Part I

An SWE solver for use on the CYBER 205

1. Introduction

In hydraulic engineering, the shallow-water equations (SWEs) are used to describe flows in shallow seas, estuaries and rivers. Numerical models based on these SWEs can be used to determine the influence of infrastructural works on the flow. Furthermore, output from these models can be used to calculate salt intrusion, the effect of waste discharges, water quality parameters, cooling water recirculation and sediment transports. An important application, in the Netherlands, is the storm surge barrier in the mouth of the Eastern Scheldt (Oosterschelde) estuary, by which this estuary can be separated from the sea during storms. In this case, a numerical model, based on the SWEs, was extensively used in the development phase of the barrier. Furthermore, after the installation, a similar numerical model provides guide lines for the operation of the barrier, not only to protect the dikes along the border of the Eastern Scheldt, but also in order to preserve the delicate ecological balance in the estuary, which has an important fish nursery as well as oyster and mussel cultures.

The nature of the applications is such that strong gradients in the solution are common, though shocks do not appear. As a consequence, it is not strictly necessary to satisfy numerically conservation of momentum or energy. (These conservation properties are indispensable for the approximation of physical shocks [23, 33].) However, the conservation of mass is important as the local amount of mass is directly connected to the depth and the latter determines largely the propagation of the waves (see [38, p. 155] and [39]). Moreover, using the model for the calculation of the dispersion of dissolved matter, mass conservation is even more needed in order to prevent loss of matter.

In the following we will briefly describe the contents of each section.

In Section 2, the problem is described, i.e. the equations, the domain, the boundary conditions and the initial values. Many of these topics are already treated by other authors (e.g. [3, 24, 38]), but it is briefly summarized for completeness. In addition, in Section 2.3.2, we propose some new boundary conditions for the SWEs in the viscid case.

Various aspects of the numerical algorithm are discussed in Section 3. With respect to the space discretization, attention will be given to the assumptions near the boundaries. Furthermore, the time discretization and its stabilization are treated. The latter will be discussed in more detail with respect to its application to the SWEs. Finally the drying and flooding procedure is described.

Section 4 is devoted to the vectorization aspects of the CYBER 205 code. The various techniques which were used to construct an efficient code are presented in detail.

The components of the developed software and their actual use are discussed in Section 5.

In Section 6, results are given of some computations for complex geometries. To obtain these results either our own system or the WAQUA system has been used. In the latter case interfaces were made such that our computational routines could replace those of Stelling in WAQUA. This enabled us to test the code on real engineering problems.

2. Problem description

2.1. The equations

In this section, the equations are given and it will be briefly indicated how they are derived from the Navier-Stokes equations. Consider Figure 2.1, where a vertical cross section of a flow field is drawn,

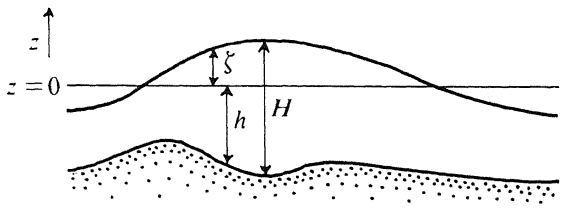


FIGURE 2.1. Vertical cross section of a flow field.

and let $z = 0$ be a reference plane, which is, for example, the mean sea level. With respect to this reference plane, we define the local bottom profile by $-h(x,y)$ and the local elevation by $\zeta(x,y,t)$; the total depth is then given by

$H = h + \zeta$. The SWEs can be derived from the Navier-Stokes equations in a few steps (see [3, p. 190]). First, the Navier-Stokes equations are simplified by assuming hydrostatic pressure and incompressibility of water. Then, the resulting equations are integrated over the total depth, where the vertical boundary conditions follow from the assumptions that the bottom as well as the water surface are stream surfaces. The integrated equations are expressed as far as possible in terms of the depth integrated horizontal velocities. Furthermore, for the stress along the bottom an empirical formula is substituted and the turbulent velocity fluctuations and the dispersion due to the non-uniform vertical distribution of the horizontal velocities are represented by viscosity (see [22, 8]). The resulting equations read

$$\begin{aligned} u_t &= -uu_x - vu_y - g\zeta_x + fv - \frac{g}{C^2}\sqrt{u^2+v^2}\,u \,/\, H + A\Delta u + F^u, \\ v_t &= -uv_x - vv_y - g\zeta_y - fu - \frac{g}{C^2}\sqrt{u^2+v^2}\,v \,/\, H + A\Delta v + F^v, \\ \zeta_t &= -(Hu)_x - (Hv)_y + F^\zeta. \end{aligned} \tag{2.1}$$

The first two equations are momentum equations describing, in this incompressible case, the change in time of the depth-averaged velocities u and v. The third one is a continuity equation. In the momentum equations appear the Coriolis force parametrized by f, which is due to the rotation of the earth, and the bottom friction parametrized by C (Chezy coefficient). Furthermore, g and A respectively denote the acceleration due to gravity and the viscosity coefficient for horizontal momentum exchange. F^u and F^v are external forcing functions such as wind stress or barometric pressure and F^ζ represents a source of water or a sink. The last is used in the model of the Eems-Dollard estuary described in Section 6.2. In this model, it represents the discharges of some rivers into the estuary. More details on these parameters can be found in [3].

2.2. The domain

The domain for these equations is to a large extent arbitrary. An example is drawn in Figure 2.2.

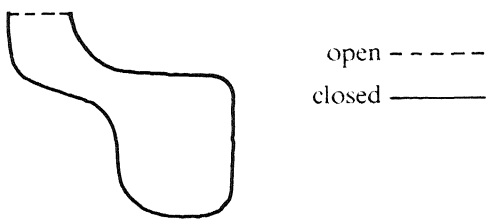


FIGURE 2.2. Example of a domain.

The contour of the domain consists of parts along "land-water" lines (e.g.,

river banks or coast lines), which are called *closed boundaries*, and parts across the flow field, which are called *open boundaries*. The latter are artificial boundaries that have been chosen judiciously across the flow field in order to restrict the size of the domain (see Section 2.3.2). However, due to assumptions near these open boundaries, it is advised to choose these boundaries far from the region of interest.

2.3. The boundary conditions

As said in the previous section, there are two types of boundaries to be distinguished: closed boundaries along "land-water" lines and open boundaries across the flow field. In this section, we present boundary conditions for both cases.

2.3.1. Closed boundaries. Let (.,.) define an inner product. Then at closed boundaries we have the conditions (see Stelling [38])

$$(\mathbf{v},\mathbf{n})=0, \tag{2.2}$$

$$(1-\alpha)(\mathbf{v},\mathbf{s})-\alpha(\nabla(\mathbf{v},\mathbf{s}),\mathbf{n})=0 \quad \text{for} \quad A\neq 0, \tag{2.3}$$

where $\mathbf{v}=[u,v]^{\mathrm{T}}$ and $\mathbf{s}$ and $\mathbf{n}$ respectively are the local tangential unit vector (direction counter clock wise) and the normal unit vector (direction inward) at the boundary. Physically, condition (2.2) describes that there is no mass flow through the boundary. Furthermore, condition (2.3) represents partial slip along the closed boundary. This partial-slip condition becomes important when the mesh size used in the numerical model is smaller than the thickness of occurring boundary layers in the flow (see e.g. [29]). The amount of "slip" is parametrized by α. For the special cases $\alpha=1$ and $\alpha=0$ this is a "perfect slip" and a "no slip" boundary condition, respectively. In general $\alpha=1$, i.e. the mesh size is much larger than the boundary layers.

2.3.2. Open boundaries. The open boundaries are artificial "water-water" boundaries. In general, the conditions at these boundaries consist of combinations of $(\mathbf{v},\mathbf{n})$, $(\mathbf{v},\mathbf{s}),\zeta$, $(\nabla(\mathbf{v},\mathbf{n}),\mathbf{n})$, $(\nabla(\mathbf{v},\mathbf{s}),\mathbf{n})$, $(\nabla\zeta,\mathbf{n})$, $\mathbf{v}_t$ and ζ_t. The data needed for the conditions are usually obtained from measurements or from a model which encloses the model at hand. In practice, it appears to be more difficult to measure accurately the velocity than the elevation. As a consequence velocity data are mainly used for the boundary conditions if the model at hand is nested in a larger model.

For the purely hyperbolic case ($A=0$) it is known that at an inflow boundary ($(\mathbf{v},\mathbf{n})>0$) two boundary conditions are needed, whereas at an outflow boundary ($(\mathbf{v},\mathbf{n})\leqslant 0$) only one boundary condition is required [29]. In the incompletely parabolic case [42] ($A\neq 0$), we need at each boundary one extra condition.

2.3.2.1. The inviscid case ($A=0$). Usually, at open boundaries the normal velocity $(\mathbf{v},\mathbf{n})$ or the elevation is prescribed. Moreover, the tangential velocity $(\mathbf{v},\mathbf{s})$

is prescribed if $(\mathbf{v},\mathbf{n})>0$. In our model, we use the modification of these conditions as proposed by Stelling [38], which are weakly-reflective for short wave components in the solution. At a "velocity boundary", i.e. a boundary where the velocity is prescribed, we specify the value of

$$(\mathbf{v},\mathbf{n})+\gamma\frac{\partial}{\partial t}R \tag{2.4}$$

and at an "elevation" boundary we specify the value of

$$\zeta+\gamma\frac{\partial}{\partial t}R. \tag{2.5}$$

Here,

$$R=(\mathbf{v},\mathbf{n})+2\sqrt{gH} \tag{2.6}$$

denotes the so-called ingoing Riemann invariant. Furthermore, in both cases we prescribe the value of the tangential velocity

$$(\mathbf{v},\mathbf{s}) \text{ if } (\mathbf{v},\mathbf{n})>0. \tag{2.7}$$

The prescription of the value of the expressions (2.4) and (2.5) needs some explanation. In these expressions, the time derivative of the ingoing Riemann invariant is introduced [29, 4, 15, 5, 6, 10], because including these Riemann invariants into the boundary conditions has the effect that these boundary conditions become weakly reflective for short wave components (see [45] and [38, p. 153]). These short wave components originate mainly from the initial condition and the eigenfrequencies of the model. If these Riemann invariants are not used, then these short wave components may disturb the solution for a long time as there is, in general, little dissipation in the model. When the value of R is not known, then (2.4) and (2.5) can still be used if the parameter γ is chosen such that after the start-up period (see Section 2.4) the expression $\gamma\partial R/\partial t$ is small with respect to the magnitude of the normal velocity in the case of (2.4) or with respect to the magnitude of elevation in the case of (2.5). We will derive these Riemann invariants for the simplified one-dimensional case. Consider the one-dimensional equations

$$\begin{aligned} u_t &= -uu_x - g\zeta_x, \\ \zeta_t &= -(Hu)_x, \end{aligned} \tag{2.8}$$

which are identical to (recall that $\zeta=H-h$)

$$\begin{aligned} u_t &= -uu_x - gH_x + gh_x, \\ H_t &= -Hu_x - uH_x. \end{aligned} \tag{2.8'}$$

Multiplying the second equation with $\sqrt{g/H}$ and adding and subtracting the equations, we obtain

$$(u\pm 2\sqrt{gH})_t = -(u\pm\sqrt{gH})\,(u\pm 2\sqrt{gH})_x + gh_x, \tag{2.9}$$

or, introducing $R^{\pm}=u\pm 2\sqrt{gH}$,

$$R_t^{\pm} = -(u \pm \sqrt{gH}) R_x^{\pm} + g h_x. \tag{2.9'}$$

These equations express that the solution of (2.8) can be described by two waves moving in opposite directions with propagation speeds $u \pm \sqrt{gH}$. Notice that, in this one-dimensional case, we have at the left boundary $R = R^+$ and at the right boundary $R = -R^-$, where R is defined by (2.6). Suppose that the Riemann invariants are available at the boundaries. Then by prescribing R^+ and R^- at the left and right boundary, respectively, we are led to a non-reflective boundary treatment. In the two-dimensional case these conditions can also be used but they yield only in very special cases a non-reflective boundary treatment, i.e. if the flow is normal to the boundary and if the Coriolis force and the bottom friction are negligible. Nevertheless, in practice the flow is often very "close" to such a special case and consequently the weakly-reflective properties of these conditions are still substantial. It should be mentioned that Verboom and Slob [45] derived boundary conditions with improved weakly-reflective properties. Currently, this type of boundary treatment is implemented in and tested for the WAQUA system (see [27]). Awaiting the results of this implementation, we used the weakly-reflective boundary conditions (2.4) and (2.5) as proposed by Stelling.
The well-posedness of the SWEs using these boundary conditions is treated by Verboom et al. [46].

2.3.2.2. The viscid case ($A \neq 0$). As already mentioned, in the viscid case at each boundary one extra condition is needed. Oliger and Sundström [29] propose to prescribe the value of the following expressions:
At an inflow boundary (R as defined by (2.6)):

$$R \tag{2.10}$$

$$(\nabla(\mathbf{v},\mathbf{n}),\mathbf{n}) \tag{2.11}$$

$$(\mathbf{v},\mathbf{s}), \tag{2.12}$$

and at an outflow boundary:

$$(\mathbf{v},\mathbf{n}) \tag{2.13}$$

or

$$R - \frac{A}{\sqrt{gH}}(\nabla(\mathbf{v},\mathbf{n}),\mathbf{n}), \tag{2.14}$$

and

$$(\nabla(\mathbf{v},\mathbf{s}),\mathbf{n}). \tag{2.15}$$

Similar conditions can be prescribed in the inviscid case as we discussed at the end of the preceding section (see also [29]).
In addition to this set of conditions, we would like to have conditions which resemble conditions (2.4) and (2.5). In order to find such conditions, we will derive a class of boundary conditions for the one-dimensional equations. We

restrict our considerations to the one-dimensional case, because we assume that the condition for the tangential velocity is given by the prescription of (2.12) or (2.15) at an inflow or outflow boundary, respectively. For the viscid case the equivalent of (2.9') is

$$R_t^{\pm} = -(u \pm \sqrt{gH})R_x^{\pm} + gh_x + \frac{A}{2}(R^+ + R^-)_{xx}. \tag{2.16}$$

In the following, we try to find boundary conditions such that (2.16) is well-posed. An important condition for the well-posedness of (2.16) is that the right-hand side should satisfy a so-called one-sided Lipschitz condition (see [2, 9]). We will explain the relevance of this condition briefly. Let a partial differential equation be given by

$$\mathbf{w}_t = \mathbf{f}(\mathbf{w}, \mathbf{w}_x, \mathbf{w}_{xx}) \quad \text{for } x_l < x < x_r \tag{2.17}$$

with appropriate boundary conditions, where $\mathbf{w}$ and $\mathbf{f}$ are functions ($\mathbf{w}: \mathbb{R} \to \mathbb{R}^n$ and $\mathbf{f}: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$). Furthermore, let an inner product be defined by

$$<\mathbf{g}, \mathbf{h}> = \int_{x_l}^{x_r} (\mathbf{g}, \mathbf{h}) dx, \tag{2.18}$$

with a generated norm denoted by $|.|$. Then the one-sided Lipschitz condition we will use is defined by

$$<\mathbf{f}(\tilde{\mathbf{w}}, \tilde{\mathbf{w}}_x, \tilde{\mathbf{w}}_{xx}) - \mathbf{f}(\mathbf{w}, \mathbf{w}_x, \mathbf{w}_{xx}), \tilde{\mathbf{w}} - \mathbf{w}> \leqslant \sigma |\tilde{\mathbf{w}} - \mathbf{w}|^2, \tag{2.19}$$

where $\sigma \in \mathbb{R}$. If this condition is satisfied then it can be proven (e.g. Dahlquist [2]) that

$$|\tilde{\mathbf{w}}(t_2) - \mathbf{w}(t_2)| \leqslant e^{\sigma(t_2 - t_1)} |\tilde{\mathbf{w}}(t_1) - \mathbf{w}(t_1)| \quad \text{for } t_2 \geqslant t_1. \tag{2.20}$$

Now, we can proof the following theorem for the frozen coefficient form of (2.16). (In this theorem the difference $\tilde{\mathbf{w}} - \mathbf{w}$ will be denoted by $\Delta \mathbf{w}$.)

THEOREM 2.3.1. *Let the flow be subcritical and the frozen coefficient form of (2.16) be given by*

$$R_t^{\pm} = -(u_0 \pm \sqrt{gH_0})R_x^{\pm} + gh_x + \frac{A}{2}(R^+ + R^-)_{xx}. \tag{2.21}$$

Let the conditions at the left boundary be prescribed by

$$\Delta u = 0 \tag{2.22}$$

or

$$A(\Delta R^+ + \Delta R^-)_x + \alpha \Delta R^+ + \beta \Delta R^- = 0 \tag{2.23}$$

with $\beta - \alpha = 2\sqrt{gH_0}$ *and* $\alpha \leqslant -\sqrt{gH_0}$ *in the case of outflow, and by*

$$\Delta u = 0 \quad \text{and} \quad \Delta \zeta = 0 \tag{2.24}$$

or

$$A(\Delta R^+ + \Delta R^-)_x + \alpha \Delta R^+ + \beta \Delta R^- = 0 \text{ and } \Delta R^- + \delta \Delta R^+ = 0 \quad (2.25)$$

with $(2\sqrt{gH_0} + \alpha) - (\alpha + \beta)\delta + \beta\delta^2 \leqslant 0$ *in the case of inflow. Let the conditions at the right boundary be given by interchanging the roles of* ΔR^+ and ΔR^- *in (2.22)-(2.25). Then the right-hand side of (2.21) satisfies the one-sided Lipschitz condition with* $\sigma = 0$.

PROOF. For this linear case, substitution of the right-hand side of (2.21) into the Lipschitz condition with $\sigma = 0$ yields the inequality

$$\{-(c^+(\Delta R^+)^2 + c^-(\Delta R^-)^2) + \quad (2.26)$$

$$A(\Delta R^+ + \Delta R^-)_x(\Delta R^+ + \Delta R^-)\}\,|_{x_l}^{x_r} - A\int_{x_l}^{x_r}(\Delta R^+ + \Delta R^-)_x^2 dx \leqslant 0$$

where $c^{\pm} = u_0 \pm \sqrt{gH_0}$. The boundary conditions for both solutions ($\tilde{R}^{\pm}$ and $R^{\pm}$) are equal. Hence, the differences $\Delta R^{\pm}$ have homogeneous boundary conditions. Furthermore, forcing terms cancel out. Now, appropriate boundary conditions have to be found such that (2.26) holds. Notice that the integral has a negative contribution to the left-hand side of (2.26). Therefore, we will omit the integral.

If one chooses the boundary condition $\Delta R^+ + \Delta R^- = 0$ (i.e. $\Delta u = 0$), then from (2.26) there remains

$$-2u_0(\Delta R^+)^2\,|_{x_l}^{x_r} \leqslant 0.$$

The term at the left boundary, i.e. at x_l, is negative at outflow ($u_0 < 0$). Hence, it is sufficient to prescribe (2.22) at an outflow boundary. If the left boundary is an inflow boundary, then the term at this boundary is positive and therefore an extra condition is needed such that $\Delta R^+ = 0$. For example by the condition $\Delta\zeta = 0$. Hence, it is sufficient to prescribe (2.24) at in inflow boundary.

Next, at the left boundary, we consider boundary conditions of the type

$$A(\Delta R^+ + \Delta R^-)_x + \alpha \Delta R^+ + \beta \Delta R^- = 0. \quad (2.27)$$

Substitution into the inequality (2.26) yields, at the left boundary, the inequality

$$(c^+ + \alpha)(\Delta R^+)^2 + (\alpha + \beta)\Delta R^+ \Delta R^- + (c^- + \beta)(\Delta R^-)^2 \leqslant 0. \quad (2.28)$$

The constants α and β should be chosen such that this quadratic form is negative definite. It is definite if its discriminant is negative. Evaluation of this discriminant leads to the condition

$$(\alpha - \beta)^2 - 4(c^+ c^- + (\alpha c^- + \beta c^+)) \leqslant 0. \quad (2.29)$$

Assuming that $u_0 = 0$, then this inequality is equal to

$$(\alpha - \beta + 2\sqrt{gH_0})^2 \leqslant 0. \quad (2.30)$$

It is now easily verified, that (2.28) is satisfied at outflow ($u_0 \leqslant 0$) for the choice $\beta - \alpha = 2\sqrt{gH_0}$ and $\alpha \leqslant -\sqrt{gH_0}$. This proves condition (2.23).
At an inflow boundary we add to (2.27) the condition $\Delta R^- + \delta \Delta R^+ = 0$. Substitution into (2.28) yields the inequality

$$c^+ + \alpha - (\alpha + \beta)\delta + (c^- + \beta)\delta^2 \leqslant 0.$$

As the flow is subcritical, we have that $c^+ \leqslant 2\sqrt{gH_0}$ and $c^- \leqslant 0$. Using these inequalities we are led to condition (2.25). □

The inflow conditions (2.10) and (2.11) proposed by Oliger and Sundström are now found for $\alpha = \beta = 0$ and $\delta = -\infty$ in (2.25). For this choice we obtain from (2.25) that we have to impose the conditions $\Delta u_x = 0$ and $\Delta R^+ = 0$, which are the perturbed one-dimensional equivalents of (2.11) and (2.10), respectively. Furthermore, at outflow we find for $\alpha = -2\sqrt{gH_0}$ the condition $A\Delta u_x - \sqrt{gH_0}\,\Delta R^+$ which is the perturbed linearized equivalent of (2.14). Furthermore, at inflow the theorem suggests to impose the conditions (chosing $\alpha = \beta$, $\delta = -1$) $A\Delta u_x + \alpha \Delta u = 0$ and $\Delta\sqrt{gH} = 0$ which are, assuming the differences to be small, the perturbed equivalents of prescribing the expressions:

$$(\mathbf{v},\mathbf{n}) + \frac{A}{\alpha}(\nabla(\mathbf{v},\mathbf{n}),\mathbf{n}) \text{ and } \zeta \quad \text{for} \quad \alpha \leqslant -\tfrac{1}{2}\sqrt{gH}. \tag{2.31}$$

At outflow we find from the theorem the condition (chosing $\beta = -\alpha = \sqrt{gH_0}$) $A\Delta u_x - \sqrt{gH_0}\,\Delta(2\sqrt{gH}) = 0$ which is for small differences ($\Delta H \ll H_0$) the perturbed equivalent of the condition imposed by prescribing the value of

$$g\zeta - A(\nabla(\mathbf{v},\mathbf{n}),\mathbf{n}). \tag{2.32}$$

The boundary conditions (2.31) and (2.32) are almost of the same form as the conditions (2.4) and (2.5).

