) then
c ## in the x-direction for every j-point
         do j = 1,ny
            do i = 1,nxa(j)
               aux(i) = zetanp1(i,j)
               aux2(i) = 0.d0
            end do
            call shapd1(aux, aux2,nxa(j),nxorder,izxfilter)
            do i = 1,nxa(j)
               zetanp1(i,j)= zetanp1(i,j) - aux2(i)
            end do
         end do
c ## in the y-direction for every i-point
         do i = 1,nx
            do j = 1,ny
               aux(j) = zetanp1(i,j)
               aux2(j) = 0.d0
            end do
    if(izyfilter.eq.5) then
                aux(ny+1) = zetanp1(i,2)
                aux2(ny+1) = 0.d0
                call shapd1(aux, aux2,ny+1,nyorder,izyfilter)
    else
```

```
               call shapd1(aux, aux2,ny,nyorder,izyfilter)
    endif
            do j = 1,ny
               zetanp1(i,j)= zetanp1(i,j) - aux2(j)
            end do
         end do
778      continue
         end if
779      continue

c ##  Reset zetanp1, qxnp1, qynp1 in the "dry" region ----------------
       do j = 1,ny
         do i = nxa(j)+1,nx
            qxnp1(i,j)=0.d0
            qynp1(i,j)=0.d0
            zetanp1(i,j)=-ht(i,j)+0.0001d0
         end do
       end do


c--------------------------------------------------------------------
c ## Calculate time-averages of all the terms in the governing eqs.
c--------------------------------------------------------------------
      if (t.ge.5.d0/delay) then
        average = average + 1.d0
        do j = 1,ny
          do i=1,nx
            htt(i,j) = ht(i,j)+zetanp1(i,j)
            adqxdx(i,j) = dqxdxstar(i,j) + adqxdx(i,j)
            adqydy(i,j) = dqydystar(i,j) + adqydy(i,j)
            africtx(i,j) = frictx(i,j) + africtx(i,j)
            adtaudy(i,j) = dtauxydy(i,j) + adtaudy(i,j)
            adzdx(i,j) = g*htt(i,j)*dzetadxstar(i,j)
     .                   + adzdx(i,j)
            aqxqxdx(i,j) = qxqxdx(i,j) + aqxqxdx(i,j)
            aqxqydy(i,j) = qxqydy(i,j) + aqxqydy(i,j)
            adispx(i,j) = dispx(i,j) + adispx(i,j)

            africty(i,j) = fricty(i,j) + africty(i,j)
            adtaudx(i,j) = dtauxydx(i,j) + adtaudx(i,j)
            adzdy(i,j) = g*htt(i,j)*dzetadystar(i,j)
     .                   + adzdy(i,j)
            aqyqydy(i,j) = qyqydy(i,j) + aqyqydy(i,j)
```

```
            aqxqydx(i,j) = qxqydx(i,j) + aqxqydx(i,j)
            adispy(i,j) = dispy(i,j) + adispy(i,j)

            azeta(i,j) = zetan(i,j) + azeta(i,j)
            aqxn(i,j) = qxn(i,j) + aqxn(i,j)
            aqyn(i,j) = qyn(i,j) + aqyn(i,j)
          end do
        end do
      end if


c-----------------------------------------------------------------------
c ## Write values for zeta and q at various locations at the current
c ## time level
c-----------------------------------------------------------------------

        if ((mod((itstep-1),jump).eq.0)) then  !qun sep.8,1999

          do i=1,nsensor
            if (xsensor(i).ne.0 .and. ysensor(i).ne.0) then
              write(i+50,996) zetan(xsensor(i),ysensor(i)),
     .          qxn(xsensor(i),ysensor(i)), qyn(xsensor(i),ysensor(i))
            endif
          end do
        end if

c     write the files for continuing with a hot start

        if (itstep.eq.(ntime-2)) then
          open(unit=30,file='hot2.dat',access='direct',recl=4*ny)
          do i=1,nx
            write(30,rec=i) (sngl(zetan(i,j)),j=1,ny)
            write(30,rec=i+nx) (sngl(qxn(i,j)),j=1,ny)
            write(30,rec=i+2*nx) (sngl(qyn(i,j)),j=1,ny)
          end do
        end if

        if (itstep.eq.(ntime-1)) then
          open(unit=30,file='hot1.dat',access='direct',recl=4*ny)
          do i=1,nx
            write(30,rec=i) (sngl(zetan(i,j)),j=1,ny)
            write(30,rec=i+nx) (sngl(qxn(i,j)),j=1,ny)
            write(30,rec=i+2*nx) (sngl(qyn(i,j)),j=1,ny)
```

```
            end do
          end if

          if (itstep.eq.(ntime)) then
             open(unit=30,file='hot0.dat',access='direct',recl=4*ny)
             do i=1,nx
               write(30,rec=i) (sngl(zetan(i,j)),j=1,ny)
               write(30,rec=i+nx) (sngl(qxn(i,j)),j=1,ny)
               write(30,rec=i+2*nx) (sngl(qyn(i,j)),j=1,ny)
             end do
          end if

c ## write only at specified time-steps ----------------------

           if ((mod((itstep-1),interval).eq.0)) then

c ## write Zeta, qx, qy
             kounter=(itstep-1)/interval
             write(6,*) itstep,t,kounter+100
             write(unit=name1,fmt=244)100+kounter
             open(unit=30,file='fort.'//name1,access='direct',recl=4*ny)
             do i=1,nx
               write(30,rec=i) (sngl(zetan(i,j)),j=1,ny)
               write(30,rec=i+nx) (sngl(qxn(i,j)),j=1,ny)
               write(30,rec=i+2*nx) (sngl(qyn(i,j)),j=1,ny)
             end do
     close(30)
244          format(I3)

c ## write variables for velocity depth profiles --------------------
        if (yprofile(1).ne.0) then
            open (unit=21,file='fort.401',status='unknown')
        j=yprofile(1)
           write(21,820) j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j
     .       ,j,j,j,j,j,j,j,j,j,j,j,j,j
        do i=1,nx
         write(21,994) d1x(i,j),e1x(i,j),fs1x(i,j),f2x(i,j),
     .                 d1y(i,j),e1y(i,j),fs1y(i,j),f2y(i,j),
     .                 ht(i,j),zetanp1(i,j),qxnp1(i,j),qynp1(i,j)
     .                 ,uda4(i,j),uda3(i,j),uda2(i,j),
     .                 vda4(i,j),vda3(i,j),vda2(i,j),
     .                 udb4(i,j),udb3(i,j),udb2(i,j),
     .                 vdb4(i,j),vdb3(i,j),vdb2(i,j),
```

```
     .               udw4(i,j),udw3(i,j),udw2(i,j),
     .               vdw4(i,j),vdw3(i,j),vdw2(i,j),
     .               udc(i,j),vdc(i,j),qwx(i,j),qwy(i,j)
      end do
   close(21)
      end if
      if (yprofile(2).ne.0) then
          open (unit=22,file='fort.402',status='unknown')
j=yprofile(2)
        write(22,820) j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j
     .     ,j,j,j,j,j,j,j,j,j,j,j,j,j
      do i=1,nx
       write(22,994) d1x(i,j), e1x(i,j), fs1x(i,j),f2x(i,j),
     .               d1y(i,j), e1y(i,j), fs1y(i,j),f2y(i,j),
     .                 ht(i,j), zetanp1(i,j), qxnp1(i,j),
     .                  qynp1(i,j)
     .               ,uda4(i,j),uda3(i,j),uda2(i,j),
     .               vda4(i,j),vda3(i,j),vda2(i,j),
     .               udb4(i,j),udb3(i,j),udb2(i,j),
     .               vdb4(i,j),vdb3(i,j),vdb2(i,j),
     .               udw4(i,j),udw3(i,j),udw2(i,j),
     .               vdw4(i,j),vdw3(i,j),vdw2(i,j),
     .               udc(i,j),vdc(i,j),qwx(i,j),qwy(i,j)
      end do
   close(22)
      end if
      if (yprofile(3).ne.0) then
          open (unit=23,file='fort.403',status='unknown')
j=yprofile(3)
        write(23,820) j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j
     .     ,j,j,j,j,j,j,j,j,j,j,j,j,j
      do i=1,nx
       write(23,994) d1x(i,j), e1x(i,j), fs1x(i,j),f2x(i,j),
     .               d1y(i,j), e1y(i,j), fs1y(i,j),f2y(i,j),
     .                 ht(i,j), zetanp1(i,j), qxnp1(i,j),
     .                  qynp1(i,j)
     .               ,uda4(i,j),uda3(i,j),uda2(i,j),
     .               vda4(i,j),vda3(i,j),vda2(i,j),
     .               udb4(i,j),udb3(i,j),udb2(i,j),
     .               vdb4(i,j),vdb3(i,j),vdb2(i,j),
     .               udw4(i,j),udw3(i,j),udw2(i,j),
     .               vdw4(i,j),vdw3(i,j),vdw2(i,j),
     .               udc(i,j),vdc(i,j),qwx(i,j),qwy(i,j)
```

```
   end do
close(23)
   end if
   if (yprofile(4).ne.0) then
       open (unit=24,file='fort.404',status='unknown')
   j=yprofile(4)
      write(24,820) j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j
     .     ,j,j,j,j,j,j,j,j,j,j,j,j,j
   do i=1,nx
    write(24,994) d1x(i,j), e1x(i,j), fs1x(i,j),f2x(i,j),
     .               d1y(i,j), e1y(i,j), fs1y(i,j),f2y(i,j),
     .                 ht(i,j), zetanp1(i,j), qxnp1(i,j),
     .                  qynp1(i,j)
     .               ,uda4(i,j),uda3(i,j),uda2(i,j),
     .               vda4(i,j),vda3(i,j),vda2(i,j),
     .               udb4(i,j),udb3(i,j),udb2(i,j),
     .               vdb4(i,j),vdb3(i,j),vdb2(i,j),
     .               udw4(i,j),udw3(i,j),udw2(i,j),
     .               vdw4(i,j),vdw3(i,j),vdw2(i,j),
     .               udc(i,j),vdc(i,j),qwx(i,j),qwy(i,j)
   end do
close(24)
    end if
   if (yprofile(5).ne.0) then
      open (unit=25,file='fort.405',status='unknown')
   j=yprofile(5)
      write(25,820) j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j
     .     ,j,j,j,j,j,j,j,j,j,j,j,j,j
   do i=1,nx
    write(25,994) d1x(i,j), e1x(i,j), fs1x(i,j),f2x(i,j),
     .               d1y(i,j), e1y(i,j), fs1y(i,j),f2y(i,j),
     .                 ht(i,j), zetanp1(i,j), qxnp1(i,j),
     .                  qynp1(i,j)
     .               ,uda4(i,j),uda3(i,j),uda2(i,j),
     .               vda4(i,j),vda3(i,j),vda2(i,j),
     .               udb4(i,j),udb3(i,j),udb2(i,j),
     .               vdb4(i,j),vdb3(i,j),vdb2(i,j),
     .               udw4(i,j),udw3(i,j),udw2(i,j),
     .               vdw4(i,j),vdw3(i,j),vdw2(i,j),
     .               udc(i,j),vdc(i,j),qwx(i,j),qwy(i,j)
   end do
close(25)
   end if
```

```
   if (yprofile(6).ne.0) then
       open (unit=26,file='fort.406',status='unknown')
   j=yprofile(6)
      write(26,820) j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j
  .     ,j,j,j,j,j,j,j,j,j,j,j,j,j
   do i=1,nx
    write(26,994) d1x(i,j), e1x(i,j), fs1x(i,j),f2x(i,j),
  .               d1y(i,j), e1y(i,j), fs1y(i,j),f2y(i,j),
  .                 ht(i,j), zetanp1(i,j), qxnp1(i,j),
  .                  qynp1(i,j)
  .               ,uda4(i,j),uda3(i,j),uda2(i,j),
  .               vda4(i,j),vda3(i,j),vda2(i,j),
  .               udb4(i,j),udb3(i,j),udb2(i,j),
  .               vdb4(i,j),vdb3(i,j),vdb2(i,j),
  .               udw4(i,j),udw3(i,j),udw2(i,j),
  .               vdw4(i,j),vdw3(i,j),vdw2(i,j),
  .               udc(i,j),vdc(i,j),qwx(i,j),qwy(i,j)
   end do
close(26)
   end if
   if (yprofile(7).ne.0) then

       open (unit=27,file='fort.407',status='unknown')
   j=yprofile(7)
      write(27,820) j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j
  .     ,j,j,j,j,j,j,j,j,j,j,j,j,j
   do i=1,nx
    write(27,994) d1x(i,j), e1x(i,j), fs1x(i,j),f2x(i,j),
  .               d1y(i,j), e1y(i,j), fs1y(i,j),f2y(i,j),
  .                 ht(i,j), zetanp1(i,j), qxnp1(i,j),
  .                  qynp1(i,j)
  .               ,uda4(i,j),uda3(i,j),uda2(i,j),
  .               vda4(i,j),vda3(i,j),vda2(i,j),
  .               udb4(i,j),udb3(i,j),udb2(i,j),
  .               vdb4(i,j),vdb3(i,j),vdb2(i,j),
  .               udw4(i,j),udw3(i,j),udw2(i,j),
  .               vdw4(i,j),vdw3(i,j),vdw2(i,j),
  .               udc(i,j),vdc(i,j),qwx(i,j),qwy(i,j)
   end do
close(27)
   end if
   if (yprofile(8).ne.0) then
       open (unit=28,file='fort.408',status='unknown')
```

```
      j=yprofile(8)
         write(28,820) j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j,j
     .     ,j,j,j,j,j,j,j,j,j,j,j,j,j,j
      do i=1,nx
       write(28,994) d1x(i,j), e1x(i,j), fs1x(i,j),f2x(i,j),
     .               d1y(i,j), e1y(i,j), fs1y(i,j),f2y(i,j),
     .                 ht(i,j), zetanp1(i,j), qxnp1(i,j),
     .                  qynp1(i,j)
     .               ,uda4(i,j),uda3(i,j),uda2(i,j),
     .               vda4(i,j),vda3(i,j),vda2(i,j),
     .               udb4(i,j),udb3(i,j),udb2(i,j),
     .               vdb4(i,j),vdb3(i,j),vdb2(i,j),
     .               udw4(i,j),udw3(i,j),udw2(i,j),
     .               vdw4(i,j),vdw3(i,j),vdw2(i,j),
     .               udc(i,j),vdc(i,j),qwx(i,j),qwy(i,j)
      end do
   close(28)
      end if
c ## write last wet point file
          open (unit=21,file='fort.701',status='unknown')
          write(21,*) (nxa(j),j=1,ny)
   close(21)


       if (momout.eq.1) then
c ## write forces in momentum eqs ----------------------------
          do iunit=21,42
             write(unit=name2(iunit),fmt=244)580+iunit
            open(iunit,file='fort.'//name2(iunit),access='direct',
     .       recl=4*ny)
          end do
          open(iunit,file='fort.331',access='direct',recl=4*ny)
          do i=1,nx
c ## x-mom eq. terms
     write(21,rec=i) (sngl(-force_xx(i,j,t)),j=1,ny)
     write(22,rec=i) (sngl(-force_xy(i,j,t)),j=1,ny)
     write(23,rec=i) (sngl(-frictx(i,j)),j=1,ny)
     write(24,rec=i) (sngl(dtauxydy(i,j)),j=1,ny)
     write(25,rec=i) (sngl(-g*htt(i,j)
     . *dzetadxstar(i,j)),j=1,ny)
     write(26,rec=i) (sngl(-qxqxdx(i,j)-qxqydy(i,j)),j=1,ny)
     write(33,rec=i) (sngl(dispx(i,j)*factor),j=1,ny)
     write(35,rec=i) (sngl(-(qxnp1(i,j)-qxn(i,j))/dt),j=1,ny)
```

