NUMERICAL SOLUTION OF THE SHALLOW-WATER EQUATIONS

F.W. WUBS

STELLINGEN
behorende bij het proefschrift

Numerical Solution of the Shallow-Water Equations

van F.W. Wubs

21 oktober 1987

I

Beschouw een numerieke methode van de orde $p+1$ (methode I) voor het beginwaardeprobleem.

$$\frac{d}{dt}y(t)=f(y),\ y(t_0)=y_0,\ t>t_0$$

dan wordt een numerieke methode van orde p (methode II) als volgt gedefinieerd. Het beginwaardewaardeprobleem wordt opgelost met methode I (met oplossing $y_n \approx y(t_n)$), maar aan de gebruiker wordt de oplossing $u_n = y_n + e(y_n,t_n)h^p$ gepresenteerd. De fout in de oplossing u_n wordt geschat door $\epsilon_n = e(y_n,t_n)h^p$.

Nu gelden de volgende stellingen.

1.a. Methode II geeft met

$$e(y_n,t_n) \equiv \left[B_p \frac{d^p y(t_n)}{dt^p} \right]$$

asymptotisch dezelfde oplossing en foutschatting als de methode van Stetter [1] (methode III). Voor de notatie zie [2].

1.b. Zowel voor methode II als III is de fout ten tijde $t=t_0$ gelijk aan nul als de volgende modificaties wordt aangebracht:
in methode II

$$e(y_n,t_n) \equiv \left[B_p \frac{d^p y(t_n)}{dt^p} \right] - \alpha / h^p,$$

en in methode III

$$k_i = hf(y_{n-1} + \sum_{j=1}^{\mu} a_{ij}k_j + \alpha), \qquad i = 1, \ldots, \mu,$$

$$y_n = y_{n-1} + \sum_{i=1}^{\mu} b_i k_i, \qquad n = 1,2, \ldots,$$

$$\epsilon_n = \sum_{i=1}^{\mu} r_i k_i - \alpha,$$

waarin $\alpha = h^p[B_p(d^p / dt^p)y(t_0)]$.

[1] H.J. Stetter (1971), Local Estimation of the Global Discretization Error, *SIAM J. Numer Anal*, 8, pp. 512-523.
[2] P. Merluzzi & C. Brosilow (1978). Runge-Kutta Integration Algorithms with Built-in Estimates of the Accumulated Truncation Error, *Computing*, 20, pp. 1-16.

II

In tegenstelling tot de methoden in de vorige stelling waar de schatting van de globale fout verkregen werd door een lineaire combinatie van rechterlidevaluaties, kan de opbouw van de fout ook meegeïntegreerd worden. Een voorbeeld van zo'n methode wordt gegeven door de tweede-orde integratieformule.

$$k_1 = f(y_n + \tfrac{3}{2}\epsilon_n)$$

$$k_2 = f(y_n - \tfrac{1}{2}\epsilon_n + \tfrac{2}{3}hk_1)$$

$$k_3 = f(y_n - \tfrac{3}{2}\epsilon_n + \tfrac{5}{12}hk_1 + \tfrac{1}{4}hk_2)$$

$$y_{n+1} = y_n + \tfrac{1}{4}hk_1 + \tfrac{3}{4}hk_2$$

$$\epsilon_{n+1} = \epsilon_n + hk_1 - hk_2$$

met $y_0 = y(0)$ en $\epsilon_0 = 0$.

III

Wanneer een convergente numerieke oplossingswaarde als functie van de verfijningsparameter een convex gedrag heeft, dan kunnen er gegarandeerde onder- en bovengrenzen voor de exacte oplossing gegeven worden.

IV

Zij een numerieke methode gegeven met een stabiliteitspolynoom van de gedaante

$$1 + P_k(x^2)(1 + \tfrac{1}{2}x)x,$$

waarin P_k een polynoom is van de graad k. Voorts geldt dat de methode

voorwaardelijk stabiel is voor beginwaardeproblemen waarvan de Jacobiaan een normale matrix is en bovendien zuiver imaginaire eigenwaarden heeft.

a. Er kan nu een expliciete splitmethode geconstrueerd worden met "stabiliteitspolynoom"

$$1+P_k(x_1^2)P_k(x_2^2)(1+\tfrac{1}{2}x_1)(1+\tfrac{1}{2}x_2)(x_1+x_2),$$

die een tweemaal zo grote effectieve imaginaire stabiliteitsgrens heeft als de oorspronkelijke methode voor problemen met: (i) zuiver imaginaire eigenwaarden (ii) Jacobianen die in tweeën gesplitst kunnen worden zodanig dat de beide delen precies voor de helft bijdragen aan de spectrale radius van de ongesplitste Jacobiaan. (Hierbij is aangenomen dat de evaluatie van het rechterlid van het ongesplitste probleem even duur is als de som van de evaluaties van het gesplitste probleem.)

b. Zij $P_1(x^2)=1+x^2/4$ dan is voor problemen met Jacobianen als onder a. beschreven de effectieve imaginaire stabiliteitsgrens 1.

V

Voor berekeningen aan 2-D stromingsproblemen op de CYBER 205, waarbij een rechthoekig rooster wordt gebruikt, kan zonder herordening van de rekenarrays tijdens de tijdstap, evenals op rechthoekige gebieden ook op "grillige" gebieden, die in één richting een constante doorsnede hebben, geheugenzuinig en efficient gerekend worden.

[3] F.W. Wubs (1987). An Explicit Shallow-Water Equations Solver for Use on the CYBER 205, in *Algoritms and Applications on Vector- and Parallel Computers*, eds. H.J.J. te Riele, Th.J. Dekker & H.A. van der Vorst, North-Holland, Amsterdam.

VI

Bij potentiaalstromingen om vleugelprofielen kan het aantal roosterpunten tot de helft worden teruggebracht door gebruik te maken van asymptotische benaderingen van de oplossing ver van de vleugel.

[4] F.W. Wubs, J.W. Boerstoel & A.J. van der Wees (1984). Grid-Size Reduction in Flow Calculations on Infinite Domains by Higher-Order Far-Field Asymptotics in Numerical Boundary Conditions, *Journal of Engineering Mathematics*, 18, pp. 157-177.

VII

De door Stelling ontwikkelde methode voor de numerieke integratie van de ondiepwatervergelijkingen genereert in het geval van een stationaire stroming een numerieke oplossing die afhangt van de gekozen tijdstap.

[5] G.S. Stelling (1983). *On the Construction of Computational Methods for Shallow Water Flow Problems*, Thesis, TU Delft, Delft.

VIII

Laat $P_{2^q-1}(z)$ een polynoom zijn van de graad 2^q-1 gegeven door

$$P_{2^q-1}(z)=\prod_{j=0}^{q-1}[1-\frac{\beta_j}{1+\beta_j}\frac{1-T_{2^j}(1+2z)}{2}]$$

met $\beta_{j+1}=\beta_j^2/(4(1+\beta_j))$ en β_0 gegeven, waar T_{2^j} een Chebyshev polynoom van de graad 2^j is.

a. Zij S een hermitische matrix die tevens normaal is, dan wordt de inverse van $I-\alpha S$ op een stabiele manier benaderd door $P_{2^q-1}(S/\gamma)$ met $\gamma \geqslant \rho(S)$ en $\beta_0=\alpha\gamma$. De spectraalnorm van de fout is maximaal β_q.

b. Zij A een scheefhermitische matrix die tevens normaal is, dan wordt de inverse van $I-\alpha A$ op een stabiele manier benaderd door $(1+\alpha A)P_{2^q-1}(A^2/\gamma)$ met $\gamma \geqslant \rho(A^2)$ en $\beta_0=\alpha^2\gamma$. De spectraalnorm van de fout is maximaal β_q.

IX

Het is ongewenst dat elke overheid in dit land bij aanstelling van een werknemer een "verklaring omtrent het gedrag" kan opvragen.

X

Voor een supercomputer is het belangrijker de gebruikersvriendelijkste te zijn dan de snelste.

XI

De ervaring dat een enorme inspanning gepleegd moet worden om een apparaat of programma van hoge complexiteit te ontwikkelen strijdt met de gedachte dat levende wezens bij toeval ontstaan zijn.

XII

Voor zowel expliciete als impliciete eenstapsmethoden voor de numerieke integratie van parabolische en hyperbolische partiële differentiaalvergelijkingen kan men volstaan met één array voor de onbekenden ter grootte van het hele veld en een werkruimte van een aantal arrays van één dimensie lager waar het aantal bepaald wordt door de gekozen tijdstap, de methode en de machinenauwkeurigheid.

XIII

Het is te hopen dat een dakloze na dit jaar tenminste op zijn eigen dak kan gaan zitten.

Numerical Solution of the Shallow-Water Equations

Part I : An SWE Solver for use on the CYBER 205

Part II : Theoretical Aspects

Numerical Solution of the Shallow-Water Equations

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit van Amsterdam, op gezag van de Rector Magnificus dr. S.K. Thoden van Velzen, hoogleraar in de Faculteit der Tandheelkunde, in het openbaar te verdedigen in de Aula der Universiteit (Oude Lutherse Kerk, ingang Singel 441, hoek Spui) op woensdag 21 oktober 1987 te klokke 16.00 uur

door

Friederik Wilhelm Wubs

geboren te Sellingen in 1957

1987
Centrum voor Wiskunde en Informatica

Promotor:	Prof. dr. P.J. van der Houwen
Co-promotor:	Dr. ir. G.K. Verboom

Preface

This thesis results from the research project "Evaluation and stabilization of numerical methods for the shallow-water equations", which started June 1983. The project was financed by the Foundation for Technical Research (STW), future Technical Science Branch Division of the Netherlands Organisation for the Advancement of Pure Scientific Research (ZWO), and carried out at the Centre for Mathematics and Computer Science (CWI) in Amsterdam under the guidance of Prof. dr. P.J. van der Houwen. The project was supervised by a user committee with members:

Ir. G.J.A. Loman (Hydronamic bv)
Dr. ir. G.K. Verboom (Delft Hydraulics)
Prof. dr. ir. P. Wesseling (Technical University Delft)
and Prof. dr. P.J. van der Houwen (University of Amsterdam, CWI).

The thesis consists of two parts; part I describes a numerical model for the shallow-water equations on the CYBER 205 and part II consists of 5 papers presenting theoretical aspects of the numerical integration of partial differential equations.

I am grateful to all those who contributed in some way to the realization of this thesis. I like to mention some of them explicitly.

In particular, I thank Prof. dr. P.J. van der Houwen for his guidance as a promotor, for his cooperation in the development of some of the papers in this thesis, and for his numerous valuable suggestions for the improvement of the final manuscript.

Special thanks go to Dr. ir. G.K. Verboom and B.P. Sommeijer; Dr. Verboom for his willingness to act as a co-promotor and for the illuminating discussions we had on the results of the various shallow-water flow computations presented in this thesis, and Mr. Sommeijer for his careful examination of the formulas in this thesis and the many constructive remarks on the manuscript.

I also wish to thank Drs. E.D. de Goede for his pleasant cooperation during the research which led to paper 5.

Furthermore, I wish to express my thanks to Drs. J.G. Blom, Mr. W.M. Lioen and Mr. D.T. Winter for their assistance in exploiting the computer facilities.

For the technical realization of this thesis, I would like to acknowledge Mr. R.T. Baanders, Mr. J. Schipper and Mr. D. Zwarst.

Finally, I wish to express my gratitude to the Centre for Mathematics and Computer Science for providing the necessary computer time for the project, and to the Delft Hydraulics and the Data Processing Division of Rijkswaterstaat who allowed me to use the WAQUA system.

Amsterdam, August 1987. F.W. Wubs

General Introduction

In recent years, numerical methods for solving problems in fluid dynamics have become more and more important, because (i) numerical methods are more flexible and now-a-days cheaper than *scale models*, and (ii) numerical models have become reliable for the simulation of a large variety of flow problems.
In the hydraulic engineering computations in the Netherlands, three models are widely used for the simulation of shallow-water flow:

1. The model of Leendertse [24] based on a *finite-difference* space discretization and on a *one-dimensional implicit* time discretization (ADI discretization).
2. The model of Stelling [38] which is a *storage economic* and *stabilized* modification of the original Leendertse model.
3. The model of Praagman [31, 35] based on a *finite element* space discretization and on an *explicit* time discretization (Runge-Kutta discretization and Sielecki discretization [7, 37]).

In general, finite differences are used on regular grids and finite elements are used on unstructured grids. Finite elements have the advantage that the spatial geometry of the problem can be approximated with higher accuracy than with finite differences, but, at the same time, the disadvantage that the associated code is less efficient because computers, especially vector computers, perform better for well-structured data (see, e.g. , the various contributions on vectorization in [34]). Since it is to be expected that vector computers will become more and more widely used, this aspect is important in developing codes for flow problems.

As far as the time discretization is concerned, explicit methods vectorize extremely well but the time step is limited by a stability condition so that they

have the undesirable property that often the time step is not dictated by accuracy considerations. To be more precise, the time step is much smaller than needed to achieve the required accuracy. The one-dimensional implicit ADI methods do not have this undesirable property, but cannot exploit the facilities of a vector computer as well as explicit methods can.

When, now four years ago, we started the project "Evaluation and stabilization of numerical methods for the shallow-water equations", the CYBER 205 computer of SARA (Stichting Academisch Rekencentrum Amsterdam) was just about to be installed. This motivated us to concentrate on methods which can take full advantage of this new architecture. In view of the above considerations, we decided to base our method on (i) a *finite-difference* space discretization and on (ii) a *fully explicit* time discretization. At the same time, we decided not only to develop a code for use on the CYBER 205, but also to study numerical techniques for optimizing finite difference discretizations and for stabilizing explicit time integration. In a later stage of the project, the code was tested on real engineering problems. The details of the numerical model for use on the CYBER 205, results of the various computations, and theoretical results which apply immediately to the shallow-water equations are reported in Part I of this thesis, and the more general investigations of space and time discretizations are presented in Part II, in the form of 5 papers (i.e. [49, 20, 19, 17, 12]), which are submitted for publication in scientific journals (3 of these papers are, at the time of writing, accepted for publication). We shall briefly indicate the subjects studied in this thesis.

Finite-difference space discretization
For the finite-difference space discretization we used as a starting point the ideas of Stelling [38]. In this work, second-order accurate differences at internal points and first-order or even zero-order accurate differences at the boundaries are used. Although, this discretization proved to be well-suited for a large variety of practical engineering problems, we have investigated whether it is possible to improve upon the efficiency of the numerical method with respect to the space discretization. Two possible approaches have been considered, (i) higher-order difference formulas and (ii) exponential fitting of dominant frequencies. The first is to use fourth-order accurate space discretizations in the internal domain. The fourth-order accuracy allows to use a coarser spatial grid by which the stability condition imposed by the explicit time discretization is relaxed. However, using a coarser grid may decrease the accuracy by which the geometry can be represented, so that the accuracy improvement obtained by the higher-order discretization is not always observed. Nevertheless, if the geometry can be approximated accurately by a small number of points then the use of high-order finite differences is relevant.
The second way to improve the efficiency of the space discretization is to use difference formulas which can be tuned to certain dominant Fourier components in the solution. If the corresponding frequencies are known, then these so-called *exponentially fitted* difference formulas represent very accurately the

dominant components and lead to a space discretization of increased accuracy (cf. [20]). However, these discretizations are less accurate for the frequencies they are not tuned to. Hence, in special cases, the exponentially fitted difference formulas improve the accuracy, but, in general, they will not, as the values of the frequencies of the dominant Fourier components depend on the location in the domain. Therefore, constructing a general purpose model, we have refrained from implementing these discretizations in our code.

Explicit time discretization
Because of its relatively large imaginary stability interval, the standard fourth-order Runge-Kutta method is often applied in the time discretization of hyperbolic problems such as the shallow-water equations (e.g., in [31]). In order to compensate for the still severe stability condition, several authors have employed a form of smoothing (cf. [25,21,44]). However, these papers all employ *implicit* smoothing which is less attractive on vector computers. Therefore, for vectorization reasons, we have developed an *explicit* smoothing technique which is extremely efficient on a vector computer (cf. [49,11]). In [19] and [17] a detailed analysis of explicit smoothing techniques is given, not only for hyperbolic problems but also for parabolic and elliptic problems.
Apart from these explicit methods we studied a class of explicit-implicit methods which look rather promising for use on a vector computer [12].

The implementation of the numerical method
In order to obtain an optimal performance on the CYBER 205, special attention is given to the implementation. The most difficult part to optimize is the boundary treatment. Boundaries can occur at any place in the domain and their location may change during the computation due to drying and flooding. To determine the location of the boundary points in the computational domain, we used so-called bit vectors (see FORTRAN 200 reference manual [1]). This type of vectors allows to execute efficiently logical operations over all grid points. Once the location of the boundary points is known, we construct the discretizations by using their factorized form (see Section 4.3.1). By this form, the bookkeeping in the program can be simplified.
For the large scale problems wa are aiming at, the CYBER 205 code turned out to run at a rate of 100 Megaflops.
The CYBER 205 code was extensively tested for a large variety of geometries. Furthermore, computations were performed for real engineering problems by incorporating it into the WAQUA system, which is a large computational system with extensive plotting facilities. This system is in use at Rijkswaterstaat and Delft Hydraulics for the simulation of water flow. Such type of computations are important for the appreciation of the practical merits of the code, because (i) it proves that the numerical method performs satisfactorily for non-academic problems, (ii) it shows that the code is capable of handling very complex geometries.
In addition to the CYBER 205 code, we also implemented an input processor and an output processor. By means of the input processor the user can

generate in a convenient way input data for a flow simulation by the CYBER 205 code. The output processor allows the user to visualize the output data of the simulation run.

Conclusions

The research project "Evaluation and stabilization of numerical methods for the shallow-water equations" has led to the following conclusions:

. Explicit methods are well-suited for the numerical integration of partial differential equations on vector computers.
. The classical Runge-Kutta method proves to be a robust integration method for shallow-water flow computations.
. The drawback of a limited time step, which is inherent in explicit methods, can be relaxed considerably by appropriate smoothing of the discretized right-hand side function, with only a modest increase of the computational effort per time step.
. Bit vectors and gather and scatter operations contribute to a large extent to the efficient implementation of the boundary treatment.
. By a large number of experiments it was shown that the physical behaviour of shallow-water flow was satisfactorily simulated (relatively to the available input data) by the numerical model.

Future research

This project has left a number of topics to be studied. We mention the most important ones.

. The representation of the boundary and the discretization near the boundary such that higher-order methods are more effective.
. Other integration methods may be considered as well. Since the solution often varies slowly in time, efficient techniques should possess a high-order of accuracy in combination with large stability regions. One such method is described in this thesis. An alternative may be fully implicit methods. However such schemes lead to the problem of solving a large algebraic system in each integration step. A third possibility is offered by using so-called explicit-implicit methods. A few first results show that such methods are promising (cf. [12]).
. Extension of the numerical model to three space dimensions. Such an ambitious project has become within the scope of numerical computations.

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,

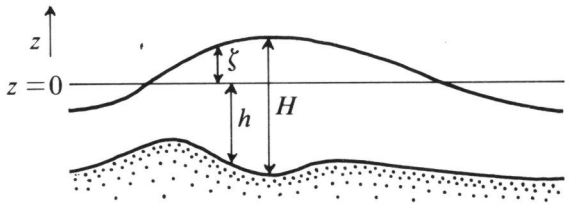


Figure 2.1. Vertical cross section of a flow field.

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.

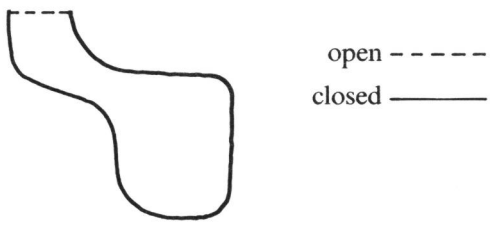


FIGURE 2.2. Example of a domain.

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 $\mathbf{v}=[u,v]^T$ 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 ($(\mathbf{v},\mathbf{n})\leq 0$) 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.

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.

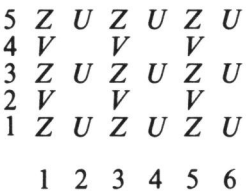


FIGURE 3.1. Position of the variables U, V and Z in space.

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.

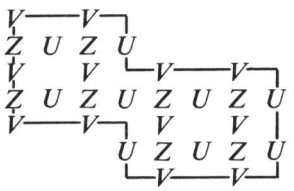


FIGURE 3.2. Boundary of the computational domain.

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)*
	$\begin{array}{ccc} & V & \mathbf{V} \\ U \mid\mid & Z & U\ Z \\ & V & \mathbf{V} \\ V & V & \mathbf{V} \\ Z \mid & U\ Z & U\ Z \\ V & V & \mathbf{V} \end{array}$	$1/(2\Delta x)\{\frac{1}{2}(E^2-E^{-2})\}V$	2	(3.5.b)*
	$\begin{array}{ccc} & \mathbf{V} & V \\ U \mid\mid & Z & U\ Z \\ & \mathbf{V} & V \\ V & \mathbf{V} & \\ Z \mid & U\ Z & \\ V & \mathbf{V} & \end{array}$	$1/(2\Delta x)\{(E^2-I)\}V$ 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.

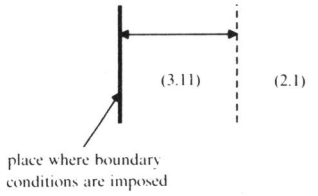


FIGURE 3.3. Domains where (3.11) and (2.1) are valid.

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.

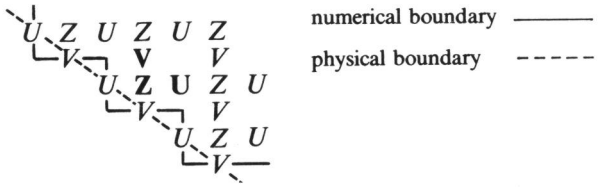


FIGURE 3.4. "Zig-zag" boundary.

A 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

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

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 $\mathbf{F}$ ($J = \partial \mathbf{F}(\mathbf{W}, t)/\partial \mathbf{W}$)

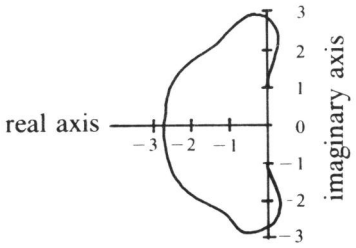


FIGURE 3.5. Stability region of the classical Runge-Kutta method.

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

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

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

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

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

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 $Q(z)=\tilde{Q}(z^2)$.

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.

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

Let the ordering of the dependent variables be given by

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

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.

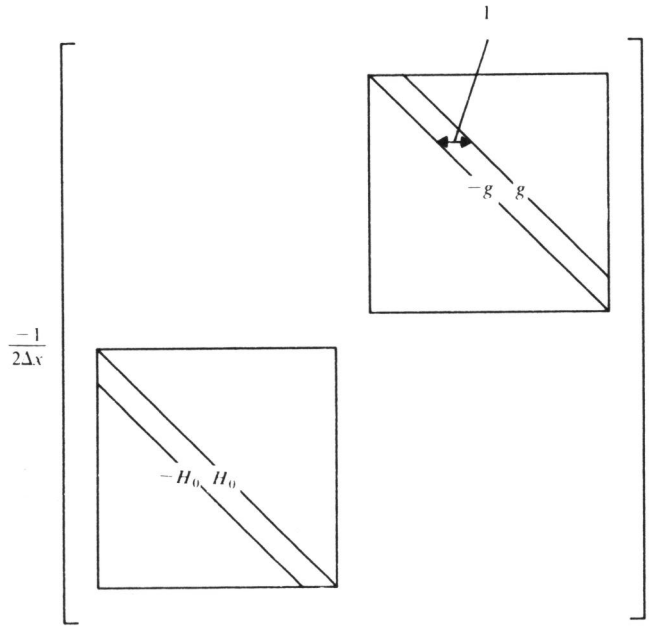


FIGURE 3.6. The form of the simplified Jacobian.

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)).

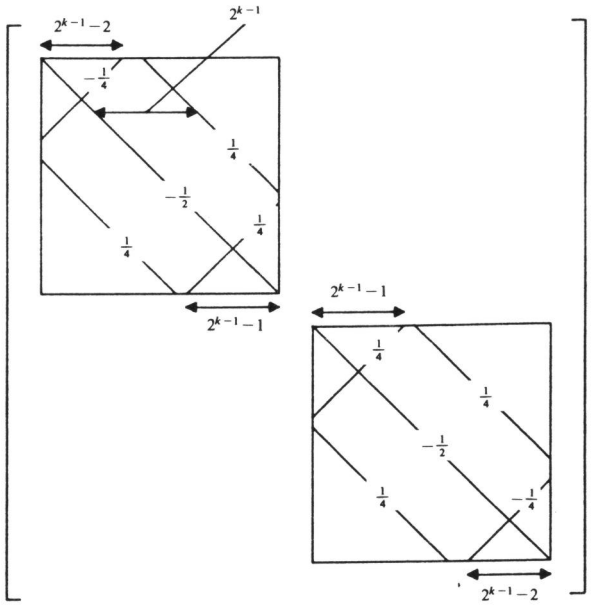


FIGURE 3.7. The structure of D_k.

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

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

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

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

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

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

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 $T_{2^k}-1=2(T^2_{2^{k-1}}-1)$ 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

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

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

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

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 $\sqrt{P_{2^q-1}(-z^2)z^2}$ for $z\in[0,1]$, which in turn is equal to the maximum of $\sqrt{-P_{2^q-1}^2(z)P_{2^q-1}(z)z}$ 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 $\Delta t g\sqrt{U^2+V^2}\,/(C^2H)$, 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 $\Delta t g\sqrt{U^2+V^2}\,/(C^2H)$

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.

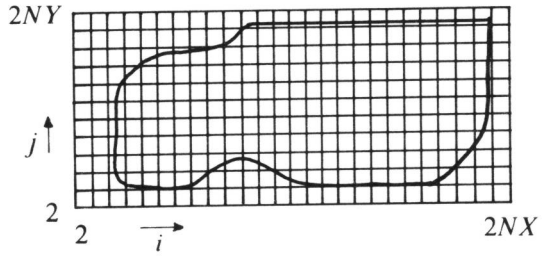


FIGURE 4.1. Example domain.

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 +.

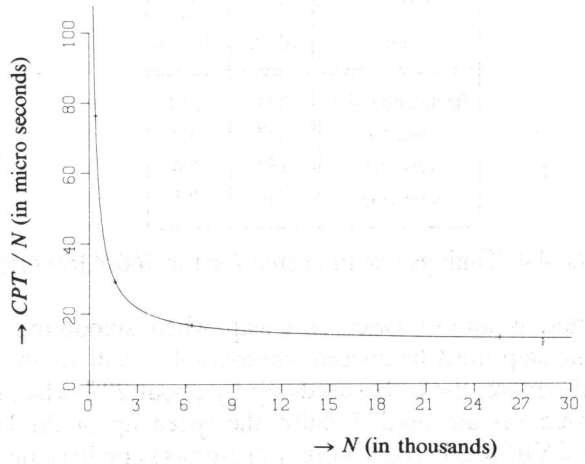


FIGURE 4.3. Plot of the CPU time per time step per grid point.

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.

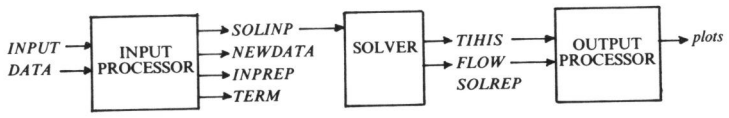


FIGURE 5.1. Flow of the system.

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.

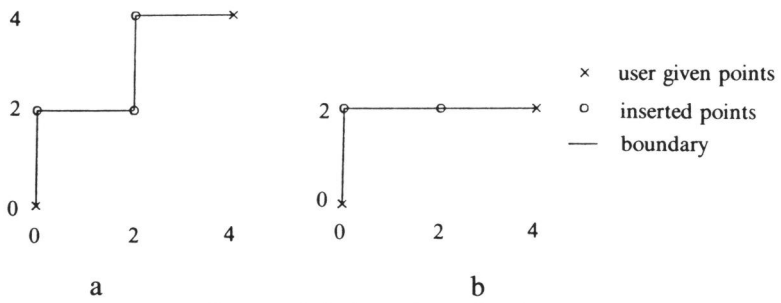


FIGURE 5.2. Insertion of points.

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.

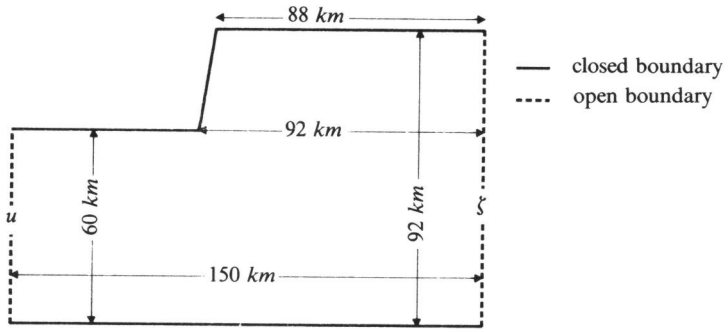


FIGURE 5.3. Example domain.

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.

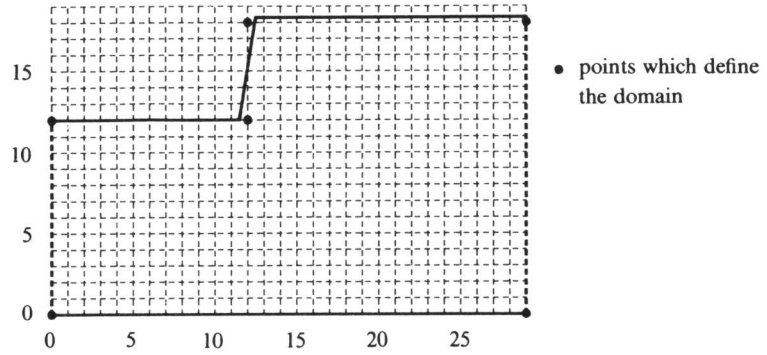


FIGURE 5.4. Example domain of Figure 5.3 covered by a 5 *km* grid.

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               0         Z
U               0         Z
U               0         Z
0 0 0 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.

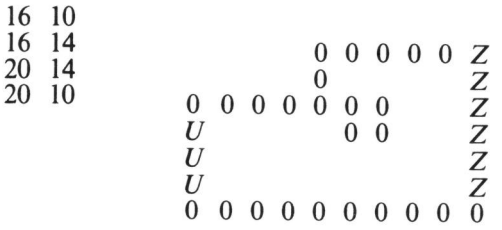


FIGURE 5.6. Domain of Example 5.2 with island. □

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])

$$C = 1.49H^{1/6}/n, \tag{5.2}$$

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.

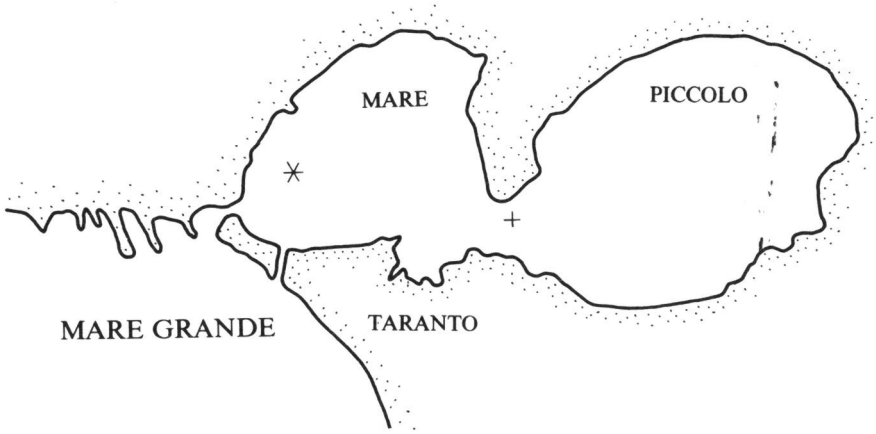
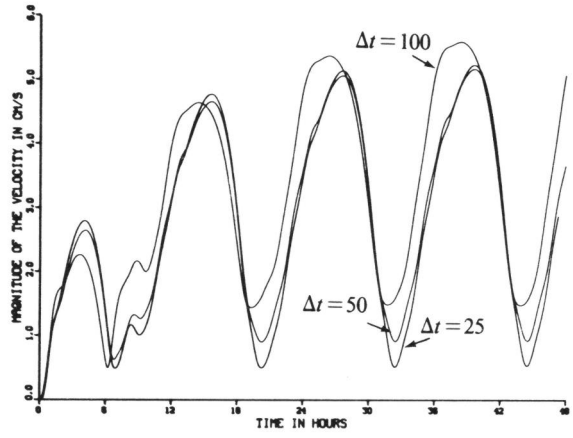
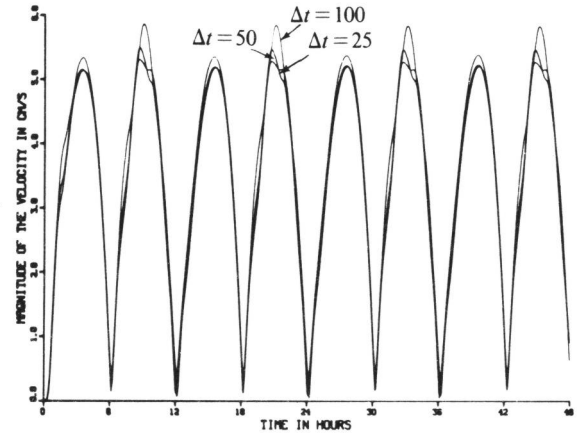


FIGURE 6.1. The Taranto bay.

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.



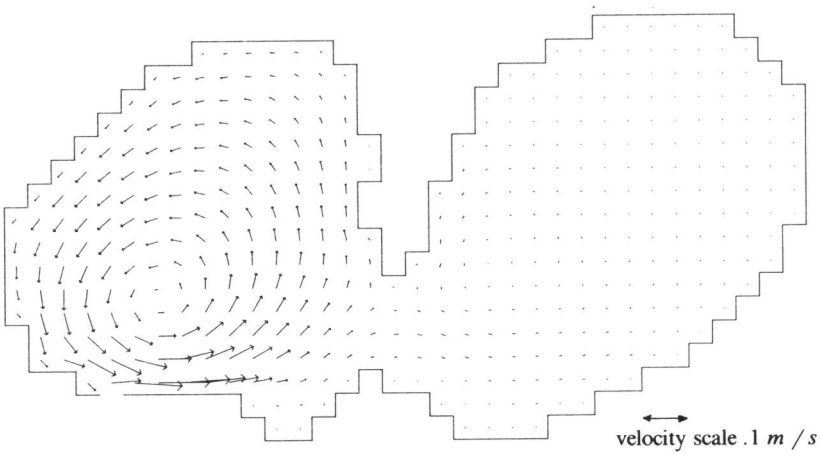
a. Time history at *.



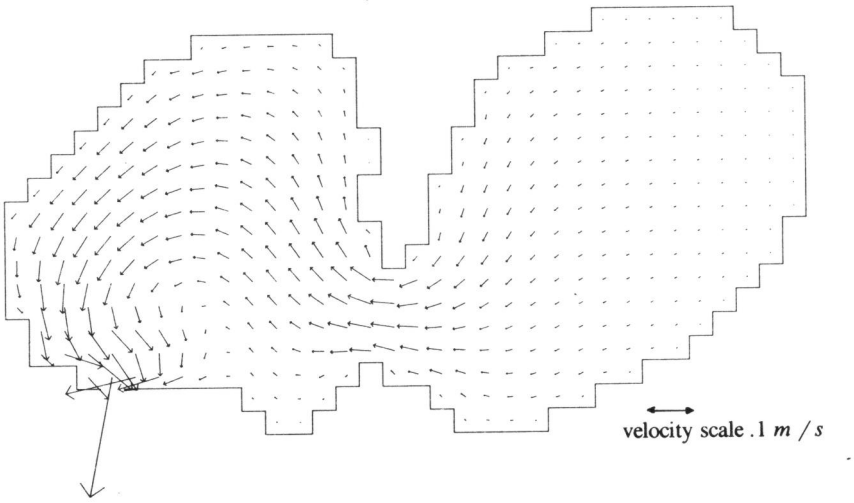
b. Time history at +.

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

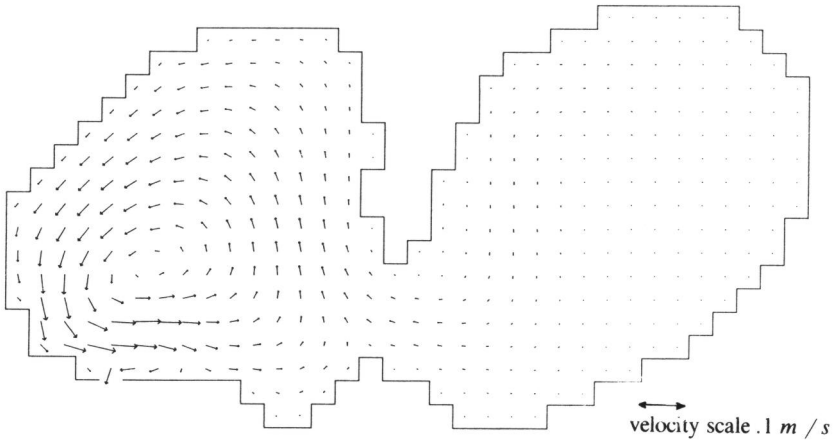
They show that the solution becomes periodic after a few tides. Moreover, we observe that the solution depends on the time step, showing the need for small time steps in this type of applications. It is interesting to see that the time step has a much larger influence on the solution at the point * than 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.



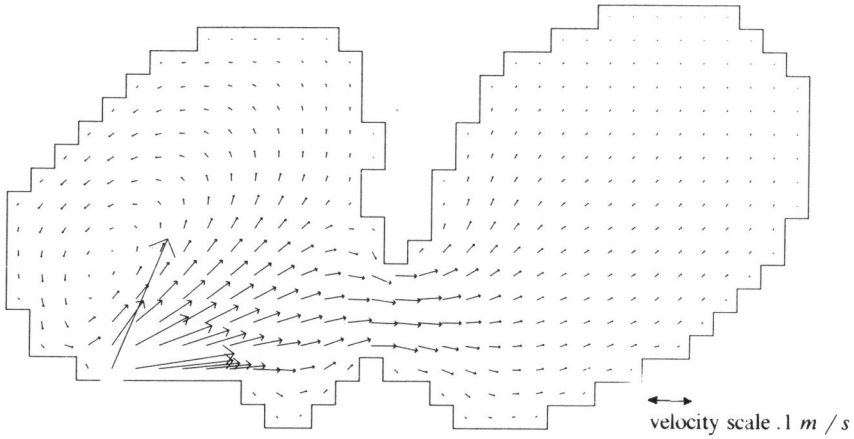
a. Flow field at 36 hours.



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).

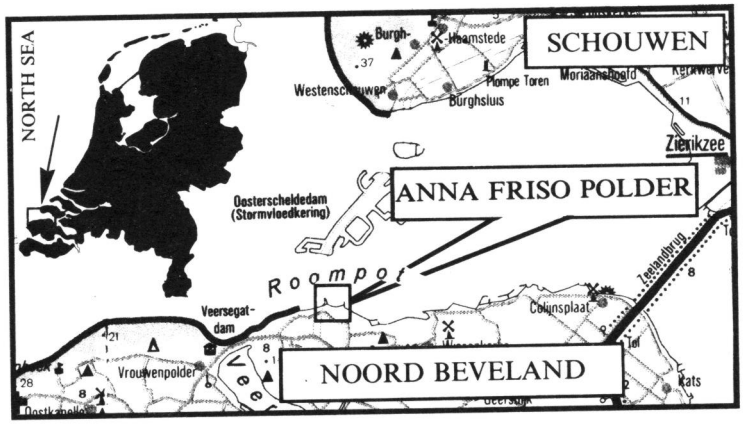


FIGURE 6.4 Location of the Roompot.

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.

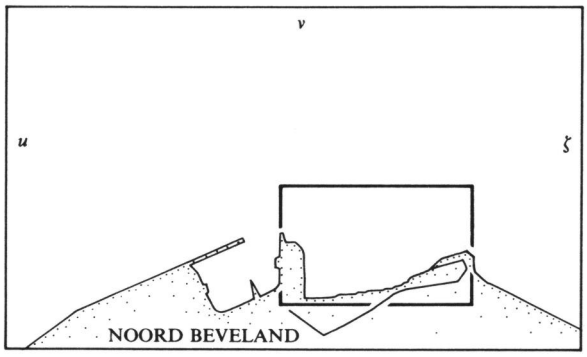


FIGURE 6.5. Computational domain.

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 variation 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.

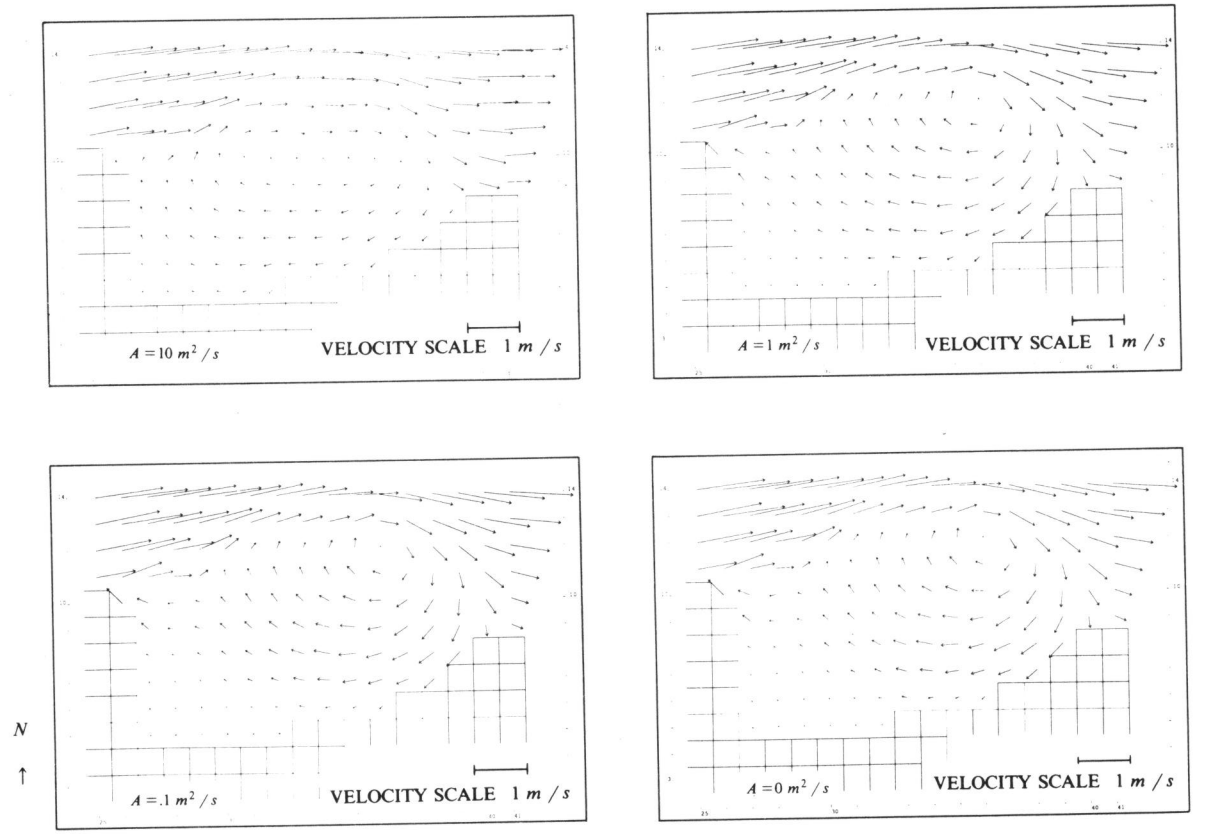


FIGURE 6.6. Vector plots for $A = 10, 1, .1, 0\ m^2/s$.

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.

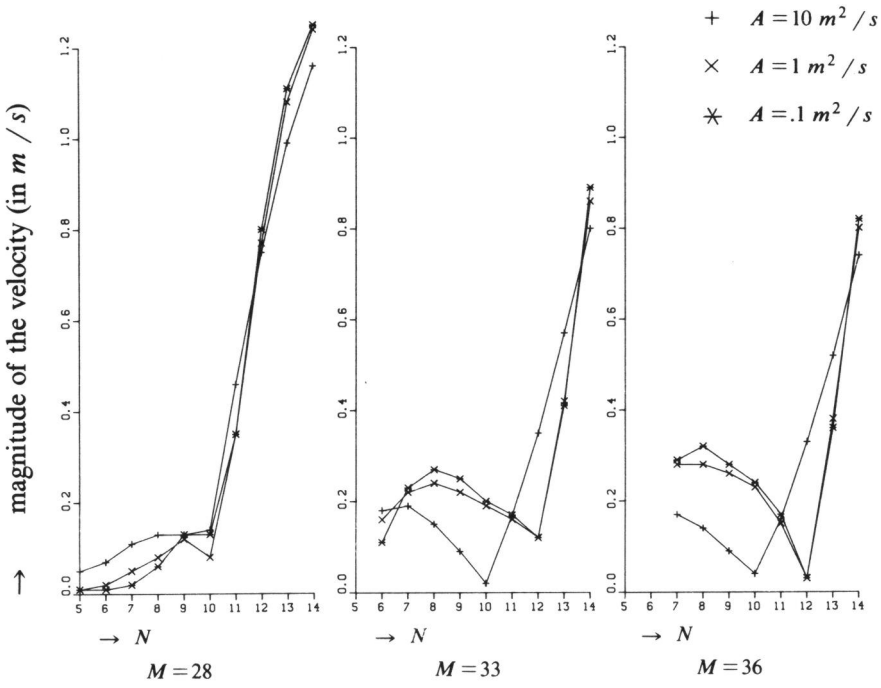


FIGURE 6.7. Cross-sections of the magnitude of the velocity at $M = 28$, 33, 36.

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.

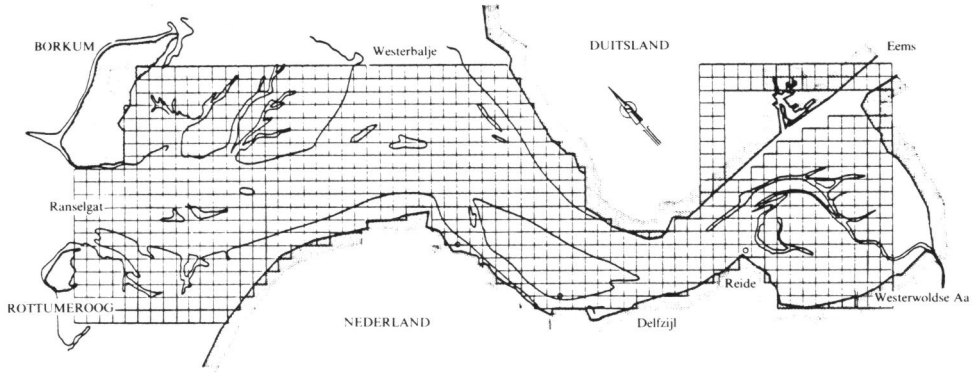


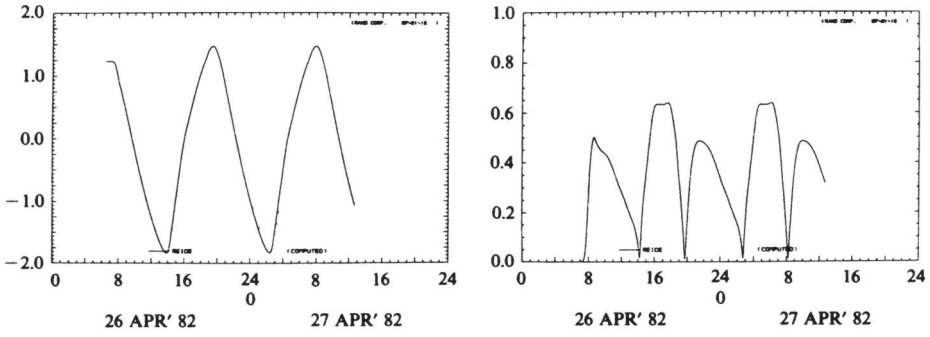
FIGURE 6.8. The Eems-Dollard estuary.

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.

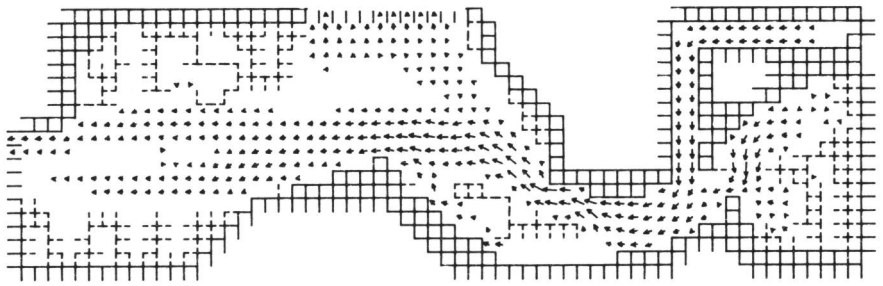


FIGURE 6.10. Flow field at low tide.

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.

Part II

Theoretical Aspects

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 6, 641–657 (1986)

STABILIZATION OF EXPLICIT METHODS FOR HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS

F. W. WUBS

Centre for mathematics and Computer Science, Amsterdam

SUMMARY

It is well known that explicit methods are subject to a restriction on the time step. This restriction is a drawback if the variation of the solution in time is so small that accuracy considerations would allow a larger time step. In this case, implicit methods are more appropriate because they do allow large time steps. However, in general, they require more storage and are more difficult to implement than explicit methods. In this paper we propose a technique by which it is possible to stabilize explicit methods for quasi-linear hyperbolic equations. The stabilization turns out to be so effective that explicit methods become a good alternative to unconditionally stable implicit methods.

1. INTRODUCTION

In numerical analysis, we distinguish explicit and implicit time integrators for partial differential equations. It is well known that explicit methods are subject to a restriction on the time step. This restriction is a drawback if the variation in time is so small that accuracy considerations would allow a larger time step. In this case, implicit methods are more appropriate because they do allow large time steps. However, in general, they require more storage and are more difficult to implement than explicit methods. In this paper, we propose a technique by which it is possible to stabilize explicit methods for quasi-linear hyperbolic equations. The stabilization turns out to be so effective that explicit methods become a good alternative to unconditionally stable implicit methods. More precisely, the stabilized explicit methods are competitive with conventional implicit methods with respect to both accuracy and computational costs. In fact, we will show, for some examples, that the technique also inherently appears in implicit methods, which explains the improved stability behaviour of implicit methods. In the fifties, explicit methods were quite popular because of their simplicity. Thereby, they were well suited for hand calculations and small computers. With the coming of more powerful computers in the sixties, having also a larger memory, implicit methods became popular. In the seventies, when the vector computers were introduced, the explicit methods became in scope again, because they allow a high degree of vectorization. Therefore, the stabilization technique given here may be of interest for the efficient use of explicit methods in a large variety of problems. In fact, our attention was focused on explicit methods when we started to construct a shallow-water equation solver for use on the vector computer CYBER 205.

In this paper, we restrict ourselves to hyperbolic problems; however the theory develops in a similar way for parabolic problems. In Section 2 the theory is presented and in Section 3 some numerical illustrations will be given.

0271–2091/86/090641–17$08.50

Received 2 May 1985
Revised 11 February 1986

2. THEORY

Consider the equation

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u}, \mathbf{u}_{x_1}, \mathbf{u}_{x_2}, \ldots, \mathbf{u}_{x_n}, \mathbf{x}, t), \quad \mathbf{x} \in R^n, \quad t > 0 \tag{1}$$

where $\mathbf{u} = (u_1(\mathbf{x}, t), u_2(\mathbf{x}, t), \ldots, u_N(\mathbf{x}, t))^T$, defining a first-order quasi-linear hyperbolic system of N equations.[1] Using explicit methods for (1), the time step is restricted by the Courant–Friedrichs–Lewy (C.F.L.) condition (see (25)). In many problems, this time step restriction is much more severe than the one following from accuracy considerations. For instance, in order to represent an irregular geometry, a fine space mesh is needed. At the same time the variation of the solution in time may be very slow. In that case, one likes to use much larger time steps than the one allowed by the C.F.L. condition. In the following sections, we will show that it is possible to stabilize an explicit method by an appropriate smoothing of the right-hand side. Smoothing of the right-hand side will not give rise to larger errors when $\mathbf{u}_t$ has small space derivatives. We will show for some examples that small time derivatives of $\mathbf{u}$ imply, under certain conditions, small space derivatives of $\mathbf{u}_t$. Moreover, this property is trivial for the limiting stationary case. We emphasize that $\mathbf{u}$ itself may have large space derivatives. For example, we may think of solutions which are close to a steady state and which can be written in the form

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{u}_0(\mathbf{x}) + \mathbf{u}_1(\mathbf{x}, t) \tag{2}$$

where $\mathbf{u}_0(\mathbf{x})$, the stationary solution, has large space derivatives and $\mathbf{u}_1(\mathbf{x}, t)$ is a smooth function in both variables $\mathbf{x}$ and t.

For the stabilization of explicit methods, smoothing is often used before, but then usually the grid function $\mathbf{u}$ is smoothed,[2,3] rather than the right-hand side. This smoothing of $\mathbf{u}$ may only be applied, without danger of loss of accuracy, if $\mathbf{u}$ itself is smooth, i.e. if $\mathbf{u}$ has small derivatives with respect to the space variables, which, in general, is not true. As an example, the famous variant of the Lax–Wendroff scheme proposed by Richtmyer and Morton[4] may be regarded as a two-stage second-order Runge–Kutta method,[5] where, in the first stage, the solution $\mathbf{u}$ is smoothed, in order to obtain a stable method.

In the field of the boundary-value problems the stabilization technique is known by the name residual averaging.[6] In this case, explicit time stepping is used to solve a boundary-value problem. The explicit method is then stabilized by using an implicit smoothing operator (see Section 2.4) in order to accelerate the convergence. Our contribution will be the construction of explicit smoothing operators which are less expensive than the implicit smoothing operators, especially if we want to use a vector computer.

2.1. *The smoothness of the right-hand side*

The assumption of a smooth right-hand side (or, equivalently, of $\mathbf{u}_t$) is important for the error introduced by smoothing. In hyperbolic equations we may expect in some cases that there is a relation between the variation of the solution in time and in space. For example, we may think of wave-like phenomena moving with some characteristic speed over the field. We will show for two examples that such a relation exists.

Example 1. Consider the one-dimensional system of equations

$$\mathbf{u}_t = \mathbf{A} \frac{\partial}{\partial x} \mathbf{u} + \mathbf{B}\mathbf{u} + \mathbf{g}(x), \; x \in R \tag{3}$$

where $\mathbf{u} = (u_1(x,t), u_2(x,t), \ldots, u_N(x,t))^T$, $\mathbf{A}$ is a non-singular, constant $N \times N$ matrix, $\mathbf{B}$ a constant $N \times N$ matrix and $\mathbf{g}$ an arbitrary continuous function of x. Differentiating (3) with respect to time yields

$$(\mathbf{u}_t)_t = \mathbf{A}\frac{\partial}{\partial x}\mathbf{u}_t + \mathbf{B}\mathbf{u}_t \tag{4}$$

Using (4) it follows that if all time derivatives of $\mathbf{u}$ up to order n are small, then the $(n\text{-}1)$th space derivative of $\mathbf{u}_t$ is small. Hence, the right-hand side in (3) has small space derivatives if $\mathbf{u}$ has small time derivatives.

Example 2. As a second example we consider the linearized shallow-water equations given by

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u}_x, \mathbf{u}_y) = -\left(\mathbf{A}\frac{\partial}{\partial x} + \mathbf{B}\frac{\partial}{\partial y}\right)\mathbf{u} + \mathbf{h}(x,y) \tag{5}$$

where $\mathbf{u} = (u, v, \zeta)^T$, $\mathbf{h}(x,y)$ is an arbitrary function of x and y, and

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & g \\ 0 & 0 & 0 \\ H & 0 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & g \\ 0 & H & 0 \end{bmatrix}, \quad g, H > 0$$

Differentiation of (5) with respect to t gives

$$(\mathbf{u}_t)_t = -\left(\mathbf{A}\frac{\partial}{\partial x} + \mathbf{B}\frac{\partial}{\partial y}\right)\mathbf{u}_t \tag{6}$$

Hence $\mathbf{u}_t$ is also a solution of the linearized shallow-water equations, in this case, without the forcing term $\mathbf{h}(x,y)$. Let us define by $\xi(x,y), \eta(x,y)$ a normalized orthogonal co-ordinate system. After transformation (in terms of these co-ordinates) (6) becomes

$$(\mathbf{u}_t)_t = -\left[\mathbf{A}\left(\frac{\partial \xi}{\partial x}\frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial}{\partial \eta}\right) + \mathbf{B}\left(\frac{\partial \xi}{\partial y}\frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial y}\frac{\partial}{\partial \eta}\right)\right]\mathbf{u}_t \tag{7}$$

Now we assume that the partial derivatives with respect to η are negligible and that all the time derivatives of $\mathbf{u}$ are small. Furthermore, we assume that

$$\frac{\partial \eta}{\partial x}\frac{\partial u_t}{\partial \xi} + \frac{\partial \eta}{\partial y}\frac{\partial v_t}{\partial \xi} \tag{8a}$$

is small. This condition says that the vector $((u_t)_\xi, (v_t)_\xi)^T$ is small in the η-direction, given by the vector $(\eta_x, \eta_y)^T$. Using the special structure of $\mathbf{A}$ and $\mathbf{B}$, it follows from the first and second equations in (6) that $(\zeta_t)_x$ and $(\zeta_t)_y$ are small. Furthermore, from the third equation in (7), it follows that

$$\frac{\partial \xi}{\partial x}\frac{\partial u_t}{\partial \xi} + \frac{\partial \xi}{\partial y}\frac{\partial v_t}{\partial \xi} \tag{8b}$$

is small. Combining (8a) and (8b), we have that $(u_t)_\xi$ and $(v_t)_\xi$ are small. As the derivatives with respect to η are negligible, it follows that $(u_t)_x, (u_t)_y$ and $(v_t)_x, (v_t)_y$ are small. Hence, all first-order space derivatives of $\mathbf{u}_t$ are small. Proceeding in the same way, under similar assumptions, it is possible to show that all higher-order space derivatives of $\mathbf{u}_t$ are small.

These two examples show that, under certain assumptions, we may expect that the right-hand side is smooth in space if the time derivatives of the solution are small. The property of a smooth

right-hand side in space can be used effectively to stabilize an explicit time integration method by smoothing the discretized form of $\mathbf{f}(\mathbf{u}, \mathbf{u}_{x_1}, \mathbf{u}_{x_2}, \ldots, \mathbf{u}_{x_n}, \mathbf{x}, t)$, which is obtained by the method of lines. In this approach, the space discretization gives rise to a system of ordinary differential equations[5]

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{U} = \mathbf{F}(\mathbf{U}, t), \tag{9}$$

where $\mathbf{U}$ is a grid function approximating $\mathbf{u}$, and $\mathbf{F}(., t)$ a vector function approximating $\mathbf{f}(., \mathbf{x}, t)$. Thereafter, an appropriate time integrator is used to solve this equation. Instead of (9), we propose to solve

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{U} = S\mathbf{F}(\mathbf{U}, t), \tag{10}$$

where S is a smoothing operator, with the property $S \to I$, the identity operator, when the mesh size tends to zero.

In fact, many unconditionally stable time integrators, applied to (9), can be written as a conditionally stable (explicit) integrator applied to (10). We will illustrate this for Euler's backward method applied to

$$\begin{aligned} u_t &= f(u_x, x) \\ f(u_x, x) &= u_x + g(x) \end{aligned} \tag{11}$$

where $g(x)$ is an arbitrary function of x. The right-hand side in (11) is discretized, on a grid with mesh size h, with the usual second-order central differences

$$F_j(\mathbf{U}) = (D\mathbf{U})_j + g(x_j),\ x_j = jh \tag{12}$$

where

$$(D\mathbf{U})_j = (U_{j+1} - U_{j-1})/(2h) \tag{13}$$

and U_j approximates $u(x_j)$. When backward Euler is applied to (9), with $\mathbf{F}$ given by (12), we find

$$U_j^{n+1} - \Delta t(D\mathbf{U})_j^{n+1} = U_j^n + \Delta t\, g(x_j) \tag{14}$$

where $\mathbf{U}^n$ approximates the exact solution $\mathbf{U}(t)$ of (9) at $t^n = n\Delta t$. This can be rewritten as

$$U_j^{n+1} - \Delta t(D\mathbf{U})_j^{n+1} = U_j^n - \Delta t(D\mathbf{U})_j^n + \Delta t F_j(\mathbf{U}^n) \tag{15}$$

As the operator $(I - \Delta tD)$ is invertible, we find

$$U_j^{n+1} = U_j^n + \Delta t\{(I - \Delta tD)^{-1}\mathbf{F}(\mathbf{U}^n)\}_j \tag{16}$$

which is simply forward Euler applied to (10) with the smoothing operator $S = (I - \Delta tD)^{-1}$. A discussion of this smoothing operator and another example can be found in the Appendix. Here, we mention that the time step appears in the smoothing operator. Because the magnitude of the time step determines the amount of smoothing needed to obtain a stable method, it will also appear in our smoothing operators. Moreover, the time step in the smoothing operator ensures the consistency of (10) with (9).

In the remainder of this section, we will illustrate the theory by the scalar equation (11) and its semi-discretization (12). We are aware of the fact that (11) is simple, but it gives relevant information for less trivial cases (see Section 3.3).

2.2. *Stability*

In order to solve the initial value problem (9) we consider explicit m-point single step Runge–

Kutta formulae, i.e. formulae of the type

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \sum_{i=0}^{m-1} \theta_i \mathbf{K}_i \tag{17}$$

$$\mathbf{K}_i = \Delta t \mathbf{F}\left(\mathbf{U}^n + \sum_{l=0}^{i-1} \alpha_{i,l} \mathbf{K}_l, t^n + \mu_i \Delta t \right)$$

$$\mu_0 = 0$$

When (17) is applied to the scalar equation

$$\frac{du}{dt} = \lambda u \tag{18}$$

we obtain

$$u^{n+1} = P_m(\Delta t \lambda) u^n$$

where $P_m(z)$ is a polynomial of the form

$$P_m(z) = \beta_0 + \beta_1 z + \cdots + \beta_m z^m \tag{19}$$

of which the coefficients β_j can be expressed in terms of the Runge–Kutta parameters. The polynomial $P_m(z)$ is the so-called stability polynomial associated with formula (17). The polynomial is compatible with a Runge–Kutta formula of order p, provided that

$$\beta_j = \frac{1}{j!}, \quad j = 0, 1, \ldots, p \tag{20}$$

Furthermore, the region defined by

$$S = \{ z | \quad |P_m(z)| < 1 \} \tag{21}$$

will be called the stability region of the Runge–Kutta formula and $P_m(\Delta\lambda)$ is called the amplification factor.[4] In this paper, scheme (17) is said to be stable when the set of points $\Delta t \lambda$, where λ is an eigenvalue of the Jacobian matrix of (9), belongs to the stability region S.

For example, if we apply (17) to (9) with the right-hand side (12) and if we assume periodic boundary conditions, then the eigenfunctions of the Jacobian matrix are Fourier components,

$$V_j = \exp(\mathrm{i} b j h), \quad b \in R \tag{22}$$

and the associated eigenvalues are

$$\lambda = \mathrm{i} \frac{\sin(bh)}{h} \tag{23}$$

Thereby, the amplification factor becomes $P_m(\Delta t \mathrm{i} \sin(bh)/h)$. Furthermore, the method is von Neumann stable if the absolute value of this amplification factor is smaller than one.[4]

The largest constant C, such that $|P_m(\mathrm{i}y)| < 1$ for $y \in [-C, C]$ and $y \in R$, is called the imaginary stability boundary. For the classical Runge–Kutta method $C = 2\sqrt{2}$.[5] If, in general, an explicit method has an imaginary stability boundary C, then when this method is applied to $\{(9), (12)\}$ we have the von Neumann stability condition

$$\Delta t < Ch \tag{24}$$

When smoothing is applied it is of interest to compare the stability condition with the C.F.L.

condition, because the stability condition can never exceed the C.F.L. condition. The C.F.L. condition says that the convex hull of the domain of dependence of the exact solution at a point $\mathbf{x}$ at time t_1 must be contained in the convex hull of the domain of dependence of the approximating solution at the same point in space and time.[7]

Lemma 1. Let (11) be discretized with central differences involving l points to the left and to the right. If an explicit Runge–Kutta method with stability polynomial $P_m(z)$ is used to solve the resulting system of ODEs then we have the C.F.L. condition

$$\Delta t \leqslant mlh \tag{25}$$

Proof

It is straightforwardly proved that (25) is the C.F.L. condition for the Runge–Kutta method.

2.3. *Explicit smoothing operators*

2.3.1. Derivation. Consider the smoothing operator S defined by

$$(S_1\mathbf{F})_j := (F_{j+1} + F_{j-1})/2 \tag{26}$$

In order to determine the maximum allowed time step, we now need the eigenvalues of $S_1 D$. These are simply the products of the eigenvalues of S_1 and D, because S_1 and D have the same eigenfunctions (22). The eigenvalues of S_1 are

$$\lambda_{S_1} = \cos(bh) \tag{27}$$

and the products of the eigenvalues of S_1 and D

$$\lambda_{S_1 D} = \cos(bh)\mathrm{i}\sin(bh)/h = \mathrm{i}\sin(2bh)/(2h) \tag{28}$$

Hence, compared with (23) the maximum eigenvalues have been reduced by a factor two. However, this may still be very restrictive. Therefore, we repeat the smoothing. Defining a second smoothing operator by

$$(S_2\mathbf{F})_j := (F_{j+2} + F_{j-2})/2 \tag{29}$$

we have, along the same line, that again a factor two is won. In general, we apply the smoothing operator

$$S := \prod_{k=1}^{n} S_k \tag{30}$$

where

$$(S_k\mathbf{F})_j := (F_{j+2^{k-1}} + F_{j-2^{k-1}})/2 \tag{31}$$

The maximum eigenvalue is now reduced by a factor 2^n. This means that the time step can be increased exponentially, whereas the costs grow linearly. Hence, as 2^n time steps are more expensive than one time step with n smoothings, smoothing makes the method much more efficient.

The reader may have noticed that in the case $g \equiv 0$ the smoothing degenerates to a discretization on a coarser grid. This appears quite natural, because of the following reasoning. The solution is of the form

$$u(x, t) = r(x + t) \tag{32}$$

where r is a function depending on the initial and boundary conditions. If in this case the time

derivatives are small, then also the space derivatives are small. Hence, if for accuracy reasons the time step may be increased then also the mesh size may be increased. If, however, g is non-zero the discretization differs essentially from the one on a coarser grid. For example, a function $g(jh) = (-1)^j$ cannot be approximated on a coarser grid.

We now will define the smoothing operator more generally by

$$S := \prod_{k=k_0}^{n} S_k \tag{33}$$

where

$$(S_k \mathbf{F})_j := \mu_k F_{j+2^{k-1}} + (1-2\mu_k)F_j + \mu_k F_{j-2^{k-1}} \tag{34}$$

Notice that the smoothing operator in (34) appears to be an identity operator plus a discretized form of a diffusion operator. For $\mu_k = \frac{1}{2}$ for all k and $k_0 = 1$ we have again (30). Another, special smoothing operator following from (33) is the case where $\mu_k = \frac{1}{4}$ for all k and $k_0 = 2$. The eigenvalue of (34) for this value of μ_k is

$$\lambda_{S_k} = \cos^2(2^{k-2}bh) \tag{35}$$

Now, when (33) is applied to (12), again the corresponding eigenvalues may be multiplied and we find

$$\lambda_{SD} = \mathrm{i} \prod_{k=2}^{n} \cos(2^{k-2}bh)\sin(2^{n-1}bh)/(2^{n-1}h) \tag{36}$$

In order to approximate the modulus of the maximum eigenvalue we need the inequality

$$|\cos(x)\sin(2x)| = |2(1-\sin^2(x))\sin(x)| \leqslant \tfrac{4}{9}\sqrt{3} \tag{37}$$

Isolating $\cos(2^{n-2}bh)$ from the product sequence (36) and combining it with $\sin(2^{n-1}bh)$, we can apply inequality (37) to find an upper bound for the maximum modulus of (36). This gives

$$|\lambda_{SD}| < \tfrac{4}{9}\sqrt{3}/(2^{n-1}h) \approx 0{\cdot}77/(2^{n-1}h) \tag{38}$$

In Reference 8 it is shown that (30), (31) defines an optimal smoothing operator for the considered equation. However, for initial-boundary value problems we found that this operator gave unstable results, whereas the operator (34) with $k_0 = 2$ and μ_k slightly smaller than $\frac{1}{4}$ gave stable results (see Section 3.3). This is possibly due to the fact that in the latter case (33) has positive eigenvalues.

With respect to the C.F.L. condition (25) we remark that an application of (33) gives rise to a differencing in which $l = 1 + \sum_{k=k_0}^{n} 2^{k-1} = 2^n - 2^{k_0-1} + 1$. In the case without smoothing $l = 1$. Application of Lemma 1 in both cases shows that with the resulting method after smoothing an increase of the time step with a factor $2^n - 2^{k_0-1} + 1$ is possible. This factor is obtained for $k_0 = 1$ and almost obtained for $k_0 = 2$, as can be seen from the reduction of the magnitude of the spectral radius of the Jacobian matrix.

2.3.2. The smoothing error. Here we will give an approximation of the error due to the smoothing operation (33), for μ_k independent of k. This is achieved by comparing the smoothed and non-smoothed right-hand sides. We will see that the smoothness of the original right-hand side and the time step determine the magnitude of the error. If, in the following, the subscript h is used in connection with a continuous function, then this denotes the restriction of that function to the grid.

Lemma 2. Let $\mathbf{A}(\xi_h)$ be a discretization of $a(\xi(x), \xi_x(x), x)$. If $\mathbf{A}(\xi_h)$ and ξ_h satisfy the condition

$$A_j(\xi_h) = a(\xi(x_j), \xi_x(x_j), x_j) + C_j h^2 + O(h^4) \tag{39}$$

$$C_{j\pm1} = C_j \pm D_j h + O(h^2) \tag{40}$$

and, moreover, $a(\xi(x), \xi_x(x), x) \in C^4$, then the error due to the smoothing operator (33) is, with $\mu_k = \mu$,

$$(S\mathbf{A}(\xi_h))_j - A_j(\xi_h) = \mu h^2 \frac{2^{2n} - 2^{2k_0-2}}{3} \frac{\partial^2}{\partial x^2} a(\xi(x_j), \xi_x(x_j), x_j) + O(h^4) \tag{41}$$

Proof

Let $\phi(x) = a(\xi(x), \xi_x(x), x)$. Using Taylor expansions, we find by substitution of $\phi(x)$ into (34)

$$(S_k \phi_h)_j = \left(1 + \mu(2^{k-1}h)^2 \frac{\partial^2}{\partial x^2}\right)\phi(x_j) + O(h^4) \tag{42}$$

Hence, we have the following error due to the smoothing (33) for $\phi(x_j)$:

$$\begin{aligned}(S\phi_h)_j - \phi(x_j) &= \prod_{k=k_0}^{n} \left(1 + \mu(2^{k-1})^2 \frac{\partial^2}{\partial x^2}\right)\phi(x_j) + O(h^4) - \phi(x_j) \\ &= \mu h^2 \left(\sum_{k=k_0}^{n} (2^{k-1})^2\right) \frac{\partial^2}{\partial x^2} \phi(x_j) + O(h^4) \\ &= \mu h^2 \frac{2^{2n} - 2^{2k_0-2}}{3} \frac{\partial^2}{\partial x^2} \phi(x_j) + O(h^4)\end{aligned} \tag{43}$$

With (39) it follows that

$$(S\mathbf{A}(\xi_h))_j - A_j(\xi_h) = (Sa_h)_j - a(\xi(x_j), \xi_x(x_j), x_j) + h^2((S\mathbf{C})_j - C_j) + O(h^4) \tag{44}$$

It follows from (40) that $(S\mathbf{C})_j - C_j$ is of $O(h^2)$. Furthermore, by assumption $\phi(x) = a(\xi(x), \xi_x(x), x)$, hence, the lemma follows by substitution of (43) into (44).

Corrollary. The error due to the smoothing operator (2.33) is of $O(h^2)$.

Theorem 1. Let the conditions of Lemma 2 be satisfied. Let $\mathbf{F}$ be defined by (12). Let C be the imaginary stability boundary of an explicit method (see (24)). Then the error due to the smoothing operator (33) is, when the maximum allowed time step is used, for the special case $\mu_k = \frac{1}{2}$, $k_0 = 1$,

$$(S\mathbf{F}(u_h))_j - F_j(u_h) = \frac{(\Delta t/C)^2 - h^2}{6} \frac{\partial^2}{\partial x^2} f(u_x, x_j) + O(h^4) \tag{45}$$

and for the special case $\mu_k = \frac{1}{4}$, $k_0 = 2$

$$(S\mathbf{F}(u_h))_j - F_j(u_h) = \frac{\frac{32}{27}(\Delta t/C)^2 - 2h^2}{6} \frac{\partial^2}{\partial x^2} f(u_x, x_j) + O(h^4) \tag{46}$$

Proof

First we prove (45). Denote by Δt_0 the maximum time step without smoothing. Hence, from (24) $\Delta t_0/h = C$. In Section 2.3.1 we have found that the time step can be increased by a factor 2^n. This gives $(\Delta t/\Delta t_0) = 2^n$. Substituting this into (41) and setting $\xi(x) = u(x, t)$ for some time t, we arrive at (45). The proof of (46) follows the same line, except that from (38), the time step can now be increased by a factor $\frac{3}{4}\sqrt{32^{n-1}}$.

2.4. An implicit smoothing operator

Another smoothing operator, we want to introduce, is an implicit one. This smoothing

operator is implicitiy defined by

$$-\mu(S\mathbf{F})_{j+1}+(1+2\mu)(S\mathbf{F})_j-\mu(S\mathbf{F})_{j-1}=F_j \tag{47}$$

For the eigenfunctions (22), the eigenvalues of this system are

$$\lambda_S=1/[1+4\mu\sin^2(bh/2)] \tag{48}$$

The reduction factor is found by the multiplication of the eigenvalues of S and D, giving

$$\begin{aligned}\lambda_{SD}&=\mathrm{i}\sin(bh)/\{h[1+4\mu\sin^2(bh/2)]\}\\&=2\mathrm{i}\sin(bh/2)\cos(bh/2)/\{h[1+4\mu\sin^2(bh/2)]\}\end{aligned} \tag{49}$$

Omiting $\cos(bh/2)$, which is less than one, and writing $x=\sin(bh/2)$ we find

$$|\lambda_{SD}|<2x/[h(1+4\mu x^2)],\quad 0<x<1. \tag{50}$$

By differentiation with respect to x we find a maximum of the right-hand side for

$$\begin{aligned}&x=1/\sqrt{(4\mu)},\quad \mu>\tfrac{1}{4},\\&x=1,\quad 0<\mu<\tfrac{1}{4}.\end{aligned} \tag{51}$$

Substitution in (50) gives

$$\max|\lambda_{SD}|<1/(2h\sqrt{\mu}) \tag{52}$$

Hence, increasing μ by a factor of four will decrease the maximal eigenvalue by a factor of two.

Notice that from the stability condition (24) and from (52) it follows that

$$\mu\geqslant\tfrac{1}{4}\Delta t^2/(C^2h^2) \tag{53}$$

If μ satisfies this condition, then we have constructed an unconditionally stable method. Compared to the usual implicit time integrators, this method is simpler to implement. Especially if the right-hand side (see (1)) becomes non-linear and complicated.

Theorem 2. Let the conditions of Lemma 2 (see Section 2.3.2) be satisfies. Let $\mathbf{F}$ be given by (12). Let C be the imaginary stability boundary of an explicit method (see (24)). Assume periodic boundary conditions and equality in (53). Then the error due to the implicit smoothing operator (47) is given by

$$(S\mathbf{F}(u_h))_j-F_j(u_h)=\frac{1}{4}\frac{\Delta t^2}{C^2}\frac{\partial^2}{\partial x^2}f(u_x,x_j)+O(h^4) \tag{54}$$

Proof

From (47) we obtain

$$(S^{-1}\mathbf{F})_j=-\mu F_{j+1}+(1+2\mu)F_j-\mu F_{j-1} \tag{55}$$

Using Gerschgorin's theorem,[9] we have that the minimum eigenvalue of S^{-1} is greater than or equal to 1. Hence the spectral radius of S is smaller than or equal to 1 and thereby S is bounded in any matrix norm. Let S' be defined by

$$(S'\mathbf{F})_j=\mu F_{j+1}+(1-2\mu)F_j+\mu F_{j-1} \tag{56}$$

then for a test function $\phi(x)\in C^2$ we have

$$\begin{aligned}(S\phi_h - S'\phi_h)_j &= \{S(\phi_h - S^{-1}S'\phi_h)\}_j \\ &= \left\{S\left[\phi_h - S^{-1}\left(1+\mu h^2\frac{\partial^2}{\partial x^2}\right)\phi_h + O(h^3)\right]\right\}_j \\ &= \{S(\phi_h - S^{-1}\phi_h)\}_j + O(h^2) \\ &= \left\{S\left[\phi_h - \left(1+\mu h^2\frac{\partial^2}{\partial x^2}\right)\phi_h\right]\right\}_j + O(h^2) \\ &= \left\{S\left(-\mu h^2\frac{\partial^2}{\partial x^2}\phi_h\right)\right\}_j + O(h^2) = O(h^2) \end{aligned} \tag{57}$$

From (39), (56) and (57) it follows that

$$\begin{aligned}(S\mathbf{F}(u_h))_j - F_j(u_h) &= \{S[\mathbf{F}(u_h) - S^{-1}\mathbf{F}(u_h)]\}_j \\ &= \{S[f_h - S^{-1}f_h + h^2(\mathbf{C} - S^{-1}\mathbf{C})]\}_j + O(h^4) \\ &= \left\{S\left[\mu h^2\frac{\partial^2}{\partial x^2}f(u_x, x)\right]_h\right\}_j + O(h^4) \\ &= \left\{S'\left[\mu h^2\frac{\partial^2}{\partial x^2}f(u_x, x)\right]_h\right\}_j + O(h^4) \\ &= \mu h^2\frac{\partial^2}{\partial x^2}f(u_x, x_j) + O(h^4) \end{aligned} \tag{58}$$

From (58) and equality in (53), we obtain (54).

2.5. *Systems*

In the case of systems, the same smoothing operators can be used again. This proceeds as follows. First the terms on the right-hand side which play an important role with respect to stability have to be determined. These terms contain, in general, a derivative in one of the space directions. Then the right-hand side of the equation containing such a term has to be smoothed in the same direction as that of the derivative. If this equation contains important derivatives with respect to stability in more directions, then this equation is successively smoothed in all these directions. We will clarify this by an example.

Example 3. The shallow-water equations can be written in the form

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u}, \mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_{xx}, \mathbf{u}_{yy}, x, y, t) \tag{59}$$

where $\mathbf{u} = (u, v, \zeta)^T$ and

$$\begin{aligned} f^u(\cdot) &= -uu_x - vu_y - g\zeta_x + \nu\Delta u - C_z\sqrt{(u^2+v^2)}u/H \\ f^v(\cdot) &= -uv_x - vv_y - g\zeta_y + \nu\Delta v - C_z\sqrt{(u^2+v^2)}v/H \\ f^\zeta(\cdot) &= -(Hu)_x - (Hv)_y \\ H(x, y, t) &= h(x, y) + \zeta(x, y, t) \end{aligned}$$

where g, ν and C_z are positive constants. These equations are again discretized by standard central differences (13). In many applications it suffices to consider for stability a reduced Jacobian of the

space discretized form of (59), i.e.

$$\mathbf{J}_r = -\begin{bmatrix} 0 & 0 & gD_x \\ 0 & 0 & gD_y \\ \tilde{H}D_x & \tilde{H}D_y & 0 \end{bmatrix} \tag{60}$$

where D_x and D_y are the discretizations of $\partial/\partial x$ and $\partial/\partial y$, respectively, and $\tilde{H}$ is a constant approximating $H(x, y, t)$. If this Jacobian is applied to a Fourier component $\exp[\mathrm{i}(b_1 x + b_2 y)]$, then we obtain the eigenvalues $0, \pm\sqrt{[g\tilde{H}(\lambda_x^2 + \lambda_y^2)]}$, where λ_x and λ_y are the eigenvalues of D_x and D_y for the Fourier component, respectively. According to this outcome, it seems appropriate to smooth the right-hand sides of the successive equations in the x-, y- and x, y-directions, respectively. Using the same one-dimensional smoothing operators as in the previous sections, the third right-hand side has to be smoothed in x- and y-directions, successively. After smoothing, the Jacobian matrix becomes

$$\mathbf{J}_r = -\begin{bmatrix} 0 & 0 & gS_xD_x \\ 0 & 0 & gS_yD_y \\ \tilde{H}S_yS_xD_x & \tilde{H}S_xS_yD_y & 0 \end{bmatrix} \tag{61}$$

where S_x and S_y denote the smoothing operators in the x- and y-directions, respectively. The eigenvalues of the Jacobian after smoothing are $0, \pm\sqrt{\{g\tilde{H}[\lambda_{S_y}(\lambda_{S_x}\lambda_x)^2 + \lambda_{S_x}(\lambda_{S_y}\lambda_y)^2]\}}$, where λ_{S_x} snd λ_{S_y} denote the eigenvalues of S_x and S_y, respectively. Hence, as long as the neglected terms in (59) do not become important with respect to stability, it is possible to reduce the modulus of the maximum eigenvalue to any desired magnitude.

3. NUMERICAL ILLUSTRATIONS

To illustrate the foregoing theory, we will give examples of the stabilization for linear and non-linear problems. Furthermore, an application to the shallow-water equations will be shown.

3.1. A linear problem

The linear problem is defined by

$$\begin{aligned} u_t &= u_x - 16\pi/L\cos(32\pi x/L), \quad 0 < t < T, \quad 0 < x < L \\ u(x, 0) &= 0{\cdot}5\sin(2\pi x/L) + 0{\cdot}5\sin(32\pi x/L) \\ u(0, t) &= u(L, t) \end{aligned} \tag{62}$$

where $L = 100$. The exact solution of this problem is

$$u(x, t) = 0{\cdot}5\sin(2\pi(x + t)/L) + 0{\cdot}5\sin(32\pi x/L) \tag{63}$$

Hence, the solution consists of a non-stationary part, which is slowly varying both in the time and in the space variable, and a stationary part which varies rapidly in the space variable only.

Therefore, the numerical approximation of the stationary part needs a finer space mesh than the non-stationary part. This fine space mesh does, when no smoothing is used, severely restrict the time step. Here, we will give the accuracy results for five methods which all have the same semi-discretization (12). The basic time integrator we use is the classical fourth order Runge–Kutta method.[5] This method, which is used by various others,[6,10,11] is conditionally stable for hyperbolic partial differential equations. The imaginary stability boundary of this method is $C = 2\sqrt{2}$. The methods are:

Table I. Numerical results using smoothing operators with $T = 2{\cdot}8 \times 128$, $h = L/N$

Δt	RK4	RK4E1	Correct digits, $N = 384$ RK4E2	RK4I	CN	Correct digits on coarser grids RK4	N
0·7	2·0	2·0(0)	2·0(0)	2·0(0)	1·9	2·0	384
1·4	-	2·1(1)		2·0(1)	1·7	1·6	192
1·866	-		2·0(1)				
2.8	-	1·9(2)		1·7(1)	1·4	0·9	96
3·733	-		1·7(2)				
5·6	-	1·4(3)		1·3(1)	0·9	0·3	48
7·466	-		1·2(3)				
11·2	-	0·8(4)		0·7(1)	0·4	−0·1	24
14·933	-		0·6(4)				

RK4 the classical Runge–Kutta method without smoothing

RK4E1 the classical Runge–Kutta method with smoothing operator (33), where $\mu_k = \frac{1}{2}$ and $k_0 = 1$, $n = [1 + \log_2(\Delta t/(2\sqrt{2h}))]$

RK4E2 the classical Runge–Kutta method with smoothing operator (33), where $\mu_k = \frac{1}{4}$ and $k_0 = 2$, $n = [2 + \log_2(\Delta t/(2\sqrt{2h}))]$ for $1 < \Delta t/(2\sqrt{2h}) < \frac{3}{2}$ and $n = [2 + \log_2 (\frac{4}{9}\sqrt{3}\Delta t/(2\sqrt{2h}))]$ for $\Delta t/(2\sqrt{2h}) > \frac{3}{2}$

RK4I the classical Runge–Kutta method with the implicit smoothing operator (47), where $\mu = \frac{1}{4}\Delta t^2/(2\sqrt{2h})^2$ for $\Delta t/(2\sqrt{2h}) > 1$, and

CN the Crank–Nicolson method.

The brackets, [ ], in the expressions for the determination of n denote the entier function. Furthermore, no smoothing is performed for $\Delta t/(2\sqrt{2h}) < 1$ in RK4E1, RK4E2 and RK4I. In Table I we give the number of correct digits produced by these integration methods, i.e. the $-\log_{10}(|\textit{maximum error}|)$, and in parentheses the number of smoothings.

The main part of the table presents the results on a grid with 384 grid points. For reference, we also added results of the RK4 method on coarser grids using the corresponding maximum allowed time steps. These time steps are given in the first column. The hyphens in the column of RK4 denote that the method is unstable for the corresponding time step. The results of RK4E1 and RK4E2 are given for time steps Δt which are the maximum allowed for the corresponding number of smoothings. For RK4E1, this results in a doubling of the allowed time step, each time a new operator is applied. If in RK4E2 the first operator of the product sequence is applied, a factor $\frac{3}{2}\sqrt{3}$ is gained (see (36) and (38)). Thereafter, as with RK4E1, a factor two is gained each time a new smoothing operator of the sequence is applied. RK4I and CN were applied using the same step sizes as RK4E1. Because n is an integer, the increase of the maximum allowed time step proceeds in a discreet way. However, for accuracy reasons it may be desirable to have a smooth increase of the time step as the right-hand side is smoothed more and more. Without going into details, we mention that this can be established by varying the coefficient μ_k of the last smoothing operator in the product sequence (34).

The results on the fine grid ($N = 384$) develop in the same way for all methods when the time step increases: at first, the number of correct digits changes slightly; then, when the time step becomes larger than about 3·5, the number of correct digits decreases rapidly. This can be understood by the following reasoning. The error due to the stationary part of the solution is independent of the time step. For this problem, this error is rather large because of the large space derivatives of the stationary part of the solution. Of course, the error due to the non-stationary part is dependent on

the time step. Hence, for a certain time step the error due to the non-stationary part becomes larger than that due to the stationary part of the solution. This time step is about 3·5 for this problem.

The results on coarser grids clearly show the need for a calculation on the fine grid, because the number of correct digits rapidly decreases on coarser grids. This error is due to the stationary part of the solution.

3.2. *A non-linear problem*

In this section, we will use the stabilization technique for a non-linear equation. The problem is given by

$$u_t = uu_x + g(x,t), \quad 0 < t < T, \quad 0 < x < L \tag{64}$$

where $L = 100$. The forcing function g is chosen such that we have a solution consisting of a part which is slowly varying in both the time and the space variables, and a part which varies relatively rapidly in the space variable only. The solution is given by

$$u(x,t) = 0{\cdot}5\sin(2\pi(x+t)/L) + 0{\cdot}5\sin(8\pi x/L) \tag{65}$$

Hence, the function g follows to be

$$\begin{aligned} g(x,t) = 2\pi/L\{&0{\cdot}5\cos(2\pi(x+t)/L) - [0{\cdot}5\sin(2\pi(x+t)/L) + 0{\cdot}5\sin(8\pi x/L)] \\ &\times [0{\cdot}5\cos(2\pi(x+t)/L) + 2\cos(8\pi x/L)]\} \end{aligned} \tag{66}$$

The initial condition is taken from the exact solution (65).

We discretized the non-linear term uu_x by

$$\{(u_{j+1} + 2u_j + u_{j-1})/4\}(u_{j+1} - u_{j-1})/(2h) \tag{67}$$

Owing to the non-linear nature of equation (3), almost any time integration will become unstable after a certain time period. In our experiments, this discretization (67) performed quite well. For more details on discretizations for non-linear problems we refer to References 12–15. For the time discretization, we applied the same time integrators as in Section 3.1, except for the CN method. This method is modified to

$$\begin{aligned} u_j^{n+1} = u_j^n &+ \tfrac{1}{2}\Delta t[((u_{j+1}^n + 2u_j^n + u_{j-1}^n)/4)(u_{j+1}^{n+1} - u_{j-1}^{n+1})/(2h)] \\ &+ \tfrac{1}{2}\Delta t[((u_{j+1}^{n+1} + 2u_j^{n+1} + u_{j-1}^{n+1})/4)(u_{j+1}^n - u_{j-1}^n)/(2h)] \\ &+ \Delta t g(x_j, t + \Delta t/2) \end{aligned} \tag{68}$$

This modification is linearly implicit and still second order in time. In the following this method is called MCN. Table II gives the results in the same form as in Table I

Globally, we observe the same effect for the explicit methods as in the previous section: at first the error of the time stepping is negligible with respect to that of the space discretization; then, when the time step becomes larger than about 5, the error due to the time stepping becomes significant. Furthermore, we find that the application of the smoothing operators gives at first a slight increase of the number of correct digits. This is possibly due to an annihilation of errors. The MCN method performs relatively poorly for this problem, which is caused by the larger error constants of its time discretization.

The implicit smoothing operator is of course more expensive than one explicit smoothing operator of the product sequence. However, as the time step increases, we need more and more applications of the explicit smoothing operators, whereas the implicit smoothing operator needs to be applied only once. Hence, after a certain number of applications of explicit smoothing operators,

Table II. Numerical results using smoothing operators with $T = 128 \times 8$, $h = L/N$

Δt	RK4	RK4E1	Correct digits, $N = 384$ RK4E2	RK4I	MCN	Correct digits on coarser grids RK4	N
0·8	2·0	2·0(0)	2·0(0)	2·0(0)	1·8	2·0	384
1·6	-	2·6(1)		2·4(1)	1·2	1·4	192
2·1	-		2·4(1)				
3·2	-	2·3(2)		2·1(1)	0·2	1·0	96
4·2	-		2·1(2)				
6·4	-	1·8(3)		1·6(1)		0·6	48
8·4	-		1·6(3)				
12·8	-	1·3(4)		0·9(1)		0·0	24
16·8	-		1·3(4)				

explicit smoothing becomes more expensive than implicit smoothing. On a vector computer this number is of course much larger, because the explicit smoothing operators vectorize very well, which is not the case for the implicit smoothing operator.

3.3. *The shallow-water equations*

In this section, we give results of a computation with the shallow-water equations where smoothing is used. These computations are performed on the CYBER 205. On such a vector computer, explicit methods are to be preferred, because they allow a high degree of vectorization. The application of the smoothing operators is such as described in Section 2.5. In this computation, the operator (34) is applied with coefficients depending on k and n, i.e.

$$\mu_k = \tfrac{1}{4}(1 - 2^{-(n+1-k)}) \tag{69}$$

For this choice the eigenvalues of (38) are positive, which appeared important for initial-boundary value problems in order to have a stable integration. With this choice, we found in experiments, that the factor 0·77 in (38) can be replaced by 1. In this computation again the classical Runge–Kutta method is used. For further details we refer to Reference 8. These results will show the relevance of smoothing for flow computations.

The test problem is a square basin, which has sides of length 5 km (see Figure 1). In the middle of the basin is a bump.

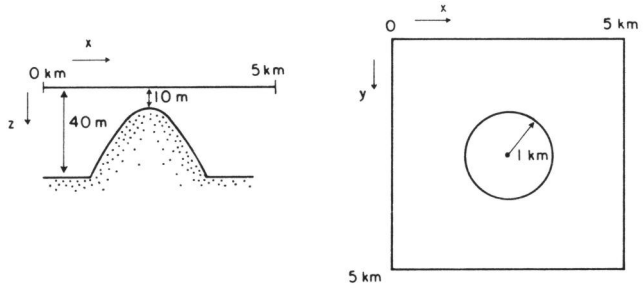


Figure 1. Geometry of a square basin with bump.

The bottom profile $h(x, y)$ is given by

$$h(x, y) = \begin{cases} 40 - 30\cos(\pi r/2)m & \text{for } r < 1 \\ 40m, & \text{elsewhere} \end{cases} \tag{70}$$

where

$$r = \sqrt{[(x - 2{\cdot}5 \times 10^3)^2 + (y - 2{\cdot}5 \times 10^3)^2]}/1000$$

At the left and right boundaries, the elevation ζ is prescribed

$$\begin{aligned} \zeta(0, y, t) &= -\sin(\omega t), \\ \zeta(5{\cdot}0 \times 10^3, y, t) &= -\sin(\omega t - \phi), \end{aligned} \tag{71}$$

where $0 < y < 5{\cdot}0 \times 10^3$ m and

$$\begin{aligned} \omega &= 2\pi/(12 \times 3600)\,\mathrm{s}^{-1} \\ \phi &= 2\pi 5/600 \end{aligned} \tag{72}$$

At the upper and lower boundaries, at $y = 0$ and $y = 5{\cdot}0 \times 10^3$ m, respectively, the normal velocity component v is zero. Furthermore, the constants in equation (59) are chosen to be

$$\nu = 10\,\mathrm{m}^2/\mathrm{s}, \quad C_z = 4 \times 10^{-3}$$

We integrated 15 hours physically with various time steps (see Table III). The calculations were performed on a 24×24 grid. At the end of each integration the solution was compared with a reference solution computed on a finer grid (96×96). The results are given in significant digits of the v-component. In this case, a root-mean-square error is used, defined by

$$\mathrm{Sd}_2 = -\log_{10}(|v - v_{\mathrm{ref}}|_2/|v_{\mathrm{ref}} - \bar{v}_{\mathrm{ref}}|_2)$$

where

$$\begin{aligned} \bar{v}_{\mathrm{ref}} &= (\sum_i v_i)/(24 \times 24) \\ |v|_2 &= \sqrt{(\sum_i v_i^2)}/(24 \times 24) \end{aligned} \tag{73}$$

in which the summation is over all grid points. The results are given in Table III.

In this table, the number in parentheses denotes the number of applied smoothing operators. In the first column again the time step is given; it increases downwards with a factor two. In the second column the computation times are given and in the last column the numbers of significant digits are given with, in parentheses, the number of smoothings. The time step $\Delta t = 8$ is the maximum time step without smoothing. The general picture is comparable with the previous cases. At first, the number of significant digits remains constant as the time step increases and then, when the time step becomes greater than 32 s, the error due to the time step becomes dominant.

The computation times show a significant reduction when smoothing is used. Furthermore, they

Table III. Significant digits for the shallow-water equations

	RK4E2	
Δt	Computation time	Sd_2
8	45	2·1(0)
16	26	2·1(1)
32	15	2·1(2)
64	9	1·6(3)

contain information about the overhead of the smoothing, because without smoothing the computation time should decrease by a factor two downwards. The overhead of one smoothing is in this case about $\frac{1}{6}$ of one right-hand side evaluation.

4. CONCLUSIONS

In Section 2, we have set up the theory for the stabilization of explicit methods for purely initial-value problems. For some numerical examples it was shown that the predicted reduction of the spectral radius is correct, even in non-linear partial differential equations. Furthermore, a significant decrease of the computation time was found (see Table III).

Our experiences are that the described stabilization is easy to implement. In fact, by its simplicity, it can be added easily to an existing program. Moreover, we think that the technique can be applied to a large variety of problems, even to problems with non-smooth solutions.[6]

APPENDIX: SMOOTHING OPERATORS OCCURRING IN OTHER TIME INTEGRATORS

In Section 2.1 we have rewritten the implicit backward Euler integrator to an explicit method in which a smoothing operator occurs. We will now show that the backward Euler method also can be considered, for problem $\{(9), (12)\}$, as a two-stage first-order Runge–Kutta scheme where an implicit smoother of the form described in Section 2.5 occurs. Furthermore, applying the well-known Crank–Nicolson method to problem $\{(9), (12)\}$, this method appears to be a second-order two-stage Runge–Kutta scheme, where the same implicit smoothing operator occurs. We rewrite (16) as

$$U_j^{n+1} = U_j^n + \Delta t\{(I - \Delta t^2 D^2)^{-1}(I + \Delta t D)\mathbf{F}(\mathbf{U}^n)\}_j \tag{74}$$

The term $(I - \Delta t^2 D^2)^{-1}$ is an implicit smoothing operator similar to the one described in Section 2.5. Furthermore, if this implicit smoothing operator is omitted from (74), then there remains a two-stage first-order Runge–Kutta scheme, applied to the linear problem $\{(9), (12)\}$. Proceeding in the same way for an application of Crank–Nicolson to $\xi\{(9), (12)\}$ we have

$$U_j^{n+1} = U_j^n + \Delta t\{(I - \Delta t^2 D^2/4)^{-1}(I + \Delta t D/2)\mathbf{F}(\mathbf{U}^n)\}_j \tag{75}$$

Here, again the implicit smoothing operator occurs. Omitting this operator, a two-stage second-order Runge–Kutta method, applied to $\{(9), (12)\}$ remains. However, this scheme, without smoothing operator, is unstable for hyperbolic problems. Hence, by smoothing it is possible to stabilize a method that otherwise would be unstable for all Δt.

ACKNOWLEDGEMENTS

These investigations were supported by the Netherlands Foundation for Technical Research (STW), future Technical Science Branch Division of the Netherlands Organization for the Advancement of Pure Research (ZWO). The experiments were done on a CYBER 750 and CYBER 205 at the expense of the Centre for Mathematics and Computer Science (CWI) and the Control Data Corporation (CDC).

REFERENCES

1. R. Courant, and D. Hilbert, *Methods of Mathematical Physics*, Interscience Publishers, 1962.
2. E. E. Rosinger, 'Nonlinear equivalence, reduction of PDEs to ODEs and fast convergent numerical methods', *Research Notes in Mathematics 77*, Pitman Advanced Publishing Program, Boston-London-Melbourne, 1982.

3. F. Shuman, 'Numerical methods in weather prediction: II, smoothing and filtering', *Monthly Weather Review*, **85,** 357–361 (1957).
4. R. D. Richtmyer, and K. W. Morton, *Difference Methods for Initial Value Problems*, Interscience Publishers, Wiley, New York, 1967.
5. J. D. Lambert, *Computational Methods in Ordinary Differential Equations*, Wiley, New York, 1973.
6. A. Jameson and D. Mavriplis, 'Finite volume solution of the two-dimensional Euler equations on a regular triangular mesh', *AIAA 23rd Aerospace Sciences Meeting*, AIAA-85-0435, Nevada, 1985.
7. J. C. Wilson, 'Stability of Richtmyer type difference schemes in any finite number of space variables and their comparison with multistep strange schemes', *J. Inst. Maths Applics*, **10,** 238–257 (1972).
8. F. W. Wubs, P. J. van der Houwen and B. P. Sommeijer, 'On the construction of optimal smoothing operators for stabilizing explicit time integrators in PDE's', *Report NM86*, Centre for mathematics and Computer Science, Amsterdam, 1986.
9. P. Lancaster, *Theory of Matrices*, Academic Press, New York and London, 1969.
10. P. J. van der Houwen, *Construction of Integration Formulas for Initial Value Problems*, North-Holland Publishing Company, Amsterdam, 1977.
11. N. Praagman, 'Numerical solution of the shallow-water equations by a finite element method', *Thesis*, TH Delft, 1979.
12. A. Grammeltvedt, 'A survey of finite-difference schemes for the Primitive equations for a barotropic fluid', *Monthly Weather Review*, **97,** (1969).
13. J. G. Verwer and K. Dekker, 'Step-by-step stability in the numerical solution of partial differential equations', *Report NW 161/83*, Mathematical Centre, Amsterdam, 1983.
14. A. Arakawa, 'Computational design for long-term numerical integration of the equations of fluid motion: 1. Two-dimensional incompressible flow', *Journal of Computational Physics*, **1,** (1), (1966).
15. A. Arakawa, and V. R. Lamb, 'The UCLA general circulation model', *Methods in Computational Physics*, **17,** (1977).

Analysis of Smoothing Operators in the Solution of Partial Differential Equations by Explicit Difference Schemes

P.J. van der Houwen, B.P. Sommeijer, F.W. Wubs
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

A smoothing technique for the "preconditioning" of the right-hand side of semi-discrete partial differential equations is analysed. For a parabolic and a hyperbolic model problem optimal smoothing matrices are constructed which result in a substantial amplification of the maximal stable integration step of arbitrary explicit time integrators when applied to the smoothed problem. This smoothing procedure is illustrated by integrating both linear and nonlinear parabolic and hyperbolic problems. The results show that the stability behaviour is comparable with that of the Crank-Nicholson method; furthermore, if the problem belongs to the problem class in which the time derivative of the solution is a smooth function of the space variables, then the accuracy is also comparable with that of the Crank-Nicholson method.

1980 Mathematics Subject Classification: Primary: 65M10, Secondary: 65M20
1982 CR Categories: 5.17
Key Words & Phrases: numerical analysis, initial boundary value problems in partial differential equations, method of lines, explicit integration methods, smoothing, stability.
Note: These investigations were supported by the Netherlands Foundation for Technical Research (STW), future Technical Science Branch Devision of the Netherlands Organization for the Advancement of Pure Research (ZWO).
Note: This report will be submitted for publication elsewhere.

1. Introduction

In a number of papers (cf. e.g., [2] and [4]), it has been observed that many initial-boundary value problems for partial differential equations of the form

$$\frac{\partial \mathbf{u}}{\partial t}(t,\mathbf{x}) = D(t,\mathbf{x},\mathbf{u}(t,\mathbf{x})) \tag{1.1}$$

possess the property that the right-hand side $D(t,\mathbf{x},\mathbf{u})$ is a smooth function of the space variable $\mathbf{x}$ if the exact solution of the initial-value problem is substituted, even when the exact solution has large space derivatives. Here, D may be a (nonlinear) differential operator of parabolic or hyperbolic type.

The situation described above arises in cases where the solution of the initial-boundary value problem tends to a steady state solution:

$$\mathbf{u}(t,\mathbf{x}) \to \mathbf{r}(\mathbf{x}) + \mathbf{s}(t,\mathbf{x}) \quad \text{as} \quad t \to \infty, \tag{1.2}$$

where $\mathbf{r}(\mathbf{x})$ is a *rapidly* varying function of $\mathbf{x}$ and $\mathbf{s}(t,\mathbf{x})$ is a *smooth* function of $(t,\mathbf{x})$. Evidently,

$$D(t,\mathbf{x},\mathbf{r}(\mathbf{x})+\mathbf{s}(t,\mathbf{x})) \to \frac{\partial \mathbf{s}}{\partial t}(t,\mathbf{x}),$$

so that the right-hand side becomes a smooth function of $\mathbf{x}$ as $t \to \infty$ (see the examples in Section 4).

For such problems it was proposed in, e.g., [2] and [4] to smooth the right-hand side of the equation (1.1) with respect to $\mathbf{x}$, before applying a numerical integration method. The effect of smoothing the right-hand side of (1.1) becomes apparent when the space variable $\mathbf{x}$ and the differential operator D in (1.1) are discretized: the resulting system of ordinary differential equations is *better conditioned* in the sense that the spectral radius of the Jacobian matrix of this system reduces considerably in magnitude by the smoothing process. It is well known that the usually large spectral radius of semi-

Report NM-R8617
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

discrete partial differential equations makes *explicit* integration methods unattractive for solving these systems, because of the rather restrictive stability condition. However, if smoothing reduces the spectral radius sufficiently in magnitude, then explicit time integration methods become of interest.

The price we have to pay for the "preconditioning" of the system of semi-discrete equations, is a possible drop in accuracy of the space discretization. To make this more clear, we consider the quasi linear equation

$$\frac{\partial \mathbf{u}}{\partial t}(t,\mathbf{x}) = A(\mathbf{u}(t,\mathbf{x}))L\mathbf{u}(t,\mathbf{x})+\mathbf{g}(t,\mathbf{x}), \tag{1.3}$$

where L is a *linear* differential operator with respect to $\mathbf{x}$, and A and $\mathbf{g}$ are given functions; let A_Δ and L_Δ represent discretizations of A and L with Δ characterizing the accuracy of the discretization, and let S_Δ denote a (linear) smoothing operator. For example, in one space variable x, we may think of

$$L = \frac{\partial}{\partial x},\ L_\Delta \mathbf{u}(t,x) = \frac{1}{2\Delta}(E_\Delta - E_\Delta^{-1})\mathbf{u}(t,x),\ \ S_\Delta \mathbf{u}(t,x) = \frac{1}{2}(E_\Delta + E_\Delta^{-1})\mathbf{u}(t,x),$$

where E_Δ is the forward shift operator defined by $E_\Delta \mathbf{u}(t,x) := \mathbf{u}(t,x+\Delta)$. Instead of solving (1.3), we try to solve the smoothed, semidiscrete equation

$$\frac{\partial \mathbf{u}}{\partial t}(t,\mathbf{x}) = S_\Delta A_\Delta(\mathbf{u}(t,\mathbf{x}))L_\Delta \mathbf{u}(t,\mathbf{x})+S_\Delta \mathbf{g}(t,\mathbf{x}). \tag{1.4}$$

Let $\mathbf{v}(t,\mathbf{x})$ and $\mathbf{w}(t,\mathbf{x})$ denote the solutions of the initial-boundary value problem for the equations (1.3) and (1.4), respectively. Then, it is easily verified that the difference $\mathbf{v}-\mathbf{w}$ satisfies the equation

$$\begin{aligned}\frac{\partial}{\partial t}(\mathbf{v}-\mathbf{w}) &= S_\Delta A_\Delta(\mathbf{w})L_\Delta(\mathbf{v}-\mathbf{w})+S_\Delta[A(\mathbf{v})L - A_\Delta(\mathbf{w})L_\Delta]\mathbf{v} \\ &\quad + [I - S_\Delta]\,[A(\mathbf{v})L\mathbf{v}+\mathbf{g}].\end{aligned} \tag{1.5}$$

This "error equation" shows the effect of the space discretization and of the smoothing operator on the accuracy by which $\mathbf{w}$ approximates $\mathbf{v}$. The second term in the right-hand side of (1.5) represents the (smoothed) *space discretization error*, whereas the last term represents the *smoothing error*. Evidently, the smoothing error vanishes if $S_\Delta = I$ (no smoothing), it is small if $A(\mathbf{v})L\mathbf{v}+\mathbf{g}$ is a smooth function of $\mathbf{x}$, and it hardly affects the accuracy of $\mathbf{w}$ if $A(\mathbf{v})L\mathbf{v}+\mathbf{g}$ is much smoother in $\mathbf{x}$ than $\mathbf{v}$.

Thus, we expect that the introduction of smoothing operators into the right-hand side of the partial differential equation (1.1) will not severely decrease the accuracy provided that the exact solution of (1.1) varies much more rapidly with $\mathbf{x}$ than its time derivative does.

In [4] a few smoothing operators were tested and shown to have the expected effect. In this paper, we analyse smoothing operators more systematically, and we derive a family of optimal operators of second-order for a parabolic and a hyperbolic model problem. In addition, a family of fourth-order smoothing operators are constructed which are not optimal, but still result in a considerable reduction of the spectral radius of the Jacobian matrix.

The various smoothing operators are tested by integrating a few initial-value problems of parabolic and hyperbolic type, both linear and nonlinear. The results obtained clearly show that the two-stage *explicit* Runge-Kutta time integrators used in our experiments, when combined with a suitable smoothing operator, exhibit a stability behaviour which is comparable with that of the *(implicit)* Crank-Nicholson method, while the accuracy is hardly lower. Since a *smoothed* Runge-Kutta step is "cheaper" than a Crank-Nicholson step, particularly in the case of *nonlinear* problems, we conclude that, for the class of problems described above, explicit Runge-Kutta methods equipped with the right smoothing operators are preferable to the Crank-Nicholson method.

2. Smoothing operators

By restricting the semi-discrete (partial) differential equation (1.4) to a grid Ω_Δ in the **x**-space, we are led to a system of ordinary differential equations (method of lines). This system will be denoted by

$$\frac{d\mathbf{y}(t)}{dt} = S\,\mathbf{f}(t, \mathbf{y}(t)), \quad t \geqslant t_0, \tag{2.1}$$

where the matrix S corresponds to the smoothing operator S_Δ introduced in (1.4). More generally, by smoothing the right-hand side of (1.1) and by discretizing **x** and D, we will always obtain a system of the form (2.1).

2.1. Relaxing the stability condition by smoothing

If the system (2.1) is integrated by an *explicit* time integrator we are faced with a stability condition on the time step Δt of the form

$$\Delta t \leqslant \frac{\beta}{\rho(SJ)}, \quad J := \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t, \mathbf{y}(t)), \tag{2.2}$$

where $\rho(SJ)$ denotes the spectral radius of the matrix SJ, and β is a constant (the so-called *stability boundary*) completely determined by the time integrator.

Since the stability boundary of explicit methods is relatively small and $\rho(J)$ usually extremely large, the condition (2.2) may be extremely restrictive if no smoothing is applied (i.e., $S = I$). This may force the method to take steps Δt that are much smaller than accuracy would require. By an appropriate choice of the smoothing matrix S we can reduce the magnitude of $\rho(SJ)$ considerably.

In general, it is too ambitious to derive optimal smoothing matrices for an arbitrary Jacobian matrix J. Therefore, we shall consider the optimization problem for two model problems which characterize, respectively, a parabolic and a hyperbolic equation. First, however, we consider the order of accuracy of the smoothing operator, that is we require

$$S = I + O(\Delta^p) \tag{2.3}$$

as the spatial grid Ω_Δ is refined.

2.2. The order of accuracy of smoothing operators

Let the vector **v** have components $v^{(j)}$ and define the shift operator E by

$$Ev^{(j)} := v^{(j+1)}. \tag{2.4}$$

Let $Q_k(z)$ be a polynomial of degree k in z with $Q_k(1) = 1$. Then we may consider smoothing matrices S of the form

$$S\mathbf{v} = \mathbf{u} := (\tfrac{1}{2}[Q_k(E) + Q_k(E^{-1})]v^{(j)}), \quad Q_k(1) = 1. \tag{2.5}$$

We shall call this matrix a *smoothing matrix* or *smoothing operator* of *degree k*.

This operator should be sufficiently close to the identity operator I. In order to define the order of the smoothing operator (2.5) we apply S to the test vector $\mathbf{v} = (v^{(j)}) := (w(j\Delta x))$, where $w(x)$ is a sufficiently differentiable function of x. We find

$$\begin{aligned} S\mathbf{v} &= (\tfrac{1}{2}[Q_k(E) + Q_k(E^{-1})]w(j\Delta x)) \\ &= (\tfrac{1}{2}[Q_k(e^{\Delta x \frac{d}{dx}}) + Q_k(e^{-\Delta x \frac{d}{dx}})]w(j\Delta x)) \\ &= ([Q_k(1) + \tfrac{1}{2}(Q'_k(1) + Q''_k(1))\Delta^2 x \frac{d^2}{dx^2} + O(\Delta^4 x)]w(j\Delta x)). \end{aligned}$$

DEFINITION 2.1. The smoothing operator (2.5) is said to be *of order p* if for all vectors $\mathbf{w} = (w(j\Delta x))$ with $w \in C^p$ we have

$$S\mathbf{w} = \mathbf{w} + O(\Delta^p x) \text{ as } \Delta x \to 0. \square$$

The following theorem is easily proved:

THEOREM 2.1. *The smoothing operator (2.5) is at least of order* $p = 2$; *it is of order* $p = 4$ *if* $Q_k(z)$ *satisfies* $Q'_k(1) + Q''_k(1) = 0$. $\square$

EXAMPLE 2.1. A two-parameter family of *second-order* smoothing operators is generated by the polynomial

$$Q_2(z) = 1 - q_1 - q_2 + q_1 z + q_2 z^2.$$

The order can be raised to four if we choose $q_1 = -4q_2$. We observe that fourth-order smoothing operators always require $k \geqslant 2$. $\square$

EXAMPLE 2.2. Let S be defined by

$$S\mathbf{v} := (\frac{1}{16}(E + 2 + E^{-1})(E^2 + 2 + E^{-2})v^{(j)}).$$

It is easily verified that this operator can be represented in the form (2.5) with

$$Q_3(z) = \frac{1}{4} + \frac{3}{8}z + \frac{1}{4}z^2 + \frac{1}{8}z^3.$$

Since $Q_3(1) = 1$, this smoothing operator is *second-order* accurate. $\square$

3. CONSTRUCTION OF OPTIMAL SMOOTHING OPERATORS

In order to investigate the operator S defined by (2.5) we will use the test vectors

$$\mathbf{e} = (e^{(j)}), \quad e^{(j)} := \exp(i\omega j\Delta x), \tag{3.1}$$

where $\omega \in \mathbb{R}$ and Δx is the space discretization parameter.

DEFINITION 3.1. Let $C(z)$ be the polynomial

$$C(z) = \sum_{l=0}^{r} c_l z^l.$$

Then we associate to C the polynomial $\hat{C}$ defined by

$$\hat{C}(z) := \sum_{l=0}^{r} c_l T_l(z), \quad T_l(z) := \cos(l \arccos z).$$

THEOREM 3.1. *The smoothing operator* S *satisfies the eigenvalue equation*

$$S\mathbf{e} = \hat{Q}_k(\zeta)\mathbf{e}, \quad \zeta := \cos(\omega\Delta x).$$

PROOF. On substitution of $\mathbf{e}$ into (2.5) we obtain

$$\begin{aligned} S\mathbf{e} &= \tfrac{1}{2}[Q_k(e^{i\omega\Delta x}) + Q_k(e^{-i\omega\Delta x})]\,\mathbf{e} \\ &= \frac{1}{2}\sum_{l=o}^{k} q_l(e^{il\omega\Delta x} + e^{-il\omega\Delta x})\mathbf{e} \\ &= \sum_{l=0}^{k} q_l \cos(l\omega\Delta x)\mathbf{e} = \sum_{l=0}^{k} q_l T_l(\zeta)\mathbf{e}. \square \end{aligned}$$

Thus, the test vector $\mathbf{e}$ is an eigenvector of S with eigenvalue $\hat{Q}_k(\zeta)$. The behaviour of the polynomial $\hat{Q}_k(z)$ on the interval $[-1,1]$ determines the properties of the smoothing operator S (notice that $-1 \leqslant \zeta \leqslant 1$). For instance, if $\hat{Q}_k(z)$ is small in magnitude for $z \to -1$, then S will damp the high frequencies in the Fourier expansion of the vector $\mathbf{v} = (w(j\Delta x))$.

In the actual derivation of the smoothing operator S from a given polynomial $\hat{Q}_k(z)$ the following corollary of Theorem 3.1 is often convenient.

COROLLARY 3.1. *Let $\hat{Q}_k(z)$ be a polynomial expression in terms of the functions $T_0(z), T_1(z), \ldots, T_\kappa(z)$:*

$$\hat{Q}_k(z) = \mathbb{S}(T_0(z), \ldots, T_\kappa(z)). \tag{3.2a}$$

Then the generated smoothing operator is given by

$$S\mathbf{v} = (\mathbb{S}(\frac{E^0+E^0}{2}, \ldots, \frac{E^\kappa+E^{-\kappa}}{2})v^{(j)}). \tag{3.2b}$$

PROOF. From Theorem 3.1 it follows that the smoothing operator $\tilde{S}$ generated by (3.2a), has the eigenvalues

$$\hat{Q}_k(\zeta) = \mathbb{S}(T_0(\zeta), \ldots, T_\kappa(\zeta)), \ \zeta = \cos(\omega\Delta x).$$

On the other hand, because $T_j(\zeta)$ is an eigenvalue of $(E^j + E^{-j})/2$, it follows from (3.2b) that the operator S has the same eigenvalues. Since S and $\tilde{S}$ are both polynomial operators in E and E^{-1} with identical eigenvalues, they are necessarily identical. □

EXAMPLE 3.1. Suppose that

$$\hat{Q}_6(z) = 2T_2(z)T_1(z) - T_3^2(z).$$

Then, S is defined by

$$S\mathbf{v} = ([\tfrac{1}{2}(E^2+E^{-2})(E+E^{-1}) - \tfrac{1}{4}(E^3+E^{-3})^2]v^{(j)}). \ \square$$

The following result is similarly proved by means of Theorem 3.1.:

COROLLARY 3.2. *Let the polynomials $\hat{Q}^{(j)}(z)$ generate smoothing operators $S^{(j)}$, and let a and b be scalars. Then the polynomial*

$$\hat{Q}(z) := a\hat{Q}^{(1)}(z) + b\hat{Q}^{(2)}(z)\hat{Q}^{(3)}(z)$$

generates the smoothing operator

$$S := aS^{(1)} + bS^{(2)}S^{(3)}. \ \square$$

The next theorem expresses the order conditions in terms of the polynomial $\hat{Q}_k(z)$.

THEOREM 3.2.(a) *The smoothing operator generated by $\hat{Q}_k(z)$ is of second-order if $\hat{Q}_k(1) = 1$, and of fourth-order if, in addition, $\hat{Q}'_k(1) = 0$.*
(b) *If $\hat{Q}_k(1) = 1$ and $\hat{Q}'_k(1) \neq 0$, then the polynomial*

$$\hat{P}_{2k}(z) := 1 - \alpha + \alpha\hat{Q}_k(z)[2 - \hat{Q}_k(z)]$$

generates a fourth-order smoothing operator for all values of α.

PROOF. (a) Since $T_l(1) = 1$ and $T'_l(1) = l^2$ we have

$$Q_k(1) = \sum_{l=0}^{k} q_l = \sum_{l=0}^{k} q_l T_l(1) = \hat{Q}_k(1)$$

and

$$Q'_k(1)+Q''_k(1) = \sum_{l=0}^{k} q_l[l+l(l-1)] = \sum_{l=0}^{k} q_l l^2$$
$$= \sum_{l=0}^{k} q_l T'_l(1) = \hat{Q}'_k(1).$$

From these relations and Theorem 2.1 assertion (a) of the theorem easily follows.
(b) The polynomial $\hat{P}_{2k}(z)$ is easily shown to satisfy for all α the conditions for fourth-order accuracy stated in (a). □

Once the polynomial $\hat{Q}_k$ has been specified, the smoothing operator S is easily found, either by using Definition 3.1 (to obtain Q_k) and formula (2.5) (to obtain S), or by using the above Corollaries 3.1 and 3.2.

In order to construct an effective operator S, in the sense that $\rho(SJ)$ is substantially smaller than $\rho(J)$, we need some additional information on the spectrum of J. We shall distinguish Jacobian matrices with *negative* eigenvalues arising in *parabolic* equations and *imaginary* eigenvalues arising in *hyperbolic* equations.

3.1. Smoothing of parabolic problems

If symmetric space discretizations are used in parabolic problems then J is usually of the form

$$J\mathbf{v} = (\tfrac{1}{2}[K(E)+K(E^{-1})]v^{(j)}), \tag{3.3a}$$

where K is a polynomial. In the same manner as we associated to Q_k the polynomial $\hat{Q}_k$ (cf. Theorem 3.1), we can associate to K the polynomial $\hat{K}$, to obtain the eigenvalue equation

$$J\mathbf{e} = \hat{K}(\zeta)\mathbf{e},\ \mathbf{e} := (e^{ij\omega\Delta x}),\ \zeta := \cos(\omega\Delta x). \tag{3.3b}$$

EXAMPLE 3.2. Consider the *parabolic model problem*

$$u_t = u_{xx} + g(x,t).$$

The standard three-point discretization leads to a system of differential equations of which the j-th equation reads:

$$\frac{dy^{(j)}}{dt} = \frac{1}{\Delta^2 x}[E-2+E^{-1}]y^{(j)}+g^{(j)}(t);$$

it is easily seen that the matrix J can be characterized by the polynomial

$$K(z) = -\frac{2}{\Delta^2 x}(1-z).$$

The polynomial $\hat{K}(z)$ turns out to be identical with $K(z)$. □

EXAMPLE 3.3. If the equation above is discretized by the standard fourth-order five-point discretization we obtain the polynomial

$$K(z) = -\frac{1}{6\Delta^2 x}(z^2-16z+15)$$

and

$$\hat{K}(z) = -\frac{1}{3\Delta^2 x}(z^2-8z+7) = -\frac{1}{3\Delta^2 x}(z-1)(z-7).$$ □

Let us return to our problem of minimizing $\rho(SJ)$ occurring in the stability condition (2.2). It follows from Theorem 3.1 and (3.3) that

$$\rho(SJ) = \max_{-1\leqslant\zeta\leqslant 1} |\hat{Q}_k(\zeta)\hat{K}(\zeta)|. \tag{3.4}$$

Thus, the right-hand side has to be minimized taking into account the order condition in Theorem 3.1. Moreover, the polynomial $\hat{Q}_k$ should be nonnegative on $[-1,1]$ (otherwise SJ would have positive eigenvalues).

In general, it is too ambitious to solve this minimax problem for arbitrary eigenvalue functions $\hat{K}(\zeta)$. Therefore, we shall write, instead,

$$\rho(SJ) \leqslant \max_{-1\leqslant\zeta\leqslant 1}[(1-\zeta)\hat{Q}_k(\zeta)]\cdot\max_{-1\leqslant\zeta\leqslant 1}[\frac{\hat{K}(\zeta)}{\zeta-1}], \tag{3.5}$$

and solve the minimax problem for the polynomial $(1-\zeta)\hat{Q}_k(\zeta)$, which is independent of the parabolic equation under consideration. This approach is justified by the observation that the resulting polynomial $\hat{Q}_k$ does generate optimal second-order smoothing operators in the case of the parabolic model problem of Example 3.2. In nonmodel problems (where $\hat{K}(\zeta)$ contains the factor $\zeta-1$), the resulting polynomial $\hat{Q}_k$ is not optimal, but it gives rise to the same reduction factor of the spectral radius as in the model problem.

On the basis of (3.5) the stability condition (2.2) becomes

$$\Delta t \leqslant \mu\,\beta \min_{-1\leqslant\zeta\leqslant 1} \frac{\zeta-1}{2\hat{K}(\zeta)}, \tag{3.6a}$$

where we introduced the *amplification factor*

$$\mu := [\max_{-1\leqslant\zeta\leqslant 1} \tfrac{1}{2}(1-\zeta)\hat{Q}_k(\zeta)]^{-1}. \tag{3.6b}$$

Notice that $\mu = 1$ ($\hat{Q}_0 \equiv 1$) if no smoothing operators are applied.

3.1.1. Second-order smoothing operators

The following lemma is basic in our subsequent discussion:

LEMMA 3.1. *Of all polynomials $P_m(z)$ of degree m in z satisfying the conditions $P_m(1) = 0$, $P'_m(1) = -1$, and $P_m(z) \geqslant 0$ on $[-1,1]$, the polynomial $P_m(z) := [1-T_m(z)]/m^2$ has the smallest maximum norm on $[-1,1]$.*

PROOF. The assertion of the lemma follows immediately from the various properties of the Chebyshev polynomial $T_m(z)$. □

With the help of this lemma the following theorem is easily proved.

THEOREM 3.3. *Let the smoothing operator S be generated by the polynomial*

$$\hat{Q}_k(z) = \frac{1-T_{k+1}(z)}{(k+1)^2(1-z)}. \tag{3.7}$$

Then, S is second-order accurate, and minimizes, for given k, the spectral radius $\rho(SJ)$ of the model problem in Example 3.2.

PROOF. It follows from Example 3.2 and from (3.4) that

$$\rho(SJ) = \frac{2}{\Delta^2 x}\max_{-1\leqslant\zeta\leqslant 1}\frac{1-T_{k+1}(\zeta)}{(k+1)^2},$$

and from Lemma 3.1 that $\rho(SJ)$ is as small as possible, while $\hat{Q}_k(z)$ is nonnegative with $\hat{Q}_k(1) = 1$. □

EXAMPLE 3.4. The first few polynomials Q_k corresponding to the optimal polynomials $\hat{Q}_k$ specified in Theorem 3.3 are given by

$$Q_1(z) = \frac{1}{2}(1+z),$$

$$Q_2(z) = \frac{1}{9}(3+4z+2z^2),$$

$$Q_3(z) = \frac{1}{8}(2+3z+2z^2+z^3).$$

Notice that $Q_3(z)$ is identical with the polynomial $Q_3(z)$ derived in Example 2.2. □

THEOREM 3.4. *Let J satisfy the conditions (3.3) and let S be generated by (3.7). Then the amplification factor μ is given by $(k+1)^2$ so that*

$$\Delta t \leqslant \beta(k+1)^2 \min_{-1\leqslant\zeta\leqslant 1} \frac{\zeta-1}{2\hat{K}(\zeta)}, \tag{3.6'}$$

where $\hat{K}(\zeta)$ is assumed to be negative.

PROOF. The proof is immediate from (3.7) and (3.6). □

We recall that for $k=0$ the stability condition (3.6') corresponds to the "unsmoothed" method because $\hat{Q}_0(z) \equiv 1$. This indicates that the *gain factor* obtained by the smoothing technique is as large as $(k+1)^2$ independent of the particular problem under consideration.

EXAMPLE 3.5. Consider the model problem in Example 3.2. For this three-point discretization we have

$$\min_{-1\leqslant\zeta\leqslant 1} \frac{\zeta-1}{2\hat{K}(\zeta)} = \frac{\Delta^2 x}{4}.$$

Substitution into (3.6') yields the stability condition

$$\Delta t \leqslant \frac{1}{4}\beta(k+1)^2\Delta^2 x.$$

We recall that, by virtue of Theorem 3.3, there exists no smoothing operator of degree k which leads to a larger maximum stable step Δt. □

EXAMPLE 3.6. Consider the discretization defined in Example 3.3. For this five-point discretization we have

$$\min_{-1\leqslant\zeta\leqslant 1} \frac{\zeta-1}{2\hat{K}(\zeta)} = \min_{-1\leqslant\zeta\leqslant 1} \frac{3\Delta^2 x}{2(7-\zeta)} = \frac{3}{16}\Delta^2 x,$$

so that, by Theorem 3.4, the stability condition becomes

$$\Delta t \leqslant \frac{3}{16}\beta(k+1)^2\Delta^2 x. \quad \square$$

The following lemma is of interest in the actual implementation of smoothing operators.

LEMMA 3.2. *If $m = 2^q$ with $q > 0$, then*

$$T_m(z) = 1-m(1-z)\prod_{l=0}^{q-1}(1+T_{2^l}(z)).$$

PROOF. It follows from the identity $T_{2l} = 2T_l^2 - 1$ that

$$1-T_m = 1-T_{2^q} = 2(1-T_{2^{q-1}}^2) = 2(1+T_{2^{q-1}})(1-T_{2^{q-1}}) =$$
$$\ldots = 2^q(1+T_{2^{q-1}})(1+T_{2^{q-2}})\ldots(1+T_1)(1-T_1).$$

This proves the lemma. □

By means of this lemma and Corollary 3.1 the following Theorem is immediate:

THEOREM 3.5. *Let $k = 2^q - 1$ with $q > 0$, then the smoothing operator based on (3.7) can be factorized according to*

$$Sv = \frac{1}{2^{2q}} \left(\prod_{l=0}^{q-1} [E^{2^l} + 2 + E^{-2^l}]v^{(l)}\right). \quad \square \tag{3.8}$$

The operator (3.8) is identical to the smoothing operator proposed in WUBS [4]. In this factorized form it allows a rather efficient implementation on a computer.

3.1.2. Fourth-order smoothing operators

Suppose that we can solve the following minimax problem:

Problem 3.1. Of all polynomials $P_m(z)$ of degree m in z satisfying the conditions $P_m(1) = 0, P'_m(1) = -1, P''_m(1) = 0$ and $P_m(z) \geqslant 0$ on $[-1,1]$, find the polynomial with the smallest maximum norm on $[-1,1]$. □

If such a minimax polynomial is found, then by defining

$$\hat{Q}_k(z) = \frac{P_{k+1}(z)}{1-z}, \quad k = m-1,$$

we obtain a polynomial satisfying the fourth-order conditions $\hat{Q}_k(1) = 1$, $\hat{Q}'_k(1) = 0$, being nonnegative on $[-1,1]$, and maximizing the amplification factor in the stability condition (3.6).

Sofar, we did not succeed in deriving closed expressions for the optimal polynomials $P_{k+1}(z)$ and the corresponding maximal amplification factor μ. The derivation of these polynomials will be subject of future investigations.

An alternative is offered by Theorem 3.2(b). By starting with the one-parameter family of fourth-order polynomials

$$\hat{Q}(z) = 1-\alpha+\alpha\hat{Q}^*(z)(2-\hat{Q}^*(z)), \tag{3.9}$$

where $\hat{Q}^*(z)$ generates a second-order smoothing operator S^*, there is only one parameter to be optimized such that $(1-z)\hat{Q}(z)$ has a minimal maximum norm on $[-1,1]$. In Table 3.1 the resulting amplification factors μ are listed for the case where $\hat{Q}^*(z)$ is given by (3.7). It seems that $\mu/(k+1)^2$, k denoting the degree of $\hat{Q}$, converges to a constant value (recall that this values is 1 in the second-order case).

We observe that the spectral radius $\rho(SJ)$ can be reduced further for $\alpha > 1$. However, then $\hat{Q}(z)$ is not nonnegative on $[-1,1]$ anymore which leads to unstable discretizations.

Finally, we remark that the operator S generated by $\hat{Q}(z)$, i.e.,

$$S = (1-\alpha)I + \alpha S^*(2I - S^*), \tag{3.10}$$

is to a high degree factorizable if S^* is factorizable.

TABLE 3.1. μ-values for (3.9) with $\hat{Q}^*(z)$ defined by (3.7)

Degree k of S	α	μ	$\mu/(k+1)^2$
2	1	2.6	.29
4	1	4.7	.19
6	1	8.3	.17
8	1	12.7	.16

3.2. Smoothing of a hyperbolic model problem

Symmetric space discretizations of hyperbolic problems often lead to Jacobian matrices defined by

$$J\mathbf{v} = (\frac{1}{2}[K(E)-K(E^{-1})]v^{(j)}), \tag{3.11a}$$

where K is a polynomial.

DEFINITION 3.2. Let $C(z)$ be defined as in Definition 3.1. Then $\tilde{C}$ is defined by

$$\tilde{C}(z) := \sum_{l=1}^{r} c_l U_{l-1}(z),$$

where U_l is the Chebyshev polynomial of the second kind. □

By means of this definition we can write the eigenvalue equation for the Jacobian matrix J in the form

$$J\mathbf{e} = \pm i\sqrt{1-\zeta^2}\,\tilde{K}(\zeta)\mathbf{e},\ \mathbf{e} := (e^{ij\omega\Delta x}),\ \zeta := \cos(\omega\Delta x), \tag{3.11b}$$

where the sign is determined by the sign of $\sin(\omega\Delta x)$.

In order to prove this, let

$$K(z) := \sum_{l=0}^{r} c_l z^l.$$

Then

$$\begin{aligned} J\mathbf{e} &= \frac{1}{2}[K(e^{i\omega\Delta x})-K(e^{-i\omega\Delta x})]\,\mathbf{e} \\ &= \frac{1}{2}\sum_{l=0}^{r} c_l(e^{il\omega\Delta x}-e^{-il\omega\Delta x})\,\mathbf{e} \\ &= i\sum_{l=1}^{r} c_l \sin(\omega l\Delta x)\mathbf{e} = i\sum_{l=1}^{r} c_l \sin(\omega\Delta x)U_{l-1}(\cos\omega\Delta x)\,\mathbf{e} \\ &= \pm i\sqrt{1-\zeta^2}\sum_{l=1}^{r} c_l U_{l-1}(\zeta)\mathbf{e}. \end{aligned}$$

EXAMPLE 3.7. Consider the *hyperbolic model problem*

$$u_t = u_x + g(x,t)$$

and its three-point discretization

$$\frac{dy^{(j)}}{dt} = \frac{1}{2\Delta x}[E-E^{-1}]y^{(j)} + g^{(j)}(t).$$

The Jacobian of this system is characterized by

$$K(z) = \frac{1}{\Delta x} z,$$

so that

$$\tilde{K}(z) = \frac{1}{\Delta x}. \quad \square$$

EXAMPLE 3.8. If the above equation is discretized by the fourth-order five-point discretization we obtain

$$K(z) = \frac{z}{6\Delta x}(8-z),$$

$$\tilde{K}(z) = \frac{1}{3\Delta x}(4-z). \quad \square$$

For hyperbolic problems we are faced with the problem of minimizing

$$\rho(SJ) = \max_{-1 \leq \zeta \leq 1} \sqrt{1-\zeta^2}\, |\hat{Q}_k(\zeta)\tilde{\tilde{K}}(\zeta)|, \tag{3.12}$$

taking into account the order conditions for $\hat{Q}_k$ stated in Theorem 3.2. Notice that, in contrast to the minimax problem for parabolic problems, the polynomial $\hat{Q}_k$ is not required to be nonnegative on $[-1,1]$. Consequently, the polynomials derived for parabolic problems are not optimal in the present case.

Instead of minimizing the right-hand side of (3.12) we shall write

$$\rho(SJ) \leq \max_{-1 \leq \zeta \leq 1} \sqrt{1-\zeta^2}\, |\hat{Q}_k(\zeta)| . \max_{-1 \leq \zeta \leq 1} |\tilde{K}(\zeta)| \tag{3.13}$$

and we solve the minimax problem for $\sqrt{1-\zeta^2}\,\hat{Q}_k(\zeta)$ independently of $\tilde{K}$ (cf. the discussion given for (3.5)). Similarly to (3.6), we derive from (3.13) the stability condition

$$\Delta t \leq \mu \beta \min_{-1 \leq \zeta \leq 1} \frac{1}{|\tilde{K}(\zeta)|}, \mu := [\max_{-1 \leq \zeta \leq 1} \sqrt{1-\zeta^2}\, |\hat{Q}_k(\zeta)|]^{-1}. \tag{3.14}$$

Again, μ is chosen such that $\mu = 1$ if no smoothing is applied.

3.2.1. Second-order smoothing operators

The following lemma plays the role that Lemma 3.1 played for parabolic problems.

LEMMA 3.3. *Of all functions of the form* $\sqrt{1-z^2}\, P_m(z)$ *where* $P_m(z)$ *is a polynomial of degree* m *in* z *satisfying the condition* $P_m(1) = 1$, *the function* $\sqrt{1-z^2}\, U_m(z)/(m+1)$ *has the smallest maximum norm on* $[-1,1]$.

PROOF. Since $U_m(1) = m+1$ the condition $P_m(1) = 1$ is satisfied. Furthermore, we deduce from the identity

$$|U_m(z)| \equiv \sqrt{\frac{1-T_{m+1}^2(z)}{1-z^2}},$$

that the function $\sqrt{1-z^2}\, U_m(z)$ satisfies the equal ripple property from which it can be concluded that this function is optimal. $\square$

By virtue of this lemma the following theorem is obvious.

THEOREM 3.6. *Let the smoothing operator S be generated by the polynomial*

$$\hat{Q}_k(z) = \frac{U_k(z)}{k+1}. \tag{3.15}$$

Then S is second-order accurate, and minimizes, for given k, de spectral radius $\rho(SJ)$ *of the model problem in Example 3.7.* □

EXAMPLE 3.9. The first few polynomials $Q_k(z)$ generated by (3.15) are given by

$$Q_1(z) = z,$$

$$Q_2(z) = \frac{1}{3}(1+2z^2),$$

$$Q_3(z) = \frac{1}{2}(z^3+z). \quad \square$$

THEOREM 3.7. *Let J satisfy the conditions (3.11) and let S be generated by (3.15). Then the amplification factor is given by* $k+1$ *leading to the stability condition*

$$\Delta t \leqslant \beta(k+1) \min_{-1\leqslant\xi\leqslant 1} \frac{1}{|\tilde{K}(\xi)|}. \tag{3.14'}$$

PROOF. Substitution of (3.15) into (3.14) leads to (3.14'). □

EXAMPLE 3.10. Consider the discretization of Example 3.8. Applying Theorem 3.7 we find that this five-point discretization is stable if

$$\Delta t \leqslant \frac{3}{5}\beta(k+1)\Delta x. \quad \square$$

As in the parabolic case the operator S generated by (3.15) can be factorized for special values of k. The counterpart of Lemma 3.2 is given by

LEMMA 3.4. *If* $m = 2^q$ *with* $q > 0$, *then*

$$U_{m-1}(z) = m \prod_{l=0}^{q-1} T_{2^l}(z).$$

PROOF. Using the identity $U_{2l-1} = 2U_{l-1}T_l$, (cf [1], p.782) we deduce that

$$U_{m-1} = U_{2^q-1} = 2U_{2^{q-1}-1}T_{2^{q-1}} = \cdots = 2^q \prod_{l=1}^{q} T_{2^{l-1}}$$

proving the assertion of the lemma. □

The analogue of Theorem 3.5 is given by

THEOREM 3.8. *Let* $k = 2^q - 1$ *with* $q > 0$, *then the smoothing operator based on (3.15) can be factorized according to*

$$S\mathbf{v} = \frac{1}{2^q}\left(\prod_{l=0}^{q-1}[E^{2^l} + E^{2^{-l}}]v^{(j)}\right). \quad \square \tag{3.16}$$

3.2.2. Fourth-order smoothing operators

For hyperbolic problems we have the following analogue of Problem 3.1.

Problem 3.2. Of all functions of the form $\sqrt{1-z^2}\,P_m(z)$ where $P_m(z)$ is a polynomial of degree m in z satisfying the conditions $P_m(1) = 1$ and $P'_m(1) = 0$, find the function with the smallest maximum norm on $[-1,1]$. □

If this problem is solved for $m = k$, we set $\hat{Q}_k(z) = P_k(z)$ to obtain the generating polynomial for a fourth-order smoothing operator with optimal amplification factor μ as defined in (3.14).

As in the parabolic case we did not yet find closed expressions for the optimal polynomials and we applied, instead, (3.9) with $\hat{Q}^*(z)$ given by (3.15). The analogue of Table 3.1 is presented by Table 3.2. Notice that here α is not restricted by a sign condition on $\hat{Q}(z)$. The resulting smoothing operators are given by (3.10) with S^* corresponding to $\hat{Q}^*$.

TABLE 3.2. μ-values for (3.9) with $\hat{Q}^*(z)$ defined by (3.15)

Degree k of S	α	μ	$\mu/(k+1)$
2	.67901	1.38	.46
4	.83512	2.06	.41
6	.84250	1.96	.28
8	.95280	2.56	.28

4. NUMERICAL EXPERIMENTS

In WUBS [4] a few first experiments are reported for hyperbolic problems using smoothing techniques in combination with conventional time integrators. Here, we present further experiments, both for parabolic and hyperbolic problems. All examples are chosen such that conventional explicit time integrators (without smoothing) require unrealisticly small time steps.

The examples are, respectively,

$$u_t = u_{xx} + g_1(t,x), \tag{4.1}$$

$$u_t = u^2 u_{xx} + g_2(t,x), \tag{4.2}$$

$$u_t = u_x + g_3(t,x), \tag{4.3}$$

$$u_t = \tfrac{1}{2}(u^2)_x + g_4(t,x), \tag{4.4}$$

where the forcing functions $g_j(t,x)$ are chosen in such a way that

$$u(t,x) = \tfrac{1}{2}[\sin(x+t) + \sin(\omega x)], \quad \omega \in \mathbb{N} \tag{4.5}$$

presents the exact solution. The initial condition is taken from the exact solution, and periodic boundary conditions are imposed at $x = 0$ and $x = 2\pi$. In all examples the integration interval is given by $[0,T]$, where T is specified in the tables of results.

The semi-discrete equations are obtained by using, respectively, the three-point discretizations of the Examples 3.2 and 3.7, and the five-point discretizations of the Examples 3.3 and 3.8. The spatial grid is given by the points $x_j = j\Delta x$, $j = 1,2,\ldots,2\pi/\Delta x$, where Δx is chosen such that the forcing function and the initial function can adequately be represented.

The time integrators used (in combination with smoothing operators specified in the tables of results) are given by the explicit Runge-Kutta methods (for the notation used see LAPIDUS & SEINFELD [3]):

RKP:

0	0		
1/8	1/8		
1/2	0	1/2	
	0	0	1

RKH:

0	0		
1/2	1/2		
1/2	0	1/2	
	0	0	1

Both methods are second-order accurate: RKP is used for the parabolic problems (4.1) and (4.2) with stability boundary $\beta = 6.26$ in the stability condition (3.6); RKH is used for the hyperbolic problems (4.3) and (4.4) with stability boundary $\beta = 2$ in the stability condition (3.14). These conditionally stable methods were respectively applied with the parabolic smoothers generated by (3.7) and Table 3.1, and with the hyperbolic smoothers generated by (3.15) and Table 3.2.

As reference method we apply the implicit Crank-Nicholson method which can be represented by the array:

CN:

0	0	0
1	1/2	1/2
	1/2	1/2

This method is also second-order accurate, but it is unconditionally stable both for parabolic and hyperbolic problems (i.e., $\beta = \infty$), and, therefore, it requires no smoothing in order to stabilize the integration process.

The integration steps Δt are chosen as large as allowed by the stability condition of the smoothed RKP or RKH methods.

In the tables of results we list the degree k of the smoothing operator used, the total number of steps $N := T/\Delta t$, and the number of correct significant digits obtained in $t_N = T$, i.e., the value of

$$sd := \min_j(-\log_{10}|y_N^{(j)} - u(T,x_j)|).$$

4.1. Problem (4.1)

This problem is given by (4.1) with solution (4.5) and with $\omega = 16$. The solution is therefore rapidly oscillating, while its time derivative is slowly varying with x; hence, the problem belongs to the problem class for which the smoothing technique described in the preceding sections should be effective. In order to represent the initial condition and the forcing function adequately on the spatial grid we choose $\Delta x = \pi/192$.

The results obtained are listed in the Tables 4.1a and 4.1b (see Section 4.5). They show that the smoothed RKP method performs stably for all integration steps. Compared with the maximal step allowed by the "unsmoothed" RKP method (i.e. $k = 0$), the gain factors for second and fourth-order smoothing are at least 64 and 32, respectively. The accuracy is hardly reduced by the smoothing procedure, except for the case where fourth-order space discretization is combined with second-order smoothing (here, an increase of the degree of the smoothing operator by 1 decreases the number of correct digits by about .25 if k is small and by about .15 if k becomes larger). In all other cases, the accuracy is comparable with that of the CN method.

4.2 Problem (4.2)

This problem is a *nonlinear* modification of problem (4.1), again with $\omega = 16$. The results listed in the Tables 4.2a and 4.2b show a similar behaviour as for the linear problem (4.1), provided that the degree of the smoothing operator is not too large ($k \leqslant 5$ for second-order smoothing and $k \leqslant 10$ for fourth-order smoothing). The respective amplification factors of the maximal stable integration step are at least 35 and 18.

4.3 Problem (4.3)

The results for the linear hyperbolic problem (4.3) with $\omega = 16$ (see the Tables 4.3a and 4.3b) again show that the smoothed RKH method performs stably for all integration steps, while the accuracy is not or only marginally less than the accuracy obtained by the CN method. The amplification factors of the maximal stable integration steps are at least 8 and 4 for second-order and fourth-order smoothing, respectively. Notice that, in contrast to the results obtained for the parabolic problems (4.1) and (4.2), the numerical error is not only determined by space discretization and smoothing errors, but also contains a time discretization error.

4.4. Problem (4.4)

When we integrated the nonlinear problem (4.4) with $\omega = 16$, rather low accuracies were obtained on a spatial grid with $\Delta x = \pi/192$, and instabilities developed in the case of fourth-order smoothers. Due to this low accuracy, the *numerical* solution did not satisfy the requirement that its time derivative is a smooth function of x. In order to overcome this unwanted behaviour we should decrease Δx, or equivalently, in order to stay within our budget available for these numerical experiments, we may decrease ω. Choosing $\omega = 8$ we obtained the results listed in the Tables 4.4a and 4.4b. We now have stability for all integration steps and accuracies which are even higher than those produced by the CN method.

4.5. Tables of results

TABLE 4.1a. *sd*-values for problem (4.1) with $\omega = 16, T = 1.0, \Delta x = \pi/192$, and with second-order smoother based on (3.7)

	3-point coupling			5-point coupling	
k	N	RKP	CN	N	RKP	CN
0	2400	2.54	2.54	3200	4.59	4.59
1	600	2.54	2.54	800	4.34	4.58
2	270	2.53	2.54	355	4.10	4.58
3	150	2.53	2.54	200	3.90	4.58
4	96	2.52	2.54	130	3.73	4.56
5	68	2.51	2.54	90	3.58	4.54
6	49	3.26	2.54	66	3.46	4.50
7	38	2.49	2.54	50	3.35	4.44

TABLE 4.1b. *sd*-values for problem (4.1) with $\omega = 16, T = 1.0, \Delta x = \pi/192$, and with fourth-order smoother based on {(3.9), $\alpha = 1$}

	3-point coupling			5-point coupling		
k	N	RKP	CN	N	RKP	CN
0	2400	2.54	2.54	3200	4.59	4.59
2	925	2.54	2.54	1250	4.59	4.59
4	540	2.54	2.54	710	4.59	4.59
6	300	2.54	2.54	400	4.58	4.58
8	192	2.54	2.54	260	4.58	4.58
10	136	2.54	2.54	180	4.58	4.57
12	98	2.54	2.54	132	4.57	4.56
14	76	2.54	2.54	100	4.55	4.55

TABLE 4.2a. *sd*-values for problem (4.2) with $\omega = 16, T = 1.0, \Delta x = \pi/192$, and with second-order smoother based on (3.7)

	3-point coupling			5-point coupling		
k	N	RKP	CN	N	RKP	CN
0	2400	0.62	0.62	3200	3.35	3.35
1	600	0.58	0.62	800	2.62	3.35
2	270	0.74	0.62	355	2.23	3.34
3	150	1.07	0.62	200	2.03	3.32
4	96	1.26	0.62	130	1.86	3.28
5	68	1.40	0.62	90	1.68	3.22

TABLE 4.2b. *sd*-values for problem (4.2) with $\omega = 16, T = 1.0, \Delta x = \pi/192$, and with fourth-order smoother based on {(3.9), $\alpha = 1$}

	3-point coupling			5-point coupling		
k	N	RKP	CN	N	RKP	CN
0	2400	0.62	0.62	3200	3.35	3.35
2	925	0.52	0.62	1250	3.13	3.35
4	540	0.59	0.62	710	3.01	3.35
6	300	0.83	0.62	400	3.18	3.34
8	192	1.09	0.62	260	3.40	3.33
10	136	1.13	0.62	180	3.35	3.31

TABLE 4.3a. *sd*-values for problem (4.3) with $\omega = 16, T = 10$, $\Delta x = \pi/192$, and with second-order smoother based on (3.15)

	3-point coupling			5-point coupling		
k	N	RKH	CN	N	RKH	CN
0	310	2.19	1.96	472	3.57	3.54
1	155	2.08	1.97	236	2.83	3.05
2	104	1.94	1.81	160	2.46	2.75
3	78	1.79	1.77	120	2.20	2.52
4	62	1.66	1.82	95	2.00	2.33
5	52	1.54	1.58	80	1.84	2.19
6	43	1.42	1.49	67	1.70	2.03
7	39	1.33	1.47	58	1.58	1.91

TABLE 4.3b. *sd*-values for problem (4.3) with $\omega = 16, T = 10, \Delta x = \pi/192$, and with fourth-order smoother based on { (3.9), Table 3.2 }

	3-point coupling			5-point coupling		
k	N	RKH	CN	N	RKH	CN
0	310	2.19	1.96	472	3.57	3.54
2	220	2.16	2.16	350	3.39	3.45
4	145	2.10	2.41	240	3.10	3.06
6	150	2.11	2.10	260	3.16	3.13
8	115	2.04	2.28	180	2.86	2.88
10	110	2.03	2.02	185	2.88	2.91
12	85	1.93	1.83	135	2.62	2.64
14	85	1.93	1.83	145	2.68	2.68

TABLE 4.4a. *sd*-values for problem (4.4) with $\omega = 8, T = 4, \Delta x = \pi/192$, and with second-order smoother based on (3.15)

	3-point coupling			5-point coupling		
k	N	RKH	CN	N	RKH	CN
0	110	1.36	1.37	145	3.12	2.86
1	50	1.63	1.44	75	2.55	2.19
2	33	1.83	1.66	45	2.19	1.71
3	22	1.67	1.27	30	1.81	1.32
4	17	1.73	1.06	25	1.82	1.20
5	14	1.42	0.84	20	1.52	1.03

TABLE 4.4b. *sd*-values for problem (4.4) with $\omega=8, T=4, \Delta x=\pi/192$, and with fourth-order smoother based on {(3.9), Table 3.2}

	3-point coupling			5-point coupling		
k	N	RKH	CN	N	RKH	CN
0	110	1.36	1.37	145	3.12	2.86
2	75	1.53	1.38	115	3.09	2.65
4	50	1.65	1.44	70	2.54	2.13
6	45	1.69	1.48	70	2.57	2.13
8	35	1.27	1.62	55	2.10	1.90
10	30	1.34	1.61	40	1.74	1.59

5. Concluding Remarks

In this paper we analysed a smoothing technique for preconditioning a special class of semi-discrete partial differential equations. It turned out that, in order to obtain optimal smoothing matrices, one should distinguish between parabolic and hyperbolic equations. The resulting smoothing matrices are quite different. For instance, application of a smoothing matrix, which is optimal for the hyperbolic model problem, would lead to instabilities when applied to a parabolic problem. However, if the smoothing operator is appropriately chosen, a *substantial amplification of the maximal stable step size* is obtained, *irrespective of the (explicit) time integrators used,* while the additional computational effort is rather limited. The price to be paid for the less restrictive stability condition is (i) a *decrease of the accuracy for large degree smoothing matrices,* and (ii) the requirement that the *right-hand side function should be provided in grid points beyond the boundary.*

The reduced accuracy for large k has two sources: firstly, the smoothing technique analysed in this paper presupposes that the right-hand side function is a smooth function of the spatial variables and rapidly looses accuracy if not; secondly, the error constant of the smoothing operator increases with k^2. On the other hand, the numerical experiments of the preceding section show that smoothing matrices of degree as high as 14 still do not reduce the accuracy very much if the problem belongs to the class of problems we are aiming at.

In Section 4, the need of providing right-hand side values outside the domain was solved by imposing periodic boundary conditions. In the case of other types of boundary conditions, a plausible approach is to generate these values by extrapolation. We repeated the series of experiments of Section 4 by employing *rational extrapolation* and we found a comparable stability behaviour and accuracy behaviour as well (polynomial extrapolation leads, of course, to severe instabilities). Alternatively, one may employ the Jacobian matrix of the right-hand side to achieve a correct amount of smoothing in the near boundary points. Both approaches will be subject of further investigations.

References

[1] M. Abramowitz, & I.A. Stegun, *Handbook of mathematical functions,* National Bureau of Standards, Applied Mathematics Series 55, U.S. Government Printing Office, Washington, 1964.

[2] T.J. Baker, A. Jameson and W. Schmidt, *A family of fast and robust Euler codes,* Proc. Workshop on Computational Fluid Dynamics (Tullahoma, 1984), pp. 17.1-17.38.

[3] L. Lapidus & J.H. Seinfeld, *Numerical solution of ordinary differential equations,* Mathematics in science and engineering, Academic Press, New York and London, 1971.

[4] F.W. Wubs, *Stabilization of explicit methods for hyperbolic initial-value problems,* to appear in: Int. J. Numer. Meth. in Fluids, 1986.

INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 24, 557–567 (1987)

THE METHOD OF LINES AND EXPONENTIAL FITTING

P. J. VAN DER HOUWEN AND F. W. WUBS

Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

SUMMARY

When the method of lines is used for solving time-dependent partial differential equations, finite differences are commonly employed to obtain the semidiscrete equations. Usually, if the solution is expected to be smooth, symmetric difference formulae are chosen for approximating the spatial derivatives. These difference formulae are almost invariably based on Lagrange type differentiation formulae. However, if it is known in advance that periodic components of given frequency are dominating in the solution, more accurate difference formulae, based on exponentials with imaginary exponent, are available. This paper derives such formulae and presents numerical results which clearly indicate that the accuracy can be improved considerably by exploiting additional knowledge on the frequencies of the solution.

1. INTRODUCTION

A widely used approach to solving time-dependent *partial* differential equations is the method of lines. This method replaces the spatial derivatives by discrete approximations and enables us to apply well-developed time integrators for solving the resulting systems of *ordinary* differential equations. When finite differences are used to obtain the semidiscrete equations, almost invariably Lagrange-type formulae, based on polynomial interpolation of the solution, are employed to derive the difference approximations. However, in many problems arising in fluid dynamics it is known in advance that the solution is dominated by one or more periodic components of known frequency. In such cases it turns out to be better to use difference formulae based on trigonometric interpolation, that is we require that the difference formulae have a reduced truncation error for certain exponential functions with imaginary argument (see section 2). We will call such formulae *exponentially fitted* difference formulae.

In Reference 1 exponentially fitted difference approximations to first-order spatial derivatives were derived and were shown to be more accurate than conventional difference formulae in oscillatory problems. These results are summarized in section 3.1. In section 3.2, similar formulae are derived for second-order derivatives and a comparison is made with conventional difference formulae. In section 3.3, we discuss the automatic estimation of dominant frequencies in grid functions. By means of a few numerical examples we show the performance of such a frequency estimator.

Section 4 provides formulae for approximating boundary conditions to be imposed on periodic solutions.

Finally, in Section 5, we show by a number of numerical experiments that using exponentially fitted difference formulae in the space discretization of partial differential equations leads to a considerable improvement of the accuracy.

The adaption of *spatial discretizations* to known frequencies of the exact solution has received

0029–5981/87/030557–11$05.50

Received 4 March 1986
Revised 15 July 1986

little attention in the literature. This is in contrast to the development of *time integrators* for solving periodic initial-value problems where a lot of work already has been done. We mention the papers of Gautschi[2] Brusa and Nigro,[3] Gladwell and Thomas[4] and van der Houwen and Sommeijer,[5] where further references to oscillatory time integrators can be found.

2. THE TRUNCATION ERROR IN THE METHOD OF LINES

We discuss the discretization of partial differential equations of the general form

$$\frac{\partial^v w}{\partial t^v} = F(w) := G\left(t, \mathbf{x}, w, \frac{\partial w}{\partial x_1}, \frac{\partial w}{\partial x_2}, \frac{\partial^2 w}{\partial x_1^2}, \frac{\partial^2 w}{\partial x_2^2}\right), \quad \mathbf{x} = (x_1, x_2)^T \in \Omega, \quad v = 1, 2 \tag{1}$$

where F is the differential operator defined by the function G, and where it is known in advance that the solution is composed of components that are periodic in the space variable $\mathbf{x}$. Applying the method of lines we replace the differential operators by difference operators:

$$\frac{\partial}{\partial x_j} \sim D_j, \quad \frac{\partial^2}{\partial x_j^2} \sim D_{2+j}, \quad j = 1, 2 \tag{2}$$

and instead of (1), we consider the equation

$$\begin{aligned} &\frac{\partial^v W}{\partial t^v} = F_\Delta(W) := G(t, \mathbf{x}, W, D_1 W, D_2 W, D_3 W, D_4 W) \\ &\mathbf{x} \in \Omega_\Delta := \{\mathbf{x} \mid \mathbf{x} = (j\Delta x_1, l\Delta x_2)^T;\ j, l = 0, \pm 1, \pm 2\} \end{aligned} \tag{3}$$

where W is a function of t and $\mathbf{x}$.

The truncation error of the semidiscrete equation (3) corresponding to a given test function $w = w(t, \mathbf{x})$ is given by

$$L(w) := \frac{\partial^v w}{\partial t^v} - F_\Delta(w) = F(w) - F_\Delta(w), \quad \mathbf{x} \in \Omega_\Delta \tag{4}$$

Suppose that the solution of (1) is given by

$$w_0 := \sum_{r=1}^{R} w_0^{(r)}(t) \exp(i\mathbf{f}^{(r)} \cdot \mathbf{x}) \tag{5}$$

where the frequency vectors

$$\mathbf{f}^{(r)} := (f_1^{(r)}, f_2^{(r)})^T, \quad r = 1, \ldots, R$$

are either known or are known to lie in a given real domain. Furthermore, let the exponential functions in (5) be eigenfunctions of the difference operators in (2) with eigenvalues defined by

$$D_j \exp(i\mathbf{f}^{(r)} \cdot \mathbf{x}) = \delta_j^{(r)} \exp(i\mathbf{f}^{(r)} \cdot \mathbf{x}), \quad j = 1, \ldots, 4 \tag{6}$$

Then from (4) and the definition of the operators F and F_Δ it follows that the magnitude of the truncation error corresponding to (5) can be reduced by minimizing the magnitude of the functions

$$\frac{\partial w_0}{\partial x_j} - D_j w_0 = \sum_{r=1}^{R} [if_j^{(r)} - \delta_j^{(r)}] w_0^{(r)}(t) \exp(i\mathbf{f}^{(r)} \cdot \mathbf{x}), \quad j = 1, 2 \tag{7a}$$

$$\frac{\partial^2 w_0}{\partial x_j^2} - D_{2+j} w_0 = \sum_{r=1}^{R} [(if_j^{(r)})^2 - \delta_{j+2}^{(r)}] w_0^{(r)}(t) \exp(i\mathbf{f}^{(r)} \cdot \mathbf{x}) \tag{7b}$$

We observe that by *symmetric* difference operators, we obtain in (6) purely imaginary eigenvalues for $j = 1, 2$ and real eigenvalues for $j = 3, 4$. Thus, it is then feasible to minimize the magnitude of the functions (7) by minimizing the extreme values of the real-valued functions

$$if_j^{(r)} - \delta_j^{(r)}, \quad (f_j^{(r)})^2 + \delta_{j+2}^{(r)}, \quad j = 1, 2; \quad r = 1, \ldots, R \tag{8}$$

by a judicious choice of the discretization weights in the difference operators. Since we do not want too many grid points involved in the discretization molecules, the minimization of (8) is only effective if R is small, that is the exact solution is dominated by only a few Fourier components.

3. EXPONENTIALLY FITTED DIFFERENCE FORMULAE

In this section we present discretization molecules for numerical differentation of periodic functions of the form (5).

3.1. *First-order derivatives*

Without derivation we give a symmetric, fourth-order, four-point line discretization:[1]

$$\begin{aligned} D_1 &= \frac{1}{\Delta x_1}[\xi_1(E_1^{+1} - E_1^{-1}) + \xi_2(E_1^2 - E_1^{-2})] \\ \xi_2 &:= \frac{\dfrac{z_+}{\sin(z_+)} - \dfrac{z_-}{\sin(z_-)}}{4[\cos(z_+) - \cos(z_-)]}, \quad \xi_1 := \frac{z_+}{2\sin(z_+)} - 2\xi_2\cos(z_+) \end{aligned} \tag{9}$$

where E_1 defines the forward shift operator over one mesh width; here

$$z_+ = f_1^{(1)}\Delta x_1, \quad z_- = f_1^{(2)}\Delta x_1 \tag{10a}$$

if we want to eliminate just two frequencies from the truncation error, and

$$z_\pm = \Delta x_1[\tfrac{1}{2}(\bar{f}_1^2 + \underline{f}_1^2) \pm \tfrac{1}{4}\sqrt{2}(\bar{f}_1^2 - \underline{f}_1^2)]^{1/2} \tag{10b}$$

if we want to minimize the truncation error for all frequencies in the interval

$$\underline{f}_1 \leqslant f_1^{(r)} \leqslant \bar{f}_1.$$

A similar definition holds for the difference operator D_2.

The formula (9) will be called an *exponentially fitted difference formula.*

3.2. *Second-order derivatives*

Consider the approximation

$$\frac{\partial^2}{\partial x_1^2} \sim D_3 := \frac{1}{(\Delta x_1)^2}\sum_{l=0}^{k}\sum_{j=0}^{k}\xi_j^{(l)}(E_1^{+j} + E_1^{-j})(E_2^{+l} + E_2^{-l}) \tag{11}$$

where E_i denotes the shift operator along the x_i-axis. It is elementary to show that this approximation is second-order accurate if

$$\begin{aligned} &\sum_{j,l=0}^{k}\xi_j^{(l)} = O[(\Delta x_1)^{p+2}], \quad \sum_{j,l=0}^{k} j^2\xi_j^{(l)} = \tfrac{1}{2} + O[(\Delta x_1)^p] \\ &\sum_{j,l=0}^{k} l^2\xi_j^{(l)} = O[(\Delta x_1)^p] \end{aligned} \tag{12a}$$

holds for $p=2$, and fourth-order accurate if (12a) holds for $p=4$ and if, in addition,

$$\sum_{j,l=0}^{k} j^4 \xi_j^{(l)} = O[(\Delta x_1)^2], \quad \sum_{j,l=0}^{k} l^4 \xi_j^{(l)} O[(\Delta x_1)^2], \quad \sum_{j,l}^{k} j^2 l^2 \xi_j^{(l)} = O[(\Delta x_1)^2] \tag{12b}$$

We remark that usually the order terms in the order equations (12a) and (12b) are set to zero, so that polynomials of sufficiently low degree are exactly differentiated. The corresponding difference formulae will be called *conventional* formulae. The introduction of the order terms does not decrease the (algebraic) order of the difference formulae and enables us to differentiate certain exponential functions with reduced errors, as will be shown below.

Let us apply the symmetric difference operator (11) to an exponential function. This leads to the eigenvalue (cf. (6))

$$\begin{aligned} \delta_3^{(r)} &= \frac{4}{(\Delta x_1)^2} \sum_{j,l=0}^{k} \xi_j^{(l)} \cos(j\mu_1^{(r)}) \cos(l\mu_2^{(r)}) \\ \mu_j^{(r)} &:= f_j^{(r)} \Delta x_j; \quad j = 1, 2 \end{aligned} \tag{13}$$

Defining the function

$$a_1(\mu) := \mu_1^2 + 4 \sum_{j,l=0}^{k} \xi_j^{(l)} \cos(j\mu_1) \cos(l\mu_2) \tag{14}$$

it follows from (8) and (14) that we should minimize

$$|(f_1^{(r)})^2 + \delta_3^{(r)}| = \frac{1}{\Delta^2 x_1} |a_1(\mu^{(r)})|, \quad r = 1, \ldots, R \tag{15}$$

In particular, we consider the minimization of (15) for five-point line discretizations, i.e.

$$D_3 = \frac{2}{(\Delta x_1)^2} [2\xi_0 + \xi_1 (E_1^{+1} + E_1^{-1}) + \xi_2 (E_1^{+2} + E_1^{-2})] \tag{16}$$

where we have omitted the super index in the discretization weights. The corresponding function (14) assumes the form

$$\begin{aligned} a_1(\mu) &= \mu_1^2 + 4[\xi_0 + \xi_1 \cos(\mu_1) + \xi_2 \cos(2\mu_1)] \\ &= \mu_1^2 + 4(\xi_0 - \xi_2) + 4\xi_1 \cos(\mu_1) + 8\xi_2 \cos^2(\mu_1) =: \bar{a}_1(\mu_1) \end{aligned} \tag{17}$$

In order to minimize the extreme values of (15) we require

$$\bar{a}_1(z_r) = 0, \quad r = 1, 2, 3 \tag{18}$$

where the three zeros of $\bar{a}_1$ are located at suitable points in the frequency interval. For instance, if $R=3$ and the three frequencies in (15) are known, then we set

$$z_r = f_1^{(r)} \Delta x_1, \quad r = 1, 2, 3 \tag{19}$$

Alternatively, when it is only known that

$$\underline{f}_1 \leqslant f_1^{(r)} \leqslant \bar{f}_1, \quad r = 1, \ldots, R \tag{20}$$

then suitable values for z_r can be obtained by identifying the zeros of $\bar{a}_1(z)$ with the zeros of a Chebyshev polynomial shifted to the interval of frequencies (20).[1] This results in

$$z_2 = \sqrt{\tfrac{1}{2}(\bar{f}_1^2 + \underline{f}_1^2)}\, \Delta x_1, \quad z_1 = \sqrt{z_2^2 - (z_2^2 - \underline{f}_1^2 \Delta^2 x_1) \cos\left(\frac{\pi}{6}\right)},$$

$$z_3 = \sqrt{z_2^2 - (z_2^2 - \bar{f}_1^2 \Delta^2 x_1)\cos\left(\frac{\pi}{6}\right)} \tag{21}$$

The conditions (18) imply that exponential functions of the form

$$\exp\left(i\frac{z_r x_1}{\Delta x_1}\right), \quad r = 1, 2, 3$$

are exactly differentiated by the difference operator (16).

For future reference, we give the solution of equation (18):

$$\begin{aligned}
\xi_2 &= \frac{1}{8}\frac{z_1^2(c_2 - c_3) + z_2^2(c_3 - c_1) + z_3^2(c_1 - c_2)}{(c_1 - c_2)(c_3 - c_1)(c_2 - c_3)} \\
\xi_1 &= -\frac{1}{4}\frac{z_1^2 - z_2^2}{c_1 - c_2} - 2\xi_2(c_1 + c_2) \\
\xi_0 &= \xi_2 - \xi_1 c_1 - 2\xi_2 c_1^2 - \tfrac{1}{4}z_1^2: \quad c_r = \cos(z_r), \quad r = 1, 2, 3
\end{aligned} \tag{22}$$

The discretization (16), (22) will be called an *exponentially fitted difference formula.*

We observe that the usual 5-point line discretization arises if $a(z)$ has all its zeros at the origin. The corresponding weights are given by

$$\xi_0 = -\tfrac{5}{8}, \quad \xi_1 = \tfrac{2}{3}, \quad \xi_2 = -\tfrac{1}{24} \tag{23}$$

This discretization satisfies (12) with $p = 4$ so that it is fourth-order accurate. It can be shown that the discretizations (16), (22), (19) and (16), (22), (21) are also fourth-order accurate.

In order to compare the truncation errors of the discretizations (16), (22) and (23), we derive expressions for the extreme values of $|\bar{a}|$ on the frequency interval (20) if the mesh size tends to zero. For (23) we easily find

$$|\bar{a}_1(\bar{f}_1 \Delta x_1)| \approx \tfrac{1}{90}(\bar{f}_1 \Delta x_1)^6 \text{ as } \Delta x_1 \to 0 \tag{24}$$

Since, in the case (22), the zeros of $\bar{a}$ vanish as the mesh size decreases, we find a similar expression to (24) only differing by the order constant; numerically we found for the case where the left end point of the frequency interval is the origin

$$|\bar{a}_1(\bar{f}_1 \Delta x_1)| \approx \frac{1}{3000}(\bar{f}_1 \Delta x_1)^6 \text{ as } \Delta x_1 \to 0 \tag{25}$$

3.3. Automatic estimation of dominant frequencies

In actual computation, it is convenient to estimate automatically the main frequencies of the numerical solution. Suppose that at $t = \bar{t}$ ($\bar{t}$ fixed) the numerical solution is expected to be an approximation to the function

$$u(\mathbf{x}) := \sum_{r=1}^{R} a_r \exp(i\,\mathbf{f}^{(r)} \cdot \mathbf{x}), \quad a_r \in \mathbb{C}, \quad \mathbf{f}^{(r)} \in \mathbb{R}^2 \tag{26}$$

A straightforward technique for determining the frequency vectors $\mathbf{f}^{(r)}$ is based on the minimization of the expression

$$\sum_{j=1}^{N} |u(\mathbf{x}_j) - U_j|^2 \tag{27}$$

where U_j denotes the numerical approximation to $u(\mathbf{x}_j)$ and $\{\mathbf{x}_j\}_{j=1}^{N}$ represents a set of grid points.

Most numerical libraries for large scale computing contain a suitable least-squares routine for solving this problem (e.g. NAG routine E04FCF). The efficiency of the least-squares algorithm for finding the frequencies $\mathbf{f}^{(r)}$ (and the coefficients a_r) that minimize (27) decreases when the number of parameters increases. Therefore, it is advantageous to replace (27) by an expression in which fewer parameters are involved. In particular, it would be nice when only the frequency parameters $\mathbf{f}^{(r)}$ are left. We illustrate the derivation of such an expression by a few examples.

Example 1. Let in (26) x be scalar and let $R = 1$, i.e.

$$u(x) = a_1 \exp(if^{(r)}x) \tag{28}$$

By applying the operator $P(E)$, where E is the forward shift operator and

$$P(z) = \sum_{j=-m}^{m} p_j z^j \tag{29}$$

we obtain the identity

$$P(E)u(x) - P(e^{if^{(1)}\Delta x})u(x) \equiv 0 \tag{30}$$

Suppose that $P(z)$ satisfies the condition

$$P(z) = P(1/z) \tag{31}$$

i.e. $p_j = p_{-j}$, and define

$$P^*(z) := p_0 + 2\sum_{j=1}^{m} p_j \cos(jz) \tag{32}$$

Then (30) assumes the form

$$P(E)u(x) - P^*(f^{(1)}\Delta x)u(x) \equiv 0 \tag{33}$$

This identity suggests the minimization of the *one-parameter expression*

$$\sum_{j=1}^{N} |[P(E) - P^*(f^{(1)}\Delta x)]U_j|^2 \tag{34}$$

Simple examples of a suitable function $P(z)$ are given by $P_1(z) = z + (1/z)$ and $P_2(z) = z - 2 + (1/z)$.

Example 2. Next we consider the case $R = 2$:

$$u(x) = a_1 \exp(if^{(1)}x) + a_2 \exp(if^{(2)}x) \tag{35}$$

Let us define the functions

$$v(x) := P(E)u(x), \quad w(x) := P^2(E)u(x) \tag{36}$$

Then we easily derive the identity

$$P^*(f^{(1)}\Delta x)P^*(f^{(2)}\Delta x)u(x) - [P^*(f^{(1)}\Delta x) + P^*(f^{(2)}\Delta x)]v(x) + w(x) \equiv 0 \tag{37}$$

As in the preceding example, this identity straightforwardly leads to a *two-parameter expression* to be minimized over the two frequency parameters.

In order to illustrate the performance of a frequency estimator based on (37) we have listed a few results in Table I for both functions $P_1(z)$ and $P_2(z)$. The choice of these functions is determined by efficiency considerations. The functions $u(x)$ correspond to the functions $w(0, x)$ used in our numerical experiments reported in Section 5. The results obtained show that the inaccuracy of the

Table I. Estimation of dominant frequencies

Problem	$2\pi/\Delta x$	$P_1(z)$ $f^{(1)}$	$P_1(z)$ $f^{(2)}$	$P_2(z)$ $f^{(1)}$	$P_2(z)$ $f^{(2)}$
1. $u(x) = \sin(\sin(x))$	8	0	1·10	0	1·45
	16	1·00	2·99	1·00	2·99
3. $u(x) = \tan(\sin(x))$	16	1·00	3·32	1·01	3·32
4. $u(x) = \sin(4x) + \sin(5x) + \sin(6x)$	16	4·05	5·90	4·05	5·90
5. $\sin(x) + \sin(1{\cdot}2x)$	8	1·00	1·20	1·00	1·20

estimated frequencies is at most 10 per cent for $P_1(z)$ and 40 per cent for $P_2(z)$. The latter error occurs for problem 1 on the coarsest grid. The other results appeared not to be sensitive to the choice of the function $P(z)$.

4. EXPONENTIALLY FITTED EXTRAPOLATION

In order to apply the symmetric difference operator (9) and (16), (22) near the boundary points we need to extrapolate, beyond the boundary, the numerical solution obtained at internal grid points. When *conventional* difference operators are used, then we may employ polynomial extrapolation; for example, the sixth-order formula

$$w(\mathbf{x}) \approx [6(E_1 + E_1^5) - 15(E_1^2 + E_1^4) + 20E_1^3 - E_1^6]w(\mathbf{x}) \tag{38}$$

However, when using *exponentially fitted* discretizations, then polynomial extrapolation is inaccurate, unless still higher order formulae are applied. A more attractive alternative is the use of exponentially fitted extrapolation formulae.

Let us start with the symmetric interpolation formula

$$w(\mathbf{x}) \approx A_1 w(\mathbf{x}) := \sum_{l=0}^{k} \sum_{j=1}^{k} \zeta_j^{(l)} (E_1^j + E_1^{-j})(E_2^l + E_2^{-l}) w(\mathbf{x}) \tag{39}$$

and require that this approximation has a small truncation error for functions of the form (5). Then, the extrapolation weights should be such that

$$\begin{aligned} w_0 - A_1 w_0 &= \sum_{r=1}^{R} [1 - \alpha_1^{(r)}] w_0^{(r)}(t) \exp(\mathbf{i}\mathbf{f}^{(r)} \cdot \mathbf{x}) \\ \alpha_1^{(r)} &:= 4 \sum_{l=0}^{k} \sum_{j=1}^{k} \zeta_j^{(l)} \cos(j\mu_1^{(r)}) \cos(l\mu_2^{(r)}) \end{aligned} \tag{40}$$

is small in magnitude. This is achieved by minimizing the magnitude of the function

$$b_1(\mu) := 1 - \alpha_1^{(r)} = 1 - 4 \sum_{l=0}^{k} \sum_{j=1}^{k} \zeta_j^{(l)} \cos(j\mu_1) \cos(l\mu_2) \tag{41}$$

over the range of frequencies. (Notice that $b_1(\mu)$ does not have the same form as the function $a_1(\mu)$ defined by (14): this can be traced back to the fact that $a_1(\mu)$ corresponds to the truncation error of a *difference* operator, whereas $b_1(\mu)$ corresponds to the truncation error of an *extrapolation* formula.) This minimax problem is similar to that discussed in section 3.2 for the function (14) and the (approximate) solution of this problem can be obtained along the same lines.

In our numerical experiments we will apply the seven-point formula that arises for

$$k = 3, \quad \zeta_j^{(l)} = 0 \quad \text{for } l \neq 0. \tag{42}$$

Defining

$$b_1(\mu) = \bar{b}_1(\mu_1) := 1 - 4[\zeta_1 \cos(\mu_1) + \zeta_2 \cos(2\mu_1) + \zeta_3 \cos(3\mu_1)] \tag{43}$$

we arrive at the fitting conditions (cf. (18))

$$\bar{b}_1(z_r) = 0, \quad r = 1, 2, 3 \tag{44}$$

where the three zeros of $\bar{b}_1$ coincide with (19) or (21). By solving (44), we obtain the extrapolation weights and the resulting extrapolation formula is then given by

$$w(\mathbf{x}) \approx \left[-\frac{\zeta_2}{\zeta_3}(E_1 + E_1^5) - \frac{\zeta_1}{\zeta_3}(E_1^2 + E_1^4) + \frac{1}{2\zeta_3} E_1^3 - E_1^6 \right] w(\mathbf{x}) \tag{45}$$

Just as the difference formula (16), (22), the extrapolation formula (45) presents an approximation to the formula that really minimizes the magnitude of the function (41). In the special case where $\zeta_j^{(l)} = 0$ for $l \neq 0$, it is possible to solve the minimax problem exactly, because $b_1(\mu)$ can then be expressed as a polynomial in $\cos(\mu_1)$ and for polynomials minimax solutions are available.

5. NUMERICAL EXPERIMENTS

By means of numerical examples we will show that the exponentially fitted discretization formulae derived in the preceding sections lead to considerably larger accuracies than the conventional discretizations, for both linear and non-linear problems. The problems are specified in Table II.

The initial conditions are taken from the exact solution. In cases where the solution is periodic with respect to the given x-interval, we compare results obtained by imposing Dirichlet boundary conditions and by imposing a periodicity condition. We confine our experiments to equations of the form

$$\frac{\partial^2 w}{\partial t^2} = G\left(x, t, w, \frac{\partial^2 w}{\partial x^2}\right) \tag{46}$$

The spatial discretization was based on 5-point formulae; we present results obtained by conventional and by exponentially fitted formulae ((16) with (23) and with (22)). In the case of Dirichlet boundary conditions, we used the polynomial extrapolation formula (38) for conventional discretizations and the exponentially fitted formula (45) otherwise.

The time integration was performed by the second-order Runge–Kutta–Nyström method generated by the Butcher array:[5]

$$\begin{array}{c|cccc} 1/2 & 0 & & & \\ 1/2 & 0 & 1/30 & & \\ 1/2 & 0 & 0 & 1/12 & \\ \hline & 0 & 0 & 0 & 1/2 \\ & 0 & 0 & 1 & 1 \end{array} \tag{47}$$

This method has zero dissipation and phase-lag order $q = 6$. The periodicity interval is given by $[0, (2{\cdot}75)^2]$.

The accuracy of the results is measured by the number of correct digits, i.e. by

$$cd := -\log_{10}|\text{maximal absolute error at the end point } t = T| \tag{48}$$

In the table of results, cd(P) and cd(D) correspond to results obtained by imposing periodic and

Table II. Numerical results

Problem	T	$2\pi/\Delta x$	$T/\Delta t$	$\{f_1^{(r)}\}$	cd(P)	cd(D)
1. $w_{tt} = w_{xx}$	1	8	16	{0, 0, 0}	1·80	1·30
$w = \sin(\sin(x+t))$				{1, 2, 3}	2·99	2·89
$0 \leqslant x \leqslant 2\pi$		16	32	{0, 0, 0}	2·92	2·17
$0 \leqslant t \leqslant T$				{1, 3, 5}	4·19	4·19
2. $w_{tt} = \dfrac{(1+w^2)w_{xx}}{1+\sin^2(\sin(x+t))}$	1	8	16	{0, 0, 0}	1·83	1·32
				{1, 2, 3}	3·00	2·90
		16	32	{0, 0, 0}	2·92	2·18
				{1, 3, 5}	4·20	4·20
$w = \sin(\sin(x+t))$						
$0 \leqslant x \leqslant 2\pi$	10	16	32	{0, 0, 0}	1·88	1·83
$0 \leqslant t \leqslant T$				{1, 3, 5}	4·17	4·06
3. $w_{tt} = \dfrac{(1+w^2)w_{xx}}{1+\tan^2(\sin(x+t))}$	1	16	32	{0, 0, 0}	1·95	1·46
				{1, 2, 3}	2·21	1·74
$w = \tan(\sin(x+t))$				{1, 3, 5}	2·70	2·45
$0 \leqslant x \leqslant 2\pi, 0 \leqslant t \leqslant T$				(0, 2, 4)	1·98	1·75
4. $w_{tt} = w_{xx}$						
$w = \sin 4(x+t) + \sin 5(x+t)$	1	16	32	{0, 0, 0}	−0·06	−0·65
$+ \sin 6(x+t)$				[3, 7]	0·62	−0·02
$0 \leqslant x \leqslant 2\pi, 0 \leqslant t \leqslant T$				[4, 6]	1·58	1·06
				[4·5, 5·5]	1·11	0·60
5. $w_{tt} = w^2\left[w_{xx} - \dfrac{1}{w} + w\right.$	1	8	16	{0, 0, 0}	−0·62	1·16
$\left. + 0{\cdot}44\sin(1{\cdot}2x+t)\right]$				[0·9, 1·3]	−0·61	3·26
$w = \sin(x+t) + \sin(1{\cdot}2x+t)$				[1, 1·2]	−0·61	3·28
$0 \leqslant x \leqslant 2\pi, 0 \leqslant t \leqslant T$				[1·05, 1·1]	−0·61	3·34
6. $w_{tt} = w_{xx} + \dfrac{6xt}{8\pi^3}(x^2 - t^2)$	1	8	16	{0, 0, 0}	0·98	1·31
$w = \sin(\sin(x+t)) + \left(\dfrac{xt}{2\pi}\right)^3$				{1, 2, 3}	0·84	1·40
		16	32	{0, 0, 0}	0·54	2·15
$0 \leqslant x \leqslant 2\pi, 0 \leqslant t \leqslant T$				{1, 3, 5}	0·52	1·84

Dirichlet boundary conditions, respectively. We observe that imposing periodic boundary conditions in cases where the initial conditions are not periodic *with respect to the given x-interval* leads to singularities in the exact solution at the boundary points (e.g. problem 5) caused by an inconsistency of the initial-boundary values.

The purpose of the experiments listed in Table II is to show that the use of exponentially fitted space discretizations, instead of conventional discretizations, will *improve the accuracy considerably in all cases where the exact solution is periodic*. This assertion is supported by the

Table III. cd(D) values for various frequency intervals $[\underline{f}_1, \bar{f}_1]$

Problem 7	$\frac{2\pi}{\Delta x}$	$\frac{1}{\Delta t}$	$\{0,0,0\}$	$[0,1]$	$[0,2]$	$[0,3]$
$w_{tt} = w_{xx}$	8	16	2·06	1·92	1·44	0·70
$w = 1/(1+x+t)$	16	32	2·78	2·71	2·48	2·06
$0 \leqslant x \leqslant 2\pi, 0 \leqslant t \leqslant 1$	32	64	3·78	3·74	3·62	3·78

Table IV. Numerical results for problem 2 with $\Delta x = 2\pi/16$, $\Delta t = 1/32$

t	$\{f_1^{(r)}\} = \{0,0,0\}$ cd(P)	cd(D)	$\{f_1^{(r)}\} = \{1,3,5\}$ cd(P)	cd(D)
0·2	3·75	3·25	4·47	4·47
0·4	3·33	2·65	4·26	4·26
0·6	3·02	2·30	4·17	4·17
0·8	2·91	2·16	4·16	4·16
1·0	2·92	2·18	4·20	4·20
2·0	2·54	2·03	4·27	4·29
4·0	2·25	1·90	4·14	4·25

results obtained for the problems 1–5. Even in a case where the true frequencies differ completely from the predicted frequencies, such as in the last row of problem 3, the exponentially fitted formulae are competitive with the conventional formulae. Also, notice that changing from periodic to Dirichlet boundary conditions decreases the accuracy of conventional space discretizations much more than the accuracy of the exponentially fitted discretizations.

The last problem of Table II was obtained from problem 1 by adding a *non-oscillatory term* to the exact solution. As a consequence, only the oscillatory part of the solution will be computed with increased accuracy by the exponentially fitted method, whereas the non-oscillatory part is computed with considerably reduced accuracy. The results in Table II indicate that in such cases there is no advantage in using exponentially fitted methods. Notice that imposing periodic boundary conditions leads to bad accuracies because of the inconsistency in the initial-boundary values.

Next, we integrated a problem with no space oscillations at all. In Table III, results are given for the conventional discretization (arising if the frequencies are $\{0,0,0\}$) and for three frequency intervals. At first, if the grid is rather coarse, the exponentially fitted method is considerably less accurate than the conventional method. On finer grids, however, the accuracies become more and more comparable because the exponentially fitted method converges to the conventional method.

Finally, we considered the error behaviour as a function of t. The results for problem 2 listed in Table II already indicate that the conventional method is more sensitive to long interval integration than the exponentially fitted method. Table IV presents more detailed information on the error behaviour of the various methods.

6. CONCLUDING REMARKS

Below we summarize the main properties of exponentially fitted space discretizations in comparison with conventional discretizations using the same number of grid points:

(i) The *additional costs* are negligible.
(ii) The *order of accuracy* does not change (cf. Table III).
(iii) The *accuracy* improves considerably if
  (a) only a few (approximately known) frequencies dominate the solution (cf. problems 1–4)
  (b) all dominating frequencies are located in a small interval (cf. problems 5).
(iv) The properties (i), (ii) and (iii) also hold for *non-linear* problems.
(v) If the solution contains *non-periodic components*, then there is no advantage in using exponentially fitted space discretizations (cf. problem 6).
(vi) If the solution contains no periodic components, then conventional discretization methods are to be preferred.

Furthermore, we remark that the frequency estimator described in Section 3.3 yields reliable results and can be used as a part of a computer implementation of exponentially fitted space discretizations.

Finally, we observe that a similar approach can be followed in designing space discretizations that are fitted to exponentials with real arguments (i.e. in (5) $i\mathbf{f}^{(r)}$ is real valued).

ACKNOWLEDGEMENTS

We are grateful to the referee for his comments and suggestions which improved the paper. These investigations were supported in part by the Netherlands Foundation for Technical Research (STW), future Technical Science Branch Division of the Netherlands Organization for the advancement of Pure Research (ZWO).

REFERENCES

1. P. J. van der Houwen, 'Spatial discretization of hyperbolic equations with periodic solutions', *Int. j. numer. methods eng.*, **23,** 1395–1406 (1986).
2. W. Gautschi, 'Numerical integration of ordinary differential equations based on trigonometric polynomials', *Numer. Math.*, **3,** 381–397 (1961).
3. L. Brusa and L. Nigro, 'A one-step method for direct integration of structural dynamic equations', *Int. j. numer. methods eng.*, **15,** 685–699 (1980).
4. I. Gladwell and R. M. Thomas, 'Damping and phase analysis for some methods for solving second-order ordinary differential equations', *Int. j. numer. methods eng.*, **19,** 493–503 (1983).
5. P. J. van der Houwen and B. P. Sommeijer, 'Explicit Runge–Kutta (–Nyström) methods with reduced phase errors for computing oscillating solutions', *Report RNM8504*, Centre for Mathematics and Computer Science, Amsterdam, (to appear in *SIAM J. Numer. Anal.*).

Analysis of Smoothing Matrices for the Preconditioning of Elliptic Difference Equations

P.J. van der Houwen, C. Boon, F.W. Wubs
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Smoothing techniques have been used for stabilizing explicit time integration of parabolic and hyperbolