Refraction and Diffraction of Nonlinear Waves: Green-Naghdi Equations

by

H. Sundararaghavan
Graduate Research Assistant

R.C. Ertekin
Professor & Principal Investigator

Department of Ocean and Resources Engineering
School of Ocean and Earth Science and Technology
University of Hawaii at Manoa
2540 Dole Street, Holmes Hall 402
Honolulu, HI 96822

OE Report No.: UHMOE - 99-207
Sea Grant Report No.: UNIHI-SEAGRANT-TR-99-03

December, 1999

Final Report submitted to the University of Hawaii Sea Grant College Program
Project No. R/EP-2 Year 28-29.

Acknowledgement

This publication is funded in part by a grant/cooperative agreement from the National Oceanic and Atmospheric Administration, Project No. R/EP-2 (Yr. 28-29) which is sponsored by the University of Hawaii Sea Grant College Program, SOEST, under Institutional Grant No. NA36RG0507 from NOAA Office of Sea Grant, Department of Commerce. The views expressed herein are those of the authors and do not necessarily reflect the views of NOAA or any of its sub-agencies. Report No. UNIHI-SEAGRANT-TR-99-03. Matching funds are received from the State of Hawaii, Department of Civil Defense. We are thankful to those agencies for their support. We gratefully acknowledge the discussions that we have had with Profs. Janet Becker and Mark Merrifield who were the Co-PIs of this and a related Sea Grant Project. Drs. Chuck Helsley and Rose Pfund of the University of Hawaii's Sea Grant Program have continually encouraged and supported us. Mahalo Nui.

Abstract

The Green-Naghdi equations are shallow water equations which can be solved to predict the nonlinear-effects of water waves propagating in shallow waters. In this study, we derive the Green-Naghdi equations for variable bathymetry and formulate the numerical model as a boundary-value problem. A finite-difference model, in conjunction with grid-generation, is developed to solve the Green-Naghdi equations. Non-linear wave propagation over a varying bathymetry is presented for solitary and cnoidal waves.

Table of Contents

List of Figures

1 Introduction

In the field of wave modeling, various theories can be adopted to predict shallow-water wave propagation. In the past, researchers have used theories based on equations such as the Boussinesq and KdV equations (see for example, Schember, 1982[8]) to successfully model shallow water waves in a variety of problems. In recent years, the Green-Nagdhi equations, originally developed by Green and Naghdi (1977)[4] to study deformable fluid sheets, have been successfully employed to simulate nonlinear shallow water waves (see for example, Ertekin, 1984[1], Qian, 1994[7] and Neill, 1996[6]). In this report, we present the theoretical formulation and numerical implementation of shallow water wave modeling using the Green-Naghdi equations in the presence of variable bathymetry. The numerical model is based on the finite-difference method in conjunction with numerical grid generation. A few test cases are presented and the results are compared with the results obtained with Boussinesq approximations.

2 Governing equations

In presenting the theoretical formulations, we adopt indicial notation (with implied Einstein's summation convention) unless otherwise noted. The derivatives are denoted by using the comma convention (see for example Mase, 1970[5]). The Green-Naghdi equations (see e.g. Green and Naghdi, 1977[4]) in the form of continuity and momentum equations are presented in this section. The subscripts, i, j, and k are used in these equations for the indicial notation. These subscripts assume values of 1 and 2, unless otherwise noted, to denote the two independent horizontal directions, i.e., 1 represents the longitudinal coordinate and 2 represents the transverse coordinate.

2.1 Continuity equation

The continuity equation representing mass conservation is given by (see Ertekin, 1984[1])

$$\eta_{,t} + (h+\eta)u_{i,i} + u_i(h_{,i} + \eta_{,i}) = 0. \tag{1}$$

2.2 Momentum equation

The equation of motion representing balance of momentum is given by (see Ertekin, 1984[1])

$$Du_i + g\eta_{,i} + \frac{1}{6}\left[-D^2h(2\eta_{,i} - h_{,i}) + (4\eta_{,i} + h_{,i})D^2\eta + (h+\eta)(2D^2\eta_{,i} - D^2h_{,i})\right] = 0 \quad (2)$$

where

$h = h(x_1, x_2, t)$ is the water depth,

$u_i = (u_1(x_1, x_2, t), u_2(x_1, x_2, t))$ is the velocity vector with components u_1 and u_2 along the x_1 and x_2 directions, respectively,

$\eta = \eta(x_1, x_2, t)$ is the surface elevation,

g is the gravitational acceleration,

D, the total derivative operator, is given by

$$D(\,) = (\,)_{,t} + u_j(\,)_{,j} \quad (3)$$

and, D^2, the second total derivative operator, is given by

$$D^2(\,) = (\,)_{,tt} + u_{j,t}(\,)_{,j} + 2u_j(\,)_{,jt} + u_ju_{k,j}(\,)_{,k} + u_ju_k(\,)_{,jk}. \quad (4)$$

It can be observed that the momentum equation, Eq. (2), includes terms involving Du_i, D^2h and $D^2\eta$ which are the total derivatives. These terms can be expanded using Eqs. (3) and (4).

Expansion of Du_i: Using Eq. (3), Du_i is expanded as

$$Du_i = u_{i,t} + u_ju_{i,j}. \quad (5)$$

Expansion of D^2h: Using Eq. (4), the second total derivative of the water depth, h, is given by

$$D^2h = h_{,tt} + u_{j,t}h_{,j} + 2u_jh_{,jt} + u_ju_{k,j}h_{,k} + u_ju_kh_{,jk}. \quad (6)$$

Expansion of $D^2\eta$: The second total derivative of the surface elevation, η, can be simplified by using Eq. (3) and the continuity equation, Eq. (1). Using Eq. (3) and substituting $\eta_{,t} = -(h+\eta)u_{j,j} - u_j(h_{,j}+\eta_{,j})$, we get

$$\begin{aligned} D\eta &= \eta_t + u_j\eta_{,j} \\ &= -(h+\eta)u_{j,j} - u_j(h_{,j}+\eta_{,j}) + u_j\eta_{,j} \\ &= -(h+\eta)u_{j,j} - u_j h_{,j}. \end{aligned} \tag{7}$$

From Eq. (3), it can be seen that $Dh = h_{,t} + u_j h_{,j}$. Using this in Eq. (7) gives

$$D\eta = -(h+\eta)u_{j,j} + h_{,t} - Dh. \tag{8}$$

Applying the total derivative operator, D, once again on Eq. (8), we get

$$D^2\eta = \left[-(h+\eta)u_{j,j} + h_{,t}\right]_{,t} + u_k\left[-(h+\eta)u_{j,j} + h_{,t}\right]_{,k} - D^2h. \tag{9}$$

Simplifying Eq. (9) gives

$$\begin{aligned} D^2\eta = &-(h+\eta)u_{j,jt} - h_{,t}u_{j,j} - \eta_{,t}u_{j,j} + h_{,tt} - u_k(h+\eta)u_{j,jk} \\ &- u_k(h_{,k}+\eta_{,k})u_{j,j} + u_k h_{,kt} - D^2h. \end{aligned} \tag{10}$$

Substituting $\eta_{,t} = -(h+\eta)u_{k,k} - u_k(h_{,k}+\eta_{,k})$ in Eq. (10) gives

$$\begin{aligned} D^2\eta = &-(h+\eta)u_{j,jt} - h_{,t}u_{j,j} + (h+\eta)u_{j,j}u_{k,k} + u_k(h_{,k}+\eta_{,k})u_{j,j} + h_{,tt} \\ &- u_k(h+\eta)u_{j,jk} - u_k(h_{,k}+\eta_{,k})u_{j,j} + u_k h_{,kt} - D^2h. \end{aligned} \tag{11}$$

Simplifying Eq. (11) gives

$$D^2\eta = (h+\eta)\left(-u_{j,jt} + u_{j,j}u_{k,k} - u_k u_{j,jk}\right) + u_k h_{,kt} + h_{,tt} - h_{,t}u_{j,j} - D^2h. \tag{12}$$

It may be noted that the expansion of $D^2\eta$ in Eq. (12) does not contain the time derivatives of η which would be present if Eq. (4) is used to obtain the expansion. This absence of the time derivatives of η aids in the numerical time-stepping scheme discussed later in Section 4.

Expansion of $D^2h_{,i}$: The $D^2h_{,i}$ term in Eq. (2) is expanded using Eq. (4) as

$$\begin{aligned} D^2h_{,i} = &\, h_{,itt} + u_{j,it}h_{,j} + u_{j,t}h_{,ij} + 2u_{j,i}h_{,jt} + 2u_j h_{,ijt} \\ &+ u_{j,i}u_{k,j}h_{,k} + u_j u_{k,ij}h_{,k} + u_j u_{k,j}h_{,ik} + u_{j,i}u_k h_{,jk} + u_j u_{k,i}h_{,jk} + u_j u_k h_{,ijk}. \end{aligned} \tag{13}$$

Expansion of $D^2\eta_{,i}$:

$$\begin{aligned} D^2\eta_{,i} =&(h_{,i}+\eta_{,i})\left(-u_{j,jt}+u_{j,j}u_{k,k}-u_k u_{j,jk}\right) \\ &+(h+\eta)\left(-u_{j,ijt}+u_{j,ij}u_{k,k}+u_{j,j}u_{k,ik}-u_{k,i}u_{j,jk}-u_k u_{j,ijk}\right) \\ &+u_{k,i}h_{,kt}+u_k h_{,ikt}+h_{,itt}-h_{,t}u_{j,ij}-h_{,it}u_{j,j}-D^2 h_{,i}. \end{aligned} \tag{14}$$

Thus, Eqs. (1)-(2), (5)-(6) and (12)-(14) represent the complete set of governing equations.

3 Boundary conditions

In this section, the boundary conditions along the boundaries that surround the domain are presented. These boundary conditions along with the governing equations provided in Section 2, complete the definition of the boundary value problem which can be solved using numerical methods such as the finite-difference method.

3.1 Entrance Boundary

The entrance boundary is the boundary through which the wave enters. In the present model, it will be assumed that the boundary of the domain will be chosen in such a way that the wave enters perpendicular to this entrance boundary. At this boundary, the values of the surface elevation and the velocity are input. Therefore, the boundary conditions are given by

$$\begin{aligned} &\eta(x_i,t)=\eta_{(e)}(t),\ \ x_i\in S_{(e)}, \\ &u_i(x_i,t)=u_{i_{(e)}}(t),\ \ x_i\in S_{(e)}, \end{aligned} \tag{15}$$

where $\eta_{(e)}(t)$ and $u_{i_{(e)}}(t)$ are the surface elevation and particle velocity of the incoming wave and S_e at the entrance boundary.

3.2 Lateral Boundary

At the lateral boundaries on the left and the right sides of the incoming wave direction, two types of boundary conditions may be considered. One is the wall condition were the assumption is that this boundary acts as a wall thus reflecting all the waves back into the

domain and not permitting any wave to radiate out of this boundary. This condition can be described and implemented, theoretically as well as numerically, very accurately. However, this may not represent the actual conditions that occur in open oceans. A more appropriate condition would be a radiation condition which permits the waves to pass through the boundary. Such a condition is difficult to obtain theoretically without making any further assumptions. In this study, we adopt a radiation condition based on the geometric principles and implement it numerically.

Wall condition

The wall condition is given by the fact there is no flow across the boundary. This is written as

$$u_i n_i^{(l)} = 0 \tag{16}$$

where $n_i^{(l)}$ represents the unit normal to the lateral boundary.

Radiation condition

To obtain the radiation condition, we make the assumption that the velocity components and the surface elevation vary uniformly near this boundary. Such an assumption will be valid only when the variation in the bathymetry is minimal resulting in very little refraction and diffraction of waves. If the boundary does not satisfy these requirements, it is essential that the boundary be moved to a location where this would be valid. Another option would be to move the boundary farther by increasing the domain size and forcing the bathymetry to be flatter by artificially changing the bathymetry near the lateral boundary. It may be noted that increasing the domain size poses a disadvantage with increased number of computations to be performed for the simulation. Thus, the decision on which boundary condition must be chosen must be made considering the specific environmental data for the particular site and the capability of the computer on which the simulation is performed.

To implement the radiation condition based on the assumption that the velocity and surface elevation vary smoothly near the lateral boundary, we use a second-order extrapolation of the values on the finite-difference grid to obtain the values at an imaginary grid

point outside the domain. The determination of this value permits the application of the governing equation which can be used to obtain the solutions at this boundary.

3.3 Open boundary

At the open boundary, the waves must be allowed to radiate outside the domain. The Orlanski radiation condition is applied at this boundary. The Orlanski condition is given by

$$\begin{aligned} \eta_{,t} &= c\frac{\partial \eta}{\partial \boldsymbol{n}_{(w)}}, \ x_i \in S_{(o)} \\ u_{i,t} &= c\frac{\partial u_i}{\partial n_{j(w)}}\delta_{ij}, \ x_i \in S_{(o)} \end{aligned} \tag{17}$$

where $c = \sqrt{gh}$ is the wave celerity, $S_{(o)}$ denotes the open boundary, $\boldsymbol{n}_{(w)} = (n_{1(w)}, n_{2(w)})$ represents the unit normal to the outgoing wave crest and δ_{ij} is the Kronecker's delta defined by

$$\begin{aligned} \delta_{ij} &= 0 \text{ when } i \neq j \\ &= 1 \text{ when } i = j. \end{aligned} \tag{18}$$

Equation (17) can also be written as

$$\begin{aligned} \eta_{,t} &= cn_{i(w)}\eta_{,i}, \ x_i \in S_{(o)}, \\ u_{i,t} &= cu_{i,j}n_{j(w)} \ x_i \in S_{(o)}. \end{aligned} \tag{19}$$

Under the assumption of a monochromatic wave, it can be shown that the unit normal to the wave crest (see for example, Yang and Ertekin, 1991[9]) can be written as

$$n_{i(w)} = \pm\frac{\eta_{,i}}{\sqrt{\eta_{,j}\eta_{,j}}} \tag{20}$$

where $\pm$ represents the fact that the wave crest normal must be pointing in the direction of wave propagation and not in the opposite direction.

3.4 Coastal Boundary

At the coastal boundary, the velocity normal to the boundary must be specified as zero. This can be written as

$$u_i n_i = 0, \ x_i \in S_{(c)}, \tag{21}$$

where n_i denotes the unit normal to the coastal boundary, $S_{(c)}$.

4 Governing equations for time-stepping

To facilitate the time-stepping of the governing equations, Eqs. (1) and (2), it is necessary to separate the terms involving the time derivatives and the rest. In this section, the continuity and momentum equations, Eq. (1) and Eq. (2), respectively, will be rewritten by gathering the terms involving the time-derivatives and separating the rest. The details of the numerical treatment of the time and spatial derivatives in the governing equations is presented in Section 5.

4.1 Continuity equation

The continuity equation, Eq. (1), can be rewritten by moving the spatially dependent terms to the right hand side and introducing a variable, $Q_{[\eta]}$, so that

$$\eta_{,t} = Q_{[\eta]} \tag{22}$$

and

$$Q_{[\eta]} = -[(h+\eta)u_{i,i} + u_i(h_{,i} + \eta_{,i})]. \tag{23}$$

4.2 Momentum equation

In Section 1, an expansion of the term $D^2\eta$ was performed using the continuity equation, Eq. (1), so that the time derivatives of η are eliminated by representing η in terms of $-[(h+\eta)u_i]_{,i}$. After performing this substitution, the momentum equation, Eq. (2), contains only terms with first derivatives of velocity, u_i, and terms with only spatial derivatives. Using Eqs. (2), (5) and (12), the momentum equation is written as

$$\begin{aligned} u_{i,t} + u_j u_{i,j} + g\eta_{,i} + \frac{1}{6}\{ & -D^2h(2\eta_{,i} - h_{,i}) + (4\eta_{,i} + h_{,i})[(h+\eta) \\ & (-u_{j,jt} + u_{j,j}u_{k,k} - u_k u_{j,jk}) + u_k h_{,kt} + h_{,tt} - h_{,t}u_{j,j} - D^2h] \\ & +(h+\eta)(2D^2\eta_{,i} - D^2h_{,i})\} = 0. \end{aligned} \tag{24}$$

In Eq. (24), it may be noted that the terms containing D^2h, $D^2\eta_{,i}$ and $D^2h_{,i}$, expanded in Eqs. (6), (13) and (14), respectively, contain time derivatives of velocity, u_i. It is necessary to distinguish the terms containing time derivatives from the rest so that a time-stepping

scheme such as the Modified-Euler method can be implemented. To aid in the separation of terms with time derivatives, and for simplicity, we rewrite Eqs. (6), (13) and (14) in more convenient forms as shown in Eqs. (25), (27) and (28) below:

$$D^2h = h_{,j}u_{j,t} + T_1 \tag{25}$$

where

$$T_1 = h_{,tt} + 2u_j h_{,jt} + u_j u_{k,j} h_{,k} + u_j u_k h_{,jk}, \tag{26}$$

$$D^2 h_{,i} = u_{j,it} h_{,j} + u_{j,t} h_{,ij} + T_{[2]i} \tag{27}$$

where

$$\begin{aligned} T_{[2]i} = {} & h_{,itt} + 2u_{j,i}h_{,jt} + 2u_j h_{,ijt} + u_{j,i}u_{k,j}h_{,k} + u_j u_{k,ij}h_{,k} + u_j u_{k,j} \\ & h_{,ik} + u_{j,i}u_k h_{,jk} + u_j u_{k,i} h_{,jk} + u_j u_k h_{,ijk}, \end{aligned} \tag{28}$$

and

$$D^2\eta_{,i} = -(h_{,i} + \eta_{,i})u_{j,jt} - (h+\eta)u_{j,ijt} + T_{[3]i} - D^2 h_{,i} \tag{29}$$

where

$$\begin{aligned} T_{[3]i} = {} & (h_{,i} + \eta_{,i})\left(u_{j,j}u_{k,k} - u_k u_{j,jk}\right) \\ & + (h+\eta)\left(u_{j,ij}u_{k,k} + u_{j,j}u_{k,ik} - u_{k,i}u_{j,jk} - u_k u_{j,ijk}\right) \\ & + u_{k,i}h_{,kt} + u_k h_{,ikt} + h_{,itt} - h_{,t}u_{j,ij} - h_{,it}u_{j,j}. \end{aligned} \tag{30}$$