```
      write(37,rec=i) (sngl(qxqxdx(i,j)),j=1,ny)
      write(38,rec=i) (sngl(qxqydy(i,j)),j=1,ny)
c ## y-mom eq. terms
      write(27,rec=i) (sngl(-force_yx(i,j,t)),j=1,ny)
      write(28,rec=i) (sngl(-force_yy(i,j,t)),j=1,ny)
      write(29,rec=i) (sngl(-fricty(i,j)),j=1,ny)
      write(30,rec=i) (sngl(dtauxydx(i,j)),j=1,ny)
      write(31,rec=i) (sngl(-g*htt(i,j)
     . *dzetadystar(i,j)),j=1,ny)
      write(32,rec=i) (sngl(-qyqydy(i,j)-qxqydx(i,j)),j=1,ny)
      write(34,rec=i) (sngl(dispy(i,j)*factor),j=1,ny)
      write(36,rec=i) (sngl(-(qynp1(i,j)-qyn(i,j))/dt),j=1,ny)
      write(39,rec=i) (sngl(qyqydy(i,j)),j=1,ny)
      write(40,rec=i) (sngl(qxqydx(i,j)),j=1,ny)
c ## other terms
      write(41,rec=i) (sngl(wind_x(t)),j=1,ny)
      write(42,rec=i) (sngl(wind_y(t)),j=1,ny)
c ## Smagorinsky Eddy Viscosity
              write(43,rec=i) (sngl(vs(i,j)),j=1,ny)
            end do
            do iunit=21,43
              close(iunit)
            end do
          end if

c<=========================qun Sep.9,1999==================>
c write quasi-3d ciefficients at steady state
      if (disp3d.eq.1.and.dispout.eq.1) then
      do iunit=36,40
       write(unit=name3(iunit),fmt=244)710+iunit
       open(iunit,file='fort.'//name3(iunit),access='direct',
     .   recl=4*ny)
      end do

      do i=1,nx
      write(36,rec=i) (sngl(Bxx(i,j)),j=1,ny)
      write(36,rec=i+nx) (sngl(Bxy(i,j)),j=1,ny)
      write(36,rec=i+2*nx) (sngl(Byy(i,j)),j=1,ny)
      write(37,rec=i) (sngl(Dxx(i,j)),j=1,ny)
      write(37,rec=i+nx) (sngl(Dxy(i,j)),j=1,ny)
      write(37,rec=i+2*nx) (sngl(Dyy(i,j)),j=1,ny)
      write(38,rec=i) (sngl(Mxx(i,j)),j=1,ny)
      write(38,rec=i+nx) (sngl(Mxy(i,j)),j=1,ny)
```

```
      write(38,rec=i+2*nx) (sngl(Myy(i,j)),j=1,ny)
      write(39,rec=i) (sngl(Axxx(i,j)),j=1,ny)
      write(39,rec=i+nx) (sngl(Axyx(i,j)),j=1,ny)
      write(39,rec=i+2*nx) (sngl(Ayyx(i,j)),j=1,ny)
      write(40,rec=i) (sngl(Axxy(i,j)),j=1,ny)
      write(40,rec=i+nx) (sngl(Axyy(i,j)),j=1,ny)
      write(40,rec=i+2*nx) (sngl(Ayyy(i,j)),j=1,ny)
      end do

      do iunit=36,40
      close(iunit)
      end do
      end if
c<=======================qun Sep.9,1999==================>


         end if

c--------------------------------------------------------------------
c ## Update values: roll time levels, calculate MASS and ENERGY
c--------------------------------------------------------------------

 mass = 0.d0
 energy = 0.d0
 qxtotal = 0.d0
 qytotal = 0.d0

c
c calculate mass based on the trapezoidal rule
c
mass=.25d0*(zetan(1,1)+zetan(1,ny)+zetan(nx,1)+zetan(nx,ny)
     .  +ht(1,1)+ht(1,ny)+ht(nx,1)+ht(nx,ny))
do j=2,ny-1
  mass=mass+(zetan(1,j)+zetan(nx,j)+ht(1,j)+ht(nx,j))*.5d0
  do i=2,nx-1
    mass =mass+zetan(i,j)+ht(i,j)
  end do
end do

do i=2,nx-1
  mass=mass+(zetan(i,1)+zetan(i,ny)+ht(i,1)+ht(i,ny))*.5d0
end do
mass=mass*dx*dy
```

```
2        do i = 1,nx
            do j = 1,ny

c             ## Final values, next time level

               zetan(i,j)   = zetanp1(i,j)
               qxn(i,j)     = qxnp1(i,j)
               qyn(i,j)     = qynp1(i,j)

c             ## Move RHS up one time level

               cnm2(i,j)  = cnm1(i,j)
               mxnm2(i,j) = mxnm1(i,j)
               mynm2(i,j) = mynm1(i,j)

               cnm1(i,j)  = cn(i,j)
               mxnm1(i,j) = mxn(i,j)
               mynm1(i,j) = myn(i,j)

       energy = 0.5d0*( zetan(i,j)*zetan(i,j)-ht(i,j)*ht(i,j)
     . +(qxn(i,j)*qxn(i,j)+qyn(i,j)*qyn(i,j))
     . /(g*(ht(i,j)+zetan(i,j))) ) + energy

            end do
    qxtotal = qxn(i,1) + qxn(i,ny) + qxtotal
    qytotal = qyn(i,1) + qyn(i,ny) + qytotal
         end do
c!!! write MASS and ENERGY into file fort.50
c!!! xfluxin is the total mass flux through the boundary i=1
 xfluxin = 0.d0
 yfluxin = 0.d0
 do j=1,ny
   xfluxin = qxn(1,j)+xfluxin
   yfluxin = qyn(1,j)+yfluxin
 end do

c        if ((mod((itstep-1),jump).eq.0))
c     .  write(50,911) mass/mass0, energy/energy0, xfluxin,
c     .   yfluxin, qxtotal, qytotal


         if ((ibc1.eq.2).or.(ibc1.eq.3)) then
```

```
            do j=1,ny
              betan(1,j)=betanp1(1,j)
              do i=2,3
                betan(i,j) = (ux(qxnp1(i,j),zetanp1(i,j),ht(i,j))
     .                      - 2.d0*cel(zetanp1(i,j),ht(i,j)))
              end do
              bnm2(j) = bnm1(j)
              bnm1(j) = bn(j)
            end do
          end if

          t = t+dt

        end do
c-----------------------------------------------------------------------
c ## END TIME LOOP !!!
c-----------------------------------------------------------------------

c ## write time-averages files ------------------------------------
        if (t.ge.5.d0/delay.and.tavgout.eq.1) then
        do iunit=11,35
         write(unit=name3(iunit),fmt=244)490+iunit
         open(iunit,file='fort.'//name3(iunit),access='direct',
     .     recl=4*ny)
        end do
        inv_average = 1.d0/average
        do i=1,nx
 write(11,rec=i) (sngl(adqxdx(i,j)*inv_average),j=1,ny)
 write(12,rec=i) (sngl(adqydy(i,j)*inv_average),j=1,ny)
c ------ x-mom eq. terms
 write(13,rec=i) (sngl(africtx(i,j)*inv_average),j=1,ny)
 write(14,rec=i) (sngl(adtaudy(i,j)*inv_average),j=1,ny)
 write(15,rec=i) (sngl(adzdx(i,j)*inv_average),j=1,ny)
 write(16,rec=i) (sngl(aqxqxdx(i,j)*inv_average),j=1,ny)
 write(17,rec=i) (sngl(aqxqydy(i,j)*inv_average),j=1,ny)
 write(18,rec=i) (sngl(adispx(i,j)*inv_average),j=1,ny)
c ------ y-mom eq. terms
 write(19,rec=i) (sngl(africty(i,j)*inv_average),j=1,ny)
 write(20,rec=i) (sngl(adtaudx(i,j)*inv_average),j=1,ny)
 write(21,rec=i) (sngl(adzdy(i,j)*inv_average),j=1,ny)
 write(22,rec=i) (sngl(aqyqydy(i,j)*inv_average),j=1,ny)
 write(23,rec=i) (sngl(aqxqydx(i,j)*inv_average),j=1,ny)
 write(24,rec=i) (sngl(adispy(i,j)*inv_average),j=1,ny)
```

```
      end do
c ------variables
      do j=1,ny
        do i=1,nx
          azeta(i,j) = azeta(i,j)*inv_average
          aqxn(i,j) = aqxn(i,j)*inv_average
          aqyn(i,j) = aqyn(i,j)*inv_average
          htot = ht(i,j) + azeta(i,j)
          qxqx(i,j) = aqxn(i,j)*aqxn(i,j)/htot
          qxqy(i,j) = aqxn(i,j)*aqyn(i,j)/htot
          qyqy(i,j) = aqyn(i,j)*aqyn(i,j)/htot
          Uo(i,j) = ux(aqxn(i,j),ht(i,j),azeta(i,j))
          Vo(i,j) = uy(aqyn(i,j),ht(i,j),azeta(i,j))
        end do
      end do
      ncode =1
      nxflag=1
      call deriv(qxqy,qxqx,qyqy,qxqydx,qxqydy,qxqxdx,qyqydy,
     .  ncode,nxflag)
      ncode =1
      nxflag=1
      call deriv(azeta,aqxn,aqyn,dzetadxn,dzetadyn,dqxdxn,dqydyn,
     .           ncode,nxflag)

      call bottom_friction(azeta,Uo,Vo,frictx,fricty,
     .    factor,itstep)

      do i=1,nx
 write(25,rec=i) (sngl(azeta(i,j)),j=1,ny)
 write(26,rec=i) (sngl(aqxn(i,j)),j=1,ny)
 write(27,rec=i) (sngl(aqyn(i,j)),j=1,ny)
 write(28,rec=i) (sngl(qxqydx(i,j)),j=1,ny)
 write(29,rec=i) (sngl(qxqydy(i,j)),j=1,ny)
 write(30,rec=i) (sngl(qxqxdx(i,j)),j=1,ny)
 write(31,rec=i) (sngl(qyqydy(i,j)),j=1,ny)
 write(32,rec=i) (sngl(g*(ht(i,j)+azeta(i,j))
     .            *dzetadxn(i,j) ),j=1,ny)
 write(33,rec=i) (sngl(g*(ht(i,j)+azeta(i,j))
     .            *dzetadyn(i,j) ),j=1,ny)
 write(34,rec=i) (sngl(frictx(i,j)),j=1,ny)
 write(35,rec=i) (sngl(fricty(i,j)),j=1,ny)
      end do
      do iunit=11,35
```

```
         close(iunit)
      end do

      end if

820   format(34I4)
902   format(6I8)
903   format(/A,3I5)
905   format(500(f10.4))
911   format( 1x, 6(f16.8,1x))
994   format( 1x, 34(f13.8,1x))
996   format( 2x, 10(f15.10,2x))




      return
      end

c ######################################################################
```

8.3 File *winc_sub1.f*

This file contains the subroutines for calculating the vertical profiles and the quasi-3D terms.

8.3.1 Subroutine wave_forcing

This subroutine calculates wave forcing terms used in calculating the vertical profiles.

```
c234567890123456789012345678901234567890123456789012345678901234567890123456789012
c LAST MODIFICATION:
c - May 6, 2002
c ######################################################################
      subroutine  wave_forcing

c   Calculate the steady wave-induced-forcing for the vertical
c   profiles: the terms designated as beta_{alpha} in FS97/4,
c   eqs.(3),(22),(23).
c   We consider: WAVE FORCING ONLY UP TO THE INITIAL SHORELINE !!!
c   This means: even with the moving shoreline, the forcing is
c   calculated only for the initally wet points; No corrections are done.
```

```
c
c   Calculate also the steady-streaming bottom shear stress: tauss
c   as in FS97/5, eq.(16)
c
c#######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, j, ncode, nxflag

      real*8 h2oh(nxm,nym), h2oh_x(nxm,nym), h2oh_y(nxm,nym),
     . theta_x(nxm,nym), theta_y(nxm,nym), aux1,
     . uwduwdx(nxm,nym), vwduwdy(nxm,nym), dw2dx(nxm,nym),
     . vwdvwdy(nxm,nym), uwdvwdx(nxm,nym), dw2dy(nxm,nym)

c=====================================================================
      write(*,903)' Steady wave-induced-forcing for the vertical prof.'

      if (files.eq.1) then
           open(unit=1, file='fx.dat')
           open(unit=2, file='fy.dat')
           open(unit=3, file='tsx.dat')
           open(unit=4, file='tsy.dat')
        do i=1,nx
read(1,*) (shwave_force_x(i,j),j=1,ny)
read(2,*) (shwave_force_y(i,j),j=1,ny)
read(3,*) (tauss_x(i,j),j=1,ny)
read(4,*) (tauss_y(i,j),j=1,ny)
      end do

       close(1)
       close(2)
       close(3)
       close(4)

       else

      do j=1,ny
        do i=1,nxa(j)-1
          h2oh(i,j) = H(i,j)*H(i,j) /ht(i,j)
        end do
        do i=nxa(j), nx
```

```
          h2oh(i,j) = 0.d0
        end do
      end do

      ncode =3
      nxflag=1
      call deriv(h2oh, theta, theta, h2oh_x, h2oh_y, theta_x, theta_y,
     .           ncode,nxflag)

       aux1 = Bo*4.d0*(pi*pi)/(period*period)

      do j=1,ny
        do i=1,nxa(j)-1

c ## calculate short-wave forcing, eqn. (18)-(23) note FS97/4:
          uwduwdx(i,j)= Bo*g*(0.5d0*costh(i,j)*costh(i,j)*h2oh_x(i,j)
     .         -h2oh(i,j)*costh(i,j)*sinth(i,j)*theta_x(i,j))
          vwduwdy(i,j)= Bo*g*(-h2oh(i,j)*sinth(i,j)*sinth(i,j)
     .         *theta_y(i,j) + 0.5d0*costh(i,j)*sinth(i,j)*h2oh_y(i,j))
          dw2dx(i,j)=aux1*(
     .       ((h2oh(i,j)*dhodxn(i,j)+ht(i,j)*h2oh_x(i,j))
     .       *sinth(i,j)**2+H(i,j)*H(i,j)*2*costh(i,j)*theta_x(i,j)) )


          shwave_force_x(i,j) = uwduwdx(i,j) -0.5d0*dw2dx(i,j)
     .         + vwduwdy(i,j)

          vwdvwdy(i,j)= Bo*g*(0.5d0*sinth(i,j)*sinth(i,j)*h2oh_y(i,j)
     .         +h2oh(i,j)*costh(i,j)*sinth(i,j)*theta_y(i,j))
          uwdvwdx(i,j)= Bo*g*(+h2oh(i,j)*costh(i,j)*costh(i,j)
     .         *theta_x(i,j) + 0.5d0*costh(i,j)*sinth(i,j)*h2oh_x(i,j))
          dw2dy(i,j)=aux1*(
     .         ((h2oh(i,j)*dhodyn(i,j)+ht(i,j)*h2oh_y(i,j))
     .         *sinth(i,j)**2+H(i,j)*H(i,j)*2*costh(i,j)*theta_y(i,j)) )

          shwave_force_y(i,j) = vwdvwdy(i,j) -0.5d0*dw2dy(i,j)
     .         + uwdvwdx(i,j)

c-- include steady-streaming in the U,V profiles
          tauss_x(i,j)= dsqrt(0.08d0/8.d0)*fcwij(i,j)*H(i,j)/ht(i,j)
     .                  *(u0(i,j)*costh(i,j))**2.d0
          tauss_y(i,j)= dsqrt(0.08d0/8.d0)*fcwij(i,j)*H(i,j)/ht(i,j)
     .                  *(u0(i,j)*sinth(i,j))**2.d0
```

```
          end do
          do i=nxa(j), nx
            uwduwdx(i,j)= 0.d0
            vwduwdy(i,j)= 0.d0
            vwdvwdy(i,j)= 0.d0
            uwdvwdx(i,j)= 0.d0
            shwave_force_x(i,j) = 0.d0
            shwave_force_y(i,j) = 0.d0
            tauss_x(i,j) = 0.d0
            tauss_y(i,j) = 0.d0
          end do
        end do

        end if

c--- averaging of values ------------------------------
c       call average_3pty(nx,ny,vwduwdy,vwduwdy)
c       call average_3pty(nx,ny,vwdvwdy,vwdvwdy)
c       do i=1,nx
c          vwduwdy(i,1)=0.25d0*vwduwdy(i,2)+0.5d0*vwduwdy(i,1)
c     .                 +0.25d0*vwduwdy(i,ny-1)
c          vwduwdy(i,ny)=vwduwdy(i,1)
c          vwdvwdy(i,1)=0.25d0*vwdvwdy(i,2)+0.5d0*vwdvwdy(i,1)
c     .                 +0.25d0*vwdvwdy(i,ny-1)
c          vwdvwdy(i,ny)=vwdvwdy(i,1)
c       end do



903     format(/A)
        return
        end
```

8.3.2 Subroutine UV_profile

This subroutine calculates the vertical profiles.