REMARK. The boundary conditions given in the theorem are not changed when also a linear bottom friction term is taken into account. Suppose that a term $-\lambda u$ is introduced in the right-hand side of the first equation of (2.8). Then we will find in (2.21) the term $-\lambda(R^+ + R^-)$ and in (2.26) the term

$$-\lambda \int_{x_l}^{x_r} (\Delta R^+ + \Delta R^-)^2 dx.$$

If the inequality (2.26) is satisfied without the last term (which is the case for the various boundary conditions specified in the theorem), then it will also hold when this term is included because the term is negative.

2.4. The initial values

In practical applications, almost any smooth initial function, consistent with the boundary conditions, will eventually lead, after the start-up period, to the same solution. This period is determined by the amount of dissipation in the equations (2.1), by the reflection at the open boundaries (parametrized by γ, cf. (2.4) and (2.5)), by the geometry and by the difference between the initial function and the true solution at the starting time. Hence, after the start-up period, the solution is completely determined by the boundary conditions and the forcing terms, and does not depend anymore on the initial values. It should be noticed that these boundary conditions may be time-dependent, which consequently yields a time-dependent solution.

3. The numerical algorithm

In this section, we will describe the discretization of the SWEs. Since the space discretization is performed on a so-called *staggered grid*, we will first describe this staggering. Next, we discuss how the boundaries of the domain are represented in this grid. Thereafter, the space discretization of the various terms is given. Further, the time discretization, its stabilization, and the discretization of the weakly-reflective boundary conditions will be described. Finally, the drying and flooding procedure used is explained.

3.1. Grid staggering

Grid staggering, originally introduced by Hansen [14], is often applied in the space discretization of partial differential equations. By this technique u, v and ζ are calculated at different grid points, which makes it possible to decrease the storage requirements by a factor four without loss of accuracy with respect to the main terms of the SWEs. The idea will be illustrated by the one-dimensional equations

$$\begin{aligned} u_t &= -g\zeta_x, \\ \zeta_t &= -H_0 u_x, \end{aligned} \tag{3.1}$$

which describe the dominant part of the SWEs in one dimension. If these equations are semi-discretized using second-order central differences, then we obtain

$$\begin{aligned} (U_t)_i &= -g(Z_{i+1} - Z_{i-1})/(2\Delta x), \\ (Z_t)_j &= -H_0(U_{j+1} - U_{j-1})/(2\Delta x), \end{aligned} \tag{3.2}$$

where $(U(t))_i$ and $(Z(t))_i$ approximate $u(i\Delta x,t)$ and $\zeta(i\Delta x,t)$, respectively. Observe that the subset of equations with i even, j odd is independent of the subset with i odd, j even. Hence, we may omit one of these sets, without loss of accuracy, thereby reducing the number of equations (and thus the number of dependent variables) by a factor two. Applying the same technique in the y-direction will lead to a final reduction by a factor four. A part of the resulting grid is drawn in Figure 3.1.

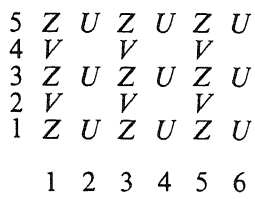


FIGURE 3.1. Position of the variables U, V and Z in space.

Those components, which are not available in a particular point can be obtained by averaging. In [38] more details can be found on the advantages of a space staggered grid.

3.2. Representation of the boundaries

In the discretization, the boundary of the domain is approximated by a polygon. This polygon consists of line pieces which are parallel to either the x-axis or the y-axis. The boundary is always parallel to the x-axis when it crosses a V-point and parallel to the y-axis when it crosses a U-point. Boundary pieces in both directions can cross through Z-points. An example of such a polygon is given in Figure 3.2.

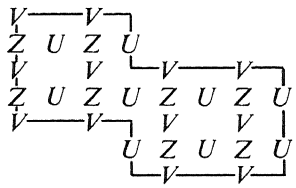


FIGURE 3.2. Boundary of the computational domain.

Due to this convention, we have that at a closed boundary either U or V is zero. Hence, this approximation does not simulate well situations where the physical boundaries are not parallel to either the x-axis or the y-axis. This has some consequences for the discretization as will be discussed in Section 3.4.2. In fact, for complex geometries this representation of the domain is only first-order accurate, i.e. the maximal distance between the numerical boundary and the true boundary decreases linearly with the mesh size.

3.3. Space discretization

In this section, the space discretization of the various terms of (2.1) will be described. As the y-derivatives are discretized similar to the x-derivatives, we only consider the x-derivatives. This similarity property is also used in the implementation, which reduces the length of the code considerably. For presentation reasons only, an exception will be made for the approximation of U at a V-point. Furthermore, with respect to the discretization near

boundaries, only the treatment at left boundaries is given. The treatment at right boundaries is analogous.
Two discretizations are implemented, a second-order and a fourth-order accurate discretization. The fourth-order accurate discretization allows the use of a coarser spatial grid by which the stability condition imposed by the explicit time discretization used is relaxed. However, this advantage cannot always be exploited, because there are many cases where the choice of the space mesh is determined by the resolution needed to represent the boundary to a sufficient accurate degree (see the previous section). In such cases, the second-order version may already simulate the flow at internal points very accurately. At the boundaries, lower-order discretizations are used in order to obtain a stable discretization. It turns out that this lower-order discretizations do not necessarily lead to a reduced accuracy (see Section 3.4.2). By Gustafsson [13] it is shown for the discretized form of hyperbolic equations that, under certain assumptions, the order of convergence is not decreased if at the boundaries approximations of one-order lower accuracy are used. The second-order discretization is almost identical to that of Stelling [38]. The fourth-order accurate discretization does not give additional problems in the implementation.

Below all discretizations are tabulated. In Table 3.1, the discretization of U at a V-point is given, whereas in Table 3.2 the other discretizations used are specified.
The Tables 3.1 and 3.2 differ only in the presentation of the quantities given in the first column. In the first column of Table 3.1 notational details are given, whereas in the first column of Table 3.2 the terms to be discretized are listed.
The second column specifies the position of the point at which the discretization is needed. For all terms, first the discretization at an internal point is given followed by the discretization in the neighbourhood of a boundary. In the latter case, the point at which the discretization is needed is denoted by a bold letter. A closed (open) boundary is indicated by | (|). In our notation, | or | directly follows the actual position of the boundary. In order to save space, we have represented several situations at the same time. For example, the discretization (3.3.b) is used to approximate U at **Z**. Here, three different cases may occur, viz. $U \mid Z\ U\ \mathbf{Z}\ U$, $U \mid Z\ U\ \mathbf{Z}\ U$, and $U\ Z \mid U\ \mathbf{Z}\ U$. These notations respectively mean a closed U-boundary, an open left U-boundary, and an elevation boundary. Hence, when more boundaries are indicated then this represents as many cases, where in each case only one of the indicated boundaries is valid. An exception is made for the case denoted by an asterisk in (3.10.b). Here a discretization is needed at an elevation point located between two closed boundaries.
The third column gives the actual discretization formulas. It is assumed that the space mesh is constant in x and y-direction and it will be denoted by Δx. This is the space mesh of the unstaggered grid (cf. (3.2)). For the notation of the discretizations we use the so-called shift operator E. Let ξ be a function defined on $\mathbb{R}^2$. Then the shift operator E is defined by $E\xi_i := \xi_{i+1}$, where

$\xi_i = \xi(i\Delta x, y)$. Likewise, the shift operator $\tilde{E}$ is defined by $\tilde{E}\xi_j := \xi_{j+1}$, where $\xi_j = \xi(x, j\Delta y)$ $(\Delta y = \Delta x)$. Below we omit the subscripts.
The order of accuracy of the discretizations is denoted by p, as given in the fourth column. The value of p is found by applying the discretization to a smooth test function.
In the fifth column the formula number of the discretization is given for later reference. Moreover, an asterisk is used in this column to indicate that the discretization is different from that used by Stelling.

As already mentioned, in Table 3.1 the discretization for the averaged value of U at a V-point is given. The construction of this averaged value proceeds in two steps; first U is approximated at a Z-point by averaging in x-direction (denoted by $\overline{U}^x$), thereafter $\overline{U}^x$ is averaged in y-direction which finally gives the approximation of U at the V-point (denoted by $\overline{U}^{xy}$).

notation	position	Discretization of U at a V-point	p	Formula number
$\overline{U}^x$	internally	$\{\frac{9}{16}(E+E^{-1})-\frac{1}{16}(E^3+E^{-3})\}U$	4	(3.3.a)*
	$U \mid\mid Z \mid U\ \mathbf{Z}\ U$	$\frac{1}{2}\{E+E^{-1}\}U$	2	(3.3.b)
	$U \mid\mid \mathbf{Z}\ U\ Z\ U$	$\frac{1}{2}\{3E-E^3\}U$	2	(3.3.c)*
$\overline{U}^{xy}$	internally	$\{\frac{9}{16}(\tilde{E}+\tilde{E}^{-1})-\frac{1}{16}(\tilde{E}^3+\tilde{E}^{-3})\}\overline{U}^x$	4	(3.3.d)*
	$Z \mid V\ Z\ \mathbf{V}\ Z$ $V \mid\mid Z\ \mathbf{V}\ Z$	$\frac{1}{2}\{\tilde{E}+\tilde{E}^{-1}\}\overline{U}^x$	2	(3.3.e)
	$Z \mid \mathbf{V}\ Z\ V$	$\frac{1}{2}\{3\tilde{E}-\tilde{E}^3\}\overline{U}^x$	2	(3.3.f)*

TABLE 3.1. Approximation of U at a V point

Stelling uses at all points (3.3.b) and (3.3.e), successively. This approach does not always lead to a first-order accurate approximation of U at a V-point near a boundary (see Section 3.4.2). Nevertheless, $\overline{U}^{xy}$ is used in (3.5.c) (see Table 3.2), which itself is a rather crude approximation (see the discussion in Sections 3.4.1 and 3.4.2). The other discretizations are given in Table 3.2. For a discussion on the choice of the discretizations, we refer to the next section.

term	position	Discretization	p	Formula number
u_x	internally	$1/(2\Delta x)\{\frac{4}{6}(E^2-E^{-2})-\frac{1}{12}(E^4-E^{-4})\}U$	4	(3.4.a)*
	$U \mid Z \mid U\ Z\ \mathbf{U}$	$1/(2\Delta x)\{\frac{1}{2}(E^2-E^{-2})\}U$	2	(3.4.b)*
	$U \mid Z\ \mathbf{U}\ Z\ U$	$1/(2\Delta x)\{\frac{1}{2}(E^2-E^{-2})\}U$ for $U\geqslant 0$ $1/(2\Delta x)\{E^2-I\}U$ for $U<0$	2 1	(3.4.c)*
	$U \mid Z \mid \mathbf{U}\ Z\ U$	$1/(2\Delta x)\{E^2-I\}U$ for $U<0$ 0 for $U\geqslant 0$	1 0	(3.4.d)
$(\Delta x)^3 u_{xxxx}$	internally	$1/(16\Delta x)\{6-4(E^2+E^{-2})+(E^4+E^{-4})\}U$	3	(3.4.d)*
v_x	internally	$1/(2\Delta x)\{\frac{4}{6}(E^2-E^{-2})-\frac{1}{12}(E^4-E^{-4})\}V$	4	(3.5.a)*
	$\begin{array}{ccc} & V & \mathbf{V} \\ U & \mid\mid & Z\ U\ Z \\ & V & \mathbf{V} \end{array}$ $\begin{array}{ccc} V & V & \mathbf{V} \\ Z & \mid & U\ Z\ U\ Z \\ V & V & \mathbf{V} \end{array}$	$1/(2\Delta x)\{\frac{1}{2}(E^2-E^{-2})\}V$	2	(3.5.b)*
	$\begin{array}{ccc} & \mathbf{V} & V \\ U & \mid\mid & Z\ U\ Z \\ & \mathbf{V} & V \end{array}$ $\begin{array}{ccc} V & \mathbf{V} & \\ Z & \mid & U\ Z \\ V & \mathbf{V} & \end{array}$	$1/(2\Delta x)\{(E^2-I)\}V$ for $\overline{U}^{xy}<0$ 0 for $\overline{U}^{xy}\geqslant 0$	1 0	(3.5.c)
$(\Delta x)^3 v_{xxxx}$	internally	$1/(16\Delta x)\{6-4(E^2+E^{-2})+(E^4+E^{-4})\}V$	3	(3.5.d)*
ζ_x	internally	$1/(2\Delta x)\{\frac{27}{24}(E^1-E^{-1})-\frac{1}{24}(E^3-E^{-3})\}Z$	4	(3.6.a)*
	$U \mid\mid Z \mid \mathbf{U}\ Z\ U$	$1/(2\Delta x)\{E^1-E^{-1}\}Z$	2	(3.6.b.)
ζ	internally	$\{\frac{9}{16}(E+E^{-1})-\frac{1}{16}(E^3+E^{-3})\}Z$	4	(3.7.a)
	$U \mid Z \mid \mathbf{U}\ Z$	$\frac{1}{2}\{E+E^{-1}\}Z$	2	(3.7.b)
	$\mathbf{U} \mid Z\ U\ Z$	$\frac{1}{2}\{3E-E^3\}Z$	2	(3.7.c)*
u_{xx}	internally	$1/((2\Delta x)^2)\{-\frac{5}{2}+\frac{4}{3}(E^2+E^{-2})-\frac{1}{12}(E^4+E^{-4})\}U$	4	(3.8.a)*
	$U \mid\mid Z \mid U\ Z\ \mathbf{U}$	$1/((2\Delta x)^2)\{E^2-2+E^{-2}\}U$	2	(3.8.b)
	$U \mid\mid Z \mid \mathbf{U}\ Z\ U$	$1/((2\Delta x)^2)\{E^2-I\}U$	0	(3.8.c)*

TABLE 3.2. Discretizations (to be continued)

term	position	Discretization	p	Formula number
v_{xx}	internally	$1/((2\Delta x)^2)\{-\frac{5}{2}+\frac{4}{3}(E^2+E^{-2})-\frac{1}{12}(E^4+E^{-4})\}V$	4	(3.9.a)*
	V **V** $U \mid\mid Z\ U\ Z$ V **V** V V **V** $Z \mid U\ Z\ U\ Z$ V V **V**	$1/((2\Delta x)^2)\{(E^2-2+E^{-2})\}V$	2	(3.9.b)
	V V $U \mid Z\ U\ Z$ **V** V V **V** $Z \mid U\ Z$ V **V**	$1/((2\Delta x)^2)\{(E^2-I)\}V$	0	(3.9.c)
	V V $U \mid Z\ U\ Z$ **V** V	$1/((2\Delta x)^2)\{(E^2-(3-2\eta)I)\}V$ $\eta=1/[1+(1-\alpha)\Delta x/\alpha)]$	1	(3.9.d)
$(Hu)_x$	internally	$1/(2\Delta x)\{\frac{27}{24}(E^1-E^{-1})-\frac{1}{24}(E^3-E^{-3})\}HU$	4	(3.10.a)*
	$U \mid \mathbf{Z}\ U\ Z$ $Z \mid U\ \mathbf{Z}\ U$ $U \mid \mathbf{Z} \mid U^*$	$1/(2\Delta x)\{E^1-E^{-1}\}HU$	2	(3.10.b)
	$U \mid \mathbf{Z}\ U\ Z$	$1/(2\Delta x)\{-\frac{25}{24}E^{-1}+\frac{26}{24}E^1-\frac{1}{24}E^3)\}HU$	0	(3.10.c)*

TABLE 3.2 (cont'd). Discretizations.

As already mentioned, we have implemented a second-order accurate version and a fourth-order accurate version. In these tables the discretizations are given exactly as they are used in the fourth-order implementation. It will be clear that the fourth-order accuracy is only obtained at internal points. The discretizations as used in the second-order implementations are found from the tables by replacing the discretization at internal points by the discretizations with number (*.b). Moreover, in the second-order case (3.3.e) is used instead of (3.3.d) at internal points.

3.4. Discussion

In this section, we motivate the choice of the preceeding discretizations. Special attention will be given to the following topics: boundary treatment, discretization near 'zig-zag boundaries', artificial diffusion and conservation of mass.

3.4.1. On the effect of the boundary treatment. The given discretizations at the boundaries are only in part consistent with the boundary conditions derived in Section 2.3. The main terms of the SWEs are treated always consistent with these boundary conditions, but the advection and viscosity terms are not. The reason for this is that the representation of the boundary may cause severe numerical errors if straightforward consistent approximations of the advection terms are used (see Section 3.4.2). In the following we analyse the effect of such an (inconsistent) discretization. The discretization at the left boundary given in the tables may be considered as an approximation of the perturbed SWEs on the strip of width Δx located at this boundary (see Figure 3.3); the perturbed SWEs are given by:

$$\mathbf{w}_t = \mathbf{f}(\mathbf{w}, \mathbf{w}_x, \mathbf{w}_y \mathbf{w}_{xx}, \mathbf{w}_{yy}, \mathbf{x}, t) + \mathbf{p}(\mathbf{w}, \mathbf{w}_x, \mathbf{w}_{xx}, \mathbf{x}, t), \tag{3.11}$$

where $\mathbf{w}=(u,v,\zeta)^T$ and $\mathbf{f}$ is the right-hand side of (2.1). Furthermore, the perturbation $\mathbf{p}$ is given by

$$\begin{aligned} p_1 &= (-u' + \frac{A}{2\Delta x})u_x + uu_x - Au_{xx}, \\ p_2 &= (-\min(u,0) + \frac{A}{2\Delta x})v_x - 2A\frac{1-\eta}{(2\Delta x)^2}v + uv_x - Av_{xx}, \\ p_3 &= 0, \end{aligned} \tag{3.12}$$

where $u' = \min(u, 0)$ at a closed boundary and at an elevation boundary, and $u' = u$ at a velocity boundary.

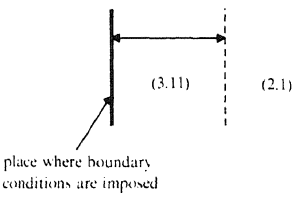


FIGURE 3.3. Domains where (3.11) and (2.1) are valid.

Furthermore, the boundary conditions are given by (2.2) if the left boundary is closed, i.e. $u=0$, and by (2.4) or (2.5) if the left boundary is open, i.e. $u+\gamma R_t = f^u(t)$ or $\zeta+\gamma R_t = f^\zeta(t)$. It should be noticed, that condition (2.7) is not imposed. This is avoided by an adaptation of the equation at an inflow boundary such that the coefficient of v_x is always non-negative (see the second equation of (3.12)). At the right-hand side of the strip for which (3.11) is valid, the SWEs are as given in (2.1).

The terms $u'u_x$ and $\min(u,0)v_x$ arise from the discretizations {(3.4.c),(3.4.d)} and (3.5.c), respectively. Furthermore, $1/(2\Delta x)\,u_x$ follows from (3.8.c) and $1/(2\Delta x)v_x$ and $2(1-\eta)/((2\Delta x)^2)\,v$ can be derived from (3.9.d) in the following way:

$$\frac{1}{(2\Delta x)^2}(E^2-(3-2\eta)I)V=$$
$$\frac{1}{(2\Delta x)}\{\frac{1}{(2\Delta x)}(E^2-I)V\}-\frac{1}{(2\Delta x)^2}(2-2\eta)V\approx$$
$$\frac{1}{2\Delta x}v_x-2\frac{1-\eta}{(2\Delta x)^2}v.$$

If we let Δx tend to zero, then we find from (3.11) that additionally (2.3) and (2.15) are imposed, i.e. $(1-\alpha)v-v_x=0$ at a closed boundary and $v_x=0$ at an outflow boundary. Moreover, we find at all types of boundaries the condition $u_x=0$ and at an inflow boundary $(u>0)$ $v_x=0$ (which replaces (2.7)). The latter causes that, for example, at a closed boundary three conditions are imposed. Hence, if Δx tends to zero the problem is overspecified. This may lead to instabilities and discontinuities (see Oliger and Sundström [29]), but so far these were not observed, which we ascribe to the fact that Δx is still very large.

REMARK. By a small adaptation of the discretization, the expression (2.13) or (2.32) can be prescribed at an outflow boundary.
The expression (2.13) is specified if at an open outflow boundary u is also used as a boundary condition for the viscosity term. Thereby, in the case of an open boundary $Au_x/(2\Delta x)$ in p_1 (see (3.12)) is replaced by Au_{xx}.
The expression (2.32) is prescribed at an outflow boundary if in the first momentum equation in the viscosity term $u_x=0$ is imposed and furthermore the elevation is specified. This is identical to prescribing the expression $Au_x+g\zeta$. In fact, this expression is specified in this way in the first momentum equation. However, an adaptation of the calculation of H, which needs ζ, should be made. Instead of discretization (3.7.b), the expression (3.7.c) should be used to approximate H at the velocity point adjacent to the boundary. We have refrained from implementing these adaptations, because it is specious to do so as long as the treatment at inflow boundaries and closed boundaries is not fully consistent.