To handle the terms in Eq. (24) that do not involve D^h, D_i^h and D_i^η , we introduce a variable, $T_{[4]i}$, such that

$$\begin{aligned} T_{[4]i} = {} & u_j u_{i,j} + g\eta_{,i} + \frac{1}{6}\left\{(4\eta_{,i} + h_{,i})[(h+\eta)(u_{j,j}u_{k,k} - u_k u_{j,jk}) \right. \\ & \left. + u_k h_{,kt} + h_{,tt} - h_{,t}u_{j,j} - T_1]\right\}. \end{aligned} \tag{31}$$

And using this definition of $T_{[4]i}$ along with Eq. (25), we can rewrite Eq. (24) as

$$\begin{aligned} & u_{i,t} - \frac{1}{6}(4\eta_{,i} + h_{,i})[(h+\eta)[u_{j,jt} + u_{j,t}h_{,j}] - \frac{1}{6}\left[D^2 h(2\eta_{,i} - h_{,i})\right] + T_{[4]i} \\ & + \frac{1}{6}(h+\eta)(2D^2\eta_{,i} - D^2 h_{,i}) = 0. \end{aligned} \tag{32}$$

Using Eqs. (25), (27) and (29) in Eq. (32), we get

$$\begin{aligned} & u_{i,t} - \frac{1}{6}(4\eta_{,i} + h_{,i})[(h+\eta)u_{j,jt} + u_{j,t}h_{,j}] - \frac{1}{6}(2\eta_{,i} - h_{,i})(u_{j,t}h_{,j} + T_1) \\ & \quad + T_{[4]i} + \frac{1}{6}(h+\eta)[-2(h_{,i} + \eta_{,i})u_{j,jt} - 2(h+\eta)u_{j,ijt} \\ & \quad + 2T_{[3]i} - 3(u_{j,it}h_{,j} + u_{j,t}h_{,ij} + T_{[2]i})] = 0. \end{aligned} \tag{33}$$

Rearranging the terms in Eq. (33) and simplifying, we get

$$\begin{aligned} & u_{i,t} - (h+\eta)\left(\eta_{,i} + \frac{1}{2}h_{,i}\right)u_{j,jt} - \frac{1}{3}(h+\eta)^2 u_{j,ijt} \\ & \quad - \frac{1}{2}(h+\eta)(u_{j,it}h_{,j} + u_{j,t}h_{,ij}) - \frac{1}{6}(2\eta_{,i} - h_{,i})T_1 - \eta_{,i}h_{,j}u_{j,t} \\ & \quad + T_{[4]i} + \frac{1}{6}(h+\eta)(2T_{[3]i} - 3T_{[2]i}) = 0. \end{aligned} \tag{34}$$

For simplicity, we introduce the variable, $Q_{[u]i}$, such that

$$Q_{[u]i} = -\left[T_{[4]i} + \frac{1}{6}(h+\eta)(2T_{[3]i} - 3T_{[2]i}) - \frac{1}{6}(2\eta_{,i} - h_{,i})T_1\right], \tag{35}$$

so that

$$\begin{aligned} & u_{i,t} - \eta_{,i}h_{,j}u_{j,t} \\ & \quad - (h+\eta)\left[\left(\eta_{,i} + \frac{h_{,i}}{2}\right)u_{j,jt} + \frac{1}{3}(h+\eta)u_{j,ijt} + \frac{1}{2}(u_{j,it}h_{,j} + u_{j,t}h_{,ij})\right] - Q_{[u]i} = 0 \end{aligned} \tag{36}$$

By the introduction of four variables, T_1, $T_{[2]i}$, $T_{[3]i}$ and $T_{[4]i}$, for convenience, the equations of motion, Eq. (23), is rewritten as Eq. (36), where the terms containing the time derivatives of velocity, u_i, are separated from the terms involving spatial derivatives only. The terms that involve spatial derivatives are represented by the variable, $Q_{[u]i}$, defined in Eq. (34). This separation of the terms results in Eq. (36) which will be used to apply the numerical time-stepping schemes as discussed in Section 5.

5 Numerical time-stepping

The continuity and momentum equations are given by Eqs. (22) and (36). These equations are solved using the finite-difference method spatially and the Modified-Euler method for

the time integration. The superscripts (1), (m) and (2) will be used to represent the initial, intermediate and final time levels of the Modified Euler method, respectively. Thus, at any time step, given the initial values $\eta^{(1)}$ and $u^{(1)}$ at time step t, the values for the next time step, Δt, $\eta^{(2)}$ and $u^{(2)}$, are determined using the Modified Euler method. The initial values, $\eta^{(1)}$ and $u^{(1)}$ are used to determine $Q^{(1)}_{[u]i}$ and $R^{(1)}_{\eta}$ from the equations mentioned above. These values are used to determine the intermediate values of $\eta^{(m)}$ and $u^{(m)}$. The Modified Euler method is second-order accurate with the error $O(\Delta t^2)$. The time stepping procedure adopted to accomplish the Modified-Euler method is described in this section.

5.1 Continuity equation

Under the Modified Euler method, the continuity equation, Eq. (22), is used to obtain the solutions for the surface elevation η. The equation to obtain the intermediate values, $\eta^{(m)}$, is given by

$$\eta^{(m)} = \eta^{(1)} + \Delta t R^{(1)}_{\eta}, \tag{37}$$

and the second-step equation is given by

$$\eta^{(2)} = \eta^{(1)} + \frac{\Delta t}{2}(R^{(1)}_{\eta} + R^{(m)}_{\eta}). \tag{38}$$

5.2 Momentum equation

The momentum equation, Eq. (36), is used to obtain the solutions for velocity u_i. The equation to obtain the intermediate values, $u^{(m)}_i$, is given by

$$\begin{aligned} & u^{(m)}_i - \eta_{,i} h_{,j} u^{(m)}_j \\ & \quad - (h+\eta)\left[(\eta_{,i} + \frac{h_{,i}}{2})u^{(m)}_{j,j} + \frac{1}{3}(h+\eta)u^{(m)}_{j,ij} + \frac{1}{2}(u^{(m)}_{j,i}h_{,j} + u^{(m)}_j h_{,ij})\right] \\ & = u^{(1)}_i - \eta_{,i} h_{,j} u^{(1)}_j \\ & \quad - (h+\eta)\left[(\eta_{,i} + \frac{h_{,i}}{2})u^{(1)}_{j,j} + \frac{1}{3}(h+\eta)u^{(1)}_{j,ij} + \frac{1}{2}(u^{(1)}_{j,i}h_{,j} + u^{(1)}_j h_{,ij})\right] - \Delta t Q^{(1)}_{[u]i} \end{aligned} \tag{39}$$

and the second-step equation is given by

$$\begin{aligned}
&u_i^{(2)} - \eta_{,i} h_{,j} u_j^{(2)} \\
&\quad - (h+\eta)\left[\left(\eta_{,i} + \frac{h_{,i}}{2}\right) u_{j,j}^{(2)} + \frac{1}{3}(h+\eta) u_{j,ij}^{(2)} + \frac{1}{2}(u_{j,i}^{(2)} h_{,j} + u_j^{(2)} h_{,ij})\right] \\
&= u_i^{(1)} - \eta_{,i} h_{,j} u_j^{(1)} \\
&\quad - (h+\eta)\left[\left(\eta_{,i} + \frac{h_{,i}}{2}\right) u_{j,j}^{(1)} + \frac{1}{3}(h+\eta) u_{j,ij}^{(1)} + \frac{1}{2}(u_{j,i}^{(1)} h_{,j} + u_j^{(1)} h_{,ij})\right] \\
&\quad + \frac{\Delta t}{2}(Q_{[u]i}^{(1)} + Q_{[u]i}^{(m)}).
\end{aligned} \tag{40}$$

It may be noted that the terms on the right hand sides in Eqs. (39) and (40) depend on the values at the previous time step, except $Q_{[u]i}^{(m)}$ in Eq. (40). Since Eq. (39) is solved prior to solving Eq. (40), the values, $Q_{[u]i}^{(m)}$ are known before we attempt to solve Eq. (40). This makes another simplification possible where we combine all the terms on the right hand side. This results in

$$\begin{aligned}
&u_i^{(m)} - \eta_{,i} h_{,j} u_j^{(m)} \\
&\quad - (h+\eta)\left[(\eta_{,i} + \frac{h_{,i}}{2}) u_{j,j}^{(m)} + \frac{1}{3}(h+\eta) u_{j,ij}^{(m)} + \frac{1}{2}(u_{j,i}^{(m)} h_{,j} + u_j^{(m)} h_{,ij})\right] \\
&= R_{[u]i} + \Delta t Q_{[u]i}^{(1)}
\end{aligned} \tag{41}$$

and

$$\begin{aligned}
&u_i^{(2)} - \eta_{,i} h_{,j} u_j^{(2)} \\
&\quad - (h+\eta)\left[\left(\eta_{,i} + \frac{h_{,i}}{2}\right) u_{j,j}^{(2)} + \frac{1}{3}(h+\eta) u_{j,ij}^{(2)} + \frac{1}{2}(u_{j,i}^{(2)} h_{,j} + u_j^{(2)} h_{,ij})\right] \\
&= R_{[u]i} + \frac{\Delta t}{2}(Q_{[u]i}^{(1)} + Q_{[u]i}^{(m)}),
\end{aligned} \tag{42}$$

such that

$$\begin{aligned}
R_{[u]i} =& u_i^{(1)} - \eta_{,i} h_{,j} u_j^{(1)} \\
&- (h+\eta)\left[(\eta_{,i} + \frac{h_{,i}}{2}) u_{j,j}^{(1)} + \frac{1}{3}(h+\eta) u_{j,ij}^{(1)} + \frac{1}{2}(u_{j,i}^{(1)} h_{,j} + u_j^{(1)} h_{,ij})\right].
\end{aligned} \tag{43}$$

Thus, the continuity and momentum equations can be integrated in time using the Modified-Euler method given by Eqs. (37)-(38) and Eqs. (41)-(42).

5.3 Boundary conditions

Among the boundary conditions presented in Section 3, only the open boundary condition given by the Orlanski condition needs to be rewritten implementing the time-stepping scheme in place of time derivatives. Applying the Modified Euler method to the open boundary condition, Eq. (19), the first and the second-order approximations are written as

$$\begin{aligned} \eta_{,t}^{(m)} &= \pm \Delta t c n_{i(w)} \eta_{,i}^{(1)}, \;\; x_i \in S_{(o)} \\ u_{i,t}^{(m)} &= \pm \Delta t c n_{i(w)} u_{i,j}^{(1)}, \;\; x_i \in S_{(o)} \end{aligned} \tag{44}$$

and

$$\begin{aligned} \eta_{,t}^{(m)} &= \pm \frac{\Delta t}{2} c n_{i(w)} (\eta_{,j}^{(m)} + \eta_{,j}^{(1)}), \;\; x_i \in S_{(o)} \\ u_{i,t}^{(m)} &= \pm \frac{\Delta t}{2} c n_{i(w)} (u_{i,j}^{(m)} + u_{i,j}^{(1)}), \;\; x_i \in S_{(o)}. \end{aligned} \tag{45}$$

6 Curvilinear coordinates and spatial derivatives

In Section 5, the methodology adopted to integrate the governing equations with respect to time, t, is presented. To obtain the solutions of Eqs. (37)-(38) and Eqs. (39)-(40), the domain is spatially discretized into a grid and the finite-difference method is used, i.e. the spatial derivatives are represented in terms of the finite-difference formulas. To facilitate the use of a non-uniform grid, a curvilinear coordinate system will be adopted in conjunction with numerical grid generation. In this section, the details of the implementation of the curvilinear coordinate system is explained and the finite difference formulas to obtain the spatial derivatives are presented. The numerical grid generation provides the mapping of the physical domain represented by the coordinate system (x_1, x_2) to a curvilinear coordinate system represented by (ξ_1, ξ_2). Namely,

$$(x_1, x_2) \mapsto (\xi_1, \xi_2). \tag{46}$$

To describe the implementation of this transformation to the equations of the problem, we introduce a generic variable, f, such that

$$f(x_1, x_2, t) = f(\xi_1, \xi_2, t). \tag{47}$$

6.1 First-order derivatives

The relationship between the first-order spatial derivative in the physical and the computational domain of variable, f, is given by the chain rule as

$$f_{,x_i} = \xi_{j,x_i} f_{,\xi_j} \tag{48}$$

where the derivative, ξ_{j,x_i}, can be obtained from the mapping achieved by the numerical grid-generation. While using numerical grid-generation to obtain the mapping, it is more convenient to represent the spatial derivatives with respect to ξ_i rather than x_i. Since ξ_{j,x_i} involves the derivative with respect to ξ_i, it is necessary to represent ξ_{j,x_i} in terms of derivatives with respect to ξ_i. This can be achieved by defining the Jacobian of the transformation as

$$\mathcal{J}_{ij} = x_{i,\xi_j}. \tag{49}$$

For two dimensional problems, the Jacobian matrix is given as

$$[\mathcal{J}] = \begin{bmatrix} x_{1,\xi_1} & x_{1,\xi_2} \\ x_{2,\xi_1} & x_{2,\xi_2} \end{bmatrix}. \tag{50}$$

Since

$$f_{,\xi_i} = x_{j,\xi_i} f_{,x_j}, \tag{51}$$

we get

$$f_{,\xi_i} = \mathcal{J}_{ij} f_{,x_j}. \tag{52}$$

Or,

$$f_{,x_i} = \mathcal{J}^I_{ij} f_{,\xi_j}, \tag{53}$$

where $\mathcal{J}^I$ represents the inverse of the Jacobian matrix $\mathcal{J}$. It may be noted that the inverse of the Jacobian matrix can be expressed in terms of the co-factor and determinant using

$$\mathcal{J}^I_{ij} = \frac{\mathcal{C}_{ij}}{J} \tag{54}$$

where $\mathcal{C}_{i,j}$ is the co-factor of the element, $\mathcal{J}_{ij}$, of the Jacobian matrix and $J, (= |\mathcal{J}|)$, is the determinant of the Jacobian matrix. Therefore, using Eq. (54) in Eq. (53), we get

$$f_{,x_i} = \frac{\mathcal{C}_{ij}}{J} f_{,\xi_j}. \tag{55}$$

It can be seen that by introducing the Jacobian of the transformation, Eq. (47) can be written as Eq. (55) where the derivatives are with respect to ξ_i instead of x_i. For two-dimensional domains, the co-factor matrix, C, is given by

$$\mathcal{C}_{ij} = \begin{bmatrix} x_{2,\xi_2} & -x_{2,\xi_1} \\ -x_{1,\xi_2} & x_{1,\xi_1} \end{bmatrix} \tag{56}$$

and the Jacobian, J, is given by

$$J = x_{1,\xi_1} x_{2,\xi_2} - x_{1,\xi_2} x_{2,\xi_1}. \tag{57}$$

Thus, Eqs. (55)-(57), using the generic variable, f, provide the expression for the first-order derivatives in terms of derivatives in the computational domain.