```
c ######################################################################
        subroutine  UV_profile(zeta,frict_x,frict_y,t,windx,windy,vs)
c
c       calculates the velocity profiles for the general 3D disperion
```

```
c     as outlined in Kevin Note 12b eqs 46-67  (V1a^0)
c
c#######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'


      real*8 force_xx, force_xy, force_yx, force_yy, wind_x, wind_y

c--- Local variables ------------------------------------------------
      integer i, j

      real*8 t, htot(nxm,nym), zeta(nxm,nym),
     . frict_x(nxm,nym), frict_y(nxm,nym),
     . f_x, f_y, inv_ht(nxm,nym), inv_vt(nxm,nym), aux1,
     . windx, windy,qwx_ht(nxm,nym),qwy_ht(nxm,nym),
     . e1xh(nxm,nym),vs(nxm,nym),
     . e1yh(nxm,nym)

c=======================================================================
      force_xx(i,j,t)= DTANH(t*delay)*dSxxdxn(i,j)
      force_xy(i,j,t)= DTANH(t*delay)*dSxydyn(i,j)
      force_yx(i,j,t)= DTANH(t*delay)*dSxydxn(i,j)
      force_yy(i,j,t)= DTANH(t*delay)*dSyydyn(i,j)
      wind_x(t) = DTANH(t*delay)*windx
      wind_y(t) = DTANH(t*delay)*windy

      aux1 = 1.d0/6.d0
      do j=1,ny
        do i=1,nxa(j)-2
  htot(i,j) = ht(i,j)+zeta(i,j)
          inv_ht(i,j) = 1.d0/htot(i,j)
          qwx_ht(i,j)  = qwx(i,j)*inv_ht(i,j)
          qwy_ht(i,j)  = qwy(i,j)*inv_ht(i,j)
          inv_vt(i,j) = 1.d0/(v_t(i,j)+vs(i,j))

c ## assemble f_x as in eq.(46), note Kevin 12.b:
          f_x = (
     .    force_xx(i,j,t)
     .    +force_xy(i,j,t)
     .     +frict_x(i,j)-tauss_x(i,j)+wind_x(t)) *inv_ht(i,j)
     .     - shwave_force_x(i,j)
```

```
c ## calculate U1 coefficients: U1=d1x*z**2+e1x*z+f1x+f2x;

          e1x(i,j)=(frict_x(i,j)-tauss_x(i,j))*inv_vt(i,j)
          e1xh(i,j)=e1x(i,j)*htot(i,j)
          d1x(i,j)=-.5d0*inv_vt(i,j)*(f_x)
          f1x(i,j)= -qwx(i,j)*inv_ht(i,j)-htot(i,j)*.5d0*e1x(i,j)
          fs1x(i,j)= -htot(i,j)*.5d0*e1x(i,j)
          f2x(i,j)=inv_vt(i,j)*(aux1*f_x*htot(i,j)*htot(i,j))

c ## assemble f_y as in eq.(47), note Kevin 12.3:
          f_y = +(force_yx(i,j,t)+force_yy(i,j,t)
     .     +frict_y(i,j)-tauss_y(i,j)+wind_y(t)) *inv_ht(i,j)
     .     - shwave_force_y(i,j)


c ## calculate V1 coefficients: V1=d1y*z**2+e1y*z+f1y+f2y;
          e1y(i,j)=(frict_y(i,j)-tauss_y(i,j))*inv_vt(i,j)
          e1yh(i,j)=e1y(i,j)*htot(i,j)
          d1y(i,j)=-.5d0*inv_vt(i,j)*(f_y)
          f1y(i,j)= -qwy(i,j)*inv_ht(i,j)-ht(i,j)*.5d0*e1y(i,j)
          fs1y(i,j)= -ht(i,j)*.5d0*e1y(i,j)
          f2y(i,j)=inv_vt(i,j)*(aux1*f_y*htot(i,j)*htot(i,j))
        end do
        do i=nxa(j)-1, nx
          e1x(i,j) = 0.d0
          d1x(i,j) = 0.d0
          f1x(i,j) = 0.d0
          fs1x(i,j) = 0.d0
          f2x(i,j) = 0.d0
          e1y(i,j) = 0.d0
          d1y(i,j) = 0.d0
          f1y(i,j) = 0.d0
          fs1y(i,j) = 0.d0
          f2y(i,j) = 0.d0
        end do
      end do

      return
      end

c ######################################################################
```

8.3.3 Subroutine coefcalc

This subroutine calculates the quasi-3D coefficients.

```
c ######################################################################
        subroutine coefcalc(Dxx,Dyy,Dxy,zeta,Mxx,Myy,Mxy,
     .     Bxx,Bxy,Byy,Axxx,Axxy,Axyx,Axyy,Ayyx,Ayyy,vs,Uo,Vo)
c
c     calculates the coefficients used in the generalized 3D dispersion
c     Currently utilizes assumptions from Ap's version only
c
c
c     Kevin Haas 5/99
c ######################################################################

      include 'winc_std_common.inc'

      integer i, j

      real*8 htot(nxm,nym),ht2(nxm,nym),ht3(nxm,nym),ht4(nxm,nym),
     . inv_vt(nxm,nym),inv_ht(nxm,nym),Dxx(nxm,nym),Dyy(nxm,nym),
     . Bxx(nxm,nym),Byy(nxm,nym),Bxy(nxm,nym),Uo(nxm,nym),Vo(nxm,nym),
     . Dxy(nxm,nym),zeta(nxm,nym),Mxx(nxm,nym),Myy(nxm,nym),Mxy(nxm,nym)
     . ,Mxxap(nxm,nym),Mxyap(nxm,nym),Myyap(nxm,nym),aux1,
     . Lxx1(nxm,nym),Lxx2(nxm,nym),Lxx3(nxm,nym),
     . Lxy1(nxm,nym),Lxy2(nxm,nym),Lxy3(nxm,nym),
     . Lyy1(nxm,nym),Lyy2(nxm,nym),Lyy3(nxm,nym),
     . Lyx1(nxm,nym),Lyx2(nxm,nym),Lyx3(nxm,nym),
     . de1xdx(nxm,nym),de1xdy(nxm,nym),de1ydx(nxm,nym),de1ydy(nxm,nym),
     . dd1xdx(nxm,nym),dd1xdy(nxm,nym),dd1ydx(nxm,nym),dd1ydy(nxm,nym),
     . dfsxdx(nxm,nym),dfsxdy(nxm,nym),dfsydx(nxm,nym),dfsydy(nxm,nym),
     . fsumx(nxm,nym),fsumy(nxm,nym),vs(nxm,nym),
     . Axxx(nxm,nym),Axxy(nxm,nym),Axyx(nxm,nym),Axyy(nxm,nym),
     . Ayyx(nxm,nym),Ayyy(nxm,nym),dUo_dx(nxm,nym),dVo_dy(nxm,nym),
     . dUo_dy(nxm,nym),dVo_dx(nxm,nym),wstar

      aux1=1.d0/6.d0


      do j=1,ny
        do i=1,nxa(j)
  htot(i,j) = ht(i,j)+zeta(i,j)
          ht2(i,j)  = htot(i,j)*htot(i,j)
```

```
      ht3(i,j)  = ht2(i,j)*htot(i,j)
      ht4(i,j)  = ht3(i,j)*htot(i,j)
      inv_ht(i,j) = 1.d0/htot(i,j)
      inv_vt(i,j) = 1.d0/(v_t(i,j)+vs(i,j))

       fsumx(i,j)=f1x(i,j)+f2x(i,j)
       fsumy(i,j)=f1y(i,j)+f2y(i,j)


 Dxy(i,j) = ht4(i,j)/63.d0*d1x(i,j)*d1y(i,j)+ht3(i,j)/36.d0*
.          (d1x(i,j)*e1y(i,j)+d1y(i,j)*e1x(i,j))
.          +ht2(i,j)/15.d0*(d1x(i,j)*(f1y(i,j)+f2y(i,j))+
.          d1y(i,j)*(f1x(i,j)+f2x(i,j))+.75*e1x(i,j)*e1y(i,j))
.          +htot(i,j)*0.125d0*(e1x(i,j)*(f1y(i,j)+f2y(i,j))+
.          e1y(i,j)*(f1x(i,j)+f2x(i,j)))
.          +1.d0/3.d0*(f1x(i,j)+f2x(i,j))*(f1y(i,j)+f2y(i,j))

 Dxy(i,j) = Dxy(i,j)*ht2(i,j)*inv_vt(i,j)

 Dyy(i,j) = ht4(i,j)/63.d0*d1y(i,j)*d1y(i,j)+ht3(i,j)/36.d0*
.          (d1y(i,j)*e1y(i,j)+d1y(i,j)*e1y(i,j))
.          +ht2(i,j)/15.d0*(d1y(i,j)*(f1y(i,j)+f2y(i,j))+
.          d1y(i,j)*(f1y(i,j)+f2y(i,j))+.75*e1y(i,j)*e1y(i,j))
.          +htot(i,j)*0.125d0*(e1y(i,j)*(f1y(i,j)+f2y(i,j))+
.          e1y(i,j)*(f1y(i,j)+f2y(i,j)))
.          +1.d0/3.d0*(f1y(i,j)+f2y(i,j))*(f1y(i,j)+f2y(i,j))

 Dyy(i,j) = Dyy(i,j)*ht2(i,j)*inv_vt(i,j)

 Dxx(i,j) = ht4(i,j)/63.d0*d1x(i,j)*d1x(i,j)+ht3(i,j)/36.d0*
.          (d1x(i,j)*e1x(i,j)+d1x(i,j)*e1x(i,j))
.          +ht2(i,j)/15.d0*(d1x(i,j)*(f1x(i,j)+f2x(i,j))+
.          d1x(i,j)*(f1x(i,j)+f2x(i,j))+.75*e1x(i,j)*e1x(i,j))
.          +htot(i,j)*0.125d0*(e1x(i,j)*(f1x(i,j)+f2x(i,j))+
.          e1x(i,j)*(f1x(i,j)+f2x(i,j)))
.          +1.d0/3.d0*(f1x(i,j)+f2x(i,j))*(f1x(i,j)+f2x(i,j))

 Dxx(i,j) = Dxx(i,j)*ht2(i,j)*inv_vt(i,j)

 Bxx(i,j)=(4.d0/63.d0*d1x(i,j)*d1x(i,j)*ht3(i,j)
.      +aux1*e1x(i,j)*d1x(i,j)*ht2(i,j)
.      +(4.d0/15.d0*d1x(i,j)*(f1x(i,j)+f2x(i,j))+.1d0
.      *e1x(i,j)*e1x(i,j))*htot(i,j)
```

```
     .     +e1x(i,j)*(f1x(i,j)+f2x(i,j))*.25d0)

      Bxx(i,j) = -Bxx(i,j)*ht3(i,j)*inv_vt(i,j)


      Bxy(i,j)=(4.d0/63.d0*d1x(i,j)*d1y(i,j)*ht3(i,j)
     .     +aux1*.5d0*(e1x(i,j)*d1y(i,j)+e1y(i,j)*d1x(i,j))*ht2(i,j)
     .     +(2.d0/15.d0*(d1x(i,j)*(f1y(i,j)+f2y(i,j))
     .     +d1y(i,j)*(f1x(i,j)+f2x(i,j)))
     .     +.1d0*e1x(i,j)*e1y(i,j))*htot(i,j)
     .     +(e1x(i,j)*(f1y(i,j)+f2y(i,j))
     .     +e1y(i,j)*(f1x(i,j)+f2x(i,j)))*.125d0)

      Bxy(i,j) = -Bxy(i,j)*ht3(i,j)*inv_vt(i,j)


      Byy(i,j)=(4.d0/63.d0*d1y(i,j)*d1y(i,j)*ht3(i,j)
     .     +aux1*e1y(i,j)*d1y(i,j)*ht2(i,j)
     .     +(4.d0/15.d0*d1y(i,j)*(f1y(i,j)+f2y(i,j))+.1d0
     .     *e1y(i,j)*e1y(i,j))*htot(i,j)
     .     +e1y(i,j)*(f1y(i,j)+f2y(i,j))*.25d0)

      Byy(i,j) = -Byy(i,j)*ht3(i,j)*inv_vt(i,j)

ckevin 8/9/2000  Aproximates V1(h)Qw with -QwQw in M

      Mxxap(i,j)=d1x(i,j)*d1x(i,j)*ht4(i,j)*htot(i,j)*.2d0
     .     +(d1x(i,j)*e1x(i,j)*.5d0*ht4(i,j))
     .     +(d1x(i,j)*(fs1x(i,j)+f2x(i,j))*2.d0+e1x(i,j)*e1x(i,j))/3.d0
     .         *ht3(i,j)
     .     +(e1x(i,j)*(fs1x(i,j)+f2x(i,j))
c     .      +2.d0*qwx(i,j)*d1x(i,j)
     .         )*ht2(i,j)
     .     +2.d0*((fs1x(i,j)+f2x(i,j))*(fs1x(i,j)+f2x(i,j))*.5d0
c     .          +qwx(i,j)*e1x(i,j)
     .     )*htot(i,j)
     .     +(qwx(i,j)*inv_ht(i,j))*qwx(i,j)*2.d0
c     .      +(fs1x(i,j)+f2x(i,j))*qwx(i,j)*2.d0

c      Mxxap(i,j)=d1x(i,j)*d1x(i,j)*ht4(i,j)*htot(i,j)*.2d0
c     .      +(d1x(i,j)*e1x(i,j)*.5d0*ht4(i,j))
c     .      +(d1x(i,j)*(f1x(i,j)+f2x(i,j))*2.d0+e1x(i,j)*e1x(i,j))/3.d0
c     .          *ht3(i,j)
```

```
c     .     +(e1x(i,j)*(f1x(i,j)+f2x(i,j)))
c     .         *ht2(i,j)
c     .     +2.d0*((f1x(i,j)+f2x(i,j))*(f1x(i,j)+f2x(i,j))*.5d0
c     .         )*htot(i,j)
c     .     +(-qwx(i,j)*inv_ht(i,j))*qwx(i,j)*2.d0

      Myyap(i,j)=d1y(i,j)*d1y(i,j)*ht4(i,j)*htot(i,j)*.2d0
     .     +(d1y(i,j)*e1y(i,j)*.5d0*ht4(i,j))
     .     +(d1y(i,j)*(fs1y(i,j)+f2y(i,j))*2.d0+e1y(i,j)*e1y(i,j))/3.d0
     .         *ht3(i,j)
     .     +(e1y(i,j)*(fs1y(i,j)+f2y(i,j))
c     .      +2.d0*qwy(i,j)*d1y(i,j)
     .         )*ht2(i,j)
     .     +2.d0*((fs1y(i,j)+f2y(i,j))*(fs1y(i,j)+f2y(i,j))*.5d0
c     .         +qwy(i,j)*e1y(i,j)
     .     )*htot(i,j)
c     .     +(fs1y(i,j)+f2y(i,j))*qwy(i,j)*2.d0
     .     +(qwy(i,j)*inv_ht(i,j))*qwy(i,j)*2.d0

c          Myyap(i,j)=d1y(i,j)*d1y(i,j)*ht4(i,j)*htot(i,j)*.2d0
c     .     +(d1y(i,j)*e1y(i,j)*.5d0*ht4(i,j))
c     .     +(d1y(i,j)*(f1y(i,j)+f2y(i,j))*2.d0+e1y(i,j)*e1y(i,j))/3.d0
c     .         *ht3(i,j)
c     .     +(e1y(i,j)*(f1y(i,j)+f2y(i,j)))
c     .         *ht2(i,j)
c     .     +2.d0*((f1y(i,j)+f2y(i,j))*(f1y(i,j)+f2y(i,j))*.5d0
c     .         )*htot(i,j)
c     .     +(-qwy(i,j)*inv_ht(i,j))*qwy(i,j)*2.d0

      Mxyap(i,j)=d1x(i,j)*d1y(i,j)*ht4(i,j)*htot(i,j)*.2d0
     .     +(d1x(i,j)*e1y(i,j)+d1y(i,j)*e1x(i,j))*.25d0*ht4(i,j)
     .     +(d1x(i,j)*(fs1y(i,j)+f2y(i,j))+d1y(i,j)*(fs1x(i,j)
     .        +f2x(i,j))+e1x(i,j)*e1y(i,j))*aux1*2.d0*ht3(i,j)
     .     +(e1x(i,j)*(fs1y(i,j)+f2y(i,j))+e1y(i,j)*(fs1x(i,j)+
     .        f2x(i,j))
c     .     +2.d0*qwx(i,j)*d1y(i,j)+2.d0*qwy(i,j)*d1x(i,j)
     .     )*0.5d0*ht2(i,j)
     .     +((fs1x(i,j)+f2x(i,j))*(fs1y(i,j)+f2y(i,j))
c     .         +qwx(i,j)*e1y(i,j)+qwy(i,j)*e1x(i,j)
     .     )*htot(i,j)
c     .     +(fs1y(i,j)+f2y(i,j))*qwx(i,j)+(fs1x(i,j)+f2x(i,j))*
c     .         qwy(i,j)
     .     +(qwx(i,j)*inv_ht(i,j))*qwy(i,j)
```