We observe that the discretizations described above lead to a simple implementation. Moreover, from (3.11) we conclude that the perturbed SWEs are close to the true SWEs if

1.a. the flow at the boundary is strongly sub-critical i.e. $|u| \ll \sqrt{gH}$ if $u>0$ at a left boundary or if $u<0$ at a right boundary,
1.b. the mesh-size is such that $A/(2\Delta x)$ is much less than $\sqrt{gH}$,

or if

2. the terms u_x and v_x are approximately zero at the boundaries and $\alpha=1$

(see Section (2.3.1)).
In many engineering problems these conditions are fulfilled to a sufficient degree (see also the discussion of Stelling and Willemse on this subject [40]).

Condition 1 can be understood from (2.9). From this equation, we have that the propagation speed of the waves is given by the factor $|u| \pm \sqrt{gH}$. Similarly, for the two-dimensional equations the propagation speed is $|\mathbf{v}| \pm \sqrt{gH}$. If, over one mesh width, we perturb this speed by a quantity of magnitude less than or equal to $|u| + A/(2\Delta x)$, which is the case when (3.11) is valid, then the error may be expected to be small if $|u| + A/(2\Delta x) << \sqrt{gH}$, i.e. for a strongly subcritical flow and for a space mesh such that $A/(2\Delta x)$ is small.
Condition 2 is derived by comparison of (3.11) and (2.1). If this condition is satisfied then (3.11) and (2.1) are equal.

3.4.2. Discretization near 'zig-zag boundaries'. The discretization of the advection terms and viscosity terms near boundaries seems to be crude. However, this treatment is more accurate than standard central differences for flows along boundaries which are neither parallel to the x-axis nor to the y-axis (so-called 'zig-zag boundaries'). For example consider the boundary drawn in Figure 3.4, which should simulate a boundary given by a "diagonal" boundary.

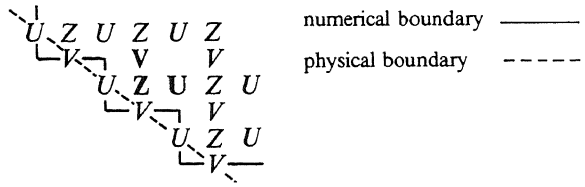


FIGURE 3.4. "Zig-zag" boundary.

flow parallel to the "diagonal" physical boundary is not disturbed by the boundary in the free slip case, i.e. $\alpha = 0$ in (2.2). In the numerical scheme where the "diagonal" boundary is represented by a "zig-zag" boundary this property should be approximated as best as possible. A straightforward central (second-order) discretization of u_x at the point indicated by U would lead to

$$[u_x] = 1/(4\Delta x)\{E^2\}U.$$

If in this case U is positive, then the discretization of $-uu_x$ will act as a bottom friction term. If, on the other hand, U is negative, then this term will have a destabilizing effect. Therefore, the differences are chosen as given in (3.4.d).
A straightforward discretization of u_{xx} at the same point would lead to

$$[u_{xx}] = 1/(2\Delta x)^2\{E^2 - 2\}U. \qquad (3.13)$$

This discretization will also act as a friction term and thereby a free slip boundary is not correctly simulated. Therefore, the discretization is chosen as given in (3.8.c). Stelling uses (3.13) in this case [38, p. 147].

A similar reasoning justifies the discretizations (3.5.c) and (3.9.c) of uv_x and v_{xx}, respectively, at the point indicated by **V**. An additional problem at this point is the computation of U at **V**. Stelling approximates U at **V** by averaging over the four neighbouring U-values. This gives only 3/4 of the real value if the flow is parallel to the boundary. Therefore, we employed approximations as given in Table 3.1, which give at least a first-order approximation in cases as discussed here.
With respect to the continuity equation, the "zig-zag" representation of the boundary has little influence. Considering the discretization of the continuity equation at the point indicated by **Z** and assuming constant depth, then the second-order discretization of the right-hand side of the continuity equation is of the form $-H(\mathbf{U}+\mathbf{V}-(U+V))/(2\Delta x)$ where U and V are zero. This discretization does not change if we set $U=-V$, where V may have an arbitrary value, i.e. a flow parallel to the boundary.
As a consequence of the above approach the deficiency in the "zig-zag" representation of a "diagonal" boundary is partly compensated by the discretization. An alternative is the transformation of the domain to another domain in which boundaries coincide with grid lines (see e.g. [48, 47]). However, when drying and flooding should be taken into account similar problems as discussed in this section can occur in the transformed domain.

3.4.3. Artificial diffusion. A known problem of the discretizations (3.4.a), (3.4.b), (3.5.a) and (3.5.b) is that they may give rise to so-called $2\Delta x$ waves (see [38] and [43]). This is caused by the fact that some eigenvalues of the operator become close to zero for high-frequency components in the solution. The occurrence of the $2\Delta x$ waves can be avoided by adding "artificial diffusion" to the momentum equations. Adding diffusion of the form $(\Delta x)u_{xx}$ to the discretized first momentum equation, where a second-order derivative is used, gives rise to a considerable amount of numerical diffusion and decreases the accuracy to first-order. As a consequence, for many practical flow problems the accuracy of the low-frequency components in the solution is seriously influenced. Therefore we applied diffusion of the form $-c(\Delta x)^3 u_{xxxx}$, where a fourth-order derivative is used and where c is a parameter which is to a large extent independent of the problem. This gives rise to a third-order discretization. For low-frequency components in the solution the damping effect of the fourth-order diffusion term is much less than for the second-order term. For high-frequency components, however, the damping effect of both treatments may well be of the same order of magnitude depending on the constants used. By numerical experiments it was found that $c \in [.2,.8]$ gives the desired robustness for a large variety of problems. For the same reasons, (3.5.d) multiplied by $-c$ is added to the second momentum equation.

3.4.4. Conservation of mass. The discretization (3.10) used in the continuity equations conserves mass near closed boundaries and in the internal domain. This can be shown by inspection of the associated matrix:

$$\frac{1}{48\Delta x}\begin{bmatrix} -25 & 26 & -1 & & & \\ 1 & -27 & 27 & -1 & & \\ & 1 & -27 & 27 & -1 & \\ & & 1 & -27 & . & . \\ & & & . & . & . \end{bmatrix},$$

where the first column corresponds with the boundary point (which has a zero value in this case). The first row originates from (3.10.c), whereas the other rows originate from (3.10.a). For conservation the column sums of this matrix, except for the first column, should be zero (see also [12, p. 6]), which is clearly the case. At closed boundaries, the discretization is zero-order consistent. The conservation property is in this case more important than consistency. In the same way it can be seen that at open boundaries the discretization does not preserve mass. The associated matrix is of the form

$$\frac{1}{48\Delta x}\begin{bmatrix} -24 & 24 & & & & \\ 1 & -27 & 27 & -1 & & \\ & 1 & -27 & 27 & -1 & \\ & & 1 & -27 & . & . \\ & & & 1 & . & . \end{bmatrix},$$

where the first column is again associated with the boundary point. Here, the first row originates from (3.10.b). Applying this matrix to the vector UH and summing over all elements of the result vector yields a non-zero contribution at the open boundary of the form

$$\frac{1}{48\Delta x}\{-23-2E+E^2\}UH=$$
$$\frac{-1}{2\Delta x}UH+\frac{1}{48\Delta x}E\{E^{-1}-2+E\}UH,$$

where the boundary point is used as a reference for the shift operator. If instead of (3.10.b) the approximation (3.10.c) is also used at open boundaries, then after the same manipulations a contribution $-1/(2\Delta x)UH$ will be found. This is considered ideal, because the only increase or decrease of the amount of mass is determined by the quantity imposed at the boundary. Using (3.10.b) there is an additional increase or decrease of mass. The amount of mass is solution-dependent. This contribution is small if the second derivative of the solution is small, which is usually the case. Therefore, we prefer to use the second-order discretization instead of the mass-conserving discretization. Nevertheless, there is no additional difficulty in implementing the mass-conserving approximation.

3.5. Time discretization

In this section, the time integration will be described. For this purpose the method of lines approach will be used. First we write (2.1) in the compact notation

$$\mathbf{w}_t = \mathbf{f}(\mathbf{w}, \mathbf{w}_x, \mathbf{w}_y \mathbf{w}_{xx}, \mathbf{w}_{yy}, \mathbf{x}, t), \; t > t_0, \; \mathbf{x} \in \Omega, \tag{3.14}$$

where $\mathbf{w} = (u, v, \zeta)^T$. After space discretization of this PDE and its boundary conditions (see Section 2.3) on the space staggered grid, we obtain the system of ODEs

$$\frac{d}{dt}\mathbf{W}(t) = \mathbf{F}(\mathbf{W}, t), \; t > t_0. \tag{3.15}$$

For the time integration of this system several integrators can be used. A survey is given in [18]. We use the classical Runge-Kutta formula given by (for a discussion of our choice we refer to Section 4.2., see also Praagman [31])

$$\mathbf{W}^{n+1} = \mathbf{W}^n + \Delta t(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4)/6, \tag{3.16}$$

where

$$\mathbf{K}_1 = \mathbf{F}(\mathbf{W}^n, t_n),$$

$$\mathbf{K}_2 = \mathbf{F}(\mathbf{W}^n + \tfrac{1}{2}\Delta t \mathbf{K}_1, t_n + \tfrac{1}{2}\Delta t),$$

$$\mathbf{K}_3 = \mathbf{F}(\mathbf{W}^n + \tfrac{1}{2}\Delta t \mathbf{K}_2, t_n + \tfrac{1}{2}\Delta t),$$

$$\mathbf{K}_4 = \mathbf{F}(\mathbf{W}^n + \Delta t \mathbf{K}_3, t_{n+1}).$$

In this formula, $t_n = t_0 + n\Delta t$ and $\mathbf{W}^n$ approximates $\mathbf{W}(t_n)$. The stability region of this formula in the complex plane is drawn in Figure 3.5. For linear stability it is needed that the eigenvalues of $\Delta t J$ are within this region. Here J is the Jacobian matrix of $\mathbf{F}$ ($J = \partial \mathbf{F}(\mathbf{W}, t)/\partial \mathbf{W}$)

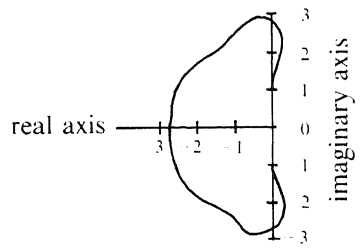


FIGURE 3.5. Stability region of the classical Runge-Kutta method.

This picture shows that the classical Runge-Kutta method is conditionally stable, i.e., given a certain problem there is a restriction on the time step. For the SWEs, the eigenvalues may vary from almost purely imaginary to real depending on the depth. The imaginary parts of the eigenvalues are due to the

main terms of the SWEs (see (3.1)) and to the advection terms. The negative real parts of the eigenvalues arise from the bottom friction and the viscosity terms. In general, the ratio $A / (\Delta x \sqrt{gH})$, reflecting the relative importance of the viscosity terms to the main terms with respect to stability, is rather small. Hence, in general, the viscosity terms are not so important with respect to stability. However, if the depth tends to zero (for example on a tidal flat), then the bottom friction term tends to minus infinity. As this will lead to an unstable calculation, an upper limit is set to this friction term in such a way that the corresponding eigenvalue is still within the stability region. This adaptation of the momentum equations does not seriously influence the accuracy of the solution as shallow regions are, in general, only important as a water storage area (see [3]). We will describe this in more detail in Section 3.8.

3.6. Stabilization of the time integration

In this subsection, the stabilization procedure as employed in the code will be described. The stabilization, based on smoothing of the discretized right-hand side function, allows to use significant larger time steps than the maximum time step dictated by the stability condition of the explicit method used. Several authors [25, 21, 44] described and applied *implicit* smoothing. However, with respect to vector computing, we prefer to use *explicit* techniques (see Section 4.2). The general concept of this type of smoothing for hyperbolic partial differential equations is treated in [49]. It is analysed more extensively in [19] for hyperbolic as well as for parabolic equations, and in [17] for solving elliptic equations. A review of the various applications of smoothing is given in [16].

The technique basically consists of solving

$$\frac{d}{dt}\mathbf{W}(t) = \mathbf{S}(\mathbf{F}(\mathbf{W},t)),\ t > t_0, \tag{3.17}$$

instead of (3.15), where $\mathbf{S}$ is a smoothing function. The function $\mathbf{S}$ should be chosen such that the spectral radius of $\partial \mathbf{S}(\mathbf{F}(\mathbf{W},t)) / \partial \mathbf{W}$ is minimized provided that the evaluation of $\mathbf{S}(\mathbf{F})$ is cheap and the error due to the smoothing is limited. Evidently, the error introduced by this smoothing depends on the difference

$$\mathbf{S}(\mathbf{F}(\mathbf{W},t)) - \mathbf{F}(\mathbf{W},t) \tag{3.18}$$

where $\mathbf{W}$ is a solution of (3.15). This error is small if $\mathbf{F}(\mathbf{W},t)$ is smooth, i.e. if successive elements of the vector $\mathbf{F}(\mathbf{W},t)$ differ slightly. For the original equation (3.14) this implies that the right-hand side $\mathbf{f}(.)$ should also be smooth if the solution $\mathbf{w}$ is substituted, i.e. it should have small space derivatives. This is trivially the case when we consider a stationary solution. In that case, the time derivative of $\mathbf{w}$ is zero and consequently all space derivatives of the right-hand side are zero. In the case, that the solution varies slowly in time, i.e. the solution is close to a steady state, we expect that the space derivatives of the right-hand side are close to zero. In [49] examples are given for which it is shown that small time derivatives of the solution result in small space derivatives of

the right-hand side. Moreover, in this paper it is shown that smoothing inherently appears in implicit time integration methods, which explains the improved stability behaviour of such methods.
It should be noticed that this type of smoothing is different from smoothing the numerical solution itself. In the latter case smoothing may only be applied, without danger of loss of accuracy, if the solution itself is smooth, i.e. if the solution has small derivatives with respect to the space variables. This is in general not the case. Smoothing of the solution is, for example, proposed by Shuman [36]. A more sophisticated example is the Richtmeyer scheme [33], which may be regarded as a two-stage second-order Runge-Kutta method, where in the first stage the solution is smoothed, in order to obtain a stable method for hyperbolic equations.
In the following we introduce the smoothing used, we derive the reduction of the spectral radius obtained after its application to the two-dimensional SWEs, and we consider its influence on the accuracy of the solution.

3.6.1. The choice of **S**. As a starting point in our presentation, we consider a smoothing based on the Jacobian matrix of (3.15), i.e. **S** is of the form

$$\mathbf{S}(\mathbf{F}) = Q(J_n)\mathbf{F} + \mathbf{g} \tag{3.19}$$

where $Q(z)$ is a rational function with $Q(z) \to 1$ for $z \to 0$, $\mathbf{g}$ is a correction term such that the error (3.18) tends to zero if the mesh size tends to zero, and J_n is the normalized Jacobian, i.e. $J_n = J / \rho(J)$. Evidently, the eigenvalues of J_n are all contained within the unit disc in the complex plane. Evaluation of $Q(J_n)\mathbf{F}$ is in general expensive. Therefore, we shall attempt to find simplified forms of J_n, which we denote by $\bar{J}_n$, such that $Q(\bar{J}_n)\mathbf{F}$ can be computed efficiently. We will start to consider the one-dimensional SWEs, which can be found from (2.1) by setting v and all y-derivatives equal to zero. For the construction of the smoothing procedure we only take into account the *main terms of the SWEs* (see (3.2)), because, in the problems we consider, these terms dominate the spectral radius. Nevertheless, the smoothing is applied to the *complete* discretized right-hand side of the SWEs (cf. (3.17)).

3.6.2. One-dimensional problems. We start with the description of our smoothing technique for one-dimensional problems. The explicit smoothing function for the one-dimensional case we use, is defined by

$$\mathbf{S}(\mathbf{F}) = \mathbf{S}_q(\mathbf{S}_{q-1}(\ldots(\mathbf{S}_1(\mathbf{F}))\ldots)), \tag{3.20}$$

where

$$\begin{aligned}
&\mathbf{S}_k(\mathbf{F}) = S_k\mathbf{F} + \mathbf{g}_k,\\
&S_k = I + \mu_k D_k,\\
&D_k = 4D_{k-1}(I + D_{k-1}),\ k \geqslant 2,\\
&D_1 = \bar{J}_n^2,
\end{aligned}$$

$$\bar{J}_n = \frac{2(\Delta x)}{\sqrt{gH_0}}\bar{J}.$$

Here, $\bar{J}$ is of the form

$$\bar{J} = \begin{bmatrix} 0 & -g\delta^T \\ H_0\delta & 0 \end{bmatrix},$$

where the submatrix $H_0\delta$ follows from the discretization (3.10.b) with constant depth H (denoted by H_0). Later on, we will show that by this smoothing function the spectral radius of the Jacobian of the SWEs can be reduced very effectively. It is straightforward to show that the eigenvalues of $\bar{J}_n$ are contained in the interval $[-i,i]$ on the imaginary axis (see also Section 3.6.4). Consequently, the eigenvalues of $\bar{J}_n^2$ are real and contained in the interval $[-1,0]$. For the smoothing operator (3.20), the function $Q(z)$ is of the form $Q(z)=Q(z^2)$.

REMARK. For the stability properties of the smoothing as applied here it is enough to consider the spectrum of the smoothed Jacobian matrix. This is due to the fact that the Jacobian matrix is similar to a symmetric matrix by means of a positive definit matrix which itself is independent of the meshsize (see for more details the proof of Lemma 3.6.2). In contrast with the famous Von Neumann analysis the boundary conditions are included in this approach. Nevertheless, the Von Neumann analysis yields the same results as the approach followed here (see [49]).

EXAMPLE 3.1. In order to illustrate the form of $\bar{J}$, S_k and $\mathbf{g}_k$, we consider the one-dimensional problem on the interval $[0,L]$, where at the left and right boundary the velocity and the elevation are respectively prescribed, i.e.

$$\begin{aligned} u(0,t) &= u_0(t), \\ \zeta(L,t) &= \zeta_L(t). \end{aligned} \tag{3.21}$$

Let the ordering of the dependent variables be given by

$$\begin{aligned} W_j(t) &= U_{2j}(t) \quad \text{for } j = 1,\ldots,N, \\ W_j(t) &= Z_{2j-2N-1}(t) \quad \text{for } j = N+1,\ldots,2N, \end{aligned} \tag{3.22}$$

where $U_{2j}(t)$ and $Z_{2j-1}(t)$ approximate $u(2j\Delta x,t)$ and $\zeta((2j-1)\Delta x,t)$, respectively. Furthermore, $\Delta x = L/(2N+1)$. The values of $W_0(t)=u_0(t)$ and $W_{2N+1}=\zeta_L(t)$ are given and occur in the forcing term of the discretized equation. For this ordering the Jacobian $\bar{J}$ assumes the form given in Figure 3.6.

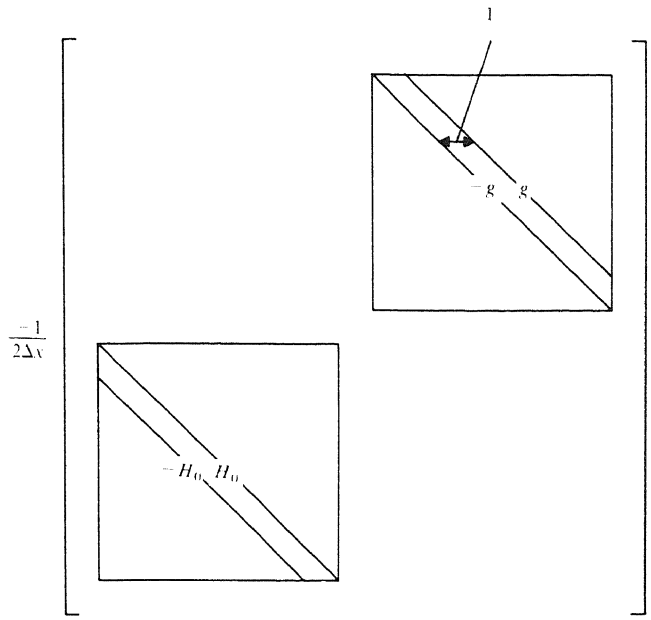


FIGURE 3.6. The form of the simplified Jacobian.

Starting from this $\bar{J}$ we find, according to (3.20), that D_k is of the form as given in Figure 3.7. A number written on a (anti-) diagonal denotes the value for all elements of the (anti-) diagonal. If an anti-diagonal and a diagonal cross through the same point then the values of the anti-diagonal and the diagonal are simply added. This only occurs if the elements of the anti-diagonal have the value -1/4 (see (3.24)).

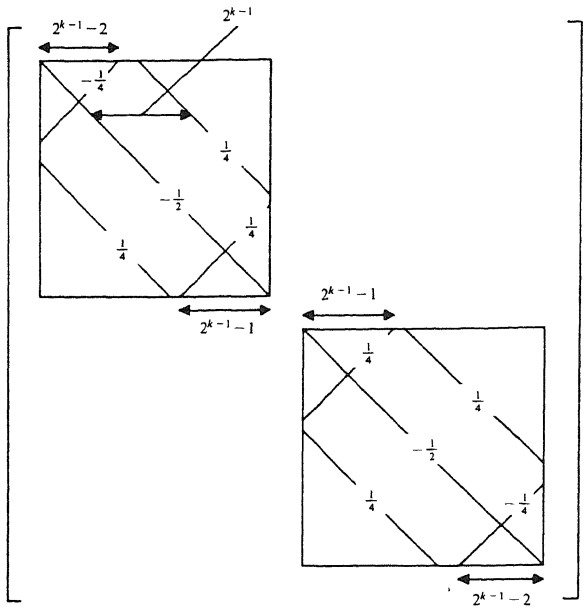


FIGURE 3.7. The structure of D_k.