6.2 Second-order derivatives

Using Eq. (55) and applying the chain rule, the second-order derivatives can be expressed in terms of derivatives in the computational domain as

$$\begin{aligned} f_{,x_i x_j} &= (f_{,x_i})_{,x_j} \\ &= \frac{\mathcal{C}_{jl}}{J}\left(\frac{\mathcal{C}_{ik}}{J} f_{,\xi_k}\right)_{,\xi_l} \\ &= \frac{1}{J^2}\mathcal{C}_{jl}\mathcal{C}_{ik} f_{,\xi_k\xi_l} + \frac{\mathcal{C}_{jl}}{J}\left(\frac{\mathcal{C}_{ik}}{J}\right)_{,\xi_l} f_{,\xi_k} \\ &= \frac{1}{J^2}\mathcal{C}_{jl}\mathcal{C}_{ik} f_{,\xi_k\xi_l} + \frac{1}{J^3}\mathcal{C}_{jl} f_{,\xi_k}(J\mathcal{C}_{ik,\xi_l} - J_{,\xi_l}\mathcal{C}_{ik}). \end{aligned} \tag{58}$$

For brevity, defining C_{ijkl} and D_{ik} such that

$$C_{ijkl} = \frac{1}{J^2}\mathcal{C}_{jl}\mathcal{C}_{ik} \tag{59}$$

and

$$D_{ijk} = \frac{1}{J^2}\mathcal{C}_{jl}(J(\mathcal{C}_{ik,\xi_l} - J_{,\xi_l}\mathcal{C}_{ik}), \tag{60}$$

we get

$$f_{,x_i x_j} = C_{ijkl} f_{,\xi_k\xi_l} + D_{ijk} f_{,\xi_k}. \tag{61}$$

6.3 Expansion of derivatives in two-dimensions

The expressions relating the derivatives in the physical domain and the computational domain are presented in Sections 6.1 and 6.2. These expressions are presented using indicial notation and are applicable to two-dimensional as well as three-dimensional computational domains. In this section, these expressions for the derivatives are expanded for the two-dimensional domain so that the indices i, j and k take the values of 1 and 2 for the two independent directions and are presented below:

$$J = x_{1,\xi_1} x_{2,\xi_2} - x_{1,\xi_2} x_{2,\xi_1}, \tag{62}$$

$$f_{,x_1} = \frac{1}{J}(x_{2,\xi_2} f_{,\xi_1} + x_{2,\xi_1} f_{,\xi_2}), \tag{63}$$

$$f_{,x_2} = \frac{1}{J}(x_{1,\xi_2} f_{,\xi_1} + x_{1,\xi_1} f_{,\xi_2}), \tag{64}$$

$$\begin{aligned} f_{,x_1x_1} =& \frac{1}{J^2}(x_{2,\xi_2}^2 f_{,\xi_1\xi_1} - 2x_{2,\xi_1} x_{2,\xi_2} f_{,\xi_1\xi_2} + x_{2,\xi_1}^2 f_{,\xi_2\xi_2}) \\ & - \frac{1}{J^3}\left[(x_{2,\xi_2}^2 x_{2,\xi_1\xi_1} - 2x_{2,\xi_1} x_{2,\xi_2} x_{2,\xi_1\xi_2} + x_{2,\xi_1}^2 x_{2,\xi_2\xi_2})(x_{1,\xi_2} f_{,\xi_1} + x_{1,\xi_1} f_{,\xi_2})\right. \\ & \left.+(x_{2,\xi_2}^2 x_{1,\xi_1\xi_1} - 2x_{2,\xi_1} x_{2,\xi_2} x_{1,\xi_1\xi_2} + x_{2,\xi_1}^2 x_{1,\xi_2\xi_2})(x_{2,\xi_2} f_{,\xi_1} + x_{2,\xi_1} f_{,\xi_2})\right], \end{aligned} \tag{65}$$

and

$$\begin{aligned} f_{,x_2x_2} =& \frac{1}{J^2}(x_{1,\xi_2}^2 f_{,\xi_1\xi_1} - 2x_{1,\xi_1} x_{1,\xi_2} f_{,\xi_1\xi_2} + x_{1,\xi_1}^2 f_{,\xi_2\xi_2}) \\ & - \frac{1}{J^3}\left[(x_{1,\xi_2^2} x_{2,\xi_1\xi_1} - 2x_{1,\xi_1} x_{1,\xi_2} x_{2,\xi_1\xi_2} + x_{1,\xi_1}^2 x_{2,\xi_2\xi_2})(x_{1,\xi_2} f_{,\xi_1} + x_{1,\xi_1} f_{,\xi_2})\right. \\ & \left.+(x_{1,\xi_2}^2 x_{1,\xi_1\xi_1} - 2x_{1,\xi_1} x_{1,\xi_2} x_{1,\xi_1\xi_2} + x_{1,\xi_1}^2 x_{1,\xi_2\xi_2})(x_{2,\xi_2} f_{,\xi_1} + x_{2,\xi_1} f_{,\xi_2})\right]. \end{aligned} \tag{66}$$

Eq. (61) in Section 6.2 provides a compact form of the second-order derivatives. The expansion of Eq. (61) and its coefficients are given below:

$$f_{,x_1,x_1} = (C_{1111} f_{,\xi_1\xi_1} + C_{1112} f_{,\xi_1\xi_2} + C_{1121} f_{,\xi_2\xi_1} + C_{1122} f_{,\xi_2\xi_2} + D_{11} f_{,\xi_1} + D_{12} f_{,\xi_2}) \tag{67}$$

and

$$f_{,x_2,x_2} = (C_{2211} f_{,\xi_1\xi_1} + C_{2212} f_{,\xi_1\xi_2} + + C_{2221} f_{,\xi_2\xi_1} + C_{2222} f_{,\xi_2\xi_2} + D_{21} f_{,\xi_1} + D_{22} f_{,\xi_2}). \tag{68}$$

where

$$
\begin{aligned}
D_{111} &= \mathcal{D}_{11}x_{1,\xi_2} - \mathcal{D}_{12}x_{2,\xi_2} \\
D_{112} &= \mathcal{D}_{12}x_{2,\xi_1} - \mathcal{D}_{11}x_{1,\xi_1} \\
D_{221} &= \mathcal{D}_{21}x_{1,\xi_2} - \mathcal{D}_{22}x_{2,\xi_2} \\
D_{222} &= \mathcal{D}_{22}x_{2,\xi_1} - \mathcal{D}_{21}x_{1,\xi_1}
\end{aligned} \tag{69}
$$

such that

$$
\begin{aligned}
\mathcal{D}_{11} &= \frac{(x^2_{2,\xi_2}x_{2,\xi_1\xi_1} - 2x_{2,\xi_1}x_{2,\xi_2}x_{2,\xi_1\xi_2} + x^2_{2,\xi_1}x_{2,\xi_2\xi_2})}{J^3} \\
\mathcal{D}_{12} &= \frac{(x^2_{2,\xi_2}x_{1,\xi_1\xi_1} - 2x_{2,\xi_1}x_{2,\xi_2}x_{1,\xi_1\xi_2} + x^2_{2,\xi_1}x_{1,\xi_2\xi_2})}{J^3} \\
\mathcal{D}_{21} &= \frac{(x^2_{1,\xi_2}x_{2,\xi_1\xi_1} - 2x_{1,\xi_1}x_{1,\xi_2}x_{2,\xi_1\xi_2} + x^2_{1,\xi_1}x_{2,\xi_2\xi_2})}{J^3} \\
\mathcal{D}_{22} &= \frac{(x^2_{1,\xi_2}x_{1,\xi_1\xi_1} - 2x_{1,\xi_1}x_{1,\xi_2}x_{1,\xi_1\xi_2} + x^2_{1,\xi_1}x_{1,\xi_2\xi_2})}{J^3}
\end{aligned} \tag{70}
$$

and

$$
\begin{aligned}
C_{1111} &= \frac{x^2_{2,\xi_2}}{J^2}, \quad C_{1112} = C_{1121} = \frac{-x_{2,\xi_1}x_{2,\xi_2}}{J^2}, \quad C_{1122} = \frac{x^2_{2,\xi_1}}{J^2} \\
C_{2211} &= \frac{x^2_{1,\xi_2}}{J^2}, \quad C_{2212} = C_{2221} = \frac{-x_{1,\xi_1}x_{1,\xi_2}}{J^2}, \quad C_{2222} = \frac{x^2_{1,\xi_1}}{J^2}.
\end{aligned} \tag{71}
$$

6.4 Finite-difference formulas

Since the problem is solved in a curvilinear coordinate system, and to estimate the derivatives with respect to x_i, it is essential to estimate the derivatives with respect to ξ_i. This can be seen from the expressions for the derivatives presented in Section 6.3. Using bracketed subscripts of the form, $[i,j]$, to represent the nodal locations along the ξ_1 and ξ_2 directions, respectively, the finite-difference formulas used to estimate the partial derivatives with respect to ξ_1 and ξ_2 are given as

$$
f_{,\xi_1} = \frac{f_{[i+1,j]} - f_{[i-1,j]}}{2}, \tag{72}
$$

$$
f_{,\xi_2} = \frac{f_{[i,j+1]} - f_{[i,j-1]}}{2}, \tag{73}
$$

$$f_{,\xi_1\xi_2} = \frac{f_{[i+1,j+1]} - f_{[i-1,j+1]} - f_{[i+1,j-1]} + f_{[i-1,j-1]}}{4}, \tag{74}$$

$$f_{,\xi_1\xi_1} = f_{[i+1,j]} - 2f_{[i,j]} + f_{[i-1,j]}, \tag{75}$$

and

$$f_{,\xi_2\xi_2} = f_{[i,j+1]} - 2f_{[i,j]} + f_{[i,j-1]}. \tag{76}$$

It may be noted that these formulas are second-order accurate. The formulas given in Eqs. (72)-(76) can be used to estimate the derivatives with respect to x_i using Eqs. (62)-(66).

7 Successive Over Relaxation

In Sections 4-6 of this report, the time-stepping procedure, Eqs. (37)-(40), and the spatial discretizations, Eqs. (62)-(76), reduces the problem into a solution of linear system of equations. With the values of the surface elevation, η, and the velocity, u_i, known at the previous time steps, the values at the current time step form the unknowns of the linear system. This linear system of equations is in the form of a banded system and the values at the previous time step are close to the unknown values at the current step. Therefore, it is more efficient to use an iterative method such as the successive over-relaxation method (SOR) compared to non-iterative methods such as Gaussian elimination. In this section, the details of the reduction of the momentum equation, Eq. (36), to a form convenient for the implementation of SOR is presented.

To facilitate the iterative scheme, it is essential to rewrite the governing equation in a form where a new iterative value can be predicted using the iterative value from the previous iteration. The momentum equation given by Eq. (36) contains the first-order and second-order derivatives of u_i. These will be approximated using the equations in Section 6.3 and finite-difference formulas in Section 6.4. Let us consider the iterations performed at any arbitrary node $[i, j]$. It may be noted that only the equations for the second-order derivatives, Eqs. (75) and (76), involve the values at the node $[i, j]$. This means that only the u_{i,x_jx_j} terms contain the unknowns $u_{i[i,j]}$. It may be noted Eqs. (39) and (40) are similar in form, with the main difference in the superscripts indicating the time-stepping levels. In this section, we

explain the implementation of the SOR. We will consider only Eq. (40) and the procedure for Eq. (39) will follow similar principles. To perform successive over relaxation, only the terms involving u_i must be retained on the left hand side and the rest must be moved to the right hand side. We have terms such as u_j giving rise to terms with u_i when $i = j$ and the other terms when $i \neq j$. Since only the terms when $i = j$ are needed on the left hand side, we need to move the terms with $i \neq j$ to the right hand side. To demonstrate this, and for simplicity, we consider the equation for the velocity component u_1 along the x_1 direction. The derivation for the other velocity component, u_2, can be obtained in a similar fashion and, therefore, will not be presented here.

Introducing the variable A_1 for simplification, we write the equation for u_1 as

$$u_1^{(2)} - \frac{1}{2}(h+\eta)h_{,11}u_1^{(2)} - h_{,1}\eta_{,1}u_1^{(2)} - \frac{1}{3}(h+\eta)^2 u_{1,11}^{(2)} = A_1 \tag{77}$$

where

$$\begin{aligned} A_1 =& \frac{1}{2}(h+\eta)u_{j,1}^{(2)}h_{,j} + \frac{1}{2}(h+\eta)h_{,12}u_2^{(2)} + h_{,2}\eta_{,1}u_2^{(2)} \\ &+ \frac{1}{3}(h+\eta)^2 u_{2,12}^{(2)} + (h+\eta)\left(\eta_{,1} + \frac{h_{,1}}{2}\right)u_{j,j}^{(2)} + R_{[u]1} \\ &+ \frac{\Delta t}{2}(Q_{[u]1}^{(1)} + Q_{[u]1}^{(m)}). \end{aligned} \tag{78}$$

Using Eq. (61) in Eq. (77), we get

$$\begin{aligned} & u_1^{(2)} - \frac{1}{2}(h+\eta)h_{,11}u_1^{(2)} - h_{,1}\eta_{,1}u_1^{(2)} \\ & - \frac{1}{3}(h+\eta)^2 C_{11jk}(u_{1,\xi_j\xi_k}^{(2)} + D_{1k}u_{1,\xi_k}^{(2)}) = A_1. \end{aligned} \tag{79}$$

Equation (79) can be expanded to give

$$\begin{aligned} & u_1^{(2)} - \frac{1}{2}(h+\eta)h_{,11}u_1^{(2)} - h_{,1}\eta_{,1}u_1^{(2)} \\ & - \frac{1}{3}(h+\eta)^2(C_{1111}u_{1,\xi_1\xi_1}^{(2)} + C_{1112}u_{1,\xi_1\xi_2}^{(2)} + C_{1121}u_{1,\xi_2\xi_1}^{(2)} + C_{1122}u_{1,\xi_2\xi_2}^{(2)}) \\ & + (D_{11}u_{1,\xi_1}^{(2)} + D_{12}u_{1,\xi_2}^{(2)}) = A_1. \end{aligned} \tag{80}$$

Simplifying Eq. (80), dropping the superscripts and substituting the derivatives using the

finite-difference formulas given by Eqs. (72)-(76), we get

$$\begin{aligned}
&\left[1 - \frac{1}{2}(h+\eta)h_{,11} - h_{,1}\eta_{,1} + \frac{2}{3}(h+\eta)^2(C_{1111} + C_{1122})\right] u_{1[i,j]} \\
&- \frac{1}{3}(h+\eta)^2[C_{1111}(u_{1[i+1,j]} + u_{1[i-1,j]}) + \\
&\frac{1}{4}(C_{1112} + C_{1121})(u_{1[i+1,j+1]} - u_{1[i-1,j+1]} - u_{1[i+1,j-1]} + u_{1[i-1,j-1]}) \\
&+ C_{1122}(u_{1[i,j+1]} + u_{1[i,j-1]})] \\
&+ \frac{1}{2}[D_{11}(u_{1[i+1,j]} - u_{1[i-1,j]}) + D_{12}(u_{1[i,j+1]} - u_{1[i,j-1]})] = A_1.
\end{aligned} \tag{81}$$

Rearranging the terms in Eq. (81), we get

$$\begin{aligned}
u_{1[i,j]} = \frac{\begin{array}{c} A_1 + \frac{1}{3}(h+\eta)^2[C_{1111}(u_{1[i+1,j]} + u_{1[i-1,j]}) \\ -\frac{1}{4}(C_{1112} + C_{1121})(u_{1[i+1,j+1]} - u_{1[i-1,j+1]} - u_{1[i+1,j-1]} + u_{1[i-1,j-1]}) \\ +C_{1122}(u_{1[i,j+1]} + u_{1[i,j-1]})] \\ +\frac{1}{2}[D_{11}(u_{1[i+1,j]} - u_{1[i-1,j]}) - D_{12}(u_{1[i,j+1]} - u_{1[i,j-1]})] \end{array}}{\left[1 - \frac{1}{2}(h+\eta)h_{,11} - h_{,1}\eta_{,1} + \frac{2}{3}(h+\eta)^2(C_{1111} + C_{1122})\right]}.
\end{aligned} \tag{82}$$

Equation (82), and a similar equation for $u_2[i,j]$, can be used, in conjunction with SOR, to solve for u_1 and u_2. The derivatives on the right hand side of Eq. (82) will be obtained by using the derivatives' representation in a curvilinear-coordinate system given by Eqs. (63)-(66) and the finite-difference formulas given by Eqs. (72)-(76).

8 Test cases

In the previous sections, the theory and numerical techniques to obtain the solutions of the Green-Nagdhi equations are presented. In this section, we consider certain example cases for which solutions have been obtained using other shallow-water wave theories such as the Boussinesq equations. A comparison of the results obtained using the Green-Nagdhi equations is made with the previously obtained results.

8.1 Ramp case

Here, we consider the diffraction of a solitary wave propagating over a ramp or, in other words, a shelf. This case was considered by Ertekin and Wehausen (1987)[3] where the

Green-Nagdhi equations are used to study soliton propagation in different setups. This case was also considered by Schember (1982)[8] where Boussinesq equations are used to study the wave propagation over different sizes of ramps and shelves. In Figure 1, a schematic diagram of the setup considered here is presented, where h_1 is the depth at the incoming boundary, h_2 is the depth at the outgoing boundary, x_L is the x_1-coordinate at the incoming boundary, x_R is the x_1-coordinate at the outgoing boundary, x_S is the location of the start of the ramp and L_R is the length of the ramp.

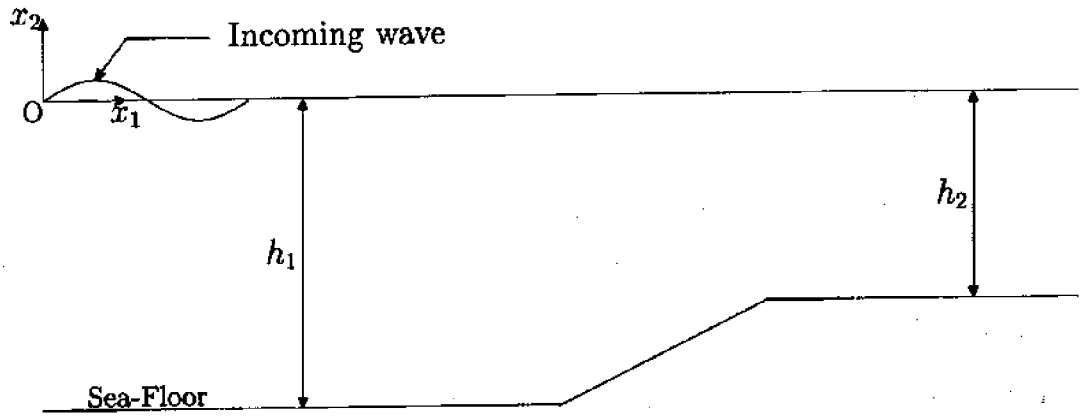

Figure 1: Schematic diagram of wave diffraction due to a ramp

To facilitate an accurate comparison of the results, we choose one of the cases considered by Ertekin and Wehausen (1987)[3] with all the parameters kept same, with the only exception being the three-dimensionality of the computer program in the case considered here.

The incoming wave is a solitary wave with a dimensionless height of 0.12 which, as in Ertekin and Wehausen (1987)[3], is incorporated using initial conditions. At time $t = 0$, the wave is set with the maximum peak at $x_1 = 30$. At this instant of time, the wave profile for a wave of height of 0.12 results in almost negligible surface elevation at the incoming boundary. It may be noted that a similar approach is adopted by Schember (1982)[8]. It is believed that such an approach avoids the numerical instability that occurs at the incoming boundary due to the inaccuracies in the finite-difference representation of the third-order derivatives of velocity at the incoming boundary.