```
     .     +(qwy(i,j)*inv_ht(i,j))*qwx(i,j)

c      Mxyap(i,j)=d1x(i,j)*d1y(i,j)*ht4(i,j)*htot(i,j)*.2d0
c     .     +(d1x(i,j)*e1y(i,j)+d1y(i,j)*e1x(i,j))*.25d0*ht4(i,j)
c     .     +(d1x(i,j)*(f1y(i,j)+f2y(i,j))+d1y(i,j)*(f1x(i,j)+
c     .         +f2x(i,j))+e1x(i,j)*e1y(i,j))*aux1*2.d0*ht3(i,j)
c     .     +(e1x(i,j)*(f1y(i,j)+f2y(i,j))+e1y(i,j)*(f1x(i,j)+
c     .         f2x(i,j))
c     .         )*0.5d0*ht2(i,j)
c     .     +((f1x(i,j)+f2x(i,j))*(f1y(i,j)+f2y(i,j))
c     .         )*htot(i,j)
c     .     +(-qwy(i,j)*inv_ht(i,j))*qwx(i,j)+(-qwx(i,j)*inv_ht(i,j))*
c     .         qwy(i,j)

        end do
      end do

      call deriv(d1x, d1y, e1x, dd1xdx, dd1xdy, dd1ydx,
     .  de1xdy, 3, 1)
      call deriv(e1y, e1x, d1y, de1ydx, de1ydy, de1xdx,
     .  dd1ydy, 3, 1)
      call deriv(fsumx, fsumy, fsumy, dfsxdx, dfsxdy, dfsydx,
     .  dfsydy, 3, 1)
      call deriv(Uo, Vo, Vo, dUo_dx, dUo_dy, dVo_dx, dVo_dy,3,1)

      do j=1,ny
      do i=1,nxa(j)

      Lxx3(i,j)=dd1xdx(i,j)
      Lxy3(i,j)=dd1xdy(i,j)
      Lyx3(i,j)=dd1ydx(i,j)
      Lyy3(i,j)=dd1ydy(i,j)
      Lxx2(i,j)=de1xdx(i,j)
      Lxy2(i,j)=de1xdy(i,j)
      Lyx2(i,j)=de1ydx(i,j)
      Lyy2(i,j)=de1ydy(i,j)
      Lxx1(i,j)=dfsxdx(i,j)
      Lxy1(i,j)=dfsxdy(i,j)
      Lyx1(i,j)=dfsydx(i,j)
      Lyy1(i,j)=dfsydy(i,j)

      Mxx(i,j)=Mxxap(i,j)
      Mxy(i,j)=Mxyap(i,j)
```

```
      Myy(i,j)=Myyap(i,j)

      Axxx(i,j)=(d1x(i,j)*Lxx3(i,j)*2.d0/63.d0*ht4(i,j)
     .     +(d1x(i,j)*Lxx2(i,j)+e1x(i,j)*Lxx3(i,j))/18.d0*ht3(i,j)
     .     +((d1x(i,j)*Lxx1(i,j)+fsumx(i,j)*Lxx3(i,j))*2.d0/15.d0
     .          +e1x(i,j)*Lxx2(i,j)*.1d0)*ht2(i,j)
     .     +(e1x(i,j)*Lxx1(i,j)+fsumx(i,j)*Lxx2(i,j))*htot(i,j)*.25d0
     .     +(fsumx(i,j)*Lxx1(i,j))*aux1*4.d0)
      Axxx(i,j)=-Axxx(i,j)*inv_vt(i,j)*ht3(i,j)

      Axxy(i,j)=(d1x(i,j)*Lxy3(i,j)*2.d0/63.d0*ht4(i,j)
     .     +(d1x(i,j)*Lxy2(i,j)+e1x(i,j)*Lxy3(i,j))/18.d0*ht3(i,j)
     .     +((d1x(i,j)*Lxy1(i,j)+fsumx(i,j)*Lxy3(i,j))*2.d0/15.d0
     .          +e1x(i,j)*Lxy2(i,j)*.1d0)*ht2(i,j)
     .     +(e1x(i,j)*Lxy1(i,j)+fsumx(i,j)*Lxy2(i,j))*htot(i,j)*.25d0
     .     +(fsumx(i,j)*Lxy1(i,j))*aux1*4.d0)
      Axxy(i,j)=-Axxy(i,j)*inv_vt(i,j)*ht3(i,j)

      Axyx(i,j)=((d1y(i,j)*Lxx3(i,j)+d1x(i,j)*Lyx3(i,j))/63.d0*ht4(i,j)
     .     +(d1y(i,j)*Lxx2(i,j)+d1x(i,j)*Lyx2(i,j)+e1y(i,j)*Lxx3(i,j)
     .         +e1x(i,j)*Lyx3(i,j))/36.d0*ht3(i,j)
     .     +((d1y(i,j)*Lxx1(i,j)+d1x(i,j)*Lyx1(i,j)
     .         +fsumy(i,j)*Lxx3(i,j)+fsumx(i,j)*Lyx3(i,j))/15.d0
     .         +(e1y(i,j)*Lxx2(i,j)+e1x(i,j)*Lyx2(i,j))*.05d0)*ht2(i,j)
     .     +(e1y(i,j)*Lxx1(i,j)+e1y(i,j)*Lyx1(i,j)+fsumy(i,j)*Lxx2(i,j)
     .         +fsumx(i,j)*Lyx2(i,j))*htot(i,j)*.125d0
     .     +(fsumy(i,j)*Lxx1(i,j)+fsumx(i,j)*Lyx1(i,j))*aux1*2.d0)
      Axyx(i,j)=-Axyx(i,j)*inv_vt(i,j)*ht3(i,j)

      Axyy(i,j)=((d1y(i,j)*Lxy3(i,j)+d1x(i,j)*Lyy3(i,j))/63.d0*ht4(i,j)
     .     +(d1y(i,j)*Lxy2(i,j)+d1x(i,j)*Lyy2(i,j)+e1y(i,j)*Lxy3(i,j)
     .         +e1x(i,j)*Lyy3(i,j))/36.d0*ht3(i,j)
     .     +((d1y(i,j)*Lxy1(i,j)+d1x(i,j)*Lyy1(i,j)
     .         +fsumy(i,j)*Lxy3(i,j)+fsumx(i,j)*Lyy3(i,j))/15.d0
     .         +(e1y(i,j)*Lxy2(i,j)+e1x(i,j)*Lyy2(i,j))*.05d0)*ht2(i,j)
     .     +(e1y(i,j)*Lxy1(i,j)+e1y(i,j)*Lyy1(i,j)+fsumy(i,j)*Lxy2(i,j)
     .         +fsumx(i,j)*Lyy2(i,j))*htot(i,j)*.125d0
     .     +(fsumy(i,j)*Lxy1(i,j)+fsumx(i,j)*Lyy1(i,j))*aux1*2.d0)
      Axyy(i,j)=-Axyy(i,j)*inv_vt(i,j)*ht3(i,j)


      Ayyx(i,j)=(d1y(i,j)*Lyx3(i,j)*2.d0/63.d0*ht4(i,j)
     .     +(d1y(i,j)*Lyx2(i,j)+e1y(i,j)*Lyx3(i,j))/18.d0*ht3(i,j)
```

```
     .     +((d1y(i,j)*Lyx1(i,j)+fsumy(i,j)*Lyx3(i,j))*2.d0/15.d0
     .          +e1y(i,j)*Lyx2(i,j)*.1d0)*ht2(i,j)
     .     +(e1y(i,j)*Lyx1(i,j)+fsumy(i,j)*Lyx2(i,j))*htot(i,j)*.25d0
     .     +(fsumy(i,j)*Lyx1(i,j))*aux1*4.d0)
           Ayyx(i,j)=-Ayyx(i,j)*inv_vt(i,j)*ht3(i,j)

           Ayyy(i,j)=(d1y(i,j)*Lyy3(i,j)*2.d0/63.d0*ht4(i,j)
     .     +(d1y(i,j)*Lyy2(i,j)+e1y(i,j)*Lyy3(i,j))/18.d0*ht3(i,j)
     .     +((d1y(i,j)*Lyy1(i,j)+fsumy(i,j)*Lyy3(i,j))*2.d0/15.d0
     .          +e1y(i,j)*Lyy2(i,j)*.1d0)*ht2(i,j)
     .     +(e1y(i,j)*Lyy1(i,j)+fsumy(i,j)*Lyy2(i,j))*htot(i,j)*.25d0
     .     +(fsumy(i,j)*Lyy1(i,j))*aux1*4.d0)
           Ayyy(i,j)=-Ayyy(i,j)*inv_vt(i,j)*ht3(i,j)


c          calculate Vd coefficients

             uda4(i,j)=(Uo(i,j)*dd1xdx(i,j)+Vo(i,j)*dd1xdy(i,j))*aux1*0.5d0
             uda3(i,j)=(Uo(i,j)*(2.d0*d1x(i,j)*dhodxn(i,j)+de1xdx(i,j))
     .          +Vo(i,j)*(2.d0*d1x(i,j)*dhodyn(i,j)+de1xdy(i,j)))*aux1
             uda2(i,j)=(Uo(i,j)*(e1x(i,j)*dhodxn(i,j)+dfsxdx(i,j))
     .          +Vo(i,j)*(e1x(i,j)*dhodyn(i,j)+dfsxdy(i,j)))*.5d0

             vda4(i,j)=(Uo(i,j)*dd1ydx(i,j)+Vo(i,j)*dd1ydy(i,j))*aux1*0.5d0
             vda3(i,j)=(Uo(i,j)*(2.d0*d1y(i,j)*dhodxn(i,j)+de1ydx(i,j))
     .          +Vo(i,j)*(2.d0*d1y(i,j)*dhodyn(i,j)+de1ydy(i,j)))*aux1
             vda2(i,j)=(Uo(i,j)*(e1y(i,j)*dhodxn(i,j)+dfsydx(i,j))
     .          +Vo(i,j)*(e1y(i,j)*dhodyn(i,j)+dfsydy(i,j)))*.5d0

             udb4(i,j)=(dUo_dx(i,j)*d1x(i,j)+dUo_dy(i,j)*d1y(i,j))*aux1*0.5d0
             udb3(i,j)=(dUo_dx(i,j)*e1x(i,j)+dUo_dy(i,j)*e1y(i,j))*aux1
             udb2(i,j)=(dUo_dx(i,j)*fsumx(i,j)+dUo_dy(i,j)*fsumy(i,j))*0.5d0
             vdb4(i,j)=(dVo_dx(i,j)*d1x(i,j)+dVo_dy(i,j)*d1y(i,j))*aux1*0.5d0
             vdb3(i,j)=(dVo_dx(i,j)*e1x(i,j)+dVo_dy(i,j)*e1y(i,j))*aux1
             vdb2(i,j)=(dVo_dx(i,j)*fsumx(i,j)+dVo_dy(i,j)*fsumy(i,j))*0.5d0

             Wstar=(Uo(i,j))*dhodxn(i,j)
     .          +(Vo(i,j))*dhodyn(i,j)

             udw4(i,j)=-d1x(i,j)*(dUo_dx(i,j)+dVo_dy(i,j))*aux1
             udw3(i,j)=-e1x(i,j)*(dUo_dx(i,j)+dVo_dy(i,j))*aux1
     .          -wstar*d1x(i,j)*aux1*2.d0
             udw2(i,j)=-e1x(i,j)*wstar*.5d0
```

```
      vdw4(i,j)=-d1y(i,j)*(dUo_dx(i,j)+dVo_dy(i,j))*aux1
      vdw3(i,j)=-e1y(i,j)*(dUo_dx(i,j)+dVo_dy(i,j))*aux1
     .      -wstar*d1y(i,j)*aux1*2.d0
      vdw2(i,j)=-e1y(i,j)*wstar*.5d0

      udc(i,j)=-(uda4(i,j)+udb4(i,j)+udw4(i,j))*ht4(i,j)*.2d0
     .    -(uda3(i,j)+udb3(i,j)+udw3(i,j))*ht3(i,j)*.25d0
     .    -(uda2(i,j)+udb2(i,j)+udw2(i,j))*ht2(i,j)*aux1*2.d0

      vdc(i,j)=-(vda4(i,j)+vdb4(i,j)+vdw4(i,j))*ht4(i,j)*.2d0
     .    -(vda3(i,j)+vdb3(i,j)+vdw3(i,j))*ht3(i,j)*.25d0
     .    -(vda2(i,j)+vdb2(i,j)+vdw2(i,j))*ht2(i,j)*aux1*2.d0

        uda4(i,j)=uda4(i,j)*inv_vt(i,j)
        uda3(i,j)=uda3(i,j)*inv_vt(i,j)
        uda2(i,j)=uda2(i,j)*inv_vt(i,j)
        vda4(i,j)=vda4(i,j)*inv_vt(i,j)
        vda3(i,j)=vda3(i,j)*inv_vt(i,j)
        vda2(i,j)=vda2(i,j)*inv_vt(i,j)
        udb4(i,j)=udb4(i,j)*inv_vt(i,j)
        udb3(i,j)=udb3(i,j)*inv_vt(i,j)
        udb2(i,j)=udb2(i,j)*inv_vt(i,j)
        vdb4(i,j)=vdb4(i,j)*inv_vt(i,j)
        vdb3(i,j)=vdb3(i,j)*inv_vt(i,j)
        vdb2(i,j)=vdb2(i,j)*inv_vt(i,j)
        udw4(i,j)=udw4(i,j)*inv_vt(i,j)
        udw3(i,j)=udw3(i,j)*inv_vt(i,j)
        udw2(i,j)=udw2(i,j)*inv_vt(i,j)
        vdw4(i,j)=vdw4(i,j)*inv_vt(i,j)
        vdw3(i,j)=vdw3(i,j)*inv_vt(i,j)
        vdw2(i,j)=vdw2(i,j)*inv_vt(i,j)
        udc(i,j)=udc(i,j)*inv_vt(i,j)
        vdc(i,j)=vdc(i,j)*inv_vt(i,j)
      end do

      do i=nxa(j)+1, nx
        Dxx(i,j) = 0.d0
        Dyy(i,j) = 0.d0
        Dxy(i,j) = 0.d0
        Mxx(i,j) = 0.d0
        Mxy(i,j) = 0.d0
        Myy(i,j) = 0.d0
```

```
          Bxx(i,j) = 0.d0
          Bxy(i,j) = 0.d0
          Byy(i,j) = 0.d0
          uda4(i,j)= 0.d0
          uda3(i,j)= 0.d0
          uda2(i,j)= 0.d0
          vda4(i,j)= 0.d0
          vda3(i,j)= 0.d0
          vda2(i,j)= 0.d0
          udb4(i,j)= 0.d0
          udb3(i,j)= 0.d0
          udb2(i,j)= 0.d0
          vdb4(i,j)= 0.d0
          vdb3(i,j)= 0.d0
          vdb2(i,j)= 0.d0
          udw4(i,j)= 0.d0
          udw3(i,j)= 0.d0
          udw2(i,j)= 0.d0
          vdw4(i,j)= 0.d0
          vdw3(i,j)= 0.d0
          vdw2(i,j)= 0.d0
          udc(i,j)= 0.d0
          vdc(i,j)= 0.d0
        end do
       end do

        return
        end
```

8.4 File *winc_std_deriv.f*

This file contains the subroutines for calculating spatial derivatives.

8.4.1 Subroutine dmixed

This subroutine calculates 2nd order mixed derivatives.