From this structure we observe that at internal points $\mathbf{S}_k(\mathbf{F})$ is given by the simple formula

$$(\mathbf{S}_k(\mathbf{F}))_j = \tfrac{1}{4}\mu_k F_{j-2^{k-1}} + (1-\tfrac{1}{2}\mu_k)F_j + \tfrac{1}{4}\mu_k F_{j+2^{k-1}}, \tag{3.23}$$

as for these points $(\mathbf{g}_k)_j$ is zero. Furthermore, near boundaries we have

$$
\left.
\begin{aligned}
&\text{for } j=1,\ldots,2^{k-1}-1:\\
&\qquad (\mathbf{S}_k(\mathbf{F}))_j = -\tfrac{1}{4}\mu_k F_{-j+2^{k-1}} + (1-\tfrac{1}{2}\mu_k)F_j\\
&\qquad\qquad + \tfrac{1}{4}\mu_k F_{j+2^{k-1}} + (\mathbf{g}_k)_j\\
&\text{for } j=2^{k-1}:\\
&\qquad (\mathbf{S}_k(\mathbf{F}))_j = (1-\tfrac{1}{2}\mu_k)F_j + \tfrac{1}{4}\mu_k F_{j+2^{k-1}} + (\mathbf{g}_k)_j\\
&\text{for } j=N-2^{k-1}+1,\ldots,N:\\
&\qquad (\mathbf{S}_k(\mathbf{F}))_j = \tfrac{1}{4}\mu_k F_{j-2^{k-1}} + (1-\tfrac{1}{2}\mu_k)F_j\\
&\qquad\qquad + \tfrac{1}{4}\mu_k F_{-j+2N+1-2^{k-1}} + (\mathbf{g}_k)_j\\
&\text{for } j=N+1,\ldots,N+2^{k-1}:\\
&\qquad (\mathbf{S}_k(\mathbf{F}))_j = \tfrac{1}{4}\mu_k F_{-j+1+2N+2^{k-1}} + (1-\tfrac{1}{2}\mu_k)F_j\\
&\qquad\qquad + \tfrac{1}{4}\mu_k F_{j+2^{k-1}} + (\mathbf{g}_k)_j\\
&\text{for } j=2N-2^{k-1}:\\
&\qquad (\mathbf{S}_k(\mathbf{F}))_j = \tfrac{1}{4}\mu_k F_{j-2^{k-1}} + (1-\tfrac{1}{2}\mu_k)F_j + (\mathbf{g}_k)_j\\
&\text{for } j=2N-2^{k-1}+1,\ldots,2N:\\
&\qquad (\mathbf{S}_k(\mathbf{F}))_j = \tfrac{1}{4}\mu_k F_{j-2^{k-1}} + (1-\tfrac{1}{2}\mu_k)F_j\\
&\qquad\qquad - \tfrac{1}{4}\mu_k F_{-j+4N+2-2^{k-1}} + (\mathbf{g}_k)_j
\end{aligned}
\right\} \quad (3.24)
$$

In order to let the error (3.18) tend to zero if Δx tends to zero, $\mathbf{g}_k$ is chosen as follows:

$$
\begin{aligned}
&(\mathbf{g}_k)_j = \tfrac{1}{2}\mu_k \frac{d}{dt}u_0(t) \quad \text{for } j=1,\ldots,2^{k-1}-1,\\
&(\mathbf{g}_k)_j = \tfrac{1}{4}\mu_k \frac{d}{dt}u_0(t) \quad \text{for } j=2^{k-1},\\
&(\mathbf{g}_k)_j = 0 \quad \text{for } j=2^{k-1}+1,\ldots,2N-2^{k-1}-1,\\
&(\mathbf{g}_k)_j = \tfrac{1}{4}\mu_k \frac{d}{dt}\zeta_L(t) \quad \text{for } j=2N-2^{k-1},\\
&(\mathbf{g}_k)_j = \tfrac{1}{2}\mu_k \frac{d}{dt}\zeta_L(t) \quad \text{for } j=2N-2^{k-1}+1,\ldots,2N. \ \square
\end{aligned}
\qquad (3.25)
$$

From this example problem with boundary conditions given by (3.21), it is straightforward to find the smoothing for problems where at both boundaries the elevation or the velocity is prescribed or for problems where at the left and right boundary the elevation and the velocity are respectively prescribed. A suitable choice of μ_k is given by (3.45).

Notice that at a closed boundary (i.e. a U-boundary) the column sum of that part of D_k operating on the right-hand side of the continuity equation is zero (see Figure 3.7). As a consequence the column sum of the matrix S_k is one. This means that, in the case that the left as well as the right boundary is closed, the sum of the right-hand sides over the grid points is preserved. This property of the smoothing is essential for the conservation of mass (see also Section 3.4.4).

The reader may wonder what the structure of the matrix D_k will be when k is so large that 2^{k-1} becomes of the same order of magnitude as N. In this case, the structure can still be found from (3.20) but in addition to its dependence on k it will also depend on N. As N varies in the case of a complex geometry we use an implicit smoothing operator when q is such that $2^{q-1} \geqslant N-2$. This operator is, for the one-dimensional problem, defined by

$$\mathbf{S}(\mathbf{F}) = (I - \frac{\mu}{4} D_1)^{-1} \mathbf{F} + \tilde{\mathbf{g}}. \tag{3.26}$$

where D_1 is given in (3.20) and $\tilde{\mathbf{g}} = \mathbf{g}_1$ in which we choose $\mu_1 = \mu$, μ being an arbitrary parameter. In this case, $Q(z)$ is of the form

$$\tilde{Q}(z) = \frac{1}{1 - \frac{\mu}{4} z}. \tag{3.27}$$

For the implicit operator a system of equations has to be solved with a tridiagonal matrix.

3.6.3. Two-dimensional problems. For the *two-dimensional case* we proceed as follows. In this case, a simplified Jacobian is given by

$$\bar{J}_x + \bar{J}_y, \tag{3.28}$$

where

$$\bar{J}_x = \begin{bmatrix} 0 & 0 & -g\delta_x^{\mathrm{T}} \\ 0 & 0 & 0 \\ H_0\delta_x & 0 & 0 \end{bmatrix}, \qquad \bar{J}_y = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -g\delta_y^{\mathrm{T}} \\ 0 & H_0\delta_y & 0 \end{bmatrix}, \tag{3.29}$$

and where the submatrices $H_0\delta_x$ and $H_0\delta_y$ follow again from the discretization (3.10.b) with constant depth H (i.e. H_0) for the x and y-direction, respectively. Starting from this Jacobian, the structure of the smoothing matrix will become complicated and consequently an expensive smoothing arises. Therefore, we applied one-dimensional smoothing in the x and y-direction, successively. This smoothing is defined in terms of Q (see 3.19) by

$$\mathbf{S}(\mathbf{F}) = Q(2\frac{\Delta x}{\sqrt{gH_0}}\bar{J}_x)(Q(2\frac{\Delta x}{\sqrt{gH_0}}\bar{J}_y)\mathbf{F} + \mathbf{g}_x) + \mathbf{g}_y, \tag{3.30}$$

where $\mathbf{g}_x$ and $\mathbf{g}_y$ are correction terms defined similarly as in the one-dimensional case (cf. (3.25)).

3.6.4. Analysis of smoothing procedures. Having developed our explicit smoothing technique for one-dimensional and two-dimensional grid functions, and having shown its implementational simplicity, we will now proceed with analysing the effect of this particular smoothing procedure on the spectral radius of the Jacobian matrix associated with the SWEs. We start with a lemma characterizing the function $Q(z)=\tilde{Q}(z^2)$ introduced in (3.19).

LEMMA 3.6.1. *The function $\tilde{Q}(z)$ for the smoothing (3.20) is given by the polynomial*

$$P_{2^q-1}(z)=\prod_{k=1}^{q}(1+\mu_k(\frac{T_{2^{k-1}}(1+2z)-1}{2})), \tag{3.31}$$

where $T_{2^{k-1}}$ is a Chebyshev polynomial of degree 2^{k-1}.

PROOF. The result follows immediately (cf. (3.20)) if we can proof that D_k is generated by the polynomial

$$D_k=\tfrac{1}{2}(T_{2^{k-1}}(I+2D_1)-I),\ k\geqslant 1. \tag{3.32}$$

This can be shown by induction as follows. Clearly, for $k=1$ (3.32) is valid. Further, from (3.20) we have

$$D_{k+1}=4D_k(I+D_k),\ k\geqslant 1;$$

and consequently, on substitution of (3.32), we obtain

$$D_{k+1}=4\frac{T_{2^{k-1}}(I+2D_1)-I}{2}\,(I+\frac{T_{2^{k-1}}(I+2D_1)-I}{2}). \tag{3.33}$$

Using $T_{2^k}-1=2(T_{2^{k-1}}^2-1)$ in (3.33), (3.32) follows. □

In the following, we will use the term reduction factor, by which we mean the factor by which the spectral radius of the Jacobian matrix is reduced when smoothing is applied. A useful lower bound for this reduction factor is given in the subsequent lemma.

LEMMA 3.6.2. *On application of the smoothing procedure (3.30) with $Q(z)=\tilde{Q}(z^2)$, the spectral radius of the Jacobian (3.29) is at least reduced by*

$$\frac{1}{\max\limits_{0\leqslant z\leqslant 1}(\tilde{Q}(-z^2)z)}. \tag{3.34}$$

PROOF. In order to derive the reduction factor, we compare the spectral radius of the smoothed Jacobian with the spectral radius of the non-smoothed Jacobian. The smoothed Jacobian can be written in the form

$$\tilde{Q}(\frac{(2\Delta x)^2}{gH_0}\bar{J}_x^2)\tilde{Q}(\frac{(2\Delta x)^2}{gH_0}\bar{J}_y^2)(\bar{J}_x+\bar{J}_y). \tag{3.35}$$

For stability, it is enough to consider the spectral radius of the smoothed Jacobian as $\bar{J}_x$, $\bar{J}_y$ and $\bar{J}_x+\bar{J}_y$ are each similar to a normal matrix by the same

diagonal transformation matrix $\Lambda = diag(\Lambda_1, \Lambda_2, \Lambda_3)$, where $\Lambda_1 = \Lambda_2 = (gH_0)^{1/4} I$ and $\Lambda_3 = (gH_0)^{-1/4} I$. Here, the size of Λ_i corresponds with the size of the diagonal matrices of (3.29) (or (3.36)). Due to this similarity property, the numerical integration by the Runge-Kutta method is stable in the L_2 norm if the eigenvalues of (3.35) multiplied by Δt are within the stability domain drawn in Figure 3.5 (see [33, p.75 and p.79]). In the following, we will derive the eigenvalues of (3.35). Elaboration of (3.35) yields

$$\begin{bmatrix} 0 & 0 & -g\bar{\delta}_x^{\mathrm{T}} \\ 0 & 0 & -g\bar{\delta}_y^{\mathrm{T}} \\ H_0\tilde{Q}(-(2\Delta x)^2\delta_y\delta_y^{\mathrm{T}})\bar{\delta}_x & H_0\tilde{Q}(-(2\Delta x)^2\delta_x\delta_x^{\mathrm{T}})\bar{\delta}_y & 0 \end{bmatrix}, \tag{3.36}$$

where

$$\bar{\delta}_x = \tilde{Q}(-(2\Delta x)^2\delta_x\delta_x^{\mathrm{T}})\delta_x,\ \bar{\delta}_y = \tilde{Q}(-(2\Delta x)^2\delta_y\delta_y^{\mathrm{T}})\delta_y.$$

Solving the eigenvalue problem for this matrix, we find that the nonzero eigenvalues are determined by

$$\det[-gH_0\{\tilde{Q}(-(2\Delta x)^2\delta_y\delta_y^{\mathrm{T}})\tilde{Q}^2(-(2\Delta x)^2\delta_x\delta_x^{\mathrm{T}})\delta_x\delta_x^{\mathrm{T}} + \tilde{Q}(-(2\Delta x)^2\delta_x\delta_x^{\mathrm{T}})\tilde{Q}^2(-(2\Delta x)^2\delta_y\delta_y^{\mathrm{T}})\delta_y\delta_y^{\mathrm{T}}\} - \lambda^2 I] = 0.$$

The matrices $\delta_x\delta_x^{\mathrm{T}}$ and $\delta_y\delta_y^{\mathrm{T}}$ are normal and they commute. Hence, the eigenvalues are found to be

$$\lambda = \sqrt{\frac{-gH_0}{(2\Delta x)^2}\{\tilde{Q}(-\lambda_y^2)\tilde{Q}^2(-\lambda_x^2)\lambda_x^2 + \tilde{Q}(-\lambda_x^2)\tilde{Q}^2(-\lambda_y^2)\lambda_y^2\}} \tag{3.37}$$

where λ_x and λ_y are the eigenvalues of the matrices $2\Delta x\sqrt{\delta_x\delta_x^{\mathrm{T}}}$ and $2\Delta x\sqrt{\delta_y\delta_y^{\mathrm{T}}}$, respectively. These eigenvalues are real and positive and contained in the interval $[0,1]$. The reduction factor is found by dividing the maximum eigenvalue without smoothing ($\tilde{Q}=1$ in (3.37)) by the maximum value with smoothing. This gives the ratio

$$\frac{\max\limits_{0\le\lambda_x,\lambda_y\le 1}[\lambda_x^2+\lambda_y^2]^{\frac{1}{2}}}{\max\limits_{0\le\lambda_x,\lambda_y\le 1}[\tilde{Q}(-\lambda_y^2)\tilde{Q}^2(-\lambda_x^2)\lambda_x^2 + \tilde{Q}(-\lambda_x^2)\tilde{Q}^2(-\lambda_y^2)\lambda_y^2]^{\frac{1}{2}}}. \tag{3.38}$$

As the numerator is equal to $\sqrt{2}$ and the denominator is less than

$$\max_{0\le\lambda_x,\lambda_y\le 1}\sqrt{\tilde{Q}^2(-\lambda_x^2)\lambda_x^2 + \tilde{Q}^2(-\lambda_y^2)\lambda_y^2},$$

we have that (3.38) is bounded below by (3.34).□

REMARK. The spectral radius of the Jacobian in case of the fourth-order space discretization is reduced by the same factor as in the case of second-order discretization, which can be shown by the following reasoning. The simplified Jacobian of the fourth-order discretization is of the form

$$(1-\epsilon\frac{(2\Delta x)^2}{gH_0}\bar{J}_x^2)\bar{J}_x+(1-\epsilon\frac{(2\Delta x)^2}{gH_0}\bar{J}_y^2)\bar{J}_y, \tag{3.39}$$

where $\epsilon=1/6$ and $\bar{J}_x$ and $\bar{J}_y$ are given in (3.29). Now, we obtain instead of (3.37)

$$\lambda=[\frac{-gH_0}{(2\Delta x)^2}\{\tilde{Q}(-\lambda_y^2)\tilde{Q}^2(-\lambda_x^2)(1+\tfrac{1}{6}\lambda_x^2)^2\lambda_x^2+ \tag{3.40}$$

$$\tilde{Q}(-\lambda_x^2)\tilde{Q}^2(-\lambda_y^2)(1+\tfrac{1}{6}\lambda_y^2)^2\lambda_y^2\}]^{\frac{1}{2}}$$

A ratio similar to (3.38) can be derived using (3.40). The resulting reduction factor is again bounded below by (3.34). Hence, the reduction factor of the fourth-order accurate discretization is estimated by the same factor as the reduction factor of the second-order accurate discretization. We remark that (3.39) is not valid near boundaries. (In order to retain (3.39) near the boundaries, we apply (3.10.c) if possible and (3.10.b) otherwise. Furthermore, a similar discretization should be used to approximate ζ_x near the boundaries.) However, the influence on the reduction factor of this simplification in the analysis was not observed in the problems we have tested.

THEOREM 3.6.1. *Let β be the imaginary stability boundary of the classical Runge-Kutta method, i.e. $\beta=2\sqrt{2}$. Let the main terms of the SWEs dominate the spectral radius of the Jacobian matrix (i.e. the spectral radius of the Jacobian matrix ρ is given by $\rho\approx(\sqrt{2gH_{\max}})/\Delta x$, where $H_{\max}$ is the maximum value of the depth in the computational domain). Then application of the smoothing generated by (3.31) to the SWEs leads for $\mu_k=1$ to the stability condition*

$$\Delta t<\beta\, 2^q\tfrac{3}{4}\sqrt{3}/\rho. \tag{3.41}$$

PROOF. According to Lemma 3.6.2, we have to find the maximum of $\tilde{Q}(-z^2)z$ for $z\in[0,1]$, where $\tilde{Q}$ is given by (3.31). For $\mu_k=1$ we have that (3.31) is equal to

$$P_{2^q-1}(z)=\frac{T_{2^q}(1+2z)-1}{4^q 2z} \tag{3.42}$$

(see [19]). The maximum of $P_{2^q-1}(-z^2)z$ for $z\in[0,1]$ is equal to the maximum of $\sqrt{P_{2^q-1}^2(-z^2)z^2}$ for $z\in[0,1]$, which in turn is equal to the maximum of $\sqrt{-P_{2^q-1}(z)P_{2^q-1}(z)z}$ for $z\in[-1,0]$. Substitution of (3.42) in the latter expression yields

$$\max_{-1\le z\le 0}\sqrt{-\frac{T_{2^q}(1+2z)-1}{4^q 2z}\frac{T_{2^q}(1+2z)-1}{4^q 2}}. \tag{3.43}$$

Using the identity $T_{2^q}-1=2(T_{2^{q-1}}^2-1)$, we have that (3.43) is equal to

$$\max_{-1\le z\le 0}\frac{1}{2^q}\sqrt{\frac{T_{2^{q-1}}(1+2z)-1}{4^{q-1}2z}}\sqrt{-\tfrac{1}{2}(T^*-1)(T^*+1)^2}, \tag{3.44}$$

where $T^* = T_{2^{q-1}}(1+2z)$. The first square root term (cf. (3.42)) is at most one (see also [19]). The second is less than $4\sqrt{3}/9$, which follows from an elementary analysis. Hence, the reduction of the spectral radius of the smoothed Jacobian is at least $3\sqrt{3}\,2^q/4$.
For the full non-linear SWEs, we assume that the method is stable if a linearized numerical model for the SWEs, with constant coefficients, is stable for every set of coefficients assumed somewhere in the domain in the non-linear numerical model (see also [33]). As the main terms of the SWEs dominate the spectral radius, we find, according to this approach, the stability condition (3.41) is found. □

The arguments given in the preceeding Remark lead us to the following corollary.

COROLLARY. *The stability condition (3.41) holds also when the fourth-order space discretization is applied except that the spectral radius is now given by* $\rho \approx (7/6\sqrt{2gH_{\max}})/\Delta x$.

Due to the simple structure of D_k (cf. Figure 3.7) the number of operations is linear in q, whereas the maximum allowed time step increases exponentially with q. Thus a very efficient smoothing is constructed.
In practical computations, μ_k is chosen less than 1 in order to obtain diagonal dominancy in (3.20) for all k. The values used are given by

$$\mu_k = 1 - 2^{-(q+1-k)}. \tag{3.45}$$

For this choice of μ_k the constant $3\sqrt{3}/4$ in (3.41) has to be replaced by 1.
Explicit methods should satisfy the Courant-Friedrichs-Lewy condition. The CFL condition says that for an hyperbolic problem the convex hull of the domain of dependence of the exact solution at some point in space and time must be contained in the convex hull of the domain of dependence of the approximating solution at the same point. From (3.20), it can be shown that the influence domain of the explicit method increases exponentially with q. According to the CFL condition this increase is optimally exploited if the time step is also allowed to increase exponentially. As argued above this is the case, so that the numerical domain of dependence is as large as the physical domain of dependence of the PDE itself.
Finally, we give a similar theorem for implicit operators. (The various quantities are defined in Theorem 3.6.1.)

THEOREM 3.6.2. *Let the main terms of the SWEs dominate the spectral radius of the Jacobian matrix. Then application of the implicit smoothing operator generated by (3.27) yields the stability condition*

$$\Delta t < \beta\sqrt{\mu}/\rho. \tag{3.46}$$

PROOF. For the implicit smoothing generated by (3.27) we have to find the maximum of the expression $z/(1+\mu z^2/4)$ for $z \in [0,1]$. This is

straightforward and leads to a reduction of the spectral radius of the Jacobian by a factor $\sqrt{\mu}$. By a similar reasoning as in the proof of Theorem 3.6.1, we arrive at the condition (3.46). □

For any time step Δt, the parameters μ and q of the implicit and explicit smoothing operator, respectively, can be chosen such that a stable method results.
In practice, the bottom profile may change considerably over the domain. This may result in a too strong smoothing in shallow regions. Therefore, we have made μ_k and μ (of the explicit and implicit smoothing, respectively) dependent on the depth.