The ramp starts at the location $x_1 = x_S = 40$ which slopes from a constant depth at the incoming boundary to a value half of that at the outgoing boundary. The location of the

ramp is such that at time $t = 0$, the surface elevations at the ramp in the initial wave profile are negligible. The ramp is of length, $L_R = 10$ which results in a slope of 1:20.

A dimensionless grid spacing of 0.1 is chosen and, in order to satisfy the Courant condition for numerical stability, the time step is set as 0.1. The simulation is carried out for 900 time steps. The wave profiles at $t = 20, 30, 50, 70$ and 90 are presented in Figure 2. These profiles compare very well with the results presented in Figure 6 of Ertekin and Wehausen (1987)[3].

The evolution of the surface elevations at different numerical gage locations are presented in Figure 3. These results also compare very well with the results presented in Ertekin and Wehausen (1987)[3].

8.2 Mount case

In this section, we consider a mount or, in other words, submerged shelf as shown in Figure 4. We consider this case which was analyzed by Ertekin and Becker (1996)[2] to study the effectiveness of this numerical model in handling the diffraction of cnoidal waves. In this section, the results of the simulation of diffraction due to a submerged shelf is presented for incoming cnoidal waves and is compared with the previous results (Ertekin and Becker, 1996)[2].

A schematic diagram of the setup is presented in Figure 4. The domain extends from $x_1 = x_L$ to $x_1 = x_R$. The description of the location of the mount is accomplished by using the variables, x_A, x_B, x_C and x_D. The water depth at the submerged shelf is represented by h_2. In the case considered in this study, we have used the following values to represent the mount:

$x_A = 203.7$, $x_B = 205.1$, $x_C = 208.1$, and, $x_D = 209.5$.

These parameters correspond to the values used in case 6 of Ertekin and Becker (1996)[2]. The height of the incoming cnoidal waves is taken as 0.1. Still water condition is used as the initial condition and therefore, in order to ensure numerical stability, the incoming wave velocities and surface elevations are modulated using an exponential decay function of the form of $1 - e^{-st}$. In this expression, s represents the modulation constant which takes values in the range of 0.5 to 1.0. In the calculations performed here, we have taken s to be 1. A rectangular grid with a grid spacing of 0.1 is used in conjunction with a time step of 0.07 ensuring the numerical stability of the computations by satisfying the Courant stability

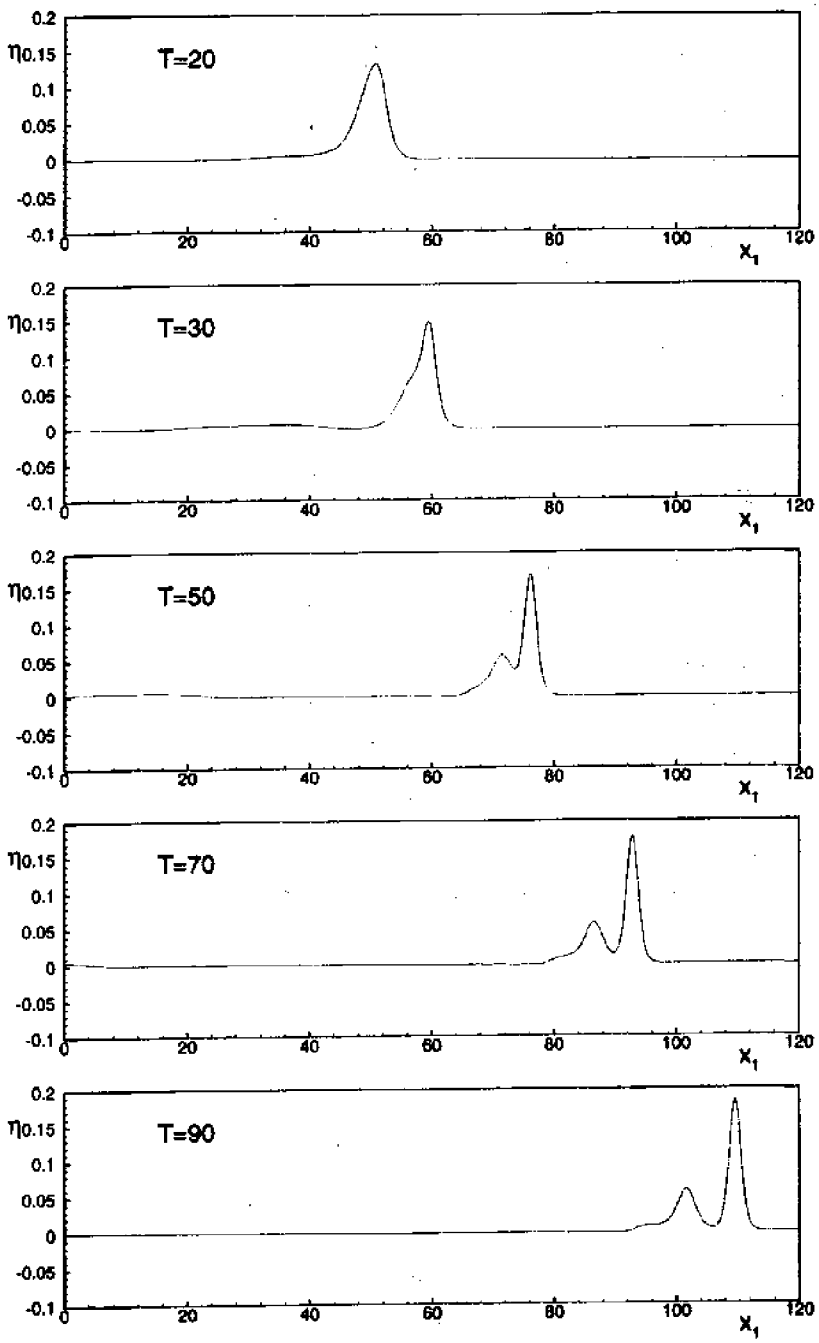

Figure 2: Wave profiles at time steps for uniform ramp case

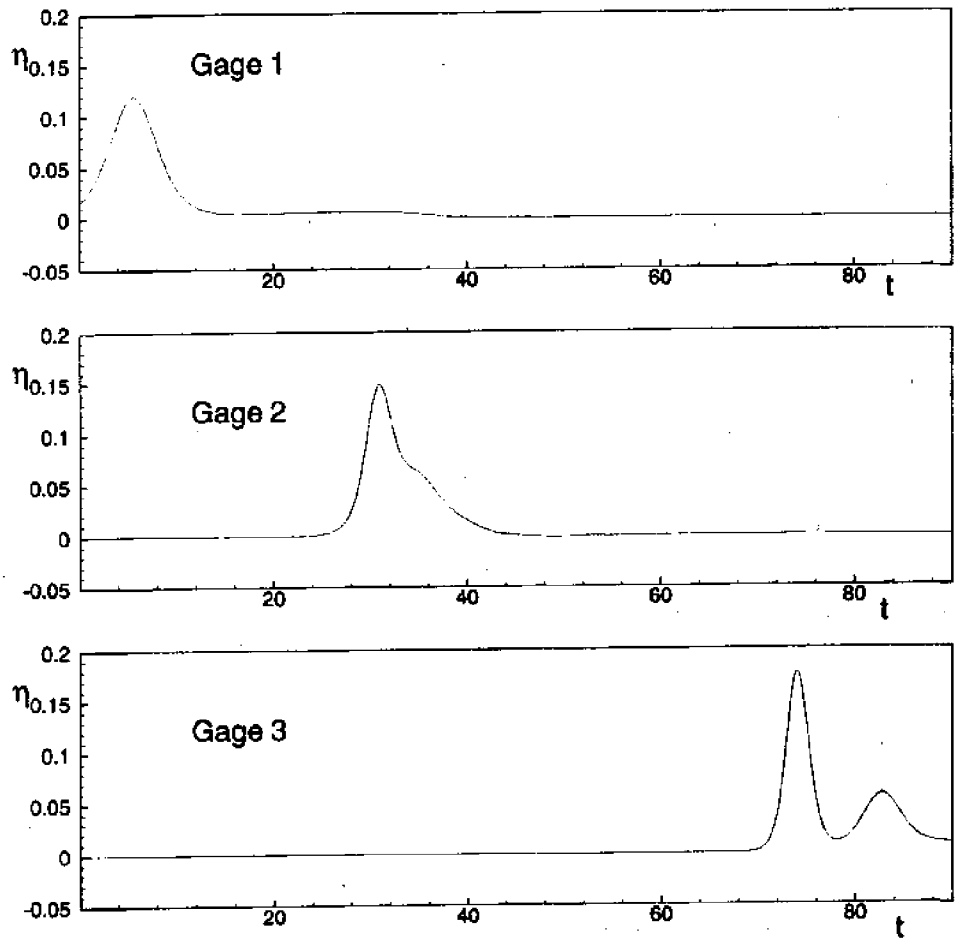

Figure 3: Time evolution of surface elevations at different gages for uniform ramp case

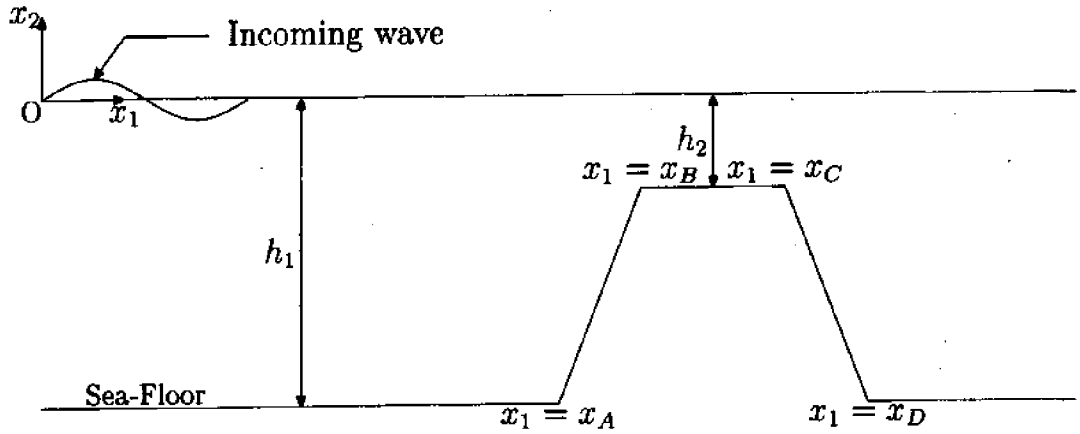

Figure 4: Schematic diagram of the setup for the mount case

condition. The simulation is performed for 8000 time steps resulting in a wave profile at time $t = 560$ as shown in Figure 5. This compares reasonably well though there are some notable differences on the right hand side of the mount with figure 2(d) of Ertekin and Becker (1996)[2]. However, the time evolution at numerical wave gages 3 and 4 located at $x_1 = 208.1$ and $x_1 = 209.5$, respectively, shown in Figure 6 are in excellent agreement with the results obtained by Ertekin and Becker (1996)[2].

8.3 Curved-ramp case

In order to confirm the validity of the computer program in a fully three-dimensional environment, we consider a ramp that is curved using a cosine-square function. This case was considered by Schember (1982)[8] where Boussinesq equations are used to simulate the diffraction of solitary waves. To obtain a curved ramp, the ramp is shifted by the function $f(x_2)$ given by

$$f(x_2) = A_{ramp} cos^2 \frac{\pi x_2}{2y_c} \quad y \leq y_c \tag{83}$$

where A_{ramp} is the shift distance and y_c is the width of the curved section. The schematic setup is shown in Figure 7. The case considered here is described as the "narrow concave" case in Schember (1982)[8] where a ramp of slope 1:10 is curved using Eq. (83) with A_{ramp} taken as 10.

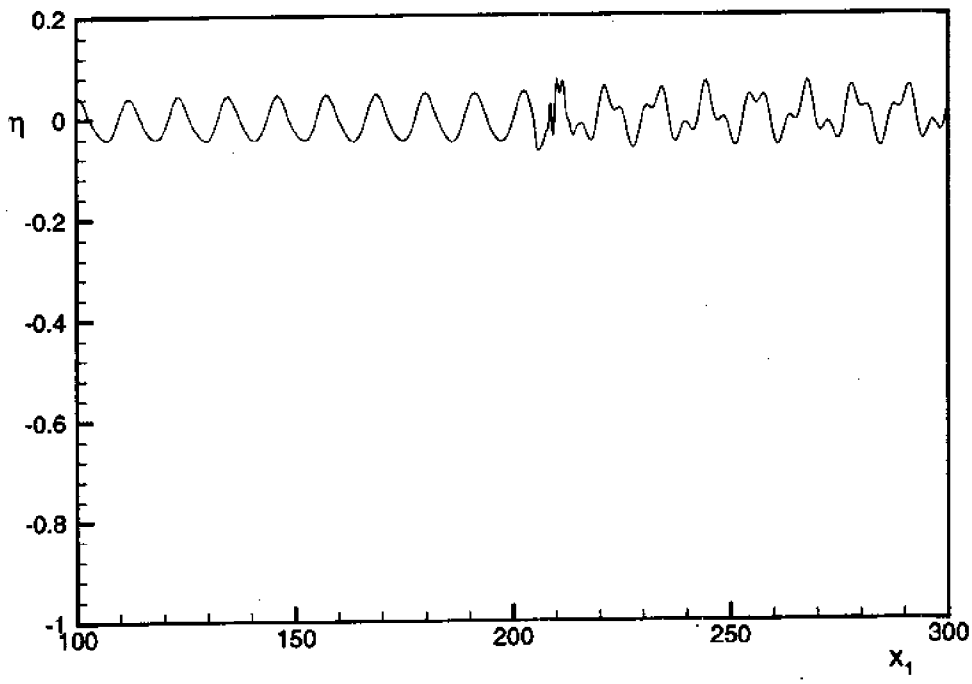

Figure 5: Wave profile at $t = 560$ for the mount case

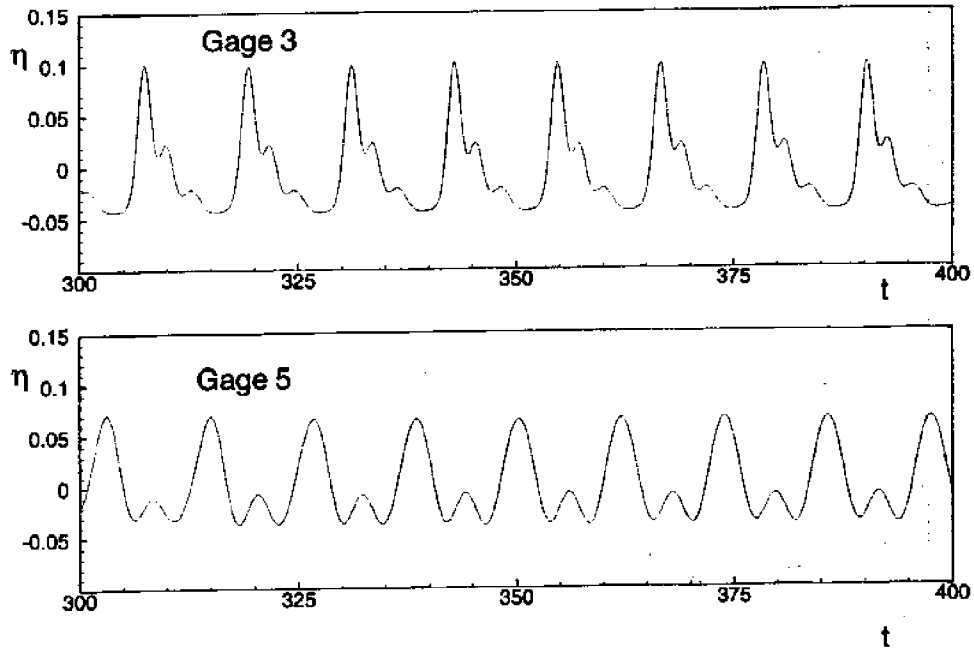

Figure 6: Time evolution of surface elevations at numerical gages for the mount case

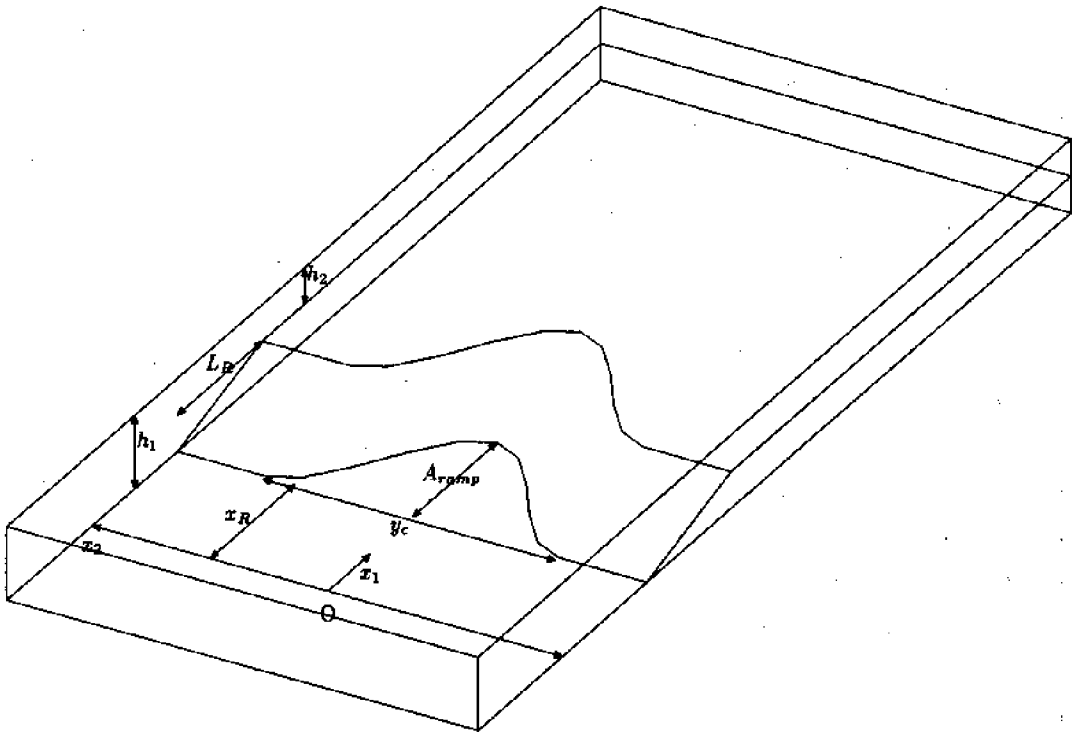

Figure 7: Schematic diagram of the setup for the curved-ramp case

To avoid numerical instabilities, the bathymetry was smoothed by taking a weighted-average of the depth values at the neighboring points at a finite-difference node. This operation is deemed necessary due to the presence of derivatives of depth upto the third-order. In Figure 8, a three-dimensional plot of the curved-ramp bathymetry as used in the present calculations is presented.