```
c ######################################################################
c Mixed DERIVATIVES
      subroutine dmixed(qx,qy,dqxdxdy,
     .                  dqydydy,nxflag)
c LAST MODIFICATION:
```

```
c - Nov  15, 2001
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, j,  nxflag, nxj(nmax), minnxa
      real*8 qx(nxm,nym), qy(nxm,nym),
     . dqxdxdy(nxm,nym), dqydxdy(nxm,nym),
     . dernum

c=====================================================================

c ##  set number of points where to take derivative based on flag
c---- nxflag=0 : perform derivatives until nx
c---- nxflag=1 : perform derivatives until last wet point
      if (nxflag .eq. 0) then
 minnxa=nx
         do j = 1,ny
    nxj(j) = nx
         end do
       else
 minnxa=nx
         do j = 1,ny
    nxj(j) = nxa(j)
         end do
       end if


c------- mixed derivatives ---
       do j = 2,ny-1
         i = 1
         dernum = -3d0*(qx(1,j+1)-qx(1,j-1))+4d0*(qx(2,j+1)-qx(2,j-1))
     .          -(qx(3,j+1)-qx(3,j-1))
         dqxdxdy(i,j) = dernum/(dx*dy*4d0)
         dernum = -3d0*(qy(1,j+1)-qy(1,j-1))+4d0*(qy(2,j+1)-qy(2,j-1))
     .          -(qy(3,j+1)-qy(3,j-1))
         dqydxdy(i,j) = dernum/(dx*dy*4d0)
       do i = 2,nxj(j)-1
         dernum = qx(i+1,j+1)+qx(i-1,j-1)-qx(i-1,j+1)-qx(i+1,j-1)
         dqxdxdy(i,j) = dernum/(dx*dy*4)
         dernum = qy(i+1,j+1)+qy(i-1,j-1)-qy(i-1,j+1)-qy(i+1,j-1)
         dqydxdy(i,j) = dernum/(dx*dy*4)
```

```
 end do
    i = nxj(j)
    dernum =3d0*(qx(i,j+1)-qx(i,j-1))-4d0*(qx(i-1,j+1)-qx(i-1,j-1))
.         +(qx(i-2,j+1)-qx(i-2,j-1))
    dqxdxdy(i,j) = dernum/(dx*dy*4d0)
    dernum =3d0*(qy(i,j+1)-qy(i,j-1))-4d0*(qy(i-1,j+1)-qy(i-1,j-1))
.         +(qy(i-2,j+1)-qy(i-2,j-1))
    dqydxdy(i,j) = dernum/(dx*dy*4d0)
 end do

 if (ipery.eq.1) then
 i=1
 j=1
    dernum =-3d0*(qx(i,j+1)-qx(i,ny-1))
.     +4d0*(qx(i+1,j+1)-qx(i+1,ny-1))
.     -(qx(i+2,j+1)-qx(i+2,ny-1))
    dqxdxdy(i,j) = dernum/(dx*dy*4d0)
    dernum =-3d0*(qy(i,j+1)-qy(i,ny-1))
.     +4d0*(qy(i+1,j+1)-qy(i+1,ny-1))
.     -(qy(i+2,j+1)-qy(i+2,ny-1))
    dqydxdy(i,j) = dernum/(dx*dy*4d0)
  i=1
  j=ny
    dernum =3d0*(qx(i,j-1)-qx(i,2))
.       -4d0*(qx(i+1,j-1)-qx(i+1,2))
.     +(qx(i+2,j-1)-qx(i+2,2))
    dqxdxdy(i,j) = dernum/(dx*dy*4d0)
    dernum =3d0*(qy(i,j-1)-qy(i,2))
.       -4d0*(qy(i+1,j-1)-qy(i+1,2))
.     +(qy(i+2,j-1)-qy(i+2,2))
    dqydxdy(i,j) = dernum/(dx*dy*4d0)
  j=ny
  i=nxj(j)
    dernum =-3d0*(qx(i,j-1)-qx(i,2))
.       +4d0*(qx(i-1,j-1)-qx(i-1,2))
.     -(qx(i-2,j-1)-qx(i-2,2))
    dqxdxdy(i,j) = dernum/(dx*dy*4d0)
    dernum =-3d0*(qy(i,j-1)-qy(i,2))
.       +4d0*(qy(i-1,j-1)-qy(i-1,2))
.     -(qy(i-2,j-1)-qy(i-2,2))
    dqydxdy(i,j) = dernum/(dx*dy*4d0)
  j=1
  i=nxj(j)
```

```
          dernum =3d0*(qx(i,j+1)-qx(i,ny-1))
     .       -4d0*(qx(i-1,j+1)-qx(i-1,ny-1))
     .      +(qx(i-2,j+1)-qx(i-2,ny-1))
          dqxdxdy(i,j) = dernum/(dx*dy*4d0)
          dernum =3d0*(qy(i,j+1)-qy(i,ny-1))
     .       -4d0*(qy(i-1,j+1)-qy(i-1,ny-1))
     .      +(qy(i-2,j+1)-qy(i-2,ny-1))
          dqydxdy(i,j) = dernum/(dx*dy*4d0)

          j=1
          do i=2,nxj(j)-1
            dernum = qx(i+1,j+1)+qx(i-1,ny-1)-qx(i-1,j+1)-qx(i+1,ny-1)
            dqxdxdy(i,j) = dernum/(dx*dy*4)
            dernum = qy(i+1,j+1)+qy(i-1,ny-1)-qy(i-1,j+1)-qy(i+1,ny-1)
            dqydxdy(i,j) = dernum/(dx*dy*4)
          end do
          j=ny
          do i=2,nxj(j)-1
            dernum = qx(i+1,2)+qx(i-1,j-1)-qx(i-1,2)-qx(i+1,j-1)
            dqxdxdy(i,j) = dernum/(dx*dy*4)
            dernum = qy(i+1,2)+qy(i-1,j-1)-qy(i-1,2)-qy(i+1,j-1)
            dqydxdy(i,j) = dernum/(dx*dy*4)
          end do
c     Not Periodic
          else
         i=1
         j=1
          dernum =9d0*qx(i,j)+16d0*qx(i+1,j+1)+qx(i+2,j+2)
     .      -12d0*(qx(i,j+1)+qx(i+1,j))+3d0*(qx(i,j+2)+qx(i+2,j))
     .      -4d0*(qx(i+1,j+2)+qx(i+2,j+1))
          dqxdxdy(i,j) = dernum/(dx*dy*4d0)
          dernum =9d0*qy(i,j)+16d0*qy(i+1,j+1)+qy(i+2,j+2)
     .      -12d0*(qy(i,j+1)+qy(i+1,j))+3d0*(qy(i,j+2)+qy(i+2,j))
     .      -4d0*(qy(i+1,j+2)+qy(i+2,j+1))
          dqydxdy(i,j) = dernum/(dx*dy*4d0)
         i=1
         j=ny
          dernum =-9d0*qx(i,j)-16d0*qx(i+1,j-1)-qx(i+2,j-2)
     .      +12d0*(qx(i,j-1)+qx(i+1,j))-3d0*(qx(i,j-2)+qx(i+2,j))
     .      +4d0*(qx(i+1,j-2)+qx(i+2,j-1))
          dqxdxdy(i,j) = dernum/(dx*dy*4d0)
          dernum =-9d0*qy(i,j)-16d0*qy(i+1,j-1)-qy(i+2,j-2)
     .      +12d0*(qy(i,j-1)+qy(i+1,j))-3d0*(qy(i,j-2)+qy(i+2,j))
```

```
.      +4d0*(qy(i+1,j-2)+qy(i+2,j-1))
       dqydxdy(i,j) = dernum/(dx*dy*4d0)
 j=1
 i=nxj(j)
       dernum =-9d0*qx(i,j)-16d0*qx(i-1,j+1)-qx(i-2,j+2)
.      +12d0*(qx(i,j+1)+qx(i-1,j))-3d0*(qx(i,j+2)+qx(i-2,j))
.      +4d0*(qx(i-1,j+2)+qx(i-2,j+1))
       dqxdxdy(i,j) = dernum/(dx*dy*4d0)
       dernum =-9d0*qy(i,j)-16d0*qy(i-1,j+1)-qy(i-2,j+2)
.      +12d0*(qy(i,j+1)+qy(i-1,j))-3d0*(qy(i,j+2)+qy(i-2,j))
.      +4d0*(qy(i-1,j+2)+qy(i-2,j+1))
       dqydxdy(i,j) = dernum/(dx*dy*4d0)
 j=ny
 i=nxj(j)
       dernum =9d0*qx(i,j)+16d0*qx(i-1,j-1)+qx(i-2,j-2)
.      -12d0*(qx(i,j-1)+qx(i-1,j))+3d0*(qx(i,j-2)+qx(i-2,j))
.      -4d0*(qx(i-1,j-2)+qx(i-2,j-1))
       dqxdxdy(i,j) = dernum/(dx*dy*4d0)
       dernum =9d0*qy(i,j)+16d0*qy(i-1,j-1)+qy(i-2,j-2)
.      -12d0*(qy(i,j-1)+qy(i-1,j))+3d0*(qy(i,j-2)+qy(i-2,j))
.      -4d0*(qy(i-1,j-2)+qy(i-2,j-1))
       dqydxdy(i,j) = dernum/(dx*dy*4d0)

       j=1
       do i=2,nxj(j)-1
          dernum=-3d0*(qx(i+1,j)-qx(i-1,j))
.              +4d0*(qx(i+1,j+1)-qx(i-1,j+1))
.              -(qx(i+1,j+2)-qx(i-1,j+2))
         dqxdxdy(i,j) = dernum/(dx*dy*4)
          dernum=-3d0*(qy(i+1,j)-qy(i-1,j))
.              +4d0*(qy(i+1,j+1)-qy(i-1,j+1))
.              -(qy(i+1,j+2)-qy(i-1,j+2))
         dqydxdy(i,j) = dernum/(dx*dy*4)
       end do
       j=ny
       do i=2,nxj(j)-1
          dernum=3d0*(qx(i+1,j)-qx(i-1,j))
.              -4d0*(qx(i+1,j-1)-qx(i-1,j-1))
.              +(qx(i+1,j-2)-qx(i-1,j-2))
         dqxdxdy(i,j) = dernum/(dx*dy*4)
         dqxdxdy(i,j) = dernum/(dx*dy*4)
          dernum=3d0*(qy(i+1,j)-qy(i-1,j))
.              -4d0*(qy(i+1,j-1)-qy(i-1,j-1))
```

```
     .           +(qy(i+1,j-2)-qy(i-1,j-2))
          dqydxdy(i,j) = dernum/(dx*dy*4)
        end do
        end if
      do j = 1,ny
         do i=nxj(j)+1,nx
            dqxdxdy(i,j)=0.d0
            dqydxdy(i,j)=0.d0
         end do
      end do
     return
     end
```

8.4.2 Subroutine deriv

This subroutine calculates the first derivatives.

```
c ######################################################################
c DERIVATIVES
      subroutine deriv(zeta,qx,qy,dzetadx,dzetady,dqxdx,
     .                 dqydy,ncode,nxflag)
c LAST MODIFICATION:
c - Sep 15, 1999
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, j, ncode, nxflag, nxj(nmax), minnxa
      real*8 zeta(nxm,nym), qx(nxm,nym), qy(nxm,nym),
     . dzetadx(nxm,nym), dzetady(nxm,nym),
     . dqxdx(nxm,nym), dqydy(nxm,nym),
     . spz(nmax),spqx(nmax),spqy(nmax),dzeta(nmax),
     . dqx(nmax),dqy(nmax)

c=====================================================================

c ##  set number of points where to take derivative based on flag
c---- nxflag=0 : perform derivatives until nx
c---- nxflag=1 : perform derivatives until last wet point
      if (nxflag .eq. 0) then
 minnxa=nx
```

```
         do j = 1,ny
    nxj(j) = nx
         end do
       else
 minnxa=nx
         do j = 1,ny
    nxj(j) = nxa(j)
         end do
       end if

c ##  Calculating the x-derivatives ----------------------------------

       if (ncode.eq.1) then
c------- 1st derivatives of higher order : dx^4 ---
         do j = 1,ny
            do i = 1, nxj(j)
               spz(i)  = zeta(i,j)
               spqx(i) = qx(i,j)
               dzeta(i) = 0.d0
               dqx(i) = 0.d0
            end do
            call splinex(nxj(j), spz, dzeta)
            call splinex(nxj(j), spqx, dqx)
            do i = 1, nxj(j)
               dzetadx(i,j) = dzeta(i)
               dqxdx(i,j)   = dqx(i)
            end do
            do i = nxj(j)+1,nx
               dzetadx(i,j) = 0.d0
               dqxdx(i,j)   = 0.d0
            end do
 end do
       elseif (ncode.eq.3) then
c------- 1st derivatives of lower order : dx^2 ---
         do j = 1,ny
            do i = 1, nxj(j)
               spz(i)  = zeta(i,j)
               spqx(i) = qx(i,j)
               dzeta(i) = 0.d0
               dqx(i) = 0.d0
            end do
            call splinex2(nxj(j), spz, dzeta)
            call splinex2(nxj(j), spqx, dqx)
```

```
            do i = 1, nxj(j)
               dzetadx(i,j) = dzeta(i)
               dqxdx(i,j)   = dqx(i)
            end do
            do i = nxj(j)+1,nx
               dzetadx(i,j) = 0.d0
               dqxdx(i,j)   = 0.d0
            end do
 end do

      elseif (ncode.eq.2) then
c------- 3rd derivatives ---
         do j = 1,ny
            do i = 1, nxj(j)
               spz(i)  = zeta(i,j)
               spqx(i) = qx(i,j)
               dzeta(i) = 0.d0
               dqx(i) = 0.d0
            end do
            call triplex(nxj(j), spz, dzeta)
            call triplex(nxj(j), spqx, dqx)
            do i = 1, nxj(j)
               dzetadx(i,j) = dzeta(i)
               dqxdx(i,j)   = dqx(i)
            end do
            do i = nxj(j)+1,nx
               dzetadx(i,j) = 0.d0
               dqxdx(i,j)   = 0.d0
            end do
 end do
      elseif (ncode.eq.4) then
c------- 2nd derivatives ---
         do j = 1,ny
            do i = 1, nxj(j)
               spz(i)  = zeta(i,j)
               spqx(i) = qx(i,j)
               dzeta(i) = 0.d0
               dqx(i) = 0.d0
            end do
            call doublex(nxj(j), spz, dzeta)
            call doublex(nxj(j), spqx, dqx)
            do i = 1, nxj(j)
               dzetadx(i,j) = dzeta(i)
```

```
               dqxdx(i,j)    = dqx(i)
            end do
            do i = nxj(j)+1,nx
               dzetadx(i,j) = 0.d0
               dqxdx(i,j)   = 0.d0
            end do
 end do
      end if


c ##  Calculating the y-derivatives ----------------------------------

      if (ncode.eq.1) then
c------- 1st derivatives  of higher order : dy^4 ---
         do i = 1,minnxa
            do j = 1,ny
               spz(j)  = zeta(i,j)
               spqy(j) = qy(i,j)
               dzeta(i) = 0.d0
               dqy(i) = 0.d0
            end do
            call spliney(ny, spz, dzeta)
            call spliney(ny, spqy, dqy)
            do j = 1,ny
               dzetady(i,j) = dzeta(j)
               dqydy(i,j)   = dqy(j)
            end do
         end do
         do j = 1,ny
               do i = nxj(j)+1,nx
                dzetady(i,j) = 0.d0
                dqydy(i,j)   = 0.d0
               end do
         end do
      elseif (ncode.eq.3) then
c------- 1st derivatives  of lower order : dy^2 ---
         do i = 1,minnxa
            do j = 1,ny
               spz(j)  = zeta(i,j)
               spqy(j) = qy(i,j)
               dzeta(i) = 0.d0
               dqy(i) = 0.d0
            end do
```

```
            call spliney2(ny, spz, dzeta)
            call spliney2(ny, spqy, dqy)
            do j = 1,ny
               dzetady(i,j) = dzeta(j)
               dqydy(i,j)   = dqy(j)
            end do
         end do
            do j = 1,ny
               do i =nxj(j)+1,nx
                 dzetady(i,j) = 0.d0
                 dqydy(i,j)   = 0.d0
            end do
         end do
      elseif (ncode.eq.2) then
c------- 3rd derivatives ---
         do i = 1,minnxa
            do j = 1,ny
               spz(j)  = zeta(i,j)
               spqy(j) = qy(i,j)
               dzeta(i) = 0.d0
               dqy(i) = 0.d0
            end do
            call tripley(ny, spz, dzeta)
            call tripley(ny, spqy, dqy)
            do j = 1,ny
               dzetady(i,j) = dzeta(j)
               dqydy(i,j)   = dqy(j)
            end do
         end do
            do j = 1,ny
                do i = nxj(j)+1,nx
                 dzetady(i,j) = 0.d0
                 dqydy(i,j)   = 0.d0
            end do
         end do
      elseif (ncode.eq.4) then
c------- 2nd derivatives ---
         do i = 1,minnxa
            do j = 1,ny
               spz(j)  = zeta(i,j)
               spqy(j) = qy(i,j)
               dzeta(i) = 0.d0
               dqy(i) = 0.d0
```

```
            end do
            call doubley(ny, spz, dzeta)
            call doubley(ny, spqy, dqy)
            do j = 1,ny
               dzetady(i,j) = dzeta(j)
               dqydy(i,j)   = dqy(j)
            end do
         end do
            do j = 1,ny
              do i = nxj(j)+1,nx
               dzetady(i,j) = 0.d0
               dqydy(i,j)   = 0.d0
            end do
         end do
      end if


      return
      end
```

8.4.3 Subroutines splinex2, splinex, spliney2 and spliney

These subroutines are called by subroutine deriv to calculate the 2nd or 4th order single derivatives.