3.6.5. Accuracy. The local error introduced by the smoothing (3.19) can be investigated by considering at an internal point the expression

$$((Q(\bar{J}_n)-I)\phi)_j, \tag{3.47}$$

where ϕ is a smooth test function and $\phi_j=\phi(j\Delta x)$. Let $Q(z)$ again be of the form $Q(z)=\tilde{Q}(z^2)$, then the error (3.47) can be written as

$$((\tilde{Q}(D_1)-I)\phi)_j, \tag{3.48}$$

with D_1 given in (3.20). For small z, a Taylor expansion of $\tilde{Q}(z)$ yields

$$\tilde{Q}(z)=1+\frac{d\tilde{Q}}{dz}(0)z+\frac{1}{2}\frac{d^2\tilde{Q}}{dz^2}(0)z^2+O(z^3). \tag{3.49}$$

Furthermore, $D_1\phi$ is given by

$$(D_1\phi)_j\approx\frac{(2\Delta x)^2}{4}\frac{\partial^2\phi}{\partial x^2}(j\Delta x). \tag{3.50}$$

Substitution of (3.50) into (3.48) reveals that the error decreases quadratically with Δx if $d\tilde{Q}(z)/dz\neq 0$. For the smoothing operator generated by (3.31), $d\tilde{Q}(z)/dz$ is found to be

$$\begin{aligned}\frac{d\tilde{Q}}{dz}(0)&=\sum_{k=1}^{q}\mu_k(2^{k-1})^2\prod_{l=1,l\neq k}^{q}\left(1+\mu_l\frac{T_{2^{l-1}}(1)-1}{2}\right) \\ &=\sum_{k=1}^{q}\mu_k 4^{k-1}.\end{aligned} \tag{3.51}$$

Hence, for $\mu_k=1$ we find the local truncation error

$$((\tilde{Q}(D_1)-I)\phi)_j=\frac{1}{3}(4^q-1)(\Delta x)^2\frac{\partial^2\phi}{\partial x^2}(j\Delta x)+O(\Delta x^4). \tag{3.52}$$

For the SWEs the magnitude of this error can be expressed in terms of the time step if the maximum allowed time step after smoothing is used. Evidently, the amount of smoothing needed in order to stabilize the method decreases with the time step and as a consequence the error due to smoothing decreases. The order by which this error decreases with the time step (see also [49])

determines the order of accuracy of the smoothing. For the error (3.52) we proceed as follows. If we use the maximum allowed time step in (3.41), then the resulting relation for Δt and Δx can be written as

$$\Delta x = \frac{\Delta t}{2^q \beta \frac{3}{4}\sqrt{3}}(\rho \Delta x), \tag{3.53}$$

where the factor $\rho\Delta x$ is, according to the definition of ρ in Section 3.6.1, independent of Δx. Substitution of (3.53) into (3.52) yields

$$((\tilde{Q}(D_1)-I)\phi)_j = \frac{16}{81}\frac{2^{2q}-1}{2^{2q}}(\frac{\Delta t}{\beta})^2(\rho\Delta x)^2\frac{\partial^2\phi}{\partial x^2}(j\Delta x)+O((\Delta t)^4). \tag{3.54}$$

Hence, the local error decreases quadratically with Δt and therefore the smoothing is second-order accurate in time. According to (3.30) the expression (3.54) corresponds to the truncation error introduced by smoothing the first momentum equation and the continuity equation. Its analogue for the y-direction is introduced by smoothing the second momentum equation and again the continuity equation.
In a similar way, the truncation error introduced by the implicit operator generated by (3.27) can be derived. For this operator we find the derivative of $\tilde{Q}(z)$ to be simply $\mu/4$. Consequently, the error is

$$((\tilde{Q}(D_1)-I)\phi)_j = \frac{\mu}{4}(\Delta x)^2\frac{\partial^2\phi}{\partial x^2}(j\Delta x)+O((\Delta x)^4). \tag{3.55}$$

Using the maximum allowed time step according to (3.46) we obtain

$$((\tilde{Q}(D_1)-I)\phi)_j = \frac{1}{4}(\frac{\Delta t}{\beta})^2(\rho\Delta x)^2\frac{\partial^2\phi}{\partial x^2}(j\Delta x)+O((\Delta t)^4). \tag{3.56}$$

Again we observe that the smoothing is second-order accurate in time. This truncation error and its analogue for the y-direction are introduced by smoothing the respective equations in exactly the same way as the case of the explicit smoothing.

3.7. Discretization of the weakly-reflective boundary conditions

In this section, details will be given on the discretization of the weakly-reflective boundary conditions as given by (2.4) and (2.5).
The discretization of (2.4) and (2.5) at a left boundary is given by

$$U^{new}+\gamma\frac{U^{new}-U^{old}+\sqrt{g/H}E(Z^{new}-Z^{old})}{t^{new}-t^{old}} = (\Phi^U)^{new} \tag{3.57}$$

and

$$Z^{new}+\gamma\frac{E(U^{new}-U^{old})+\sqrt{g/H}(Z^{new}-Z^{old})}{t^{new}-t^{old}} = (\Phi^Z)^{new}, \tag{3.58}$$

respectively. Here, E is the shift operator as defined in Section 3.3 and Φ^U and Φ^Z respectively are the value of U and the value of Z as given at the boundary. The superscripts in (3.57) and (3.58) depend on the stage of the

four-stage Runge-Kutta time integrator in which the various quantities are computed (see Section 3.5). For the first stage *new* is at time level n and *old* at time level $n-1$. In the other stages *new* is at time levels $n+1/2$, $n+1/2$ and $n+1$, respectively, and *old* is at time level n.
The weakly-reflective boundary conditions (2.4) and (2.5) have also implications for the boundary treatment of the stabilization. But in the present version we have refrained from implementing this treatment, because of complexity. Nevertheless, we found in the experiments that the implementation of (3.57) and (3.58) results in a satisfactory weakly-reflective behaviour of the open boundaries.

3.8. Drying and flooding

In many problems, it occurs that during the tide some part of the domain becomes dry land. Such dry flats, if not handled correctly may cause numerical instabilities. Therefore, following the ideas of Stelling [38, p. 153], prior to every time step the following actions with respect to drying and flooding are performed:

1. In all velocity points it is checked whether

$$H < H_{\min} \tag{3.59}$$

   where H is the total depth and $H_{\min}$ is an a-priori given minimum depth.
2. If the answer of the check in 1. is true at a certain velocity point, then the velocity at this point is set to zero and the point will be treated as a closed boundary.

Furthermore, as it is possible that in the performance of the time step (i.e. in the second, third or fourth stage of the time integrator, see (3.16)) the depth becomes very close to zero or even negative, the following procedure is applied throughout the stages.

a. If after the calculation of H it appears that $H < \epsilon$ at certain velocity points, where ϵ is a small quantity, then we set $H = \epsilon$ at these points. This avoids that the depth becomes negative during the time step and furthermore it avoids overflow during the division by H in the bottom friction term. This approach is different from that of Stelling. In the latter case, such a point is treated as a closed boundary point.
b. In shallow regions, i.e. where H is small, the factor $\Delta t g \sqrt{U^2+V^2}\,/(C^2 H)$, occurring in the bottom friction term, may become very large. (In that case, the flow is slowed down strongly.) The classical Runge-Kutta method is unstable if this factor is greater than 2.78 (see Figure 3.5). Therefore, we test whether the factor is greater than 2 (below we explain why we use 2 instead of 2.78). If at a certain velocity point the outcome of the test is true then we set the factor at this point equal to 2. We do not want to set this factor equal to 2.78, because the amplification factor of points on the boundary of the stability domain (2.78 is on the boundary) is equal to one, whereas the amplification factor is almost minimal if we set $\Delta t g \sqrt{U^2+V^2}\,/(C^2 H)$

equal to 2. A minimal amplification factor is to be preferred because it represents better the strong damping behaviour of the bottom friction term in very shallow regions.

4. VECTORIZATION ASPECTS

In this section, we will describe the vectorization aspects of the SWEs solver. The subjects that will be dealt with are: the choice of the time-integration method and its stabilization, the boundary treatment, the drying and flooding procedure and the data structure.

4.1. Preliminaries

On the CYBER 205 we used the language FORTRAN 200 [1], which contain (vector) extensions with respect to FORTRAN 77. In this section, some typical vector programming features of this language will be briefly described.

Vectors. On the CYBER 205 a vector is defined as a series of values that are stored in contiguous memory locations. Vectors can be referenced by so-called *vector references* or by *descriptors*. A vector reference or descriptor specifies the following information: the first element of the vector, which must be an array element, the length of the vector, and the data type of the vector.

EXAMPLE 4.1. Declare an array by `DIMENSION A(10)`. Then the vector reference, compactly denoted as `A(3;5)`, refers to the vector `A(3),A(4),A(5),A(6),A(7)`.
Furthermore, declare a descriptor by `DESCRIPTOR ADESC`. Then by the assignment `ASSIGN ADESC, A(3;5)` we achieve that `ADESC` denotes the same vector as `A(3;5)` .□

Some "DO-loops" can be rewritten by using these vector references or descriptors.

EXAMPLE 4.2. The "DO-loop"

```
      DO 1 I=1,5
         A(I)=A(I)+A(5+I)
    1 CONTINUE
```

can be written in the form

```
      A(1;5)=A(1;5)+A(6;5)
```

using vector references, or in the form

```
      DESCRIPTOR ADESC1, ADESC2
      ASSIGN ADESC1, A(1;5)
```

```
ASSIGN ADESC2, A(6;5)
ADESC1=ADESC1+ADESC2
```

using descriptors. □

In some cases, a temporary vector is needed for an intermediate result. Using descriptors, it is possible to define this storage dynamically.

EXAMPLE 4.3. A dynamical vector of length N is defined by

```
ASSIGN ADESC, .DYN.N  □
```

Gather and Scatter operations. "DO-loops" in which indirect addressing is used, do not vectorize well on the CYBER 205 as the data are not stored in contiguous memory locations. Therefore, there exist optimized gather instructions which create vectors from these data on which vector operations can be performed. Furthermore, optimized instructions exist which scatter elements of a vector to non-contiguous memory locations. For our purpose, gather and scatter operations are extremely helpful. We will show by some examples what the effect of these operations is.

EXAMPLE 4.4. In standard FORTRAN the gather operation reads

```
      DIMENSION V1(5),U1(4),I1(4)
      DO 1 I=1,4
         U1(I)=V1(I1(I))
    1 CONTINUE
```

Due to the indirect addressing this "DO-loop" does not vectorize automatically. However, there exists an optimized alternative for this "DO-loop":

```
DIMENSION V1(5),U1(4),I1(4)
U1(1;4)=Q8VGATHR(V1(1;4),I1(1;4);U1(1;4))
```

The scatter operation given in standard FORTRAN is

```
      DIMENSION V1(5),U1(4),I1(4)
      DO 1 I=1,4
         V1(I1(I))=U1(I)
    1 CONTINUE
```

This operation is optimized by

```
DIMENSION V1(5),U1(4),I1(4)
V1(1;5)=Q8VSCATR(U1(1;4),I1(1;4);V1(1;5))
```

Notice that the gather and scatter operations are each others inverse when the same index array `I1` is used. □

Bit vectors An important feature of the FORTRAN 200 language is the availability of the data type `BIT`. Bit vectors are important in the handling of "IF statements". In this case bit vectors are used in connection with `WHERE` constructions.

EXAMPLE 4.5. Consider the "DO-loop"

```
      DIMENSION U1(100),V1(100)
      DO 1 I=1,100
         IF (U1(I) .LT. .0 ) THEN
            V1(I)=100.
         ELSE
            V1(I)=-100.
         ENDIF
    1 CONTINUE
```

Such a "DO-loop" is not vectorized automatically by the FORTRAN 200 compiler. However, using a `WHERE` construction this is vectorized by

```
      DIMENSION U1(100),V1(100)
      WHERE (U1(1;100) .LT. .0)
         V1(1;100)=100.
      OTHERWISE
         V1(1;100)=-100.
      END WHERE
```

An equivalent form is

```
      DIMENSION U1(100),V1(100)
      BIT BITV(100)
      BITV(1;100)=U1(1;100) .LT .0
      WHERE (BITV(1;100))
         V1(1;100)=100.
      OTHERWISE
         V1(1;100)=-100.
      END WHERE
```

In the latter case the information stored in the bit array `BITV` can be used several times. □

Timings. To give some impression of the performance of the CYBER 205, timings and relative costs (with respect to a vector addition) will be given of some elementary operations. The timings are given for `N=1000` in full precision.

```
Declaration
DIMENSION U(N),V(N),W(N),IND(N)
```

Instruction	timings 10^{-5}sec	relative costs
U(1;N)=V(1;N) + W(1;N)	2.1	1.0 (by def.)
U(1;N)=V(1;N) * W(1;N)	2.1	1.0
U(1;N)=(V(1;N) + W(1;N))*C	2.1	1.0
U(1;N)=V(1;N) / W(1;N)	12.6	6.0
U(1;N)=SQRT(V(1;N);U(1;N))	12.6	6.0
U(1;N)=Q8VGATHR(V(1;N), IND(1;N);U(1;N))	3.6	1.7×np for F.P. 3.4×np for H.P.
U(1;N)=Q8VSCATR(V(1;N), IND(1;N);U(1;N))	3.6	1.7×np for F.P. 3.4×np for H.P.

TABLE 4.1. Timings of some elementary operations.

In general, a vector instruction speeds up linearly with the number of vector pipes used (denoted by np in the table). Furthermore, it speeds up by a factor two when changing from full precision (F.P.) representation (14 decimal digits representation) to half precision (H.P.) representation (7 digits representation). These properties do not hold for operations acting on non-contiguous data such as gather and scatter operations. This explains why the gather operation in Table 4.1 becomes, relatively, more expensive with respect to a vector addition, when changing from full precision to half precision or when more vector pipes are used.

4.2. Explicit or implicit methods

In this section, we motivate the choice we made for the numerical time integration of the SWEs. In Table 4.2 we have indicated the vectorizability of the various operations occurring in time integration methods.

type of time integrator	right-hand side evaluation	construction of Jacobian matrix	taking linear combinations of right-hand sides	solving systems of equations
implicit	fully	fully	fully (if occurring)	partly
explicit	fully	-	fully	-

TABLE 4.2. Vectorizability of operations in time integration methods.

The words "fully" and "partly" denote that the operation at hand is fully or partly vectorizable, whereas "-" denotes that the operation is not occurring in the time integrator.
In the table it is indicated that solving a system of algebraic equations is only partly vectorizable. This is mainly due to the inherent recursiveness of the solution process of such systems. Moreover, it is difficult to avoid in such a process operations on non-contiguous data and operations on vectors of moderate length (say less than 50 elements). On the CYBER 205, these operations do not accelerate when we change from full precision to half precision calculations or when a computer with more vector pipes is used. Hence, it is this type of operations which causes an upper limit to the performance of an implicit method on a CYBER 205. For this reason, we decided to use an explicit method which does not have such a limit. A drawback of explicit methods is that the time step may be restricted for stability reasons. This drawback may become important if the variation of the solution in time is small. Therefore, we developed the fully vectorizable stabilization technique as discussed in Section 3.6 by which the stability condition, as we have shown, is relaxed considerably. From the above discussion it is clear that, on the CYBER 205, the explicit approach is to be preferred.

4.3. Boundary treatment

As we assume that the solver should be able to handle arbitrary domains, the vectorization of the boundary treatment needs special attention. First we will describe how the differences are calculated at internal points and thereafter how this is done at boundary points.
Consider the domain given in Figure 4.1, which is covered by a rectangular grid.

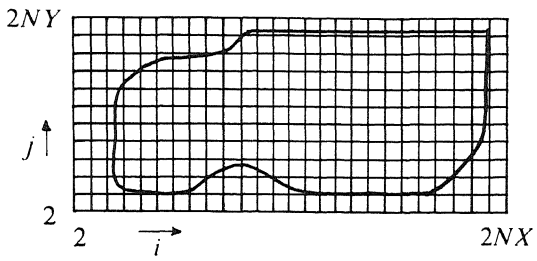


FIGURE 4.1. Example domain.

The variables defined at position (i,j) on this grid (see also Figure 3.1) are stored at location $\{[i/2]-1\}NY+[j/2]$ of the associated array, where NY as well as NX are given in Figure 4.1 and [.] denotes the integer part function. Hence, the points are counted in the y-direction. We will call this storage structure a rectangular storage structure (see also Section 4.5). In the following, we denote by NN the total number of components of one dependent variable, i.e. $NN = NX \times NY$.
Using this storage structure, the calculation of x and y-differences vectorizes well, i.e. the operation can be performed on vectors of length determined by the number of grid points NN. Long vectors are to be preferred because start-up times of the vector instructions become negligible in this case. An x-difference of U is calculated by

```
DIMENSION UX(NN),UY(NN),U(NN)
UX(1;NN)=(U(1+NY;NN)-U(1-NY;NN))/(4*DX)
```

and a y-difference is calculated by

```
UY(1;NN)=(U(1+1;NN)-U(1-1;NN))/(4*DX)
```

The boundary treatment is performed using so-called index arrays. Such an array contains all locations of points which need the same boundary treatment; for example the locations of all left closed boundaries. Using these arrays, the boundary points and, if needed, its neighbours can be gathered from the computational array. Thereafter, the boundary operations can be performed with vector instructions on the gathered arrays and, finally, the results are scattered into the result array.
In the following, we will describe the implementation of the discretization of $(Hu)_x$ given by (3.10). We assume that H is already calculated at U-points.

Due to the staggering (see Figure 3.1) and the way the components of the dependent variables are stored in the arrays (see above), a second-order central difference is computed in all points by the operation

```
      DIMENSION HUX(NN),HU(NN)
      HUX(1;NN)=(HU(1+NY;NN)-HU(1;NN))/(2*DX)
```

A straightforward implementation of the fourth-order discretization of $(Hu)_x$ together with its boundary treatment is given by

```
C ============================================================
C DESCRIPTION OF VARIABLES
C
C   HU(NN)  ARRAY CONTAINING H*U
C   HUX(NN) RESULT ARRAY CONTAINING D(UH)/DX AT EXIT
C   HUXH    DESCRIPTOR; DUMMY VARIABLE
C   HUL1H   DESCRIPTOR; DUMMY VARIABLE
C   HUR1H   DESCRIPTOR; DUMMY VARIABLE
C   HUL2H   DESCRIPTOR; DUMMY VARIABLE
C   HUR2H   DESCRIPTOR; DUMMY VARIABLE
C   I1(I1T) INDEX ARRAY INDICATING THE POINTS WHERE (3.10.b)
C           HAS TO BE APPLIED
C   IL(ILT) INDEX ARRAY INDICATING THE POINTS WHERE (3.10.c)
C           HAS TO BE APPLIED
C   IR(IRT) INDEX ARRAY INDICATING THE POINTS WHERE THE
C           RIGHT-HAND ANALOGUE OF (3.10.c) HAS TO BE APPLIED
C ============================================================
      C1=27./24. * 1/(2*DX)
      C2=-1./24. * 1/(2*DX)
      C3=1./C1   * 1/(2*DX)
      C4=1./C1   * 25./24. * 1/(2*DX)
      C5=C2/C1
C ------------------------------------------------------------
C   CALCULATION OF CENTRAL DIFFERENCES USING ONLY TWO POINTS
C ------------------------------------------------------------
      HUX(1;NN)=(HU(1+NY;NN)-HU(1;NN))*C1
C ------------------------------------------------------------
C   SAVING OF CENTRAL DIFFERENCES NEAR BOUNDARIES
C ------------------------------------------------------------
      ASSIGN HUXH,.DYN.I1T
      HUXH=Q8VGATHR(HUX(1;NN),I1(1;I1T);HUXH)
      ASSIGN HUL1H,.DYN.ILT
      ASSIGN HUL2H,.DYN.ILT
      HUL1H=Q8VGATHR(HUX(1+NY;NN),IL(1;ILT);HUL1H)
      HUL2H=Q8VGATHR(HUX(1+2*NY;NN),IL(1;ILT);HUL2H)
      ASSIGN HUR1H,.DYN.IRT
```

```
      ASSIGN HUR2H,.DYN.IRT
      HUR1H=Q8VGATHR(HUX(1;NN),IR(1;IRT);HUR1H)
      HUR2H=Q8VGATHR(HUX(1-NY;NN),IR(1;IRT);HUR2H)
C -------------------------------------------------------------
C   CALCULATION OF FOURTH-ORDER DIFFERENCES
C -------------------------------------------------------------
      HUX(1;NN)=HUX(1;NN)+(HU(1+2*NY;NN)-HU(1-NY;NN))*C2
C -------------------------------------------------------------
C   CALCULATION OF DIFFERENCES NEAR BOUNDARIES USING SAVED
C   CENTRAL DIFFERENCES
C -------------------------------------------------------------
      HUXH=HUXH*C3
      HUX(1;NN)=Q8VSCATR(HUXH,I1(1;I1T);HUX(1;NN))
      HUL2H=C4*HUL1H+C5*HUL2H
      HUX(1;NN)=Q8VSCATR(HUL2H,IL(1;ILT);HUX(1;NN))
      HUR2H=C4*HUR1H+C5*HUR2H
      HUX(1;NN)=Q8VSCATR(HUR2H,IR(1;IRT);HUX(1;NN))
```

In this approach, an extra index array `I1` is needed for the application of (3.10.b). The index arrays have to be constructed every time step, due to the drying and flooding. Hence, it is important to minimize the number of index arrays. This can be accomplished by factorizing the discretization, which will be described in the subsequent section. Using this factorization, only 12 index arrays are needed. These result from the three boundary types, viz. elevation, velocity or closed boundary, which can each occur at four boundary locations, viz. at the left, at the right, at the bottom or at the top (see Figure 3.2).

4.3.1. Factorization of discretizations. For the numerical approximation of a term of the equations the location of the computational point, under consideration, in the domain determines which variant of the discretization should be used (e.g. the fourth-order, the second-order or the one-sided variant). In general, it is needed to know the position of the point with respect to the boundaries. However, using the factorized form the only information needed is the location of the boundaries themselves. As a consequence the number of index arrays can be minimized and the programming of the discretizations is simplified.
For example, we consider again the discretization (3.10.a). This discretization can also be written in the factorized form

$$[(Hu)_x]=(1+\alpha E^{-2})(1+\alpha E^{2})\beta(E-E^{-1})HU, \tag{4.1}$$

where α and β follow from

$$\begin{aligned} &\alpha\beta=-1/24\times 1/(2\Delta x), \\ &(1+\alpha^2-\alpha)\beta=27/24\times 1/(2\Delta x). \end{aligned} \tag{4.2}$$

Obviously, there are two solutions $\alpha_{\pm}=-13\pm 2\sqrt{42}$. Here, we choose $\alpha=\alpha_{+}$,

because it is small in modulus with respect to 1 and consequently we have diagonal dominance in the factors $(1+\alpha E^{\pm 2})$. In addition to the factorization (4.1), (3.10.c) can be factorized in the form

$$[(Hu)_x]=((1+\alpha+\alpha^2)+\alpha E^2)\beta(E-E^{-1})HU \tag{4.3}$$

with α and β as given in (4.2). The factors of (4.1) are applied successively, each with a boundary treatment. This treatment is such that at the end we have (3.10.a), (3.10.b), (3.10.c) and the analogue of (3.10.c) at the right boundary at the appropriate places. To be more precise, we perform successively the operations ($R1$ and $R2$ are used for intermediate results)

$$\begin{aligned} R1&=\beta(E-E^{-1})HU, \\ R2&=(1+\alpha E^2)R1. \end{aligned} \tag{4.4}$$

At the left boundary, we overwrite $R2$ by

$$R2=(\gamma+\delta E^2)R1.$$

The values of the constants γ, δ and below of ϵ, η, θ, κ are given at the end of this section and are found by comparing the resulting boundary treatment of the factorized form with the discretizations such as given in (3.10).
Successively, at the right boundary, we overwrite $R2$ by

$$R2=\epsilon R1. \tag{4.5}$$

The order in these operations is important because the effect of this particular sequence is that the last equation holds also in the case where there is only one computational point between two boundaries. Thereafter, we perform

$$[(Hu)_x]=(1+\alpha E^{-2})R2.$$

At the right boundary, this is followed by

$$[(Hu)_x]=(\eta+\theta E^{-2})R2.$$

At the left boundary, we finally evaluate

$$[(Hu)_x]=\kappa R2.$$

After these operations we obtain (4.1) ($\equiv$ (3.10.a)) in the interior. Furthermore, we have at the left boundary

$$[(Hu)_x]=\kappa(\gamma+\delta E^2)R1,$$

and at the right boundary (notice that R2 is given by (4.5) at the right boundary and by (4.4) at a point adjacent to this boundary)

$$[(Hu)_x]=(\eta\epsilon+\theta E^{-2}(1+\alpha E^2))R1=(\eta\epsilon+\theta\alpha+\theta E^{-2})R1.$$

Furthermore, in the case where there is only one point between two boundaries we have

$$[(Hu)_x]=\kappa\epsilon R1.$$

Comparing these resulting equations with (4.3), its analogue at the right boundary, and with (3.10.b), the following conditions have to be satisfied:

$$\begin{aligned} &\kappa\gamma = 1+\alpha+\alpha^2 \\ &\kappa\delta = \alpha \\ &\eta\epsilon+\theta\alpha = 1+\alpha+\alpha^2 \\ &\theta = \alpha \\ &\kappa\epsilon = 1/\beta \times 1/(2\Delta x) \end{aligned} \tag{4.6}$$

A solution of these equations is $\theta=\alpha$, $\gamma=1+\alpha+\alpha^2$, $\delta=\alpha$, $\epsilon=1/(\beta\, 2\Delta x)$, $\eta=(1+\alpha)/\epsilon$, $\kappa=1$. We do not know whether there are better choices, but for this solution the factors are also diagonal dominant at the boundaries.

4.4. Drying and flooding

The drying and flooding procedure, described in Section 3.8, may be rather time consuming due to tests which have to be performed to determine the location of the boundary. Hence it is important to vectorize this procedure. Bit arrays play an important role in this vectorization. We will treat this again by an example. For simplicity we consider the one-dimensional case.
Suppose that, after a certain time step, the geometry is as given in Figure 4.2.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

FIGURE 4.2. Geometry after a certain time step.

We assume that at the left boundary (i.e. point 1) the velocity is prescribed and at the right boundary the elevation is prescribed (i.e. physically in the middle between point 14 and 15). Condition (3.59) is checked by the statement

```
BITDR(1;NX)=H(1;NX) .LT. HMIN
```

where $NX=15$ in this example. The bit array `BITDR` contains the following information after this check

0 0 0 0 1 1 1 1 0 1 0 0 0 0 ?

where the question mark indicates that the result of the check is undefined.

The check is undefined for points outside the computational domain such as point 15. Furthermore, a bit array BITOUT is constructed which has elements 1 for velocity-boundary points and for velocity points that are outside the computational domain during the complete simulation. The elements of this bit array for the geometry drawn in Figure 4.2 are given by

```
1  0  0  0  0  0  0  0  0  0  0  0  0  0  1
```

Combining these two bit arrays,

```
BITH(1;NX)=BITDR(1;NX) .OR. BITOUT(1;NX)
```

gives

```
1  0  0  0  1  1  1  1  0  1  0  0  0  0  1
+           *        +     *              *
                             +
```

In this array, we want to determine the location of the left boundaries, indicated by +, and the location of the right boundaries, indicated by *. We perform now

```
BIT2(1;NX)=BITH(1;NX) .XOR. BITH(0;NX)
```

which results in

```
?  1  0  0  1  0  0  0  1  1  1  0  0  0  1
```

Combination of BIT2 and BITH gives

```
BIT3(1;NX)=BIT2(1;NX) .AND. BITH(1;NX)
```

with elements

```
?  0  0  0  1  0  0  0  0  1  0  0  0  0  1
```

We have now obtained 1 bits at right boundary locations. As the first point cannot be a right boundary the corresponding first element is set to zero. The index array follows from:

```
      LIND = Q8SCNT(BIT3(1;NX))
C     LIND=3
      INDR(1;LIND)= Q8VCMPRS(IND(1;NX),BIT3(1;NX);INDR(1;LIND))
C CONTENTS OF IND
C     1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
C CONTENTS OF INDR
C     5 10 15
```

The first statement counts the number of 1 bits and stores it in `LIND`, i.e. 3 in this case. The second statement compresses the elements of the array `IND` indicated by the bit array `BIT3` into `INDR`.
If we now perform

```
      BIT3(1;NX)=BIT2(2;NX) .AND. BITH(1;NX)
```

then `BIT3` contains

```
      1  0  0  0  0  0  0  1  0  1  0  0  0  0  ?
```

`BIT3` has 1 bits at left boundary locations. Again the question mark should be replaced by a zero because at this place no left boundary can occur. In the same way as before, we now find the index array `INDL(1;LIND)` with elements

```
      1  8 10
```

It should be noticed that `INDL(I)` and `INDR(I)` respectively give the start and end point of a row of "wet" points. Prior to the time integration, index arrays are constructed giving the indices of the open boundaries. These index arrays are now used to mark the indices in `INDL` and `INDR` which correspond to open boundaries. Once this is performed, the non-marked indices represent closed boundaries which can be derived straightforwardly from `INDL` and `INDR`.

4.5. Data structure

In order to obtain an optimal performance of the solver on the CYBER 205 it is important to consider the data structure carefully. A computation on a rectangular domain (see Section 4.3) is to be preferred from a vectorization stand-point. However, often the geometries are very complex, which may lead to a substantial overhead in the computational costs if the domain is simply covered by a rectangle. Therefore, we considered in [50] a number of techniques to reduce this overhead (see also [41]). The essence of these techniques is that an x and an y-ordering is constructed for the computational arrays. If the arrays are ordered according to the x-ordering, then the x-differences can be calculated efficiently. Likewise, if the arrays are ordered according to the y-ordering, then the y-differences can be calculated efficiently. These two

orderings imply that during the performance of the right-hand side evaluation reorderings have to be performed to change from x-ordering to y-ordering and vice versa. The x and y-ordering should be such that the reordering operation is as efficient as possible.
We have refrained from implementing such a technique as the geometries encountered in many practical problems can be enclosed in rectangular region with only introducing a relatively small number of dummy grid points. Nevertheless, such a technique can be implemented without much effort.

4.6. On the computational costs of the CYBER 205 code
In this section, we discuss the computational costs of the numerical method implemented. It appears that the CPU (Central Processing Unit) time per grid point per time step depends on the number of grid points in the actual application. In order to quantify this dependence, we have performed computations on various grids for a square geometry. At the left and right boundary of this square the velocity and the elevation are respectively prescribed, whereas the upper and lower boundary are closed. The conditions at the open boundaries are time dependent. In Table 4.3, we give the timings of the computations including smoothing ($q=3$ in (3.20)).

	$N=20\times20$			$N=40\times40$			$N=80\times80$			$N=160\times160$		
type of operation	CPT $10^{-3}s$	$\frac{CPT}{\sqrt{N}}$ $10^{-5}s$	$\frac{CPT}{N}$ $10^{-6}s$	CPT $10^{-3}s$	$\frac{CPT}{\sqrt{N}}$ $10^{-5}s$	$\frac{CPT}{N}$ $10^{-6}s$	CPT $10^{-3}s$	$\frac{CPT}{\sqrt{N}}$ $10^{-5}s$	$\frac{CPT}{N}$ $10^{-6}s$	CPT $10^{-3}s$	$\frac{CPT}{\sqrt{N}}$ $10^{-5}s$	$\frac{CPT}{N}$ $10^{-6}s$
UPDBC	.43	2.1	1.1	.68	1.7	.4	1.2	1.4	.18	2.2	1.4	.086
CHECK	.24	1.2	.6	.28	.7	.2	.44	.55	.07	1.0	.63	.039
ADAP	1.85	9.3	4.6	2.6	6.5	1.6	5.0	6.3	.78	11.	6.8	.43
TIMEST	28.0	14.0	70.0	43.2	108.	27.0	100.	125.	15.6	320.	200.	12.5
TOTAL	30.5	152.	76.3	46.6	116.	29.1	107.	134.	16.7	334.	208.	13.1

TABLE 4.3. Timings for various grids per time step in case of a fourth-order space discretization.

The number of grid points (N) is chosen 20×20, 40×40, 80×80 and 160×160 in these runs. In the first column, the timed operations are specified. UPDBC computes the values at the boundaries at the new time level from a sine series (see Section 5.2.2), CHECK checks prior to every time step whether the geometry has been changed since the previous time step, ADAP adapts the index arrays if the geometry is changed, and TIMEST performs the actual time step. The next four columns give the data corresponding to the grid specified in their respective headers. Each of these columns consists of three subcolumns. In the first subcolumn the observed computation times (indicated by CPT) of the various operations are listed. In order to compare these values for the various grids, we have given in the second subcolumn the computation

times divided by the square root of the number of grid points and in the third subcolumn the computation times divided by the total number of grid points. (Note that the square root of the total number of grid points gives, up to a constant, the number of boundary points.)
Globally, we observe from this table that the computation speed drops substantially if the number of grid points decreases; for the grid with 400 points the (overall) speed is more than 5 times smaller than for the grid with 25600 points. Furthermore, we observe that we can distinguish operations whose costs increase linearly with $\sqrt{N}$ (e.g., UPDBC and ADAP), and operations whose costs increase linearly with N.
In accordance with this observation we assume that the total computation time of the method per time step is determined by an expression of the form

$$a+b\sqrt{N}+cN, \tag{4.7}$$

where a,b and c are constants. A least squares fit of (4.7) to the values for the total computation times given in the table yields

$$a=22576.\ 10^{-6},\ b=156.22\ 10^{-6},\ c=11.24\ 10^{-6}. \tag{4.8}$$

In Figure 4.3, we have drawn the curve of the CPU time per grid point per time step using the coefficients given by (4.8). Hence, the generating formula is obtained by (4.7) divided by N. Furthermore, the values given in Table 4.3 are indicated in Figure 4.3 by the symbol +.

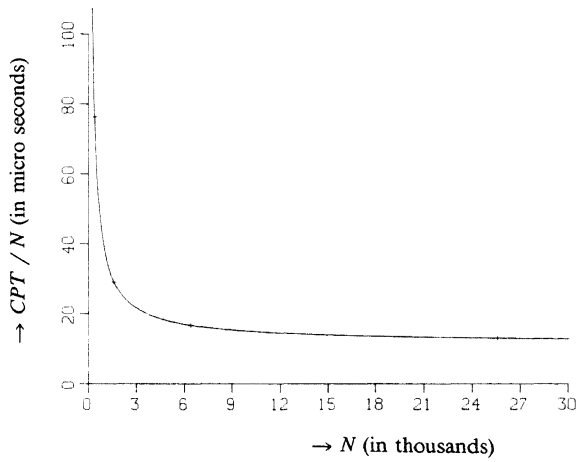


FIGURE 4.3. Plot of the CPU time per time step per grid point.

This figure shows that the computation speed is close to its maximum if the number of grid points is larger than say 4000. For smaller values the start-up times slow down the speed significantly.

Apart from the measurements listed in Table 4.3, the value of c can also be obtained by counting all vector operations in the code, which act on vectors of

length N. As a unit for measuring the costs of the various vector operations we used a vector addition (cf. Table 4.1). It turned out that the code contained 1020 of such unit vector instructions per time step. As a unit vector instruction in half precision produces one result per 10 nano seconds, we are led to the value $10.2\ 10^{-6}$ for c. This value is within 10% of the observed value.
Furthermore, we have computed the megaflop rate of the code. Therefore, we counted the number of floating point operations. This number is about $3.3\ 10^{7}$ for the 160×160 grid. The time needed for these operations is found in the table, i.e. .334 seconds. Hence the code runs at about 100 megaflops.

In addition to the timings in Table 4.3, we performed the following computations on the 160×160 grid:

(i) without smoothing,
(ii) without smoothing and second-order space discretization,
(iii) case (ii) on a two-pipe CYBER 205.

Control Data Corporation is greatly acknowledged for offering us the opportunity to perform case (iii). The results are given in Table 4.4.

	$N=160\times160$	
type of run	CPT $10^{-3}s$	$\frac{CPT}{N}$ $10^{-6}s$
from Table 4.3	334	13.1
case (i)	177	6.9
case (ii)	139	5.4
case (iii)	79	3.1

TABLE 4.4. Timings per time step for the 160×160 grid

Hence, the method is almost twice as cheap when smoothing is not used. However, the time step must be chosen considerably smaller, which offsets the advantage. Furthermore, the code speeds up by about 20% when (in addition) second-order differences are used. Finally, the speed up of the latter method using a two-pipe CYBER 205 (instead of a one-piper) confirms our expectation that the method becomes almost twice as fast when changing from a one-pipe to a two-pipe CYBER 205 (see Section 4.2).

5. THE PROGRAM SYSTEM

5.1. The system parts

The system consists of three program parts: the INPUT PROCESSOR, the SOLVER and the OUTPUT PROCESSOR. The flow chart of the system is given schematically in Figure 5.1.

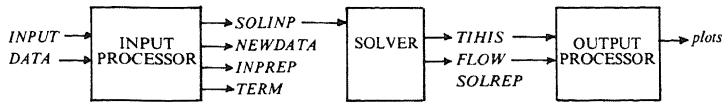


FIGURE 5.1. Flow of the system.

The INPUT PROCESSOR is an interactive program running on the front end of the CYBER 205. By this program part, the user can specify his problem. The input for this part is given by means of two files:

INPUT This file is connected to the terminal by the INPUT PROCESSOR. By means of this file the user can give input to the program. The input consists of answers to questions written by the program to the terminal display by means of the connected file *TERM*. These questions deal with the data needed to define the problem and with the control of the INPUT PROCESSOR.

DATA This file contains data defining the problem specified by the user in a previous run of the INPUT PROCESSOR. The file is obtained by renaming the file *NEWDATA*, which was created in the previous run, to *DATA*. The input on *DATA* is read by the program and written to the terminal display part by part. The user can change this data and thereby create a new model. It should be noted that the INPUT PROCESSOR can be executed without this file giving all input by means of the file *INPUT*.

The INPUT PROCESSOR generates four files:

SOLINP This file contains job control statements and input for the execution of the SOLVER.

NEWDATA All user-given input is written on this file. It is of the same format as the file *DATA*. On renaming it to *DATA*, the user can create a related model in a convenient way.

INPREP On this file a report of the user-defined problem is written.

TERM This file is used by the program to write questions and data to the terminal display.

The SOLVER, running at the CYBER 205, performs the actual computation and generates three files:

TIHIS This file contains time-history data at user-specified space

points. It is only generated if time histories are requested by the user.

FLOW This file contains flow-field data from the flow at the end time of the simulation and from the flow at user-specified times during the simulation.

SOLREP On this file a report of the simulation is written.

The OUTPUT PROCESSOR, running at the front end, generates plots of the time-history data and vector plots of the flow-field data.
In the following sections the program parts will be described in more detail.

5.2. *The* INPUT PROCESSOR

The input for the INPUT PROCESSOR consists of six parts:

the domain definition,
specification of the boundary conditions,
initialization of the U, V and Z-field,
definition of the depth and Manning values,
definition of problem and integration parameters,
definition of output parameters.

The Manning values, which are not mentioned before, are used for the calculation of the Chezy coefficient C (see Section 5.2.4 and formula (2.1)).
To some extent the program checks the user-given data on consistency, in order to obtain at the end of the input process a well-defined model. However, it is a very time consuming task to construct an input processor which guarantees a well-defined model on exit. This is beyond the scope of the project. Therefore our aim was to construct an input processor by which a skilled user can specify his problem in a convenient way.
In the following, the six parts of the INPUT PROCESSOR will be discussed in more detail.

5.2.1. Domain definition. The contour of the domain is approximated by a polygon. This polygon consists of line pieces which are parallel to either the x-axis or the y-axis. The staggering of the grid (see Section 3.1) has some consequences for the definition of the polygon.
The polygon is defined by its angle points, $\{(X_i, Y_i) \in \mathbb{N} \times \mathbb{N} \mid i = 1, \ldots, n\}$, where the integers X_i and Y_i are the numbers of the grid lines defining the original (i.e. non-staggered) grid (see Figure 3.1). The contour is found from this sequence by connecting successive points by straight lines. Furthermore, according to Figure 3.1, the type of the boundaries is specified as follows. If X_i is even (odd) then there is a U-boundary (Z-boundary) in the "vertical" direction. Likewise, if Y_i is even (odd) then there is a V-boundary (Z-boundary) in the "horizontal" direction. The Z boundaries are always open but a U or V-boundary is open or closed. The type of the *velocity* boundaries is defined by a parameter B_i; $B_i = 0$ or 1 means that the boundary between

(X_i, Y_i) and (X_{i+1}, Y_{i+1}) is open or closed, respectively. For programming reasons, the value of B_i for Z-boundaries, which are always open, should also be zero. Furthermore, as will become clear below, we require that the contour is passed in clockwise order when passing through the sequence of angle points.

In the next section, on boundary conditions, it will be pointed out that B_i may also have the value 100 which means that the next part is open and that at this point boundary condition parameters have to be prescribed. Default, the program will detect the necessary points at which boundary condition data have to be specified in order to have a well-posed problem. If the user wants to specify data at other points, then these points should be marked by setting $B_i = 100$. We will return to this matter in the next section.

Thus, the polygon the program accepts is defined by a set of 3-tuples $\{(X_i, Y_i, B_i) \in \mathbb{N} \times \mathbb{N} \times \{0,1,100\} \mid i = 1, \ldots, n\}$. These 3-tuples must have the property that either $X_i = X_{i+1}$ or $Y_i = Y_{i+1}$ for $i = 1, \ldots, n-1$ and $X_n = X_1$ or $Y_n = Y_1$. If this property does not hold for a closed boundary, then points are inserted such that the property holds. In some cases, this insertion may not be unique. In such a case, the program chooses that point which is closest to the straight line drawn between its two neighbours. From the two points resulting from this approach the program will choose the one which is outside the domain. The points can only be inserted correctly if the boundary data is given in clockwise order. We will clarify the functioning of the insertion routine by some examples.

EXAMPLE 5.1. Let the first two points of the input be given by {(0,0),(4,4)}. Then the insertion routine starts at the first point and checks whether a point should be inserted. This is the case and the unique intermediate result is {(0,0),(2,2),(4,4)}. Thereafter, it checks again whether a point should be added after the first point. This is again the case and the next intermediate result is {(0,0),(0,2),(2,2),(4,4)}. The point (0,2) is the point, outside the domain, which is closest to the straight line connecting (0,0) and (2,2). This is known because the data is given in clockwise order. Now, the routine checks again whether a point should be inserted after the first point. Since, this is not needed and the program proceeds to the second point etc.. Finally the result will be {(0,0),(0,2),(2,2),(2,4),(4,4)}. This result is drawn in Figure 5.2.a.

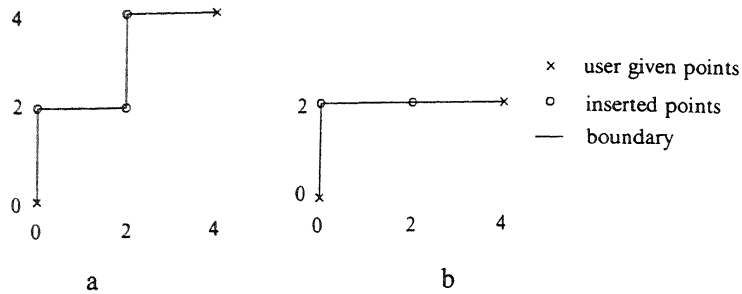


FIGURE 5.2. Insertion of points.

Next consider the input {(0,0),(4,2)}. Then the routine generates successively {(0,0),(2,2),(4,2)} and {(0,0),(0,2),(2,2),(4,2)}. This result is drawn in Figure 5.2.b. □

Below, an example is given of a domain definition.

EXAMPLE 5.2. Consider the domain in Figure 5.3, where at the left and right boundary the velocity and the elevation are respectively prescribed.

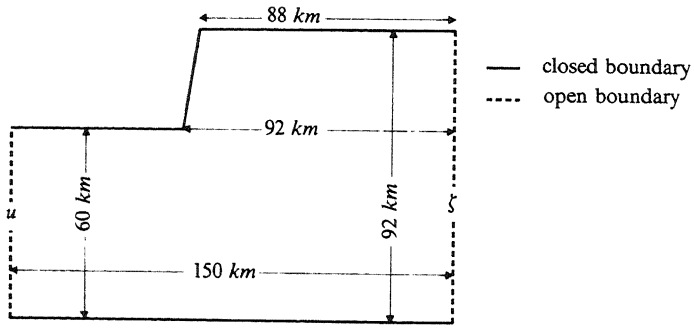


FIGURE 5.3. Example domain.

First, the user should determine the mesh width used in the numerical simulation (say 5 *km*). This defines the grid to be used. In Figure 5.4 the domain of Figure 5.3 is covered by the grid in which the grid lines are already numbered. The user should be aware of the fact that closed boundaries can only be represented by a grid line with an even X_i or Y_i-grid coordinate. Furthermore, a Z-boundary can only be represented by a grid line with an odd X_i or Y_i-coordinate.

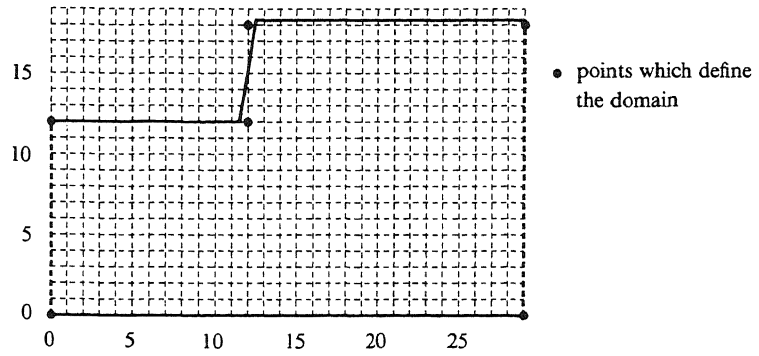


FIGURE 5.4. Example domain of Figure 5.3 covered by a 5 *km* grid.

In Figure 5.5, the ordered set of angle points defining the numerical domain according to Figure 5.4 is given. Furthermore, a picture of the domain is drawn as generated by the program.

```
0  0  0
0  12 1
12 12 1
12 18 1
29 18 0
29 0  1
```

```
                0 0 0 0 0 Z
                0         Z
0 0 0 0 0       0         Z
U                         Z
U                         Z
U                         Z
0 0 0 0 0 0 0 0 0 0       0
```

FIGURE 5.5. Input example plus resulting domain. □

Islands. After the domain is defined, the user can specify islands. Evidently, the boundaries of an island are closed. Hence, an island is specified by an ordered set of angle points which have always even values, $\{(X_i, Y_i) \in \mathbb{N} \times \mathbb{N} \mid i = 1, \ldots, n,\ X_i \text{ even},\ Y_i \text{ even}\}$. The parameter B_i, needed to define the type of the boundary, is not requested here by the program.

EXAMPLE 5.3. In Figure 5.6 an ordered set of angle points of an island is given together with the resulting domain when this island is placed in the domain of Example 5.2.

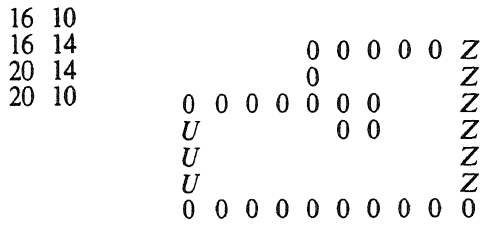


FIGURE 5.6. Domain of Example 5.2 with island. □

After the definition of the domain, there is a possibility to check the given data and to correct them if necessary. After this is completed a simple plot of the domain, such as given in Figures 5.5 and 5.6, will be given. Thereafter, there is again a possibility to change the data.

When the domain has been defined the program can generate the so-called row-column table. This table which is used in SOLVER specifies on the staggered grid the start and end point of each row and column in the computational domain. This table is used to construct the index arrays of the boundaries. For the construction of this table the sorting routine M01AQF from the NAG library is used.

5.2.2. Boundary conditions. After the domain has been specified, the program detects the points at which boundary condition data have to be prescribed in order to make the problem well-posed. These points will be called prescription points and are usually the corners of the polygon at which a transition from one boundary type to another takes place, i.e. from open to closed or conversely. The values at the boundary between two such points are obtained by interpolation.

The user should specify whether the problem is stationary or time dependent. Thereafter, the program lists the points at which the boundary condition data have to be prescribed. These points also include the points which are given by the user during the domain definition by setting $B_i = 100$.

In the stationary case, the user should give the values of each prescription point separately. If the problem has time-dependent boundary conditions then it is assumed that the boundary condition data, i.e. the values of a U, V or Z prescription point in time, can be constructed from a series of sine functions,

$$\sum_{i=1}^{n} A_i \sin(W_i t + F_i). \tag{5.1}$$

The user should specify the number of terms. Furthermore, at each point the user should specify A_i, W_i and F_i. After this specification the user is offered the opportunity to change the given values.

5.2.3. Initialization of the U, V and *Z-field.* After the specification of the boundary condition data, the values are calculated at each boundary part and the program can proceed with the initialization of U, V and Z. With respect to the initialization of the U and V field, the program checks whether the boundary values are zero. If this is the case, then the the initial field is, on request of the user, set to zero everywhere. Otherwise, the user can specify points, together with values at these points, from which the program interpolates values at all other points in the field by using cubic B-splines. Here we used the NAG library routines E02ZAF, E02DAF and E02DBF. First the program will ask for values at special points needed for a correct interpolation. Thereafter, the user may specify additional points and values.
After the interpolation the field is shown to the user at the points of the staggered grid. If the user is not satisfied with the result of the interpolation then he can correct or add data.
For the Z field the situation is the same, except that the initial field can be a constant unequal to zero when the initial boundary values are constant.

5.2.4. Definition of the depth and Manning values. The definition of the depth and Manning values proceeds in the same way as the initialization of the Z-field. The Manning values are needed to calculate the Chezy values using the formula (see [3])

$$C = H^{1/6}/n, \tag{5.2}$$

where n is the space varying Manning field. Again, the user can specify the field to be constant or to be space dependent. Additionally, there is a default value for the Manning coefficient, which is equal to 0.022.

5.2.5. Definition of problem and integration parameters. The problem and integration parameters which have to be specified are:

the mesh size (on the unstaggered grid),
the time step,
the number of time steps,
the viscosity coefficient A (see (2.1)),
the coefficient γ for the weakly-reflective boundary conditions (see (2.5))
a parameter which specifies whether second-order or fourth-order finite differences should be used.

When asking for the time step the program suggests a realistic value.

5.2.6. Definition of time history points and flow-field output parameters. In this part the user should specify the number of history points and their position. Furthermore, the user can specify the start time and the time period defining the times at which the flow field should be written to the file *FLOW* during the simulation. The start time and the time period have to be multiples of the time step.

5.3. The SOLVER

The SOLVER, running at the CYBER 205, consists of three main parts: the part which reads the input given on *SOLINP*, the part which performs the actual computation and the part which writes the output to *TIHIS* and *FLOW*. The SOLVER is activated by submitting the file *SOLINP* to the CYBER 205.

The computation part performs the user-specified number of time steps (see Section 5.2.5). Before each time step the drying and flooding conditions are checked (see Section 3.8 and 4.4). The time step requires four right-hand side evaluations (see Section 3.5). After each right-hand side evaluation the stabilization described in Section 3.6 is performed.

Apart from the computation, at each time step the solution at the time history points is written to the file *TIHIS* (see Section 5.1) and flow fields are written to the file *FLOW* at the user-specified times.

Finally, we remark that the sorting routine M01AQF from the NAG library has been used for initializing the index arrays.

5.4. The OUTPUT PROCESSOR

The OUTPUT PROCESSOR runs at the front-end of the CYBER 205. It generates plots of the time histories given on *TIHIS* and of the flow fields given on *FLOW*. For the time histories, the user can choose the type of the plot; plots of the following entities can be drawn:

the *U*-velocity,
the *V*-velocity,
the elevation,
the magnitude of the velocity,
the direction of the velocity.

The flow field is represented by means of vectors positioned at elevation points. The length and the direction of such a vector represent the magnitude and the direction of the flow, respectively. The length of the vectors can be scaled by the user.

6. NUMERICAL RESULTS

In this section, results obtained by the described solver will be given. First we present results from flow computations in a bay near Taranto in Italy. To define this problem the system described in Section 5 is used. Thereafter, we give results for a stationary flow in the Anna Friso Polder and for a time-dependent flow in the Eems-Dollard estuary. For the last two experiments the solver is incorporated into the WAQUA system, a large computational system used for the simulation of water flow and water quality at Rijkswaterstaat and Delft Hydraulics [32]. Incorporation into this system gives the possibility to test the model on real engineering problems. Furthermore, it provides a wide variety of plot facilities.

6.1. A time-dependent flow in the Taranto bay

In this section, we present results from a computation of the time-dependent flow in a bay near Taranto (Mare Piccolo), which is situated in the south of Italy; a map of this bay is drawn in Figure 6.1 (for a more detailed figure see [30]). The schematization of the bay is adopted from Notarnicola and Pontrelli [28]. Currently, there is no data available from the bottom profile of the bay. Therefore, a constant value is assumed, viz. 7 meters, which approximates the mean depth.

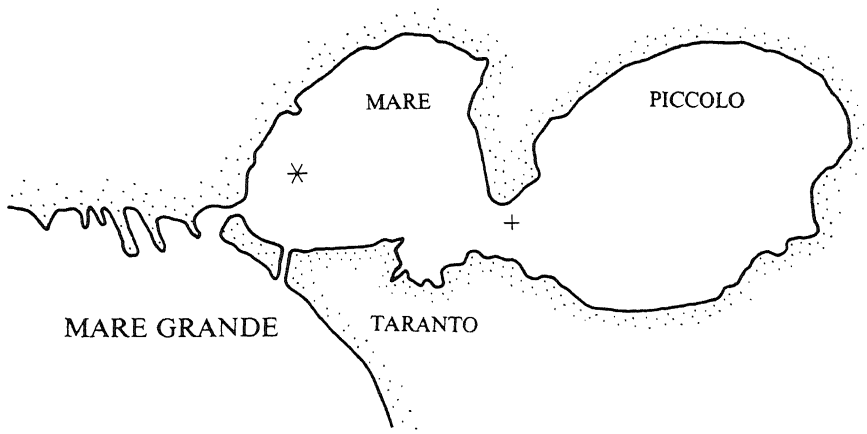
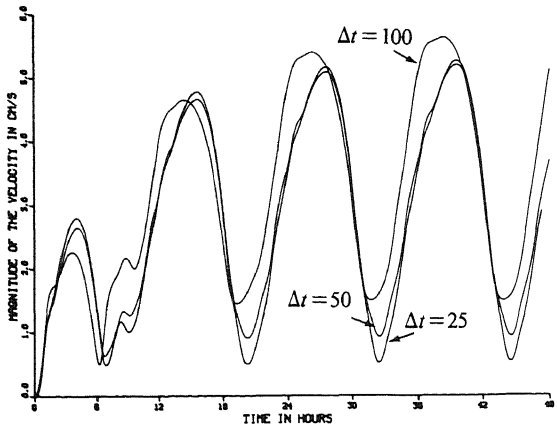
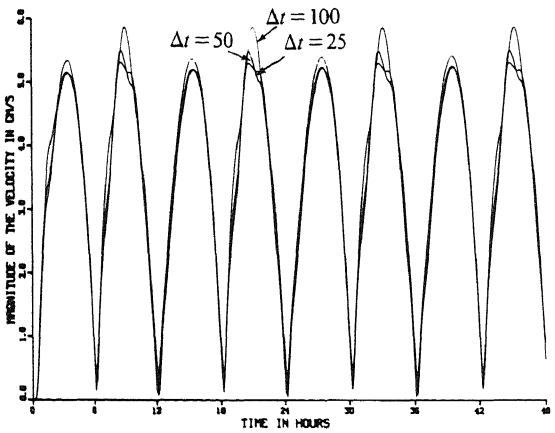


FIGURE 6.1. The Taranto bay.

The boundaries are closed, except for a small open part; here we prescribe the elevation, $\zeta(t)=.2\cos(2\pi t/(3600\times 12))$. Furthermore, we set the viscosity coefficient A equal to 5 m^2/s and the value of c equals .8 for this problem (see Section 3.4.3). In the numerical model, the fourth-order space discretizations is used with a mesh size of 111 meters (on the unstaggered grid). The flow is simulated over the (real time) period of 48 hours, i.e. over 4 full periods of the tide. The initial field of the velocity is zero and of the elevation .2 *m*. The flow is computed for three values of the time step, viz. $\Delta t = 100$, 50 and 25 seconds. In Figure 6.2, time histories of the magnitude of the velocity are drawn at the point indicated by an * and a + in Figure 6.1.



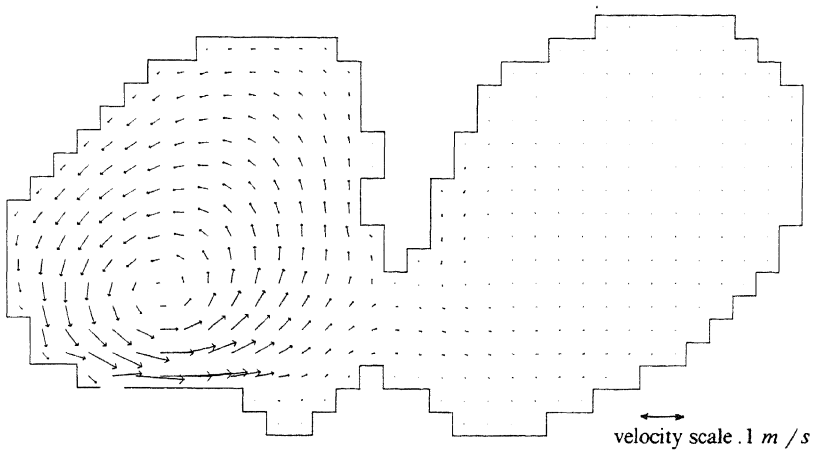
a. Time history at *.



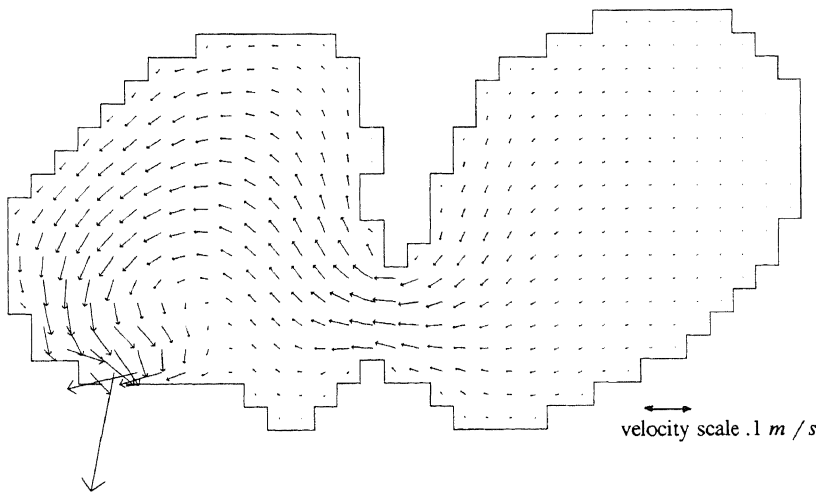
b. Time history at +.

FIGURE 6.2. Time histories of the magnitude of the velocities.

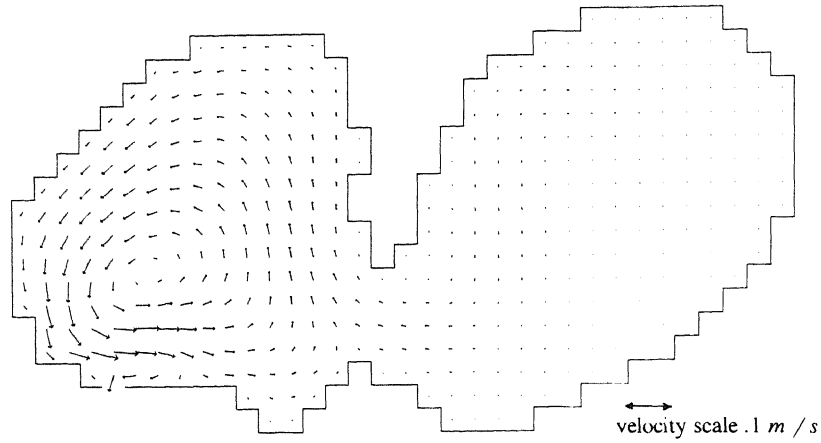
They show that the solution becomes periodic after a few tides. Moreover, we observe that the solution depends on the time step, showing the need for small time steps in this type of applications. It is interesting to see that the time step has a much larger influence on the solution at the point * than at the point +. This can be explained by considering the flow fields. In Figure 6.3, these are given at times 36, 39, 42 and 45 hours.



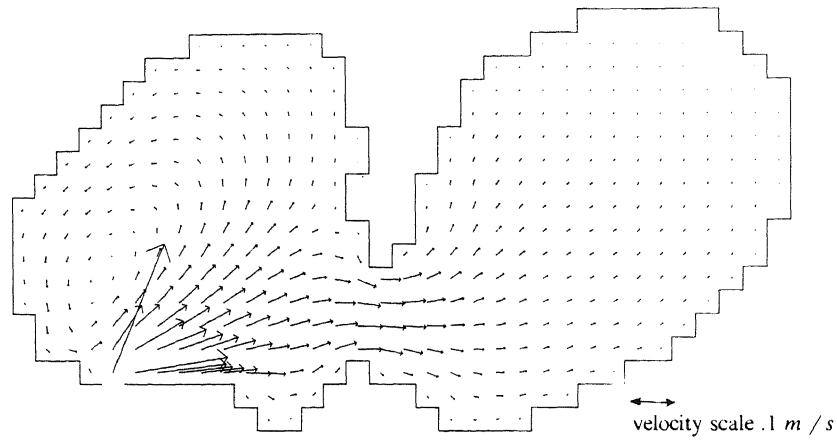
a. Flow field at 36 hours.



b. Flow field at 39 hours.



c. Flow field at 42 hours.



d. Flow field at 45 hours.

FIGURE 6.3. Flow fields.

Due to the periodicity of the solution, the flow field at 48 hours is equal to the flow field at 36 hours. We observe that the tide gives rise to a recirculating flow in that part of the bay where the open boundary is located. It is known (see [8] and Section 6.2) that, for stationary problems, the recirculating flow is determined by delicate balances. As we expect a similar behaviour in the nonstationary case, it is not surprising that the influence of the time integration error is much larger in the point indicated by *, since this point is in the recirculation zone.

6.2. A stationary flow in the Anna Friso Polder

In order to test the spatial discretization we shall consider in this section numerical solution of stationary flows in the Anna Friso Polder (AFP). Solutions will be given for various values of the viscosity parameter A. The AFP is a small recess at the southern coast of the south-west entrance of the Eastern-Scheldt estuary, the so-called Roompot (see Figure 6.4).

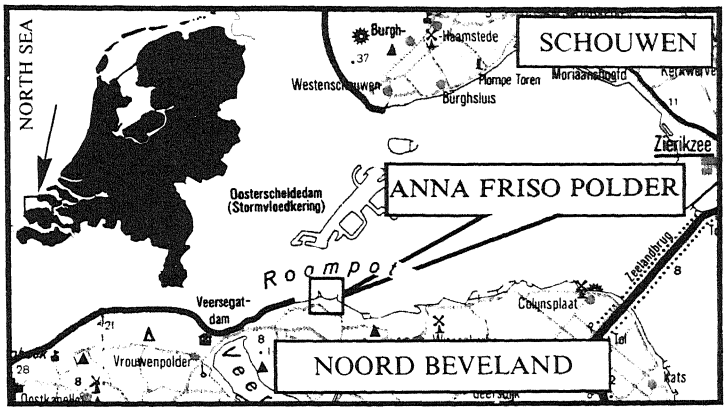


FIGURE 6.4 Location of the Roompot.

The area modelled is about $2.5 \times 1.5\ km^2$ with a complex shore line and a pronounced bottom profile. A typical cross-section normal to the coast of AFP shows a rather shallow area with a near shore depth well below 10 m, a steep slope region with slopes up to 1:5, followed by a rather flat main channel with depth up to 35 m. The boundary conditions are taken from a steady-state maximum flood situation which was simulated by a hydraulic scale model at the Delft Hydraulics. We prescribe at the left and upper boundary of the mathematical model the normal velocity component and at the right boundary the water level (see Figure 6.5). Furthermore, the mesh width Δx of the unstaggered grid is $22.5m$. This model is extensively discussed in [8]. It is of interest for the study of steady recirculating flow. A plot of the computational domain is drawn in Figure 6.5.

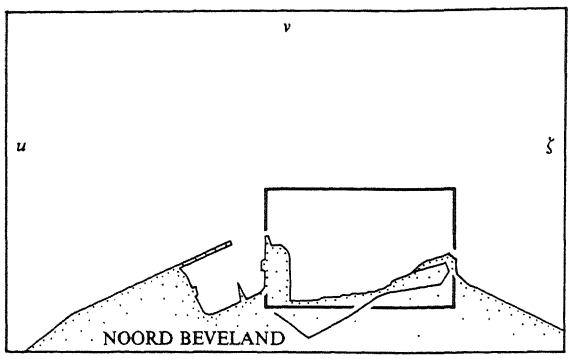


FIGURE 6.5. Computational domain.

In the computations, the time step is 7.5 seconds, the constant c is .24 (see Section 3.4.3) and the constant γ in (2.4) and (2.5) is set equal to 50. The time step used is four times larger than the maximum time step without smoothing (see Sections 3.5 and 3.6). It is assumed that the steady-state is reached if the amplitude of the elevation has a magnitude less than 1 *mm*. This requires about 6 hours of simulation. As we are only interested in the recirculating flow, we will give plots of the indicated area only. In Figure 6.6, vector plots are given for $A = 10, 1, .1, 0\ m^2/s$, respectively.

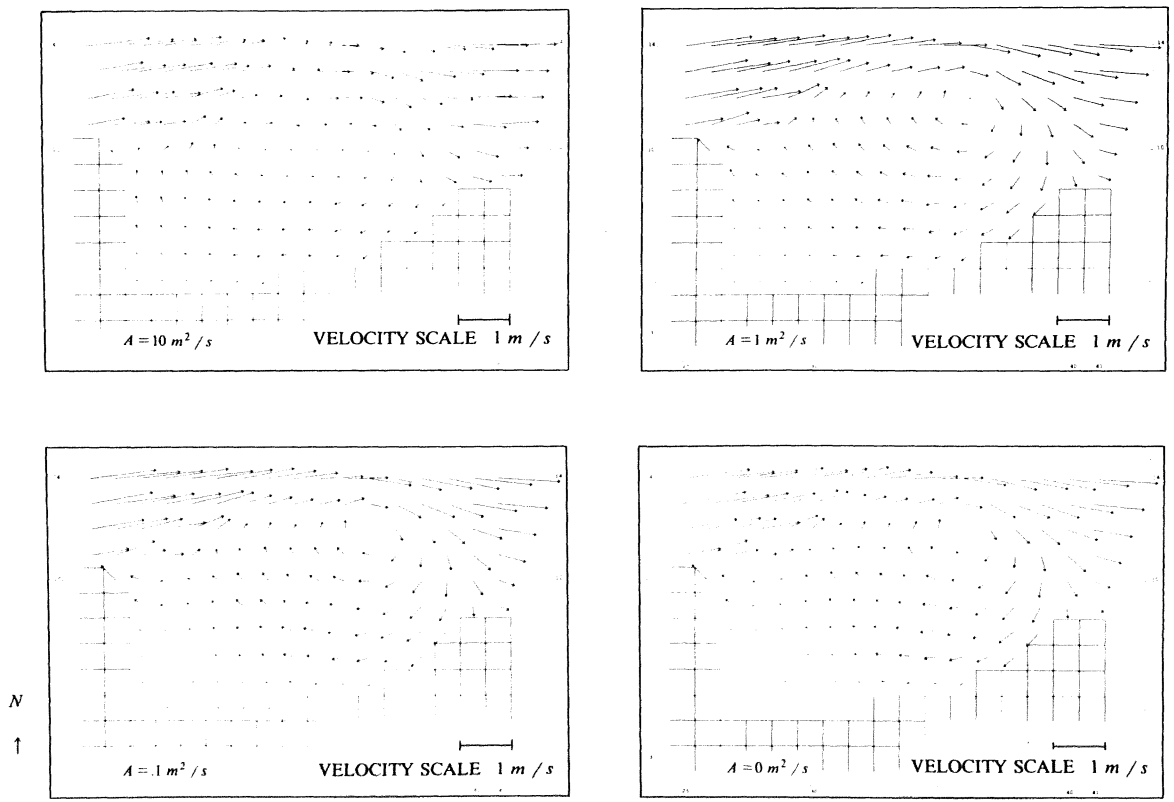


FIGURE 6.6. Vector plots for $A = 10, 1, .1, 0\ m^2 / s$.

These plots show a significant change of the flow when A is decreased from 10 to 1, but a further decrease of A hardly effects the flow pattern. This is even more clear when we consider vertical cross-sections of the magnitude of the velocities at $M = 28, 33, 36$ (see Figure 6.7). The variables M and N are the cell indices for the horizontal and vertical axis, respectively, as used in the plots.

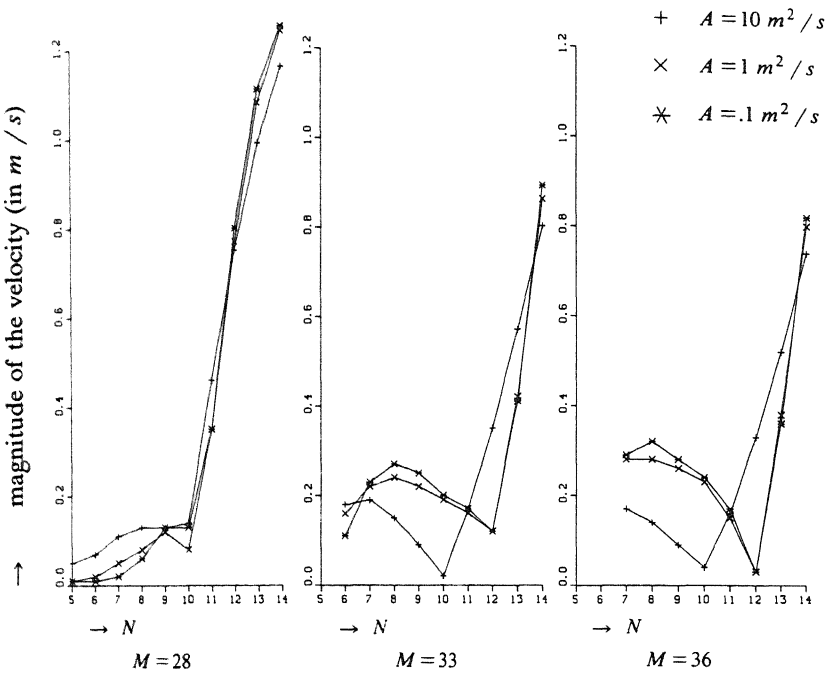


FIGURE 6.7. Cross-sections of the magnitude of the velocity at $M = 28, 33, 36$.

Flokstra et al. [8] explain these results qualitatively by argueing that for $A = 10\ m^2 / s$ the dissipation of momentum due to turbulent viscosity is more important for the flow pattern than the dissipation due to bottom friction. In the cases, $A = 1$ and $.1\ m^2 / s$ bottom friction determines largely the flow pattern. Therefore, the pattern does hardly change when the eddy viscosity is decreased from 1 to $.1\ m^2 / s$. In [8], additional computations are reported for the same model, however, (i) with perturbed bottom friction and (ii) with a perturbed bottom profile.

The results given in this section are compared with those reported in [8] obtained by the ADI method designed by Stelling. It appeared that the above plots are almost indistinguishable for the region of interest. Small differences occur near boundaries. This can be traced back to a difference in the

discretization of vu_y and uv_x and the viscosity terms near boundaries (see Section 3.4.2).

6.3. A time-dependent problem in the Eems-Dollard estuary

In many engineering problems, flows have to be calculated in estuaries in which drying and flooding occurs during the tide. The Eems-Dollard estuary is an example of such a problem. Hence, this model provides a good case to test our drying and flooding procedure. Details on this model can be found in [26].

The Eems-Dollard estuary is situated in the north of the Netherlands. In Figure 6.8, the computational domain is drawn together with the used grid.

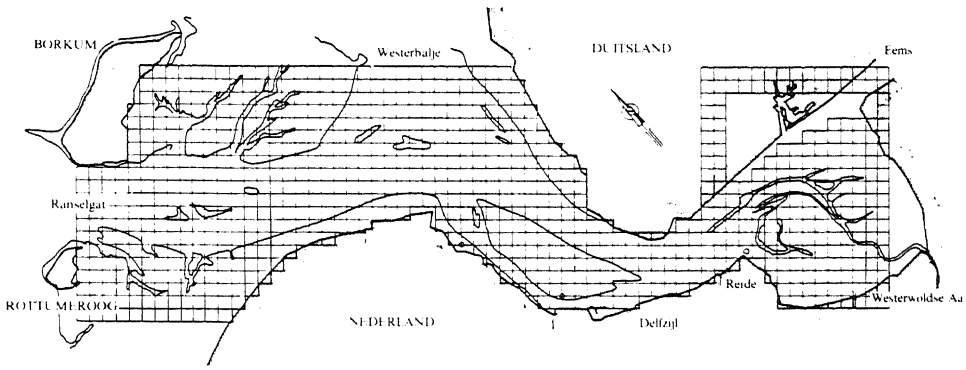


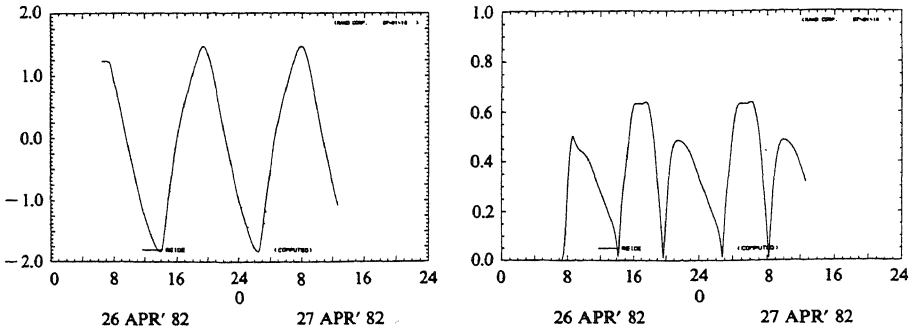
FIGURE 6.8. The Eems-Dollard estuary.

Closed boundaries are modelled from the coast of Groningen to Rottumeroog and from Borkum to Westerbalje. Water level boundaries are modelled at the Ranselgat, i.e. the opening between Rottumeroog and Borkum, and from Westerbalje to the coast of Germany. The inflows from the rivers Eems and Westerwoldse Aa as well as industrial discharges at Delfzijl are modelled as sources. The mesh size of this grid is 800 m, whereas the mesh size of the unstaggered grid is 400 m. The second-order space discretization is applied. The time step in this simulation is 150 seconds, the eddy viscosity $A = 60\ m^2/s$, and $c = .24$. The boundary conditions are derived from a Fourier analysis of measurements. They are adapted such that the tide is purely periodic with period 12 hours and 30 minutes (a motivation for this approach is given in [26]). In this computation γ (the coefficient in the weakly reflective boundary conditions) is zero.

With respect to drying and flooding, the minimal allowed water depth at a velocity point is 9.25 cm.

At the start of the simulation the elevation is set equal to 1.23 m the velocity

to zero.
We first present time histories associated with the elevation and the magnitude of the velocity at Reide (see Figure 6.8 for this location).



a. Water level (m). b. Magnitude of the velocity (m / s).

FIGURE 6.9. Time histories at Reide.

These plots show that the periodic behaviour of this flow is reached very soon after the start of the simulation (within one period of the tide). Furthermore, we give in Figure 6.10 a vector plot of the flow field at low tide (27-th April, 1 hrs. 13 min.). The closed boundaries resulting from the drying and flooding procedure are drawn as dashed lines.

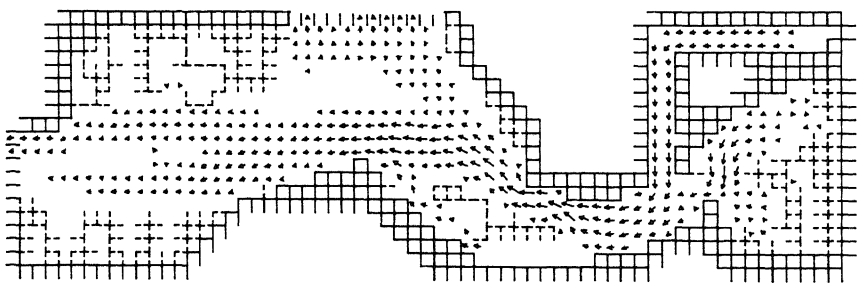


FIGURE 6.10. Flow field at low tide.

This plot shows that significant tidal flats occur during the tide. From both plots (Figures 6.9 and 6.10), we conclude that the drying and flooding

procedure as used in our method does not give rise to instabilities or unwanted phenomena in the solution.

REFERENCES

1. CONTROL DATA CORPORATION (1986). *FORTRAN 200 VERSION 1; Reference manual*, Publications and Graphics Division, California.
2. G. DAHLQUIST (1959). *Stability and Error Bounds in the Numerical Integration of Ordinary Differential Equations,* no. 130, Trans. Roy. Inst. Techn..
3. J.J. DRONKERS (1964). *Tidal Computations*, North-Holland Publishing Company, Amsterdam.
4. T. ELVIUS and A. SUNDSTRÖM (1973). Computational Efficient Schemes and Boundary Conditions for a Fine-Mesh Barotropic Model Based on the Shallow Water Equations, *Tellus*, 25, pp. 132-156.
5. B. ENGQUIST and A. MAJDA (1977). Absorbing Boundary Conditions for the Numerical Simulation of Waves, *Math. Comp.*, 31, pp. 629-651.
6. B. ENGQUIST and A. MAJDA (1979). Radiation Boundary Conditions for Acoustic and Elastic Wave Calculations, *Comm. Pure Appl. Math.*, 32, pp. 313-357.
7. G. FISCHER (1956). Ein numerisches Verfahren zur Errechnung von Windstau und Gezeiten in Randmeeren (German), *Tellus*, 11, pp. 289-300.
8. C. FLOKSTRA, G.K. VERBOOM, and A.K. WIERSMA (1986). *Computation of Steady Recirculating Flow,* Report R1150-II, Delft Hydraulics, Delft.
9. R. FRANK, J. SCHNEID, and C.W. UEBERHUBER (1981). The Concept of B-Convergence, *SIAM J. Numer. Anal.*, 18, pp. 753-780.
10. H. GERRITSEN (1982). *Accurate Boundary Treatment in Shallow-Water Flow Computations,* Thesis, TU Twente.
11. E.D. DE GOEDE (1986). *Stabilization of the Lax-Wendroff Method and a Generalized One-Step Runge-Kutta Method for Hyperbolic Initial Value Problems,* Report NM-R8613, to appear in Appl. Numer. Math., CWI, Amsterdam.
12. E.D. DE GOEDE and F.W. WUBS (1987). *Explicit-Implicit Methods for Time-Dependent Partial Differential Equations,* Report NM-R8703, CWI, Amsterdam.
13. B. GUSTAFSSON (1975). The Convergence Rate for Difference Approximations to Mixed Initial Boundary Value Problems, *Math. Comp.*, 29, pp. 396-406.
14. W. HANSEN (1956). Theorie zur Errechnung des Wasserstandes und der Strömungen in Randmeeren nebst Anwendungen (German), *Tellus*, 8, pp. 289-300.
15. G.W. HEDSTROM (1976). Nonreflecting Boundary Conditions for Nonlinear Hyperbolic Systems, *J. Comput. Phys.*, 30, pp. 333-339.
16. P.J. VAN DER HOUWEN (1987). Stabilization of Explicit Difference Schemes by Smoothing Techniques, to appear in *Proceedings of the 4th International Seminarium on Numerical Analysis of Ordinary Differential Equations*, Halle.
17. P.J. VAN DER HOUWEN, C. BOON, and F.W. WUBS (1987). *Analysis of*

Smoothing Matrices for the Preconditioning of Elliptic Difference Equations, Report NM-R8705, to appear in Z. Angew. Math. Mech..

18. P.J. VAN DER HOUWEN, B.P. SOMMEIJER, J.G. VERWER, and F.W. WUBS (1986.). Numerical Analysis of The Shallow-Water Equations, in *Mathematics and Computer Science: Proceedings of the CWI symposium, November 1983, CWI-Monographs no.1*, ed. J.W. de Bakker, M. Hazewinkel and J.K. Lenstra, North-Holland, Amsterdam.
19. P.J. VAN DER HOUWEN, B.P. SOMMEIJER, and F.W. WUBS (1986). *Analysis of Smoothing Operators in the Solution of Partial Differential Equations by Explicit Difference Schemes,* Report NM-R8617, CWI, Amsterdam.
20. P.J. VAN DER HOUWEN and F.W. WUBS (1987). The Method of Lines and Exponential Fitting, *Internat. J. Numer. Methods Engrg.*, 24, pp. 557-567.
21. A. JAMESON (1983). The Evolution of Computational Methods in Aerodynamics, *J. Appl. Mech.*, 50, pp. 1052-1076.
22. J. KUIPERS and C.B. VREUGDENHIL (1973). *Berekeningen van Twee-Dimensionale Horizontale Stromingen (Dutch),* Report-S163, Delft Hydraulics, Delft.
23. P.D. LAX (1954). Weak Solutions of Non-Linear Hyperbolic Equations and their Numerical Computation, *Comm. Pure Appl. Math.*, 7, pp. 159-193.
24. J.J. LEENDERTSE (1967). *Aspects of a Computational Model for Long-Period Water-Wave Propagation,* Memorandum RM-5294-PR, Rand Corporation, Santa Monica.
25. A. LERAT (1979). Une Classe de Schémas aux Différences Implicites pour les Systémes Hyperboliques de Lois de Conservation (French), *C.R. Acad. Sci. Paris t. 288 (18 juin 1979) Série A*, pp. 1033-1036.
26. K.D. MAIWALD, L. POSTMA, and A.K. WIERSMA (1984). *WAQUA/DELWAQ Berekeningen Eems-Dollard Estuarium (Dutch),* S296.02, Delft Hydraulics, Delft.
27. J. MOOIMAN (1987). *Implementatie van Zwak-Reflecterende Randvoorwaarden in DELFLO (Dutch),* Report Z117, Delft Hydraulics, Delft.
28. F. NOTARNICOLA and G. PONTRELLI (1987). *Un Modello Idrodinamico per Acque Basse con Termini Sorgenti e sue Integrazione Numerica (Italian),* Internal Report/1, Institute for Research of Applied Mathematics -CNR-, Bari.
29. J. OLIGER and A. SUNDSTRÖM (1978). Theoretical and Practical Aspects of some Initial Boundary Value Problems in Fluid Dynamics, *SIAM J. Appl. Math.*, 35, pp. 419-446.
30. P. PARENZAN (1984). *Il Mar Piccolo di Taranto (Italian),* C.C.I.A.A., Taranto.
31. N. PRAAGMAN (1979). *Numerical Solution of the Shallow Water Equations by a Finite Element Method,* Thesis, TU Delft, Delft.
32. M.A.M. RAS and G.S. STELLING (1984). *WAQUA, een Simulatie pakket voor Twee-Dimensionale Waterbeweging en Waterkwaliteit,* DIVISIE 1984-4, Rijkswaterstaat, Rijswijk.
33. R.D. RICHTMYER and K.W. MORTON (1967). *Difference Methods for Initial*

Value Problems, Interscience Publishers, Wiley, New York, London.

34. W. SCHÖNAUER and W. GENTZSCH (EDS.) (1985). *The Efficient Use of Vector Computers with Emphasis on Computational Fluid Dynamics*, Notes on Numerical Fluid Mechanics, 12, Friedr. Vieweg & Sohn, Braunschweig/Wiesbaden.
35. A. SEGAL and N. PRAAGMAN (1986). A Fast Implementation of Explicit Time Stepping Algorithms with the Finite Element method for a Class of Non-Linear Evolution Problems, *Internat. J. Numer. Methods Engrg.*, 23, 155-168.
36. F. SHUMAN (1957). Numerical Methods in Weather Prediction: II, Smoothing and Filtering, *Monthly Weather Review*, 85, pp. 357-361.
37. A. SIELECKI (1968). An Energy Conserving Difference Scheme for Storm Surge Equations, *Monthly Weather Review*, 96, pp. 150-156.
38. G.S. STELLING (1983). *On the Construction of Computational Methods for Shallow Water Flow Problems*, Thesis, TU Delft, Delft.
39. G.S. STELLING, A.K. WIERSMA, and J.B.T.M. WILLEMSE (1986). Practical Aspects of Accurate Tidal Computations, *J. Hydr. Engrg., ASCE*, 112, pp. 802-817.
40. G.S. STELLING and J.B.T.M. WILLEMSE (1984). Remarks about a Computational Method for the Shallow Water Equations that works in Practice, in *Colloquium Topics in Applied Numerical Analysis*, pp. 337-362, ed. J.G. Verwer, CWI, Amsterdam.
41. G.S. STELLING, J.B.T.M. WILLEMSE, and A. ROOZENDAAL (1986). A Computational Model for Shallow Water Flow Problems on the Cyber 205, *Supercomputer*, 11.
42. J.C. STRIKWERDA (1976). *Initial Boundary Value Problems for Incompletely Parabolic Systems*, Thesis, Stanford University, Stanford.
43. L.N. TREFETHEN (1982). *Wave Propagation and Stability for Finite Difference Schemes*, Thesis, Stanford University, Stanford.
44. E. TURKEL (1985). Acceleration to a Steady State for the Euler Equations, in *Numerical Methods for the Euler Equations of Fluid Dynamics*, pp. 281-311, SIAM, Philadelphia, PA.
45. G.K. VERBOOM and A. SLOB (1984). Weakly-Reflective Boundary Conditions for Two-Dimensional Shallow Water Flow Problems, *Adv. Water Resources*, 7, pp. 192-197.
46. G.K. VERBOOM, G.S. STELLING, and M.J. OFFICIER (1982). Boundary Conditions for the Shallow Water Equations, in *Engineering Applications for Computational Hydraulics*, ed. M.B. Abbott and J.A. Cunge, Pitman Publishing.
47. J.H.A. WIJBENGA (1985). Determination of Flow Patterns in Rivers with Curvilinear Coordinates, in *Proceedings of the XXI Congress of the International Association for Hydraulic Research*, Melbourne.
48. J.B.T.M. WILLEMSE, G.S. STELLING, and G.K. VERBOOM (1985). Solving the Shallow Water Equations with an Orthogonal Coordinate Transformation, in *Proceedings of the International Symposium on Computational Fluid Dynamics*, Tokyo.

49. F.W. WUBS (1986). Stabilization of Explicit Methods for Hyperbolic Partial Differential Equations, *Internat. J. Numer. Methods Fluids*, 6, pp. 641-657.
50. F.W. WUBS (1987). An Explicit Shallow-Water Equations Solver for Use on the CYBER 205, in *Algorithms and Applications on Vector- and Parallel Computers*, ed. H.J.J. te Riele, Th. J. Dekker and H.A. van der Vorst, North-Holland, Amsterdam.