It can be seen from Eq. (83) that the domain is symmetric about the $x_1 - x_2$ plane and therefore, by using a symmetry condition at $x_2 = 0$, we need to obtain solutions in only one half of the domain and the mirror image will provide the solution for the other half.

The length of the domain for the problem under consideration is taken as 120 with the width as 32. To facilitate the input of the solitary wave as the initial condition, similar to the two-dimensional uniform ramp considered earlier, the computational domain is set from -30.0 to 90.0 in the x_1 direction and 0 to 16 in the x_2 direction. A grid spacing of 0.4 is used to discretize the domain into a rectangular grid. A time step of 0.4 is used in these calculations just as in Schember (1982)[8].

In Figs. 9-12, we present three-dimensional plots of the surface elevations at times $t = 20, 40, 60$ and 80. These plots compare very well with the three-dimensional plots presented by Schember. The corresponding contour plots of the surface elevations are presented in Figs. 13-16. It may be noted that due to different contouring algorithms used and due to the fact that the surface elevations are close to zero at most of the domain, a very close comparison of the contour lines with Schember's results is not possible. To provide a more lucid comparison of the results, we present the wave profiles at sections along the x_1 direction at the wall and the center lines in Figs. 17 and 18. It can be seen that these results do compare very well providing the necessary validation for the three-dimensional cases.

To evaluate the ability of the model to simulate the propagation of periodic waves subject to varying bathymetry, we consider cnoidal waves as the incoming waves. The other input parameters are the same as in the solitary wave case presented above. We assume the still water condition as the initial condition and cnoidal waves are modulated to avoid numerical instabilities.

In Figs. 19 and 20, the diffracted waves are shown in the surface elevation plot of the cnoidal waves at time, t=160. The two cases differ in the application of the lateral boundary condition. In Figure 19, a wall condition was used, while in Figure 20, the radiation condition

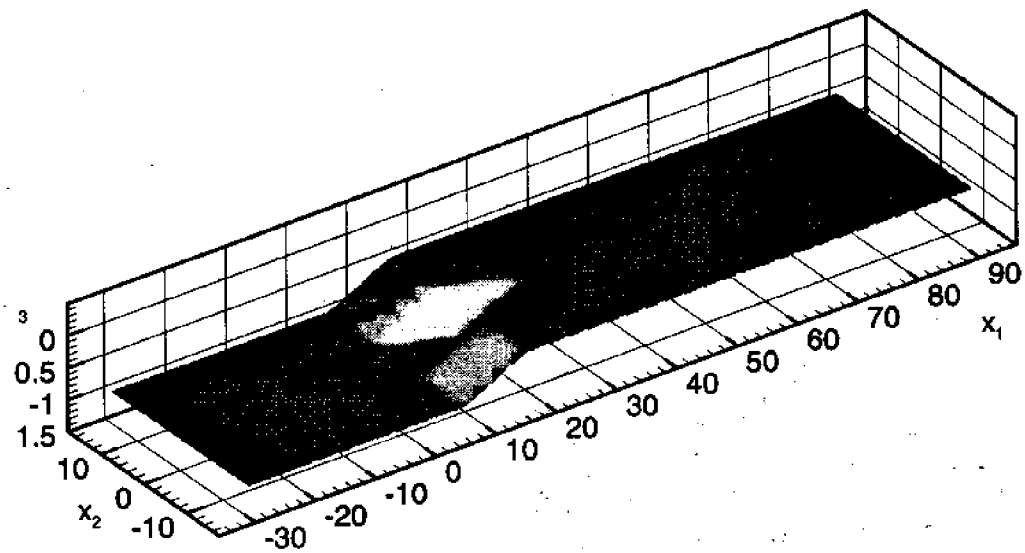

Figure 8: Bathymetry for the curved-ramp case

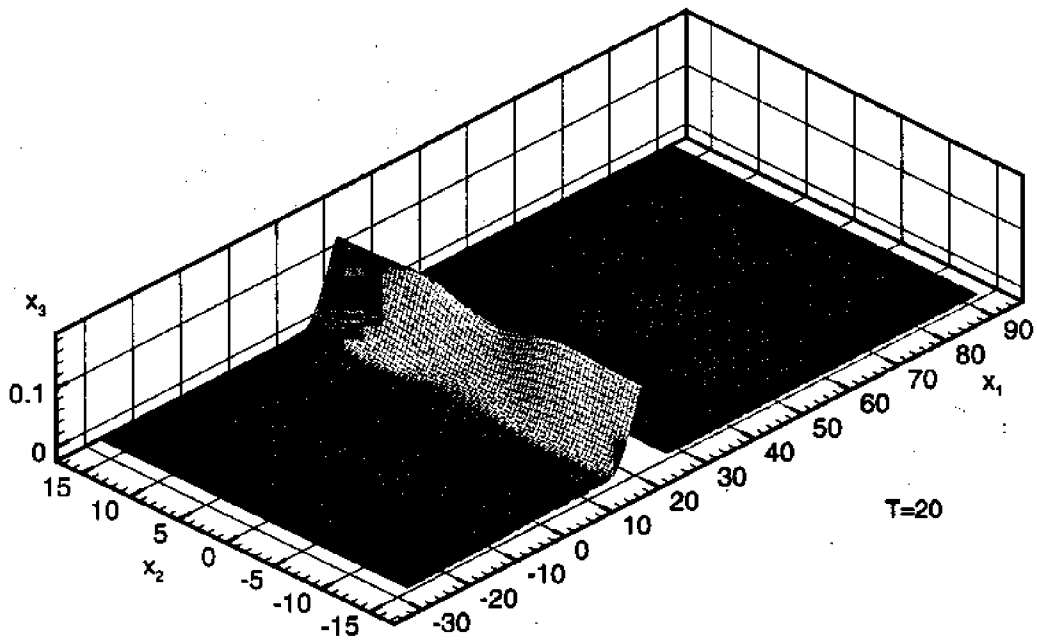

Figure 9: Wave profile at $t = 20$ for the curved-ramp case

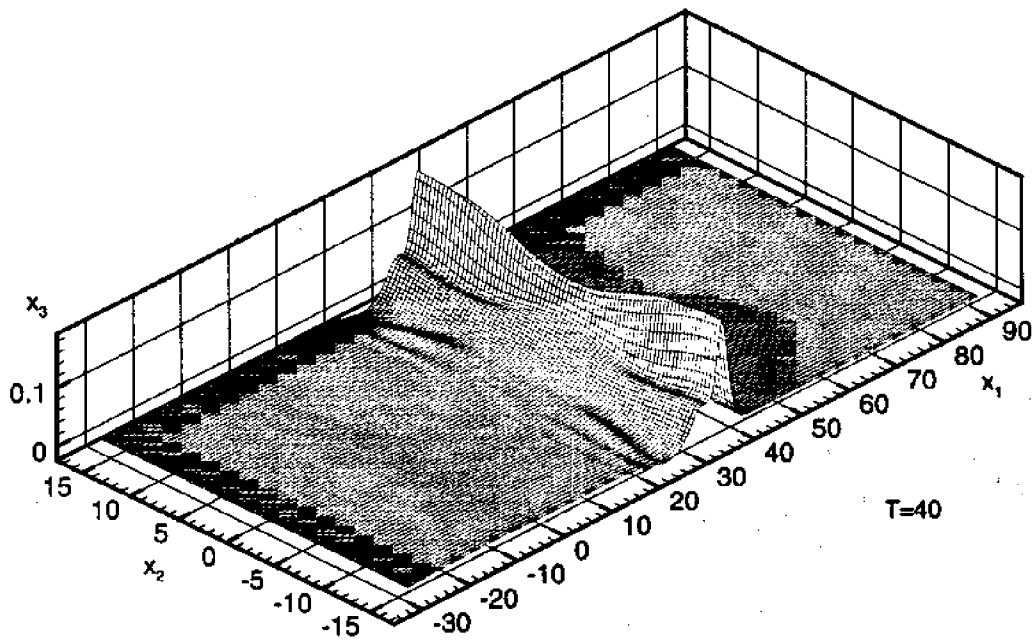

Figure 10: Wave profile at $t = 40$ for the curved-ramp case

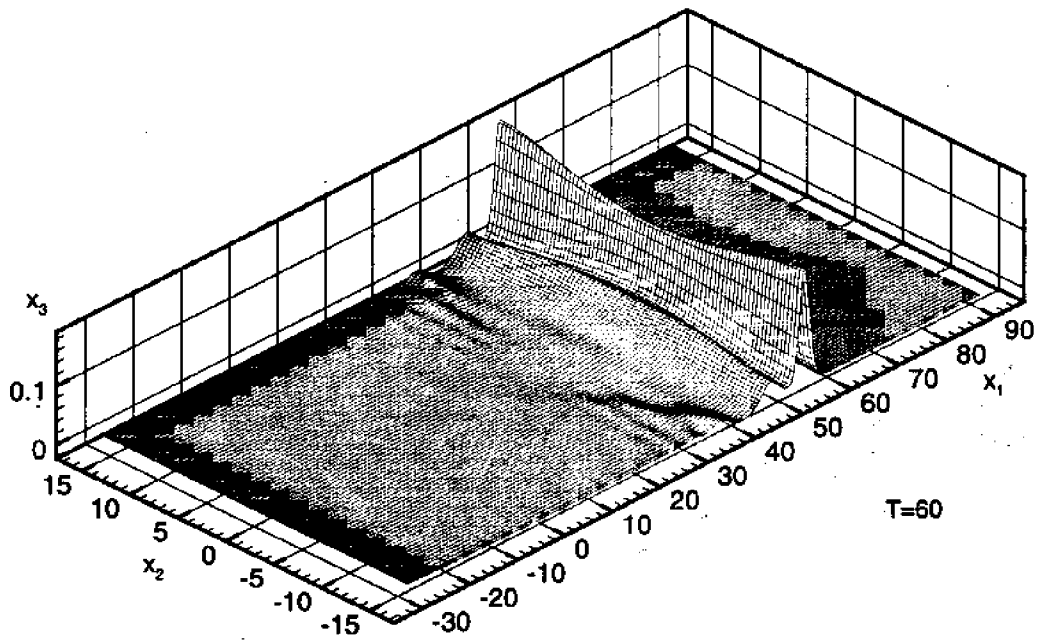

Figure 11: Wave profile at $t = 60$ for the curved-ramp case

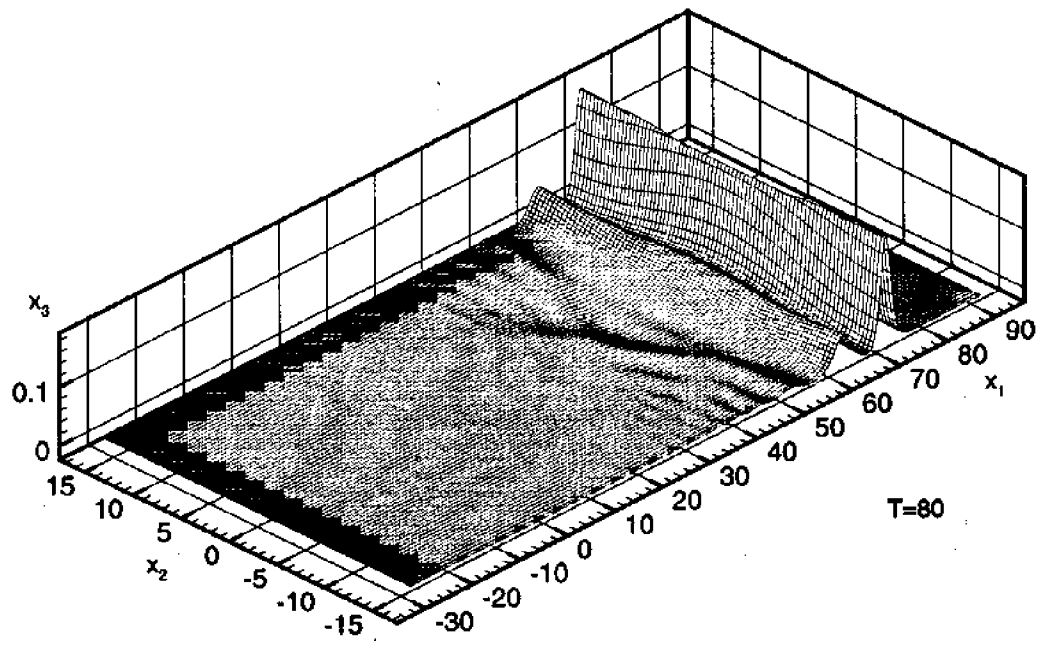

Figure 12: Wave profile at $t = 80$ for the curved-ramp case

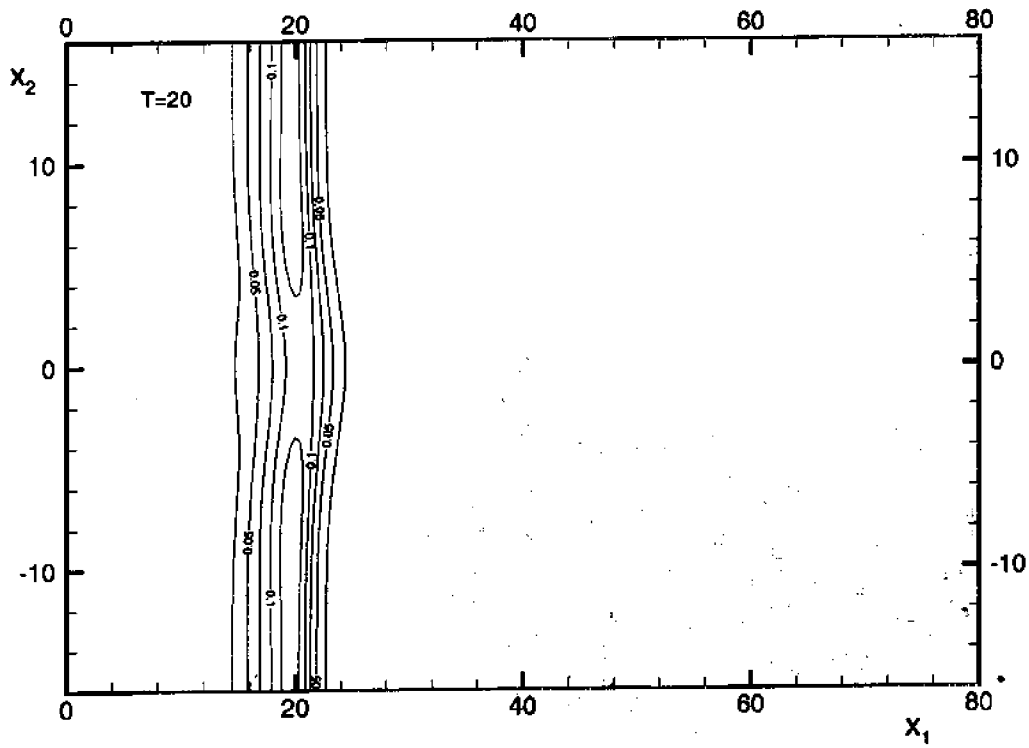

Figure 13: Surface elevation contours at $t = 20$ for the curved-ramp case

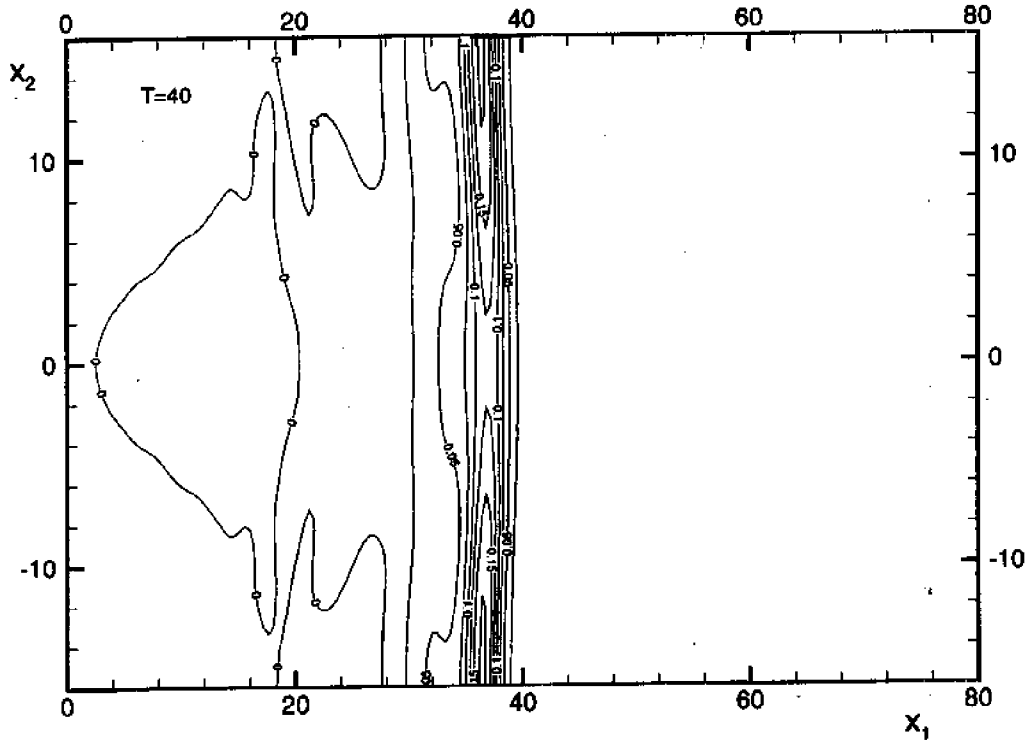

Figure 14: Surface elevation contours at $t = 40$ for the curved-ramp case Schematic diagram of the setup for the curved-ramp case

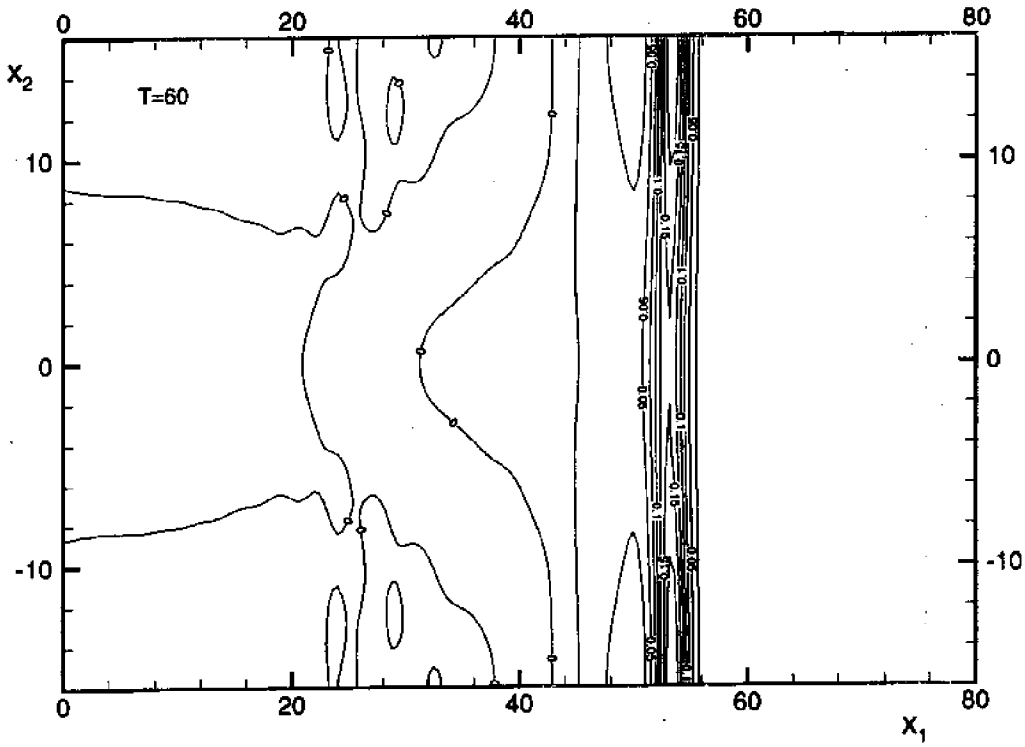

Figure 15: Surface elevation contours at $t = 60$ for the curved-ramp case

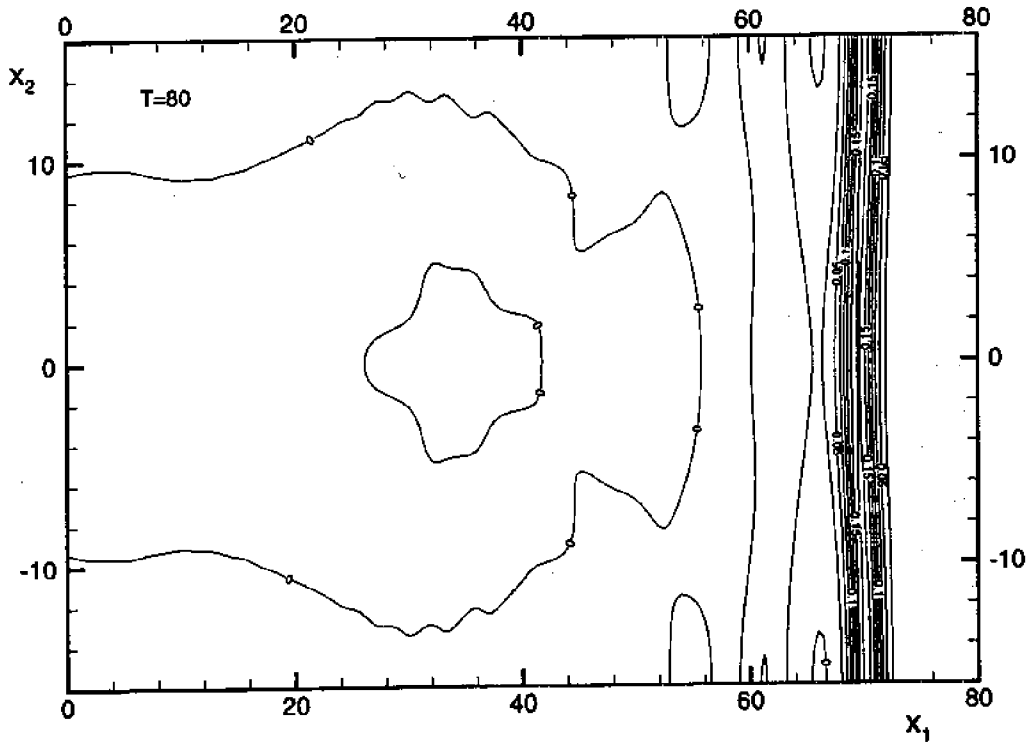

Figure 16: Surface elevation contours at $t = 80$ for the curved-ramp case

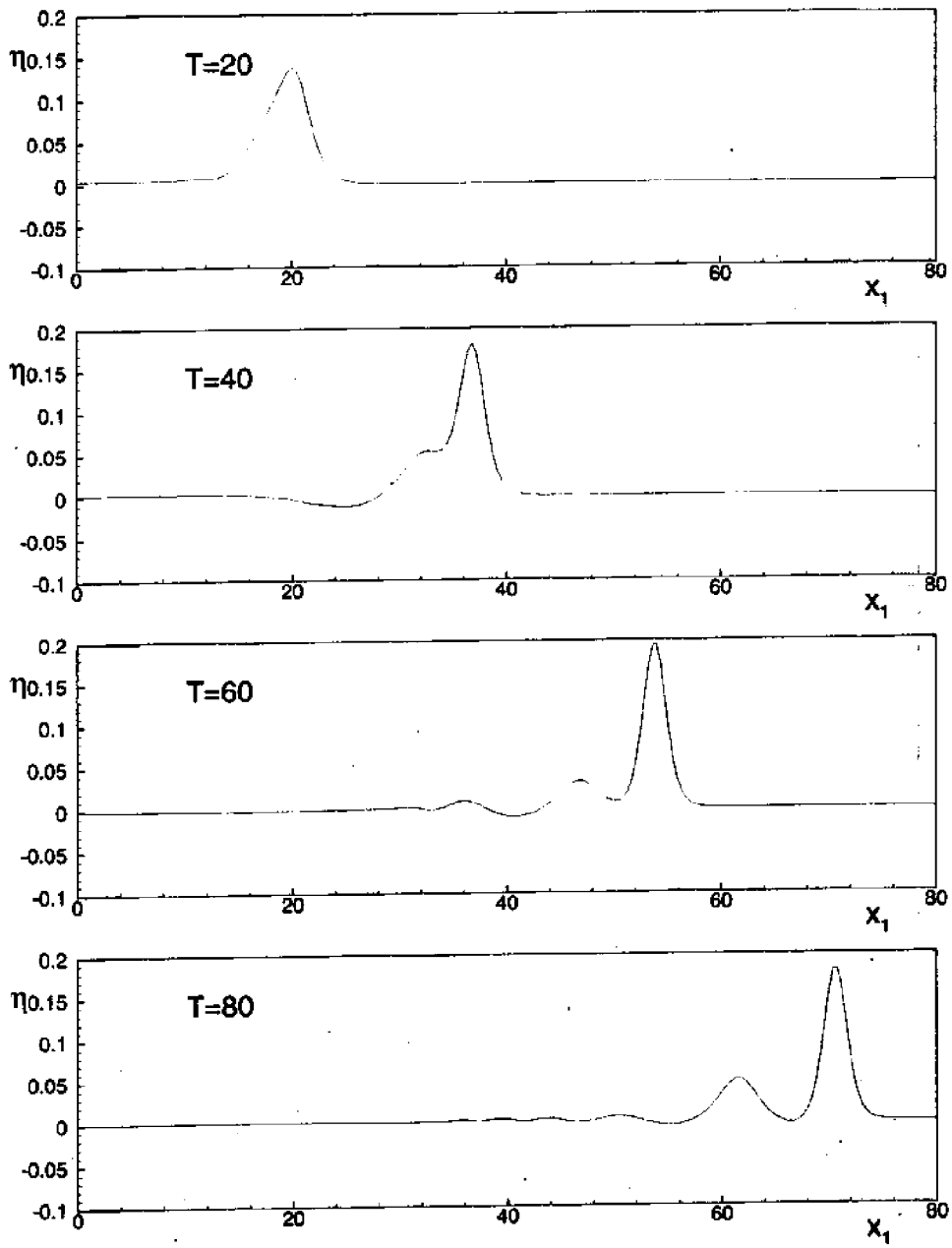

Figure 17: Wave profiles at wallcut for the curved-ramp case

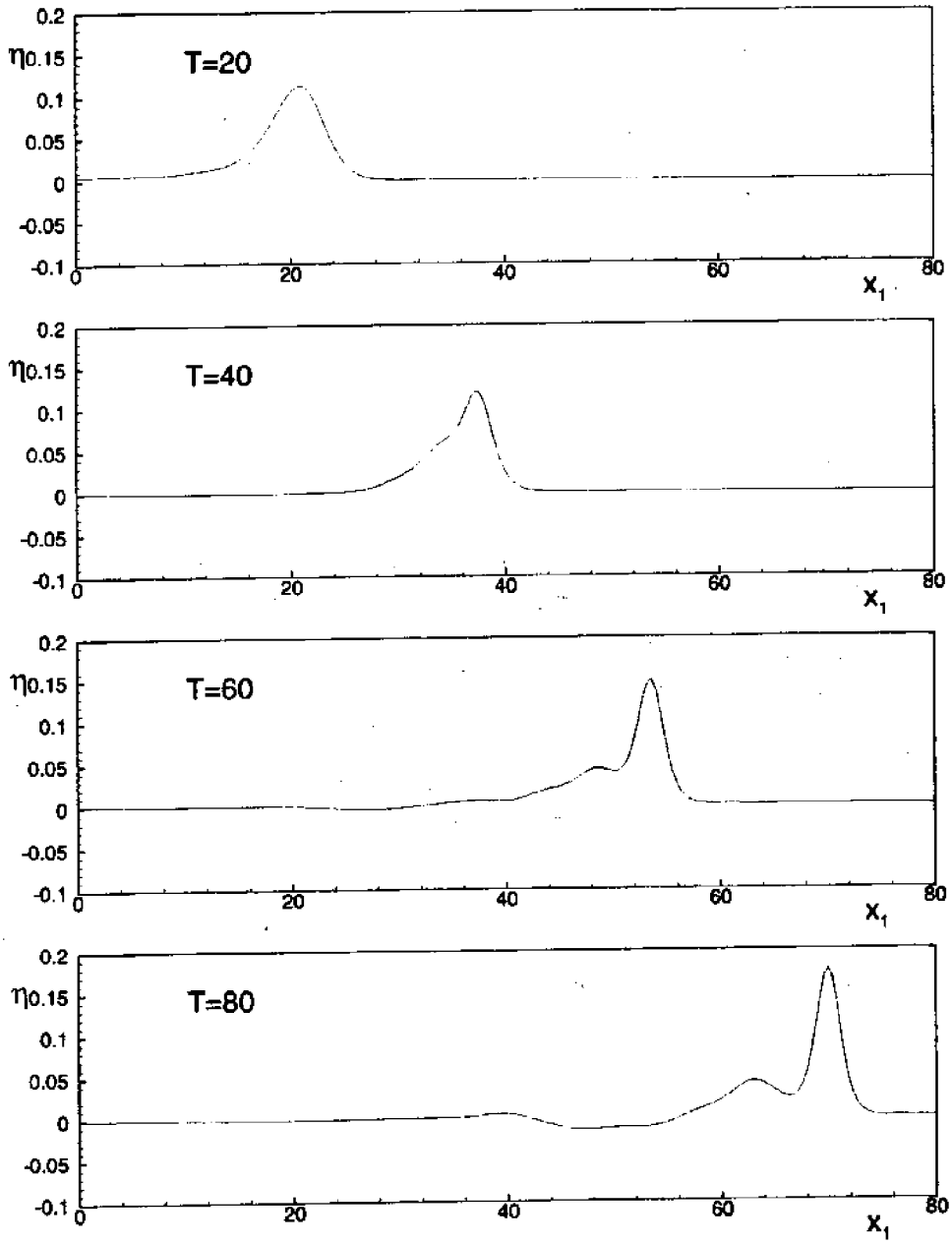

Figure 18: Wave profiles at centerline for the curved-ramp case

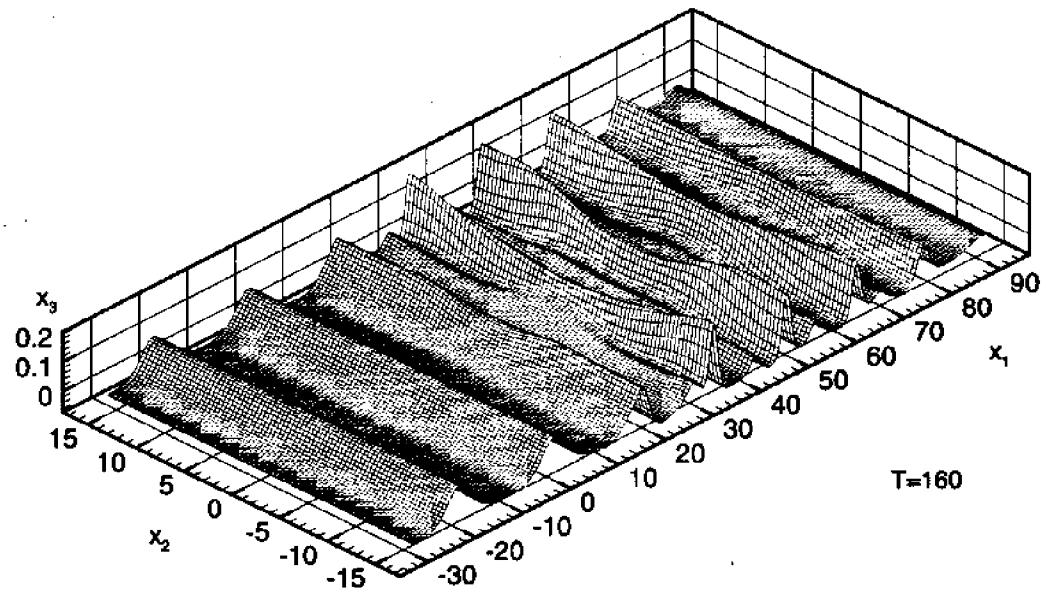

Figure 19: Cnoidal-wave profiles for the curved-ramp case with wall lateral-boundary condition

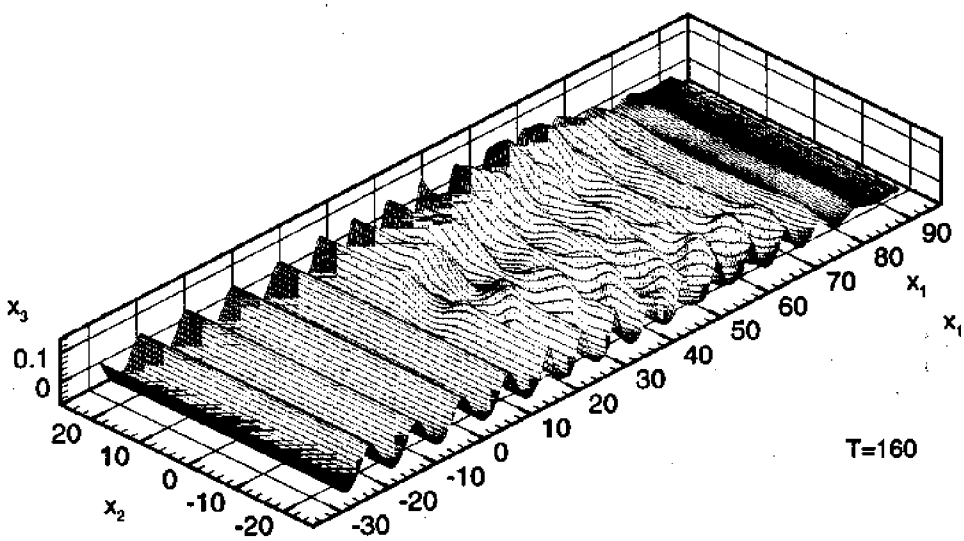

Figure 20: Cnoidal-wave profiles for the curved-ramp case with radiating lateral-boundary condition

was used. It can be seen that the surface elevations are close to normal in the case of Figure 19 due to the wall condition, while this is not the case in Figure 20. In Figs. 21 and 22, we show the contours of surface elevations for the same instant in time.

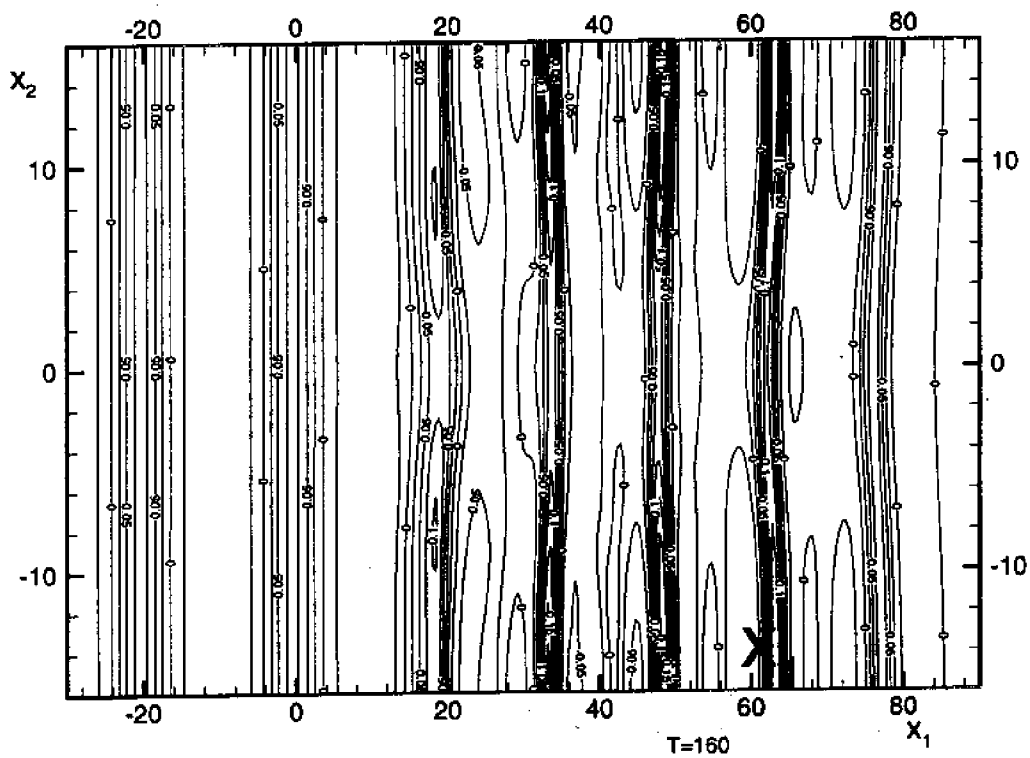

Figure 21: Cnoidal-wave contours for the curved-ramp case with wall lateral-boundary condition

9 Summary

The problem of wave propagation subject to varying bathymetry is considered using the Green-Naghdi equations. The theoretical formulations of the physical problem is presented. The governing equations are written in a simplified manner to facilitate the numerical imple-

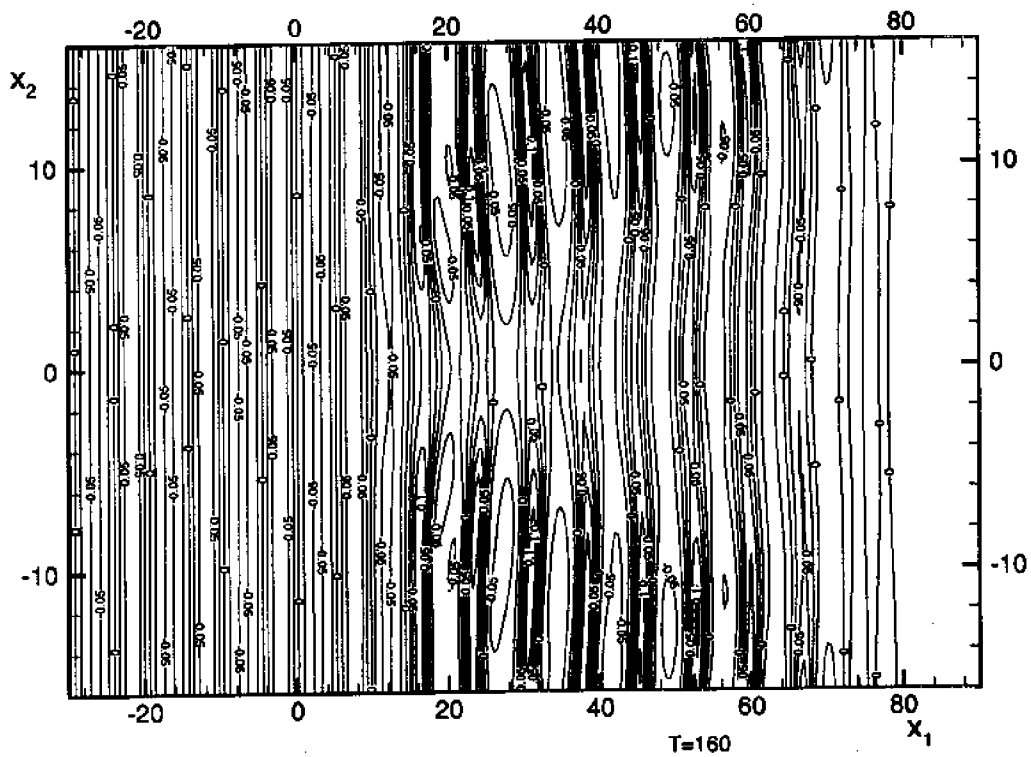

Figure 22: Cnoidal-wave contours for the curved-ramp case with radiating lateral-boundary condition

mentation. The finite-difference method in conjunction with the grid generation is adopted to spatially discretize the domain and the Modified-Euler method is used to perform time-integration. The special treatment of the boundary conditions have also been presented. The capabilities of the model have been demonstrated and validated by considering three cases: (a) ramp, (b) mount and (c) curved-ramp. Results indicate very good comparison with the previously known results.

Bibliography

[1] Ertekin, R. (1984), *Soliton Generation by Moving Disturbances in Shallow Water: Theory, Computation and Experiment*, Ph.D. thesis, University of California, Berkeley.

[2] Ertekin, R. C. and J. M. Becker (1996), Nonlinear diffraction of waves by a submerged shelf in shallow water, In *Proc. Offshore Mechanics and Arctic Engineering*, Volume Vol. I-A, pp. 105–112. ASME.

[3] Ertekin, R. C. and J. V. Wehausen (1987), Nonlinear diffraction of waves by a submerged shelf in shallow water, In *Proc. 16th Symp. on Naval Hydrodynamics*, Washington D. C., pp. 167–184. Berkeley Nat. Acad. Press.

[4] Green, A. and P. Naghdi (1977), Water waves in a non-homogeneous incompressible fluid, *Journal of Applied Mechanics Vol. 44*, 523–528.

[5] Mase, G. E. (1970), *Theory and Problems of Continuum Mechanics*, Schaum's Outline Series. McGraw Hill.

[6] Neill, D. (1996), *The Nonlinear Interaction of Waves with Multiple, Vertical, In-line Cylinders*, Ph.D. thesis, University of Hawaii at Manoa.

[7] Qian, Z. (1994), *Calculations of three dimensional nonlinear ship waves and ship resistance in a shallow water channel*, Ph.D. thesis, University of Hawaii at Manoa.

[8] Schember, H. R. (1982), *A new model for the three-dimensional nonlinear dispersive long waves*, Ph.D. thesis, California Institute of Technology.

[9] Yang, C. and R. C. Ertekin (1991), Numerical simulation of nonlinear wave diffraction by a vertical cylinder, In *Proc. Offshore Mechanics and Arctic Engineering*, Volume Vol. I-A, Stavanger, Norway, pp. 105–112. ASME.

Appendix A

gn3d and Utilities - User's Manual

1 Introduction

This manual describes the usage of the following programs:

gn3d - This is the main program which performs the simulation of wave diffraction due to varying bathymetry (See Sections 2-5).

gngrid - This program generates the data for the grid which becomes the input to gn3d (See section 6).

gndepth - This program generates the data for the depth which becomes the input to gn3d (See section 7).

gnwvemkr - This program generates the data for the incoming wave which becomes the input to gn3d (See section 8).

gn3dtec - This program generates tecplot data files from the binary output files of gn3d (See section 10).

2 gn3d Commands syntax

The programs require inputs in the form of batch files where commands are written one after the other with one command per line. The format of the command syntax defining

any general command is described below. The reader is referred to section 4 where a sample batch file is presented. The programs are executed at command line as follows:

```
$ gngrid wave.grd
$ gndepth wave.dep
$ gnwvemkr wave.wve
$ gn3d wave.gn
$ gn3dtec wave.tec
```

where wave.grd, wave.dep, wave.wve, wave.gn and wave.tec are the batch files for programs gngrid, gndepth, gnwvemkr, gn3d and gn3dtec respectively.

2.1 Command types

The main input to gn3d and utility programs is done through a batch file consisting of commands. The commands are of three types:

Comments

Any line that starts with '#' character is treated as comment.

Assignments

The parameters are assigned using an assignment statement of the following syntax:

```
<parameter name> = <parameter value>;
```

For example:

```
vel_filter_on = .TRUE.;
```

where `vel_filter_on` is the parameter name and `.TRUE.` is the value.

Execution statements

The execution statements are commands with arguments placed inside parenthesis. These are used to execute certain tasks and have the following syntax

```
command_name(arg1, arg2, arg3, ...);
```

For example

```
quad_bdry_gen(4001,6,0.,0.,400.,0.,400.,2.,0.,2.);
```

where quad_bdry_gen is the command that generates a quadrilatel boundary and the values 4001, 5, ... specify how this is to be performed. The arguments can be to type real, integer or character string and the number of arguments depends on the command.

2.2 Input data types

The parameter value will belong to one of the following data-types:

(real) The real numbers are to be entered either using a non-precision format such as 3.14159 or using double precision format such as 3.14159d0.

(integer) The integers numbers are to be entered as a series of digits.

(string) The strings are entered within double quotes, for example, as "Hello" in assignment statements and without quotes in execution statements.

(boolean) The boolean values are entered using .TRUE. and .FALSE just as in Fortran language.

3 Input commands of gn3d

The parameters used in gn3d are discussed in the following sections.

3.1 Filtering related inputs

Filtering and Successive over relaxation related inputs are described below:

Filter time step interval

The interval in terms of time-steps during which filtering is to be done is represented by `filter_time_step_interval (integer)`.

Example:

```
filter_time_step_interval = 1;
```

Filter velocity

The boolean parameter, `vel_filter_on (logical)`, determines whether filtering of velocity must be performed or not.

Example:

```
vel_filter_on = .TRUE.;
```

Filter surface elevation

The boolean parameter, `zeta_filter_on (logical)`, determines whether filtering of surface elevation must be performed or not.

Example:

```
zeta_filter_on = .FALSE.;
```

Depth smoothing

The real parameter, `depth_filter_weight (real)`, is used to control the filtering. A weight of 1.0 means that the importance to the value at a node is 100% compared to its neighboring nodes and, therefore, results in no filtering. Any value less than 1.0 can be used to achieve smoothing of the depth with a value of 1.0 representing zero smoothing and a value of 0 representing maximum.

Example:

```
depth_filter_weight = 0.6d0;
```

The integer parameter, `no_of_depth_filters (integer)`, is used apply smoothing repeatedly. A value of 10, for example, means that smoothing will be applied 10 times.

Example:

```
no_of\_depth_filters = 10;
```

Therefore, a combination `depth_filter_weight` and `no_of_depth_filters` can be used to control depth smoothing.

3.2 SOR related inputs

Filtering and Succcssive over relaxation related inputs are described below:

Maximum iterations

The maximum number of iterations that is permitted while performing Successive over relaxation (SOR) is specified using `max_no_of_iterations (integer)`.

Example:

```
max_no_of_iterations = 100;
```

SOR tolerances

Successive over relaxation is used to solve the Modified-Euler method which involves two steps. The tolerances for these two steps can be specified using `tolerance_1st_level (real)` and `tolerance_2nd_level (real)` respectively. In general, using the same tolerances for both levels is recommended.

Example:

```
tolerance_1st_level = 1.d-3;
tolerance_2nd_level = 1.d-4;
```

3.3 Input files

Binary input files are used to specify the grid, bathymetry and the incoming wave. The filenames of the input files are entered as described below:

Grid input file

The grid used for the simulation is input from the grid input file which is a FORTRAN binary file. The format of the file is described in section 6. The file name is input using the parameter, **grid_input_file (string)**.

Example:

```
grid_input_file = "./inputs/grid.dat";
```

Bathymetry input file

The bathymetry used for the simulation is input from the bathymetry input file which is a FORTRAN binary file. The format of the file is described in section 7. The file name is input using the parameter, **bathymetry_input_file (string)**.

Example:

```
bathymetry_input_file = "./inputs/depth.dat";
```

Wave input file

The wave used for the simulation is input from the wave input file which is a FORTRAN binary file. The format of the file is described in section 8. The file name is input using the parameter, **wave_input_file (string)**.

Example:

```
wave_input_file = "./inputs/wave.dat";
```

Initial condition input file

The initial condition used for the simulation is input from a FORTRAN binary file whose format is described in section 8. The file name is input using the parameter, `initcdn_input_file (string)`.

Example:

```
initcdn_input_file = "./inputs/initcdn.dat";
```

3.4 Output related inputs

The parameters related to outputs are described below:

Output time step interval

As the program undergoes time-marching, output of the crucial parameters such as velocity and surface elevations may be performed in a regular time-step intervals. The parameter, `output_time_step_interval (real)` can be used to specify this.

Example:

```
output_time_step_interval = 10;
```

Surface elevation output file

The surface elevations are stored in FORTRAN binary format into the file specified by the parameter `zeta_output_file (string)`.

Example:

```
zeta_output_file = "../outputs/zeta.out";
```

The details of the output format is presented in 5.3.

Mesh output file

The mesh or grid details are stored in FORTRAN binary format into the file specified by the parameter **mesh_output_file (string)**.

Example:

```
mesh_output_file = "../outputs/mesh.out";
```

Velocity output file

The velocity components are stored in FORTRAN binary format into the file specified by the parameter **velocity_output_file (string)**.

Example:

```
velocity_output_file = "../outputs/vel.out";
```

The details of the output format is presented in 5.3.

Case details output file

The case details are stored in text format into the file specified by the parameter **case_details_output_file (string)**.

Example:

```
case_output_file = "../outputs/case.out";
```

Numerical gages

Numerical gages can be installed to obtain time series of values at any location. The gage details are stored in text format into the file specified by the parameter **gage_output_file (string)**.

Example:

```
gage_output_file = "../outputs/gages.jnk";
```

The location of the gages can be specified using add_gage command. The right hand side of the assignment has two components with the first component being the x_1 location and second being the x_2 location of the gage.

Example:

```
add_gage = 100.d0, 200.d0;
```

Log file

A log of actions performed is stored into the log file specified by the parameter `log_output_file (string)`.

Example:

```
log_output_file = "../outputs/gn.log";
```

Delete old output files

Since the output files are new files, before specifying the output files, it is essential to ensure that files with the same paths and names do not exist on the system. The user can use the `rm` command to delete the files from the system. The user is cautioned that the files will be deleted with no questions asked.

Example:

```
rm "../outputs/case.jnk";
```

3.5 Execution statements

There are two simple commands to execute the program. The first one is `start` and the other is `end` These commands do not have any arguments and are, therefore, entered in a simple fashion as shown below.

Example:

```
# ----- Start the actual run -----
start;

# ----- End the program -----
end;
```

4 Sample commands file

To run a particular case, the batch commands are input into a batch file. This batch file becomes an argument to the executable at the command-line. For example,

```
$ gn wave.gn
```

will run the case described in the file "wave.gn". A sample set of commands which goes into such batch file are given below:

```
# ----- Open log file -----
rm "../../outputs/mount/gn.log";
log_output_file = "../../outputs/mount/gn.log";

# ---- Delete old files ---
rm "../../outputs/mount/gages.jnk";
rm "../../outputs/mount/case.jnk";
rm "../../outputs/mount/zeta.jnk";
rm "../../outputs/mount/vel.jnk";
rm "../../outputs/mount/mesh.jnk";
rm "../../outputs/mount/incwave.jnk";

# ---- Input variables ----
case_description = "Test Case";
case_number = 1000;
```

```
max_no_of_iterations = 300;
filter_time_step_interval = 1;
vel_filter_on = .TRUE.;
zeta_filter_on = .TRUE.;
tolerance_1st_level = 1.d-4;
tolerance_2nd_level = 1.d-4;
sor_relax_factor = 0.6d0;

output_time_step_interval = 500;
no_of_time_steps = 8000;
grid_input_file = "../../tests/mount/grid.jnk";
bathymetry_input_file = "../../tests/mount/depth.jnk";
wave_input_file = "../../tests/mount/wave.jnk";
initcdn_input_file = "../../tests/mount/initcdn.jnk";
no_of_depth_filters = 3;
depth_filter_weight = 0.9;

gage_output_file = "../../outputs/mount/gages.jnk";
add_gage = 199.7d0, 2.5d0;
add_gage = 205.1d0, 2.5d0;
add_gage = 208.1d0, 2.5d0;
add_gage = 209.5d0, 2.5d0;
add_gage = 213.7d0, 2.5d0;

# ----- Set output files -------
zeta_output_file = "../../outputs/mount/zeta.jnk";
velocity_output_file = "../../outputs/mount/vel.jnk";
case_output_file = "../../outputs/mount/case.jnk";
mesh_output_file = "../../outputs/mount/mesh.jnk";
incoming_wave_output_file = "../../outputs/mount/incwave.jnk";

# ----- Start the actual run -----
```

```
start;

# ----- End the program -----
end;
```

5 Outputs

The outputs of gn3d are presented in this section.

5.1 Case details

All the small details of the case is sent to the file specified by the input parameter `case_output_file`.

The data is written using sequential unformatted fashion which is described in below:

```
write (UNIT = zeta_output_unit)  itstep
write (UNIT = zeta_output_unit)
&  ((zeta(i, j), i = 1, n1), j = 1, n2)
```

where `itstep` is the time step number and `zeta` is the surface elevation.

5.2 Surface elevations

The surface elevations at a time step specified by the time step interval parameter, `output_time_step_interval`, is saved in FORTRAN binary file specified through the input parameter `zeta_output_file`.

The data is written using sequential unformatted fashion which is described in below:

```
write (UNIT = mesh_output_unit)
&  n1, n2, ((x1(i, j), i = 1, n1), j = 1, n2)
write (UNIT = mesh_output_unit)
&  n1, n2, ((x2(i, j), i = 1, n1), j = 1, n2)
```

```
      write (UNIT = mesh_output_unit)
     &  n1, n2, ((h(i, j), i = 1, n1), j = 1, n2)
```

where `itstep` is the time step number and `(x1,x2)` are the coordinates of the nodal point at `(i,j)`.

5.3 Velocity components

The velocity components at a time step specified by the time step interval parameter, `output_time_step_interval` is saved in FORTRAN binary file specified through the input parameter `zeta_output_file`.

The data is written using sequential unformatted fashion which is described in below:

```
      write (UNIT = vel_output_unit, ERR = 110) itstep
      write (UNIT = vel_output_unit, ERR = 110)
     &   ((u1(i, j), i = 1, n1), j = 1, n2),
     &   ((u2(i, j), i = 1, n1), j = 1, n2)
```

where `itstep` is the time step number and `(u1,u2)` are the velocity components.

6 gngrid - Elliptical grid generation

The program `gngrid` is used to generate the mapping between the physical domain and the computational domains. The commands supported by `gngrid` are presented below.

6.1 Commands

Load boundary data

The command `load_bdry(file_name)` takes the file name containing the boundary data as the argument. Internally, the boundary data is read using the following code:

```
read (UNIT=10, FMT=*) m, n
read (UNIT=10, FMT=*) (x(1, i), i = 1, m)
read (UNIT=10, FMT=*) (y(1, i), i = 1, m)
read (UNIT=10, FMT=*) (x(n, i), i = 1, m)
read (UNIT=10, FMT=*) (y(n, i), i = 1, m)
read (UNIT=10, FMT=*) (x(j, 1), j = 1, n)
read (UNIT=10, FMT=*) (y(j, 1), j = 1, n)
read (UNIT=10, FMT=*) (x(j, m), j = 1, n)
read (UNIT=10, FMT=*) (y(j, m), j = 1, n)
```

where m and n are the two dimensions of the grid and x and y represents the two coordinates at all boundary nodes.

Example:

```
load_bdry_data(bdry.dat);
```

Generate quadrilatrral boundary

A quadrilateral shaped domain can be generated using quad_bdry_gen.

Syntax:

```
quad_bdry_gen(nx, ny, x1, y1, ..., x4, y4);
```

where nx and ny are the number of nodes along x and y directions. x1, y1, etc. represents the coordinates of the four corners of the quadrilateral.

Example:

```
quad_bdry_gen(4001,6,0.,0.,400.,0.,400.,2.,0.,2.);
xo
```

Generate initial guess grid

The command `generate_initial_grid ()` is simply called to generate the guess mesh before grid generation begins. This command has no arguments.

Example:

```
generate_initial_grid();
```

Generate curvilinear grid

To generate the curvilinear grid, the command `curvilinear_gridgen ()` must be issued. This command has no arguments.

Example:

```
generate_curvilinear_grid();
```

Find maximum grid spacing

Sometimes, it is useful to determine the maximum grid spacing. This can be done by calling `find_max_spacing ()`. This command has no arguments.

Example:

```
find_maximum_grid_spacing();
```

Delete a file

Since the output files are new files, before specifying the output files, it is essential to ensure that files with the same paths and names do not exist on the system. The user can use the `rm (file_name)` command to delete the files from the system. The user is cautioned that the files will be deleted with no questions asked.

Example:

```
rm(../outputs/case.jnk);
```

Save the grid in gn3d binary format

The generated grid can be saved in the following binary format using the command: `save_grid_for_gn3d (file_name)`.

```
write(UNIT = 10) m, n
write(UNIT = 10) ((x(j, i), i = 1, m) , j = 1, n),
      ((y(j, i), i = 1, m) , j = 1, n)
```

This format is understood by gn3d and thus, this output file becomes input file for gn3d.

Example:

```
save_grid_for_gn3d(grid.bin);
```

Save the grid in tecplot format

The generated grid can be saved in tecplot binary format using the command: `save_grid_for_tecplot (file_name)` This format is understood by `Tecplot` and thus, this output file becomes input file for `Tecplot`.

Example:

```
save_grid_for_tecplot(grid.plt);
```

Save SOR relaxation coefficients

Along with grid generation, gn3d can also generate relaxation coefficients for SOR. These can be saved into a file using `save_sor_relax_coefs (file_name)`.

Example:

```
save_sor_relax_coeffs(relax.jnk);
```

Display this help message

The help command, `help ()`, displays the syntax of all the commands of **gngrid**. This command has no arguments.

Example:

```
help();
```

Quit the program

To quit, simply call `quit ()`.

Example:

```
quit();
```

6.2 Sample commands file

This is a sample commands file for **gngrid**.

```
quad_bdry_gen(4001,6,0.,0.,400.,0.,400.,2.,0.,2.);
generate_initial_grid();
curvilinear_gridgen();
rm(grid.plt);
rm(grid.jnk);
save_grid_for_gn3d(grid.bin);
save_grid_for_tecplot(grid.plt);
quit();
```

7 gndepth - Bathymetry generation

gndepth is a utility program that can be used to generate the bathymetry inputs for **gn3d**. The commands supported by **gn3d** are given below.

Load grid

To generate a bathymetry, one must load a grid file so that the depth at the nodal point locations can be determined. This is achieved using `load_grid (grid_input_file)`. The grid input file is of the same format as grid input file for **gn3d**. See section 6 for more details.

Example:

```
load_grid(grid.jnk);
```

Load bathymetry

Typically, the bathymetry data is available a random locations. This means that some kind of interpolation must be performed to obtain the depths at the nodal points. The idea adopted in **gndepth** is as follows:

1. Obtain depths at random locations,
2. Using programs that can triangulate (Tecplot, matlab, etc.) obtain a finite element mesh made of linear triangle, and
3. Use this finite element mesh and linear shape functions to interpolate and determine depth at any point inside the domain.

Therefore, a finite-element mesh containing the bathymetry data is input to **gndepth** and must be loaded using the command, `load_bathymetry (bathymetry_input_file)`. Internally, this command reads binary data using:

```
read(UNIT = 10) nmnpb, nmelb
print *, 'Reading coordinates...'
read(UNIT = 10) (xb(i), yb(i), zb(i), i=1, nmnpb)
print *, 'Reading elements...'
read(UNIT = 10) (ielnmb(1, i), ielnmb(2, i), ielnmb(3, i), i=1, nmelb)
```

where **nmnpb** is number of nodes, **nmelb** number of elements, **xb** and **yb** coordinates of the nodal points and **ielnmb** represents connectivity.

Example:

```
load_bathymetry_input_file(grid.jnk);
```

Convert latitide-longitude to meters

When digitizing bathymetry using nautical maps, a coordinate transformation between latitude-longitude system to meter system must be performed. The command `convert_lat_long_to_meters ()` performs this transformation. This command has no arguments.

Example:

```
convert_lat_long_to_meters();
```

Determine depths at nodes on grid

To determine the depths at the node locations using interpolation of finite-element mesh with linear triangles, the command `find_depths_at_grid_nodes ()` can be used. This command has no arguments.

Example:

```
find_depths_at_nodes();
```

Generate ramp-like shelf bathymetry

gndepth can produce a ramp-like bathymetry using the command, `generate_shelf_bathymetry (h1, h2, xstart, xend)`, where `h1` and `h2` are the depths at two sides of the ramp and `xstart` and `xend` are the starting and ending points of the ramps.

Example:

```
generate_shelf_bathymetry(10.d0, 7.d0, 40.d0, 50.d0);
```

Generate submerged shelf bathymetry

A submerged-mount type bathymetry can be generated using **generate_mount_bathymetry (h1, h2, x1, x2, x3, x4)** where **h1** and **h2** are the depths at outside and inside the mount respectively. **x1, x2, x3** and **x4** are the **x**-locations of the four points that represents the mount.

Example:

```
generate_mount_bathymetry(1.d0, .3d0, 203.d0, 205.d0, 208.d0, 209.d0);
```

Save bathymetry for gn3d

The generated bathymetry can be saved in the following binary format using **save_grid_bathymetry_for_gn3d (bathymetry_output_file)**:

```
write(UNIT = 10) m, n
write(UNIT = 10) ((z(i,j), i=1,m), j=1,n)
```

where **m** and **n** are dimensions of the grid and **z** is the depth values at the nodes of the grid.

This format is understood by **gn3d** and thus, this output file becomes input file for **gn3d**.

Example:

```
save_grid_bathymetry_for_gn3d(depth.bin);
```

Save grid bathymetry for tecplot

The generated bathymetry can be saved in the **Tecplot** binary format using **save_grid_bathymetry_for_tecplot (bathymetry_tec_file)**

This format is understood by **Tecplot** and thus, this output file becomes input file for **Tecplot**.

Example:

```
save_grid_bathymetry_for_tecplot(depth.plt);
```

Display this help message

The help command, `help ()`, displays the syntax of all the commands of **gndepth**. This command has no arguments.

Example:

```
help();
```

Quit the program

To quit, simply call `quit ()`.

Example:

```
quit();
```

7.1 Sample commands file

This is a sample commands file for **gndepth**.

```
load_grid(grid.jnk);
generate_mount_bathymetry(1.d0, .3d0, 203.d0, 205.d0, 208.d0, 209.d0);
rm(depth.jnk);
rm(depth.plt);
save_grid_bathymetry_for_gn3d(depth.jnk);
save_grid_bathymetry_for_tecplot(depth.plt);
quit;
```

8 gnwvemkr - Numerical Wave maker

The numerical wave maker has the capability generating cnoidal and solitary waves. The program is prompt-driven and commands are entered by the user interactively.

Amplitude

The amplitude of the incoming wave is specified using `set_amplitude` which takes the value of the amplitude (`real*8`) as argument.

Example:

```
set_amplitude(0.01);
```

Wavelength

The amplitude of the incoming wave is specified using `set_wave_length` which takes the value of the wave length (`real*8`) as argument.

Example:

```
set_wave_length(10.d0);
```

Wave modulation factor

The incoming wave can be modulated. The modulation factor is specified using `set_wave_modulation_factor` with the value of the factor (`real*8`) as argument.

Example:

```
set_wave_modulation_factor(1.d0);
```

Number of time steps

The total number of time steps needed to simulate the time duration of intereset is set using the command, `set_no_of_time_steps`.

Example:

```
set_no_of_time_steps(10.d0);
```

Time step

The time step interval is set using the command **set_time_step** with the value of time step (**real*8**) as argument.

Example:

```
set_time_step(0.01d0);
```

Gravity

The gravititational acceleration is set using the command **set_gravity** with the value (**real*8**) as argument. This value determines the unit-system used in the program. Consistent units must be used for all inputs.

Example:

```
set_gravity(9.81d0);
```

Depth

set_depth (real)

The depth value at the incoming boundary is set using the command **set_depth** with the value (**real*8**) as argument.

Example:

```
set_depth(10.0d0);
```

Lateral condition

Two types of lateral conditions are possible (1) Wall condition (2) radiation condition. There are two lateral boundaries, one on the left of the incoming wave direction and the other on the right. These are specified using **set_lateral_cdn (integer, integer)**. A value of 1 represents wall condition and a value of 2 represents radiation condition.

Example:

```
set_lateral_cdn(1,1);
```

Shift along x

```
set_x_shift (real)
```

Example:

```
set_x_shift(-10.d0);
```

x at incoming boundary

The value of `x` at the incoming boundary is set using `set_x_at_incoming_bdry` with a `(real*8)` value as the argument. This value is used to generate initial wave profile. In some cases, we may start the simulation with wave already (partially) inside the domain.

Example:

```
set_x_at_incoming_bdry(-10.d0);
```

x at outgoing boundary

The value of `x` at the outgoing boundary is set using `set_x_at_outgoing_bdry` with a `(real*8)` value as the argument. This value is used to generate initial wave profile. In some cases, we may start the simulation with wave already (partially) inside the domain.

Example:

```
set_x_at_going_bdry(-10.d0);
```

Grid spacing

The value of grid spacing is set using `set_delta_x_` with a `(real*8)` value as the argument. In some cases, we may start the simulation with wave already (partially) inside the domain and this value is used to generate initial wave profile.

Example:

```
set_delta_x(0.02d0);
```

Incoming wave

The incoming wave can be generated by calling `generate_incoming_wave ()`. This command has no arguments.

Example:

```
generate_incoming_wave();
```

Wave profile

`generate_wave_profile ()`

The initial wave profile can be generated by calling `generate_wave_profile ()`. This command has no arguments.

Example:

```
generate_wave_profile();
```

Save wave

The incoming wave and other pertinent info can be save in **gn3d** binary format using `save_incoming_wave_for_gn3d (file_name)`. This format is understood by **gn3d** and thus, this output file becomes input file for **gn3d**.

Example:

```
save_incoming_wave_for_gn3d(wave.jnk)
```

The incoming wave can be saved in Tecplot binary format using `save_incoming_wave_for_tecplot (file_name)`. This format is understood by Tecplot and thus, this output file becomes input file for Tecplot.

```
save_incoming_wave_for_tecplot ()
```

Example:

```
save_incoming_wave_for_tecplot(wave.plt)
```

The initial wave profile can be saved in gn3d binary format using `save_wave_profile_for_gn3d (file_name)`. This format is understood by gn3d and thus, this output file becomes input file for gn3d as the initial condition for the case.

Example:

```
save_wave_profile_for_gn3d(initcdn.jnk)
```

The initial wave profile can be saved in Tecplot binary format using `save_wave_profile_for_tecplot (file_name)`. This format is understood by Tecplot and thus, this output file becomes input file for Tecplot.

Example:

```
save_wave_profile_for_tecplot(initcdn.plt)
```

9 Sample commands file

This is a sample commands file for gnwvemkr.

```
set_gravity(1.d0)
set_depth(1.d0)
set_wave_height(0.1d0)
set_wave_length(11.35d0)
set_x_shift(0.d0)
set_x_at_incoming_bdry(0.d0)
set_wave_modulation_factor(1.d0)
load_grid(grid.jnk)
set_no_of_waves(60)
set_time_step(0.07d0)
generate_incoming_wave();
rm(wave.plt)
rm(wave.jnk)
save_incoming_wave_for_tecplot(wave.plt)
save_incoming_wave_for_gn3d(wave.jnk)
rm(initcdn.jnk)
save_wave_profile_for_gn3d(initcdn.jnk)
quit
```

10 gn3dtec - Tecplot file generation

gn3dtec is a utility that converts Fortran binary output from gn3d to Tecplot format.

11 Commands

11.1 Save mesh

The command mesh2tec can be used to convert the mesh output file from gn3d to tecplot format. The syntax is mesh2tec(gn3dmesh.out, mesh.plt); where gn3dmesh.out is the gn3d mesh-output file and mesh.plt is the Tecplot file.

Example:

```
mesh2tec(mesh.out, mesh.plt);
```

11.2 Save velocity

The command `vel2tec` can be used to convert the velocity output file from gn3d to tecplot format. The syntax is `vel2tec(gn3dvel.out, mesh.out, datasetnum1, datasetnum2, vel.plt);` where `gn3dvel.out` is the gn3d velocity-output file, `gn3dmesh.out` is the gn3d mesh-output file and `vel.plt` is the Tecplot file. `datasetnum1` and `datasetnum2` represent the starting and ending numbers of the datasets that are to be saved, respectively.

Example:

```
vel2tec(vel.out, mesh.out, 1, 10, vel.plt);
```

11.3 Save surface elevation

The command `zeta2tec` can be used to convert the surface elevation output file from gn3d to tecplot format. The syntax is `zeta2tec(gn3dzeta.out, mesh.out, datasetnum1, datasetnum2, data_step, zeta.plt);` where `gn3dzeta.out` is the gn3d surface elevation-output file and `zeta.plt` is the Tecplot file. `datasetnum1`and `datasetnum2` represent the starting and ending numbers of the datasets that are to be saved, respectively. `data_step` is the dataset-interval at which the datasets are to be saved

Example:

```
zeta2tec(zeta.out, mesh.out, 1, 10, 2, zeta.plt);
```

11.4 Save gage surface elevation

The command `gage2tec` can be used to convert the gage surface elevation output file from gn3d to tecplot format. The syntax is `gage2tec(gn3dgage.out, datasetnum, gage.plt);` where `gn3dgage.out` is the gn3d gage surface elevation-output file and `gage.plt` is the Tecplot file. `datasetnum` represents the number of the dataset that is to be saved.

Example:

```
gage2tec(gage.out, 10, gage.plt);
```