```
c ######################################################################
c SPLINE X, 2nd Order
      subroutine splinex2(ndata,fdata,fxdata)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, ndata
      real*8 fdata(ndata), fxdata(ndata), dernum

c=====================================================================

      do i = 2,ndata-1
         dernum = fdata(i+1) - fdata(i-1)
         fxdata(i) = dernum/(2.d0*dx)
      end do
```

```
      if (iperx.eq.1) then
         dernum = fdata(2) - fdata(ndata-1)
         fxdata(1) = dernum/(2.d0*dx)
         fxdata(ndata) = fxdata(1)
      else
         i = 1
         dernum = -3.d0*fdata(i) + 4.d0*fdata(i+1) - fdata(i+2)
         fxdata(i) = dernum/(2.d0*dx)
c         dernum = -fdata(i) + fdata(i+1)
c         fxdata(i) = dernum/dx
         i = ndata
         dernum = 3.d0*fdata(i) - 4.d0*fdata(i-1) + fdata(i-2)
         fxdata(i) = dernum/(2.d0*dx)
c         dernum = fdata(i) - fdata(i-1)
c         fxdata(i) = dernum/dx
      end if

      return
      end


c ######################################################################
c SPLINE X, 4th Order
      subroutine splinex(ndata,fdata,fxdata)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, ndata
      real*8 fdata(ndata), fxdata(ndata), dernum, aux1, aux2

c=====================================================================

      aux1 = 1.d0/(12.d0*dx)
      aux2 = 1.d0/(2.d0*dx)
      if (iperx.eq.1) then
         i = 1
         dernum = -fdata(i+2)+8.d0*fdata(i+1)-8.d0*fdata(ndata-1)
     .          +fdata(ndata-2)
         fxdata(i) = dernum *aux1
         i = 2
```

```
         dernum = -fdata(i+2)+8.d0*fdata(i+1)-8.d0*fdata(i-1)
     .          +fdata(ndata-1)
         fxdata(i) = dernum *aux1
      else
         i = 1
         dernum = -3.d0*fdata(i) + 4.d0*fdata(i+1) - fdata(i+2)
         fxdata(i) = dernum *aux2
         i = 2
         dernum = fdata(i+1) - fdata(i-1)
         fxdata(i) = dernum *aux2
      end if

      do i = 3,ndata-2
         dernum = -fdata(i+2)+8.d0*fdata(i+1)-8.d0*fdata(i-1)+fdata(i-2)
         fxdata(i) = dernum *aux1
      end do

      if (iperx.eq.1) then
         fxdata(ndata) = fxdata(1)
         i = ndata-1
         dernum = -fdata(2)+8.d0*fdata(i+1)-8.d0*fdata(i-1)+fdata(i-2)
         fxdata(i) = dernum *aux1
      else
         i = ndata-1
         dernum = fdata(i+1) - fdata(i-1)
         fxdata(i) = dernum *aux2
         i = ndata
         dernum = 3.d0*fdata(i) - 4.d0*fdata(i-1) + fdata(i-2)
         fxdata(i) = dernum *aux2
      end if

      return
      end


c ######################################################################
c SPLINE Y, 2nd Order
      subroutine spliney2(ndata,fdata,fxdata)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'
c--- Local variables ------------------------------------------------
      integer j, ndata
```

```
      real*8 fdata(ndata), fxdata(ndata), dernum

c=======================================================================

      do j = 2,ndata-1
         dernum = fdata(j+1) - fdata(j-1)
         fxdata(j) = dernum/(2.d0*dy)
      end do

      if (ipery.eq.1) then
         dernum = fdata(2) - fdata(ndata-1)
         fxdata(1) = dernum/(2.d0*dy)
         fxdata(ndata) = fxdata(1)
      else
         j=1
         dernum = -3.d0*fdata(j) + 4.d0*fdata(j+1) - fdata(j+2)
         fxdata(1) = dernum/(2.d0*dy)
         j = ndata
         dernum = 3.d0*fdata(j) - 4.d0*fdata(j-1) + fdata(j-2)
         fxdata(j) = dernum/(2.d0*dy)
      end if

      return
      end


c ######################################################################
c SPLINE Y, 4th Order
      subroutine spliney(ndata,fdata,fxdata)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'
c--- Local variables ------------------------------------------------
      integer j, ndata
      real*8 fdata(ndata), fxdata(ndata), dernum, aux1, aux2

c=======================================================================

      aux1 = 1.d0/(12.d0*dy)
      aux2 = 1.d0/(2.d0*dy)
      if (ipery.eq.1) then
         j = 1
         dernum = -fdata(j+2)+8.d0*fdata(j+1)-8.d0*fdata(ndata-1)
```

```
.          +fdata(ndata-2)
   fxdata(j) = dernum *aux1
   j = 2
   dernum = -fdata(j+2)+8.d0*fdata(j+1)-8.d0*fdata(j-1)
.          +fdata(ndata-1)
   fxdata(j) = dernum *aux1
else
   j = 1
   dernum = -3.d0*fdata(j) + 4.d0*fdata(j+1) - fdata(j+2)
   fxdata(1) = dernum *aux2
   j = 2
   dernum = fdata(j+1) - fdata(j-1)
   fxdata(j) = dernum *aux2
end if

do j = 3,ndata-2
   dernum = -fdata(j+2)+8.d0*fdata(j+1)-8.d0*fdata(j-1)+fdata(j-2)
   fxdata(j) = dernum *aux1
end do

if (ipery.eq.1) then
   fxdata(ndata) = fxdata(1)
   j = ndata-1
   dernum = -fdata(2)+8.d0*fdata(j+1)-8.d0*fdata(j-1)+fdata(j-2)
   fxdata(j) = dernum *aux1
else
   j = ndata-1
   dernum = fdata(j+1) - fdata(j-1)
   fxdata(j) = dernum *aux2
   j = ndata
   dernum = 3.d0*fdata(j) - 4.d0*fdata(j-1) + fdata(j-2)
   fxdata(j) = dernum *aux2
end if

return
end
```

8.4.4 Subroutines doublex and doubley

These subroutines are called by subroutine deriv to calculate the double derivatives.

```
c ######################################################################
```

```
c DOUBLE X
      subroutine doublex(ndata,fdata,fxdata)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer i, ndata
      real*8 fdata(ndata), fxdata(ndata), dernum

c=====================================================================
c
      if (iperx.eq.1) then
         i = 1
         dernum = fdata(2)-2.d0*fdata(1)+fdata(ndata-1)
         fxdata(i) = dernum/(dx**2)
      else
         i = 1
         dernum = 2.d0*fdata(1)-5.d0*fdata(2)+4.d0*fdata(3)-fdata(4)
         fxdata(i) = dernum/(dx**2)
      end if
      do i = 2,ndata-1
         dernum = fdata(i+1)-2.d0*fdata(i)+fdata(i-1)
         fxdata(i) = dernum/(dx**2)
      end do
      if (iperx.eq.1) then
         fxdata(ndata) = fxdata(1)
      else
         i = ndata
         dernum= 2.d0*fdata(ndata)-5.d0*fdata(ndata-1)
     .           +4.d0*fdata(ndata-2)-fdata(ndata-3)
         fxdata(i) = dernum/(dx**2)
      end if

      return
      end


c ######################################################################
c DOUBLE Y
      subroutine doubley(ndata,fdata,fxdata)
c ######################################################################
CDIR$ FLOW
```

```
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer j, ndata
      real*8 fdata(ndata), fxdata(ndata), dernum

c======================================================================

c     only for ipery

      if (ipery.eq.1) then
       j = 1
       dernum = fdata(2)-2.d0*fdata(1)+fdata(ndata-1)
       fxdata(j) = dernum/(dy**2)
      end if
c
      do j = 2,ndata-1
         dernum = fdata(j+1)-2.d0*fdata(j)+fdata(j-1)
         fxdata(j) = dernum/(dy**2)
      end do
c
      if (ipery.eq.1) then
       fxdata(ndata) = fxdata(1)
       j = ndata-1
      end if
c
      return
      end
```

8.4.5 Subroutines triplex and tripley

These subroutines are called by subroutine deriv to calculate the triple derivatives. They are presently not used by SHORECIRC.

```
c ######################################################################
c TRIPLE X
      subroutine triplex(ndata,fdata,fxdata)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
```

```
      integer i, ndata
      real*8 fdata(ndata), fxdata(ndata), dernum

c=====================================================================
c
      if (iperx.eq.1) then
       i = 1
       dernum = fdata(3)-2.d0*fdata(2)+2.d0*fdata(ndata-1)
     . -fdata(ndata-2)
       fxdata(i) = dernum/(2.d0*(dx**3))
       i= 2
       dernum = fdata(4)-2.d0*fdata(3)+2.d0*fdata(1)-fdata(ndata-1)
       fxdata(i) = dernum/(2.d0*(dx**3))
      else
       i = 1
       dernum = fdata(1)-3.d0*fdata(2)+3.d0*fdata(3)-fdata(4)
       fxdata(i) = dernum/((2.d0*dx)**3)
       i = 2
       dernum = -fdata(1)+3.d0*fdata(2)-3.d0*fdata(3)+fdata(4)
       fxdata(i) = dernum/(2.d0*(dx**3))
      end if
c
      do i = 3,ndata-2
         dernum = fdata(i+2)-2.d0*fdata(i+1)+2.d0*fdata(i-1)
     . -fdata(i-2)
         fxdata(i) = dernum/(2.d0*(dx**3))
      end do
c
      if (iperx.eq.1) then
       fxdata(ndata) = fxdata(1)
       i=ndata
       dernum = fdata(3)-2.d0*fdata(2)+2.d0*fdata(ndata-1)
     . -fdata(ndata-2)
       fxdata(i) = dernum/(2.d0*(dx**3))
       i = ndata-1
       dernum = fdata(2)-2.d0*fdata(1)+2.d0*fdata(ndata-2)
     . -fdata(ndata-3)
       fxdata(i) = dernum/(2.d0*(dx**3))

      else
       i = ndata
       dernum = -fdata(ndata)+3.d0*fdata(ndata-1)-3.d0*fdata(ndata-2)
     .          +fdata(ndata-3)
```

```
        fxdata(i) = dernum/((2.d0*dx)**3)
        i = ndata-1
        dernum = fdata(ndata)-3.d0*fdata(ndata-1)+3.d0*fdata(ndata-2)
     .           -fdata(ndata-3)
        fxdata(i) = dernum/(2.d0*(dx**3))

      end if
c

      return
      end


c ######################################################################
c TRIPLE Y
      subroutine tripley(ndata,fdata,fxdata)
c ######################################################################
CDIR$ FLOW
      include 'winc_std_common.inc'

c--- Local variables ------------------------------------------------
      integer j, ndata
      real*8 fdata(ndata), fxdata(ndata), dernum

c=====================================================================

      if (ipery.eq.1) then
       j = 1
       dernum = fdata(3)-2.d0*fdata(2)+2.d0*fdata(ndata-1)
     .    -fdata(ndata-2)
       fxdata(j) = dernum/(2.d0*(dy**3))
       j= 2
       dernum = fdata(4)-2.d0*fdata(3)+2.d0*fdata(1)-fdata(ndata-1)
       fxdata(j) = dernum/(2.d0*(dy**3))
      else
       j = 1
       dernum = fdata(1)-3.d0*fdata(2)+3.d0*fdata(3)-fdata(4)
       fxdata(j) = dernum/((2.d0*dy)**3)
       j = 2
       dernum = -fdata(1)+3.d0*fdata(2)-3.d0*fdata(3)+fdata(4)
       fxdata(j) = dernum/(2.d0*(dy**3))
      end if
c
```

```
      do j = 3,ndata-2
         dernum = fdata(j+2)-2.d0*fdata(j+1)+2.d0*fdata(j-1)-fdata(j-2)
         fxdata(j) = dernum/(2.d0*(dy**3))
      end do
c
      if (ipery.eq.1) then
       fxdata(ndata) = fxdata(1)
       j = ndata-1
       dernum = fdata(2)-2.d0*fdata(1)+2.d0*fdata(ndata-2)
     . -fdata(ndata-3)
       fxdata(j) = dernum/(2.d0*(dy**3))

      else
       j = ndata
       dernum = -fdata(ndata)+3.d0*fdata(ndata-1)-3.d0*fdata(ndata-2)
     .          +fdata(ndata-3)
       fxdata(j) = dernum/((2.d0*dy)**3)
       j = ndata-1
       dernum = fdata(ndata)-3.d0*fdata(ndata-1)+3.d0*fdata(ndata-2)
     .          -fdata(ndata-3)
       fxdata(j) = dernum/(2.d0*(dy**3))

      end if
c
      return
      end
```

8.4.6 Subroutine predcor

This subroutine supplies the predictor and corrector coefficients used in the time intergration.

```
c ######################################################################
c!!!! Predictor/corrector coeff.s as computed by Uday
      subroutine predcor(nord,coeff1,coeff2,denom1,denom2)
c ######################################################################


c--- Local variables ------------------------------------------------
      integer i, nord
      real*8 coeff1(10), coeff2(10), denom1, denom2
```

```
c=====================================================================

      do i = 1,10
         coeff1(i) = 0.d0
         coeff2(i) = 0.d0
      end do
c
      if (nord.eq.1) then
         coeff1(1) = 1.d0
          coeff2(1) = 1.d0
          coeff2(2) = 1.d0
         denom1 = 1.d0
         denom2 = 2.d0
      else
         if (nord.eq.2) then
              coeff1(1) = 23.d0
              coeff1(2) = -16.d0
              coeff1(3) = 5.d0
               coeff2(1) = 5.d0
               coeff2(2) = 8.d0
               coeff2(3) = -1.d0
              denom1 = 12.d0
              denom2 = 12.d0
         else
              if (nord.eq.3) then
                   coeff1(1) = 23.d0
                   coeff1(2) = -16.d0
                   coeff1(3) = 5.d0
                    coeff2(1) = 9.d0
                    coeff2(2) = 19.d0
                    coeff2(3) = -5.d0
                    coeff2(4) = 1.d0
                   denom1 = 12.d0
                   denom2 = 24.d0
              else
                   if (nord.eq.4) then
                        coeff1(1) = 55.d0
                        coeff1(2) = -59.d0
                        coeff1(3) = 37.d0
                        coeff1(4) = -9.d0
                         coeff2(1) = 251.d0
                         coeff2(2) = 646.d0
                         coeff2(3) = -264.d0
```

```
                coeff2(4) = 106.d0
                coeff2(5) = -19.d0
               denom1 = 24.d0
               denom2 = 720.d0
          else
               if (nord.eq.5) then
                    coeff1(1) = 1901.d0
                    coeff1(2) = -2774.d0
                    coeff1(3) = 2616.d0
                    coeff1(4) = -1274.d0
                    coeff1(5) = 251.d0
                     coeff2(1) = 251.d0
                     coeff2(2) = 646.d0
                     coeff2(3) = -264.d0
                     coeff2(4) = 106.d0
                     coeff2(5) = -19.d0
                    denom1 = 720.d0
                    denom2 = 720.d0
               end if
          end if
       end if
    end if
c
    return
    end
c#########################################################################
```

8.5 File *winc_std_filter.f*

This file contains the subroutine for the Shapiro filter from Shapiro (1969).

```
c-----------------------------------------------------------------------
c ## Shapiro filter, from Shapiro(1969), J Geoph and Space Ph
c-----------------------------------------------------------------------
      subroutine shapd1(b1,bout,ndim,norder,scheme)

c      implicit undefined (a-z)
      integer nxm, nym, nmax
      parameter (nxm = 200, nym = 200, nmax=200)
```

```
      integer ndim, norder, scheme
      real*8  b1(nmax),bn(nmax),bout(nmax),fourk(20),nordsq
c  Local variables
      integer      i, i1, i2, k, kit, nord2, ndm1

c      REAL      b1(ndim),bn(ndim),bout(ndim),fourk(20)
c  changed ndim to nxm here to conform to aux and aux2 which we call
c  b1(ndim)            vector to be filtered
c  ndim                dimension of b1,bout   (ndim<200)
c  norder              order of the shapiro filter
c  scheme              flag for the type of boundary scheme to be used
c                      if scheme=
c                          1       no change at wall - constant order
c                          2       smoothing at wall - constant order
c                          3       no change at wall - reduced order
c                          4       smoothing at wall - reduced order
c                          5       periodic          - constant order
c    6    no change at wall - no overwriting
c                                  (1=ndm1 and 2=ndim)
c                      any other scheme is rejected
c  fourk  (out)              scaling factors
c
c  output:
c
c  bout(ndim)          correction field (not filtered field)



      nord2 = norder/2
      ndm1 = ndim-1
      nordsq= (0.5d0)**(norder)
c  scheme 1:  constant order and no change at wall

      if (scheme.eq.1) then
        do 120 k=1,nord2
          if(k.ne.nord2)  then
            bn(1) = 2.*(b1(1) - b1(2))
            bn(ndim) = 2.*(b1(ndim)-b1(ndm1))
          else
            bn(1) = 0.
            bn(ndim) = 0.
          end if
          do 100 i=2,ndm1
```

```
            bn(i) =(2*b1(i)-b1(i-1)-b1(i+1))
  100       continue
            do 110 i=1,ndim
              b1(i)=bn(i)
  110       continue
  120     continue
          do 130 i=1,ndim
            bout(i) = bn(i)*nordsq
c fourk(nord2)
  130     continue
          return

c  scheme 2:  constant order, smoothed at edges

        else if (scheme.eq.2) then
          do 160 k=1,nord2
            bn(1) = 2.*(b1(1) - b1(2))
            bn(ndim) = 2.*(b1(ndim)-b1(ndm1))
            do 140 i=2,ndm1
              bn(i) =(2*b1(i)-b1(i-1)-b1(i+1))
  140       continue
            do 150 i=1,ndim
              b1(i)=bn(i)
  150       continue
  160     continue
          do 170 i=1,ndim
            bout(i) = bn(i)*nordsq
c fourk(nord2)
  170     continue
          return

c  scheme 3:  reduced order and no change at wall

        else if (scheme.eq.3) then
          do 210 k=1,nord2
            kit = k
            i1 = k
            i2 = ndim-k+1
            if(k.eq.1) then
              do 180 i=2,ndm1
                bn(i) = 2*b1(i)-b1(i-1)-b1(i+1)
  180         continue
              bn(1) = 2.*(b1(1) - b1(2))
```

```
            bn(ndim) = 2.*(b1(ndim)-b1(ndm1))
          else
            do 190 i=i1,i2
              bn(i) = 2*b1(i)-b1(i-1)-b1(i+1)
  190       continue
          end if
          bout(i1) = bn(i1)*(.5d0)**(2.*kit)
c fourk(kit)
          bout(i2) = bn(i2)*(.5d0)**(2.*kit)
c fourk(kit)
          do 200 i=1,ndim
            b1(i) = bn(i)
  200     continue
  210   continue
        do 220 i=i1,i2
          bout(i) = bn(i)*(.5d0)**(2.*kit)
c fourk(kit)
  220   continue
        bout(1) = 0.
        bout(ndim) = 0.
        return

c  scheme 4:  reduced order, smoothed at edges

      else if (scheme.eq.4) then
        do 260 k=1,nord2
          kit = k
          i1 = k
          i2 = ndim-k+1
          if(k.eq.1) then
            do 230 i=2,ndm1
              bn(i) = 2*b1(i)-b1(i-1)-b1(i+1)
  230       continue
            bn(1) = 2.*(b1(1) - b1(2))
            bn(ndim) = 2.*(b1(ndim)-b1(ndm1))
          else
            do 240 i=i1,i2
              bn(i) = 2*b1(i)-b1(i-1)-b1(i+1)
  240       continue
          end if
          bout(i1) = bn(i1)*(.5d0)**(2.*kit)
c fourk(kit)
          bout(i2) = bn(i2)*(.5d0)**(2.*kit)
```

```
c fourk(kit)
          do 250 i=1,ndim
            b1(i) = bn(i)
  250     continue
  260   continue
        do 270 i=i1,i2
          bout(i) = bn(i)*(.5d0)**(2.*kit)
c fourk(kit)
  270   continue
        return

c  scheme 5:  constant order, periodic

      else if (scheme.eq.5) then
        do 300  k=1,nord2
          do 280  i=2,ndm1
            bn(i) = (2*b1(i)-b1(i-1)-b1(i+1))
  280     continue
          do 290  i=2,ndm1
            b1(i) = bn(i)
  290     continue
          b1(1) = b1(ndm1)
          b1(ndim) = b1(2)
  300   continue
        do 310  i=1,ndim
          bout(i) = b1(i)*nordsq
c fourk(nord2)
  310   continue
        return

c  scheme 6:  constant order and no change at wall (without overwriting)

      else if (scheme.eq.6) then
        do 520 k=1,nord2
            bn(1) = 0.d0
            bn(ndim) = 0.d0
          do 500 i=2,ndm1
            bn(i) =(2*b1(i)-b1(i-1)-b1(i+1))
  500     continue
          do 510 i=1,ndim
            b1(i)=bn(i)
  510     continue
  520   continue
```

```
        do 530 i=1,ndim
          bout(i) = bn(i)*nordsq

  530   continue
        return

      end if
      end
```

8.6 File *winc_std_common.inc*

This file contains the common variables shared by all subroutines.

```
c--- File with commands common to all routines -----

c      implicit undefined (a-z)
       integer nxm, nym, nmax, nsensor
       parameter (nxm = 200, nym = 200, nmax=200, nsensor=20)

c--- Common variables ------------------------------------------------

       common/bath/ht, depthmax, slope, depthmin,httide
      .      /beta_char/betan, bnm2, bnm1
      .      /boundaries/ibc1, ibc2, ibc3, ibc4, iperx, ipery
      .      /break/Hbr, ibr, ibrk
      .      /depvars/zetan, qxn, qyn
      .      /explcont/cn, cnm1, cnm2
      .      /explxmom/mxn, mxnm1, mxnm2
      .      /explymom/myn, mynm1, mynm2
      .      /grids/x ,y, xsensor, ysensor,yprofile,jump
      .      /initial/kold_start, interval, delay
      .      /initsetup/setup
      .      /longwave/alpha, k, ampli, wlec, sigma
      .      /math/pi
      .      /numscheme/cr, dx, dy, dt, nx,nxwall, ny, ntime, nxa
      .      /physical/g,rho,rhod,vtshear,mdiss, vtnormal,fcw,fcwij,Cs
      .      /shear/velocity,disp3d
      .      /shortwave/H,theta,costh,sinth,u0,period,c_sw,Qw,qwx,qwy,
      .                 shwave_force_x,shwave_force_y,Bo,amerge,
```

```
.         freq,wn,dissipation,wc,files
.       /uday/hto
.       /visco/v_t, vt_normal
.       /xmom/dSxxdxn, dhodxn, dSxydyn
.       /ymom/dSxydxn, dhodyn, dSyydyn
.       /U_parameters/a1x,b1x,c1x,d1x,d2x,e1x,f1x,f2x,fs1x
.       /V_parameters/a1y,b1y,c1y,d1y,d2y,e1y,f1y,f2y,fs1y
.       /u1_parameters/uda4,uda3,uda2,udb4,udb3,udb2,
. udw4,udw3,udw2,udc
.       /v1_parameters/vda4,vda3,vda2,vdb4,vdb3,vdb2,
. vdw4,vdw3,vdw2,vdc
.       /streaming/tauss_x, tauss_y
.       /wind/wind_vel,wind_dir
.       /output/tavgout,momout,dispout
 real*8
. cr, dx, dy, dt,amerge,httide,freq(nxm,nym),wn(nxm,nym),
. alpha(nxm), k(nxm), ampli, wlec, sigma,dissipation(nxm,nym),
. H(nxm,nym), theta(nxm,nym), u0(nxm,nym), period, c_sw(nxm,nym),
. costh(nxm,nym), sinth(nxm,nym),
. Qw(nxm,nym), qwx(nxm,nym), qwy(nxm,nym), Bo,
. shwave_force_x(nxm,nym), shwave_force_y(nxm,nym),
. ht(nxm,nym), depthmax, slope, x(nxm), y(nym), depthmin,
. g, rho,rhod, vtshear,mdiss,Cs,vtnormal,fcw,pi,fcwij(nxm,nym),
. zetan(nxm,nym), qxn(nxm,nym), qyn(nxm,nym),
. cn(nxm,nym), cnm1(nxm,nym), cnm2(nxm,nym),
. mxn(nxm,nym), mxnm1(nxm,nym), mxnm2(nxm,nym),
. myn(nxm,nym), mynm1(nxm,nym), mynm2(nxm,nym),
. dSxxdxn(nxm,nym), dhodxn(nxm,nym), dSxydyn(nxm,nym),
. dSxydxn(nxm,nym), dhodyn(nxm,nym), dSyydyn(nxm,nym),
. betan(3,nym), bnm2(nym), bnm1(nym),
. delay,
. Hbr, setup(nxm,nym),
. v_t(nxm,nym), vt_normal(nxm,nym),
. velocity(nxm), hto(nxm,nym),
. a1x(nxm,nym), b1x(nxm,nym), c1x(nxm,nym),d1x(nxm,nym),
. d2x(nxm,nym), e1x(nxm,nym),f1x(nxm,nym), f2x(nxm,nym),
. fs1x(nxm,nym),
. a1y(nxm,nym), b1y(nxm,nym),c1y(nxm,nym),d1y(nxm,nym),
. d2y(nxm,nym), e1y(nxm,nym),f1y(nxm,nym),f2y(nxm,nym),
. fs1y(nxm,nym), wind_vel,wind_dir,
. tauss_x(nxm,nym), tauss_y(nxm,nym),
. uda4(nxm,nym),uda3(nxm,nym),uda2(nxm,nym),udb4(nxm,nym),
. udb3(nxm,nym),udb2(nxm,nym),
```

```
     . udw4(nxm,nym),udw3(nxm,nym),udw2(nxm,nym),udc(nxm,nym),
     . vda4(nxm,nym),vda3(nxm,nym),vda2(nxm,nym),vdb4(nxm,nym),
     . vdb3(nxm,nym),vdb2(nxm,nym),
     . vdw4(nxm,nym),vdw3(nxm,nym),vdw2(nxm,nym),vdc(nxm,nym)

      integer
     . nx, ny, ntime, nxa(nym),nxwall,files,
     . ibc1, ibc2, ibc3, ibc4, iperx, ipery,
     . kold_start, interval,wc,disp3d,yprofile(8),
     . ibr(nym), ibrk(nxm,nym), xsensor(nsensor), ysensor(nsensor),
     . tavgout,momout,dispout,jump
```

8.7 File *pass.inc*

This file contains the common variables passed between the wave driver and the circulation model.

```
      integer cnx, cny

      parameter (cnx = 501, cny = 501)

      real*8 ctheta(cnx,cny), ccel(cnx,cny),cwh(cnx,cny),
     . ccg(cnx,cny)

integer cibr(cnx,cny)

      common/pass/ctheta,ccel,ccg,cwh, cibr
```

8.8 File *create_input.f*

This file contains the program used to generate the input files *input_winc.dat* and *indat.dat*.

```
c**********************************************************************
c*---------------------------------------------------------------------
c*    create_input.f
c*
c*    Used to create input_winc.dat and indat.dat
c*    Based on Ref/Dif's indat-createv25.f
c*
```

```
c*    Last revision May 6, 2002.
c*
c*--------------------------------------------------------------------------
c**************************************************************************

      program createinput

      include 'winc_std_common.inc'
      include 'param.h'
      dimension md(ixr), dconv(2), iff(3)
      dimension freqs(ncomp), edens(ncomp), nwavs(ncomp)
      dimension amp(ncomp,ncomp), dir(ncomp,ncomp), tide(ncomp)
      real*8 amp,dir,kapp,gamm

      integer numsens,ans

      character*255 fname1,fname2,fname3,fname4,fname5,fname6,fname7,
     1              fname8,fname9,fname10,fname11,fname12,fname13,
     1               fname14,fname15,fnamein

      data fname1 /'refdat.dat'/, fname2 /'outdat.dat'/,
     1     fname3/'subdat.dat'/,fname4/'wave.dat'/,
     1     fname5/'owave.dat'/,fname6/''/,
     1     fname7/'bottomu.dat'/,fname8/'angle.dat'/,
     1     fname9/' '/,fname10/'refdif1.log'/,
     1     fname11/'height.dat'/,fname12/'sxx.dat'/,
     1     fname13/'sxy.dat'/,fname14/'syy.dat'/,
     1     fname15/'depth.dat'/


      namelist /ingrid/ mr, nr, iu, ntype, icur, ibc,ismooth, dxr, dyr,
     1                  dt, ispace, nd, iff, isp, iinput, ioutput
     1        /inmd/  md
     1        /fnames/ fname1,fname2,fname3,fname4,fname5,fname6,
     1                 fname7,fname8,fname9,fname10,fname11,fname12,
     1                 fname13,fname14,fname15
     1        /waves1a/ iwave, nfreqs, kapp,gamm
     1        /waves1b/ freqs, tide, nwavs, amp, dir
     1        /waves1c/ thet0, freqs, tide, edens, nwavs, nseed
     1        /waves2/ freqin, tidein


      namelist
```

```
     .        /boundary/ibc1,ibc2,ibc3,ibc4
     .        /grids/dx,dy,nx,ny,nxwall,httide
     .        /short_wave/period,wc,amerge,files
     .        /physics/fcw, vtshear,mdiss,Cs,wind_vel,wind_dir,disp3d
     .        /control/cr,ntime,kold_start,depthmin,delay
     .        /sensor/jump,xsensor, ysensor, yprofile
     .        /output/interval,tavgout,momout,dispout



      write(*,*)'**************************************'
      write(*,*)'generating input_winc.dat for SHORECIRC'
      write(*,*)'**************************************'
      open(unit=10,file='input_winc.dat')

c  Enter boundary data.

      write(*,*)' Enter Offshore Boundary Condition'
      write(*,*)'   (2) Abs/Gen BC'
      write(*,*)'   (4) Wall BC'
      read(*,*) ibc1

      write(*,*)' Enter Shoreline Boundary Condition'
      write(*,*)'   (4) Wall BC'
      write(*,*)'   (6) No-flux BC following SWL'
      read(*,*) ibc2

      write(*,*)' Enter Lateral Boundary Condition'
      write(*,*)'   (1) Flux Specified'
      write(*,*)'   (4) Wall BC'
      write(*,*)'   (5) Periodicity'
      read(*,*) ibc3
      ibc4=ibc3


c  Enter grids data.
 500  continue
      write(*,*)'Use dimensions from dim.dat written by createbath?'
      write(*,*)'  (1) yes'
      write(*,*)'  (0) no'
      read(*,*)ans
      if (ans.eq.1) then
```

```
         open(unit=50,file='dim.dat',status='old')
         read(50,*)nx,ny,dx,dy,depthmin
         close(50)
      else if (ans.ne.0) then
         write(*,*)'Invalid response'
         goto 500
      else
         write(*,*)' Enter Grid Dimensions nx, ny'
         read(*,*) nx,ny
         write(*,*)' Enter Grid Spacings dx, dy (m)'
         read(*,*) dx, dy
      end if
c  Wall and tidal levels
      if (ibc2.eq.4) then
      write(*,*)' Enter the shoreline location for the wall (i= ?)'
      read(*,*) nxwall
      else
         nxwall=nx
      end if

      write(*,*)' Enter the tidal level (m)'
      read(*,*) httide
c  Enter Shortwave data.

      write(*,*)' Enter Short Wave Period (s)'
      read(*,*) period

      write(*,*)' Enter Interval For Wave/Current Interaction
     .     (# of time steps)'
      write(*,*)'    (0) For None'
      read(*,*) wc
      icur=1
      write(*,*)' Enter value of amerge for roller transistion
     .     (5 typical)'
      read(*,*) amerge

      write(*,*)' Do you want to import the wave forcing from files?'
      write(*,*)'   (0) No '
      write(*,*)'   (1) Yes'
      read(*,*) files
```

```
c  Enter Control data.

      write(*,*)' Enter Courant Number'
      write(*,*)'    recommend 0.5'
      read(*,*) cr

      write(*,*)' Enter Number of Time Steps and Snapshot Interval'
      read(*,*) ntime,interval

      write(*,*)' Cold Start?'
      write(*,*)'   (0) No '
      write(*,*)'   (1) Yes'
      read(*,*) kold_start

c      if (ans.ne.1) then
      write(*,*)' Enter Minimum Depth (m)'
      read(*,*) depthmin
c      end if

      write(*,*)' Enter Delay (s)'
      read(*,*) delay

c  Enter Sensor data.
      write(*,*)' Enter jump between time steps in time series
     .     (# of time steps)'
      read(*,*) jump


      write(*,*)' Enter Number of Sensors to record time series'
      read(*,*) numsens

       do i=1,numsens
222      continue
        write(*,*)' Enter x,y location for sensor ',i,'  (i,j)'
        read(*,*)xsensor(i),ysensor(i)
        if (xsensor(i).gt.nx.or.ysensor(i).gt.ny) then
          write(*,*)' Sensor outside of domain'
          goto 222
        endif
      end do
      xsensor(numsens+1)=0
      ysensor(numsens+1)=0
```

```
      do i=1,8
223     continue
      write(*,*)'Enter location',i,' for vertical profiles (j=?)'
      write(*,*)'Enter 0 for none'
        read(*,*)yprofile(i)
        if (yprofile(i).eq.0) goto 665
        if (yprofile(i).gt.ny) then
          write(*,*)' Section outside of domain'
          goto 223
        endif
      end do

665   continue
c  Enter Output data

      write(*,*)' Output time averages?'
      write(*,*)'   (0) No '
      write(*,*)'   (1) Yes'
      read(*,*) tavgout

      write(*,*)' Output momentum balances?'
      write(*,*)'   (0) No '
      write(*,*)'   (1) Yes'
      read(*,*) momout

      write(*,*)' Output 3D disperison coeffs.?'
      write(*,*)'   (0) No '
      write(*,*)'   (1) Yes'
      read(*,*) dispout


c  Enter Physics data.

      write(*,*)' Enter Friction Factor (typically 0.02)'
      read(*,*) fcw

      write(*,*)' Enter vtshear and mdiss (typically 0.08 and 0.1) '
      read(*,*) vtshear, mdiss

      write(*,*)' Enter Smagorinsky coefficient Cs (typically 0.2)'
      read(*,*) Cs
```

```
      write(*,*)' Enter Wind Velocity (m/s) and Direction (degrees)'
      read(*,*) wind_vel,wind_dir

      write(*,*)' Enter Dispersive Mixing Mode'
      write(*,*)'   (0) Depth Uniform '
      write(*,*)'   (1) Quasi 3-D'
      read(*,*) disp3d

      write(10,nml=boundary)
      write(10,nml=grids)
      write(10,nml=short_wave)
      write(10,nml=physics)
      write(10,nml=control)
      write(10,nml=sensor)
      write(10,nml=output)

      close(10)

c*************************************************************************
c     CREATE indat.dat FOR REF/DIF  taken from indat-createv25.f
c*************************************************************************

      write(*,*)'*******************************'
      write(*,*)'Generating indat.dat for Ref/Dif'
      write(*,*)'*******************************'
      write(*,*)''
      write(*,*)'** Refer questions to the Ref/Dif manual **'

      call infile(fnamein)
      open(unit=10,file=fnamein)

c file names generated automatically

      FNAME1 = "'refdat.dat'"
      FNAME2 ="'outdat.dat'"
      FNAME3 = "'subdat.dat'"
      FNAME4 = "'wave.dat'"
      FNAME5 = "'owave.dat'"
      FNAME6 = "'surface.dat'"
      FNAME7 ="'bottomu.dat'"
      FNAME8 ="'angle.dat'"
      FNAME9 ="' '"
```

```
      FNAME10 ="'refdif1.log'"
      FNAME11 = "'height.dat'"
      FNAME12 = "'sxx.dat'"
      FNAME13 ="'sxy.dat'"
      FNAME14 = "'syy.dat'"
      FNAME15 ="'depth.dat'"
      write(10,nml=fnames)

c  Enter control data.
      mr=nx
      nr=ny
      dxr=dx
      dyr=dy

      ismooth=0

      write(*,*)' enter  depth tolerance dt'
      read(*,*)  dt

      write(*,*)' input iu: 1=mks, 2=english'
      read(*,*) iu

      write(*,*)' input dispersion relationship; ntype: 0=linear,'
      write(*,*)'                              1=composite, 2=stokes'
      read(*,*)ntype

      if (ibc3.eq.4) then
         ibc=0
      else
         ibc=1
      end if

      write(*,*)' input ispace (0=program picks x spacing,',
     1'  1=user chooses)'
      read(*,*) ispace

      write(*,*)' input nd (# y divisions, 1 is minimum)'
      read (*,*) nd

      if(ispace.eq.0) go to 105

      write(*,*)' constant or variable x spacing?(0 for constant)'
      read(*,*) ians1
```

```
      if(ians1.eq.0) then
      write(*,*) ' input constant md'
      read(*,*) mdc
      do 103 iko=1,mr-1
      md(iko)=mdc
  103 continue
      else
      write(*,104)
  104 format(' input md(i) for i=1 to mr-1')
      read(*,*) (md(i),i=1,mr-1)
      endif

  105 write(*,106)
  106 format(' input if(1) turbulent, if(2) porous, if(3) laminar')
      write(*,*) '  standard  choice:  1, 0, 0'
      read(*,*) iff(1), iff(2), iff(3)

      write(*,*)' input isp (subgrid features) :standard  0'
      read(*,*) isp

      write(*,108)
  108 format(' input values of iinput, ioutput:'/
     1' iinput: 1 standard, i.e., not starting from previous run'/
     1'         2 if starting from previous run'/
     1' ioutput: 1 standard, not saving restart data'/
     1'          2 if saving restart data')
      read(*,*)iinput,ioutput

      write(*,115)
  115 format(' input value of isurface:'/
     1'  isurface = 0:   no surface picture generated'/
     1'  isurface = 1:   surface picture generated')
      read(*,*) isurface

      if(isurface.eq.0) fname6 = ' '

      write(10,nml=ingrid)

      if(ispace.eq.1) write(10,nml=inmd)

      if(iinput.eq.1) then
c*--------------------------------------------------------------------
```

```
c*    write waves1 portion of indat.dat
c*--------------------------------------------------------------------
      nfreqs=1
      iwave=1
      write(*,*)'Enter the value desired for kappa'
      write(*,*)'kappa = 0.78  Normally'
      read(*,*)kapp
       write(*,*)'Enter the value desired for gamma'
      write(*,*)'gamma = 0.4  Normally'
      read(*,*)gamm


      write(10, nml=waves1a)

      do 113 ifreq=1,nfreqs
c*--------------------------------------------------------------------
c*    line 10, iinput=1
c*--------------------------------------------------------------------
c      write(*,109)
c 109  format(' input tide stage')
c      read(*,*)  tide(ifreq)
      tide(ifreq)=0
      freqs(ifreq)=period
c*--------------------------------------------------------------------
c*    line 11, iwave=1, iinput=1
c*--------------------------------------------------------------------
      if(iwave.eq.1) then

      write(*,110)
 110  format(' input # of waves per frequency, nwavs')
      read(*,*) nwavs(ifreq)

c*--------------------------------------------------------------------
c*    line 12, iwave=1, iinput=1
c*--------------------------------------------------------------------
      do 111 iwavs=1,nwavs(ifreq)
      write(*,*)' input amplitude and direction'
      read(*,*) amp(ifreq,iwavs), dir(ifreq,iwavs)

 111  continue

      else
```

```
c*--------------------------------------------------------------------
c*    iwave=2, iinput=1
c*--------------------------------------------------------------------
      write(*,112)
 112  format('input en. density and on next line, directional',
     1'  spreading factor')
      read(*,*) edens(ifreq)
      read(*,*) nwavs(ifreq)

      nseed=500

      endif
 113  continue

      if (iwave .eq. 1) write(10, nml=waves1b)
      if (iwave .eq. 2) write(10, nml=waves1c)

      endif

      if ( iinput .eq. 2) then
c*--------------------------------------------------------------------
c*    line 9, iinput=2
c*--------------------------------------------------------------------
      write(*,*)' input wave period'
      read(*,*) freqin
      tidein=0

      write(10, nml=waves2)

      endif

      close(10)

       write(*,*)' Input can be changed by editing input_winc.dat'
       write(*,*)' and indat.dat directly.'


      stop
      end
```

8.9 File *create_bath.f*

This file contains the program used to generate the bathymetry files *bath.dat* and *dim.dat*.

```
c**************************************************************************
c*-------------------------------------------------------------------------
c*    create_bath.f
c*
c*    Used to create bath.dat for various bathymetries
c*
c*    Last revision Feb 7, 2002.
c*    Kevin A Haas
c*-------------------------------------------------------------------------
c**************************************************************************
      program createbath

      include 'winc_std_common.inc'

      integer bathtype,ans,bar,channel,numchan,xtoe
      real*8 Lx, Ly, slpe,xc,wb,hc,ht2(500,500),chanpos(500),
     .   chanwid(500)  ,lxflat,depthflat,A,n
c********* Begin the Program ***********************************************
      write(*,*)''
      write(*,*)'*********************************'
      write(*,*)'Creating Bathymetry for Shorecirc'
      write(*,*)'*********************************'
      open(unit=10,file='bath.dat')
500   continue
c********* Ask the type of bathymetry **************************************
      write(*,*)''
      write(*,*)'Enter type of bathymetry'
      write(*,*)'  (1) plane beach'
      write(*,*)'  (2) closed basin (flat bottom/ plane beach)'
      write(*,*)'  (3) equilibrium profile'
      read (*,*)bathtype
c********* Check for valid answer ******************************************
      if (bathtype.eq.1) then
         write(*,*)'Creating a Plane beach'
      else if (bathtype.eq.2) then
         write(*,*)'Creating a closed basin'
      else if (bathtype.eq.3) then
         write(*,*)'Creating an equilibrium profile '
```

```
      else
       write(*,*)'  Bad Choice, Try Again'
       goto 500
      end if
c********* Find length of domain ***********************************************
501   continue
      write(*,*)''
      write(*,*)'Enter the length of domain Lx, Ly (m)'
      read (*,*)Lx,Ly
c********* Check for valid answer **********************************************
      if (Lx.le.0.or.Ly.le.0) then
       write(*,*)'Invalid Dimensions Try Again'
       goto 501
      end if
c********* Find number of grid points ******************************************
502   continue
      write(*,*)''
      write(*,*)'Enter the nx and ny'
      read (*,*)nx,ny
c********* Check for valid answer **********************************************
      if (nx.le.0.or.ny.le.0) then
       write(*,*)'Invalid Number of grid points, Try Again'
       goto 502
      end if
c********* Calculate dx and dy *************************************************
      dx=Lx/(nx-1)
      dy=Ly/(ny-1)
c********* Ask if dx and dy are OK   *******************************************
503   continue
      write(*,*)''
      write(*,*)'dx = ',dx,'  dy = ',dy
      write(*,*)'Is this OK?'
      write(*,*)'  (1) yes'
      write(*,*)'  (0) no'
      read (*,*)ans
c********* Check for valid answer.  If OK then calculate x and y arrays ********
      if (ans.eq.0) then
         write(*,*)' Then Try again'
         goto 502
      else if (ans.eq.1) then
         do i=1,nx
            x(i)=(i-1)*dx
         end do
```

```
         do j=1,ny
            y(j)=(j-1)*dy
         end do
         write(*,*)' Continuing'
      else
         write(*,*)'Invalid answer'
         goto 503
      end if

c********* Ask for depth at shoreline ****************************************
      write(*,*)''
      write(*,*)'Enter the depth at the shoreline (m)'
      read(*,*)depthmin
c********* Ask for slope for plane beach *************************************
       if (bathtype.eq.1) then
504   continue
      write(*,*)''
      write(*,*)'Enter the slope as a positive decimal'
      write(*,*)'     ie 1/10 is 0.1 '
      read(*,*)slpe
c********* Check for valid answer ********************************************
      if (slpe.lt.0) then
       write(*,*)'Negative Slope Try Again'
       goto 504
      end if
c********* Calculate bathymetry for plane beach ******************************
      do i=1,nx
         do j=1,ny
            ht(i,j)=slpe*Lx+depthmin-x(i)*slpe
         end do
      end do
      end if

c********* Ask for slope for closed basin ************************************
       if (bathtype.eq.2) then
604   continue
      write(*,*)''
      write(*,*)'Enter the slope as a positive decimal'
      write(*,*)'     ie 1/10 is 0.1 '
      read(*,*)slpe
c********* Check for valid answer ********************************************
      if (slpe.lt.0) then
       write(*,*)'Negative Slope Try Again'
```

```
       goto 604
      end if
c********* Ask for length of flat ****************************************
 605  continue
      write(*,*)''
      write(*,*)'Enter length and depth of flat portion (m)'
      write(*,*)''
      read(*,*)lxflat,depthflat
c********* Check for valid answer ****************************************
      if (lxflat.le.0) then
       write(*,*)'Negative distance'
       goto 605
      end if
      if (depthflat.le.0) then
       write(*,*)'Negative depth'
       goto 605
      end if
c********* Calculate bathymetry for closed basin *****************************
      xtoe=int(lxflat/dx)+1
      do i=1,xtoe
         do j=1,ny
            ht(i,j)=depthflat
         end do
      end do
      do i=xtoe+1,nx
         do j=1,ny
            ht(i,j)=depthflat-(x(i)-x(xtoe))*slpe
         end do
      end do
      end if
c********* equilibrium profile ******************************
       if (bathtype.eq.3) then
       write(*,*)''
      write(*,*)'Enter A, n for equation h=Ax^(n)'
      write(*,*)''
      read(*,*)A,n
c********* Calculate bathymetry for equilibrium profile  ***************
      do j=1,ny
         do i=1,nx
            ht(i,j)=A*(x(nx)-x(i))**n+depthmin
         end do
      end do
```

```
          end if

c********* Ask about a bar ***************************************************
506   continue
      write(*,*)'Do you want a bar?'
      write(*,*)'  (1) yes'
      write(*,*)'  (0) no'
      read (*,*)bar
c********* Check for valid answer.  If OK ask for location *******************
      if (bar.eq.1) then
507     continue
        write(*,*)'x location for the bar (m) (x=0 offshore)?'
        read(*,*)xc
c********* Check for valid answer ********************************************
      if (xc.lt.0.or.xc.gt.x(nx)) then
       write(*,*)'Invalid location Try Again'
       goto 507
      end if
c********* Ask for Bar width and height **************************************
508     continue
        write(*,*)'Enter bar width and height (m)'
        read(*,*)wb,hc
      if (wb.le.0.or.hc.le.0) then
       write(*,*)'Invalid width or height'
       goto 508
      end if
c********* Ask if wants a channel ********************************************
 510  continue
      write(*,*)'Do you want to add channels?'
      write(*,*)'  (1) yes'
      write(*,*)'  (0) no'
      read (*,*)channel
c********* Add the bar only to the bathymetry ********************************
      do j=1,ny
         do i=1,nx
  ht2(i,j)=-(exp(log(.05)*16/wb**4*(dx*(i-1)-xc)**4)*hc)
         end do
      end do
c********* Add channels ******************************************************
      if (channel.eq.1) then
 511     continue
         write(*,*)'Enter the number of channels desired.'
         read(*,*)numchan
```

```
         if (numchan.le.0) then
            write(*,*)'Invalid number of channels'
            goto 511
         else
         do m=1,numchan
 512        continue
            write(*,*)'Enter y position for channel (m) ',m
            read(*,*)chanpos(m)
            if (chanpos(m).lt.0.or.chanpos(m).gt.y(ny)) then
               write(*,*)'Invalid channel position'
               goto 512
            end if
 513        continue
            write(*,*)'Enter width for channel (m) ',m
            read(*,*)chanwid(m)
            if (chanwid(m).lt.0.or.chanwid(m).gt.y(ny)) then
               write(*,*)'Invalid channel width'
               goto 513
            end if
c********* calculate channels  ***********************************************
            do i=1,nx
               do j=1,ny
                  ht2(i,j)=-ht2(i,j)*(exp(log(.05)*16/chanwid(m)**4
     .              *((dy*(j-1)-chanpos(m))**4))-1)
               end do
            end do
         end do
         end if
c********* Invalid response to  channels query  ******************************
      else if (channel.ne.0) then
         write(*,*)'Invalid response'
         goto 510
      end if
c********* Calculate total bathymetry ****************************************
      do i=1,nx
         do j=1,ny
           ht(i,j)=ht(i,j)+ht2(i,j)
         end do
      end do
c********* Invalid response to bar query  ************************************
      else if (bar.ne.0) then
         write(*,*)'Invalid response'
         goto 506
```

```
      end if
c********* write parameters to screen and check to see if all OK **************
      write(*,*)''
      write(*,*)'********************************'
      write(*,*)'Final parameters'
      write(*,*)'********************************'
      write(*,*)''
      if (bathtype.eq.1) then
      write(*,*)'Plane beach'
      write(*,*)'Slope = ',slpe
      else if (bathtype.eq.2) then
      write(*,*)'Closed Basin'
      write(*,*)'Width of flat portion',lxflat,' (m)'
      write(*,*)'Depth of flat portion',depthflat,' (m)'
      write(*,*)'Slope = ',slpe
      else if (bathtype.eq.3) then
      write(*,*)'Equilibrium Profile'
      write(*,*)'h = ',A,' x^',n
      end if
      write(*,*)'Offshore depth is  ',ht(1,1),' (m)'
      write(*,*)'Shore depth is  ',ht(nx,1),' (m)'
      write(*,*)'dx = ',dx,'  dy = ',dy
      write(*,*)'Lx = ',Lx,'  Ly = ',Ly
      write(*,*)'nx = ',nx,'  ny = ',ny
      if (bar.eq.1) then
         write(*,*)'Bar position = ',xc
         write(*,*)'Bar height = ',hc,'  Bar width = ',wb
         if (channel.eq.1) then
            do m=1,numchan
               write(*,*)'Channel ',m,' located at y = ',chanpos(m)
               write(*,*)'Channel ',m,' width = ',chanwid(m)
            end do
         end if
      end if
      write(*,*)''
c********* Ask if OK ***********************************************************
505   continue
      write(*,*)'Is this OK?'
      write(*,*)'  (1) yes'
      write(*,*)'  (0) no'
      read (*,*)ans
c********* Check for valid answer.  If OK create bath.dat **********************
      if (ans.eq.0) then
```

```
         write(*,*)' Start Over'
         goto 500
      else if (ans.eq.1) then
         write(*,*)'Writing the bathymetry to bath.dat'
         do j=1,ny
  do i=1,nx
    write(10,*)x(i),y(j),ht(i,j)
  end do
 end do
      else
         write(*,*)'Invalid Response'
         goto 505
      end if
      close(10)
c********* Writes dimensions to dim.dat for use in createdat ******************
      open(unit=10,file='dim.dat')
      write(*,*)'Writing dimensions to dim.dat'
      write(10,*)nx,ny,dx,dy,depthmin
      close(10)

c*********  End of Program ***************************************************
      stop
      end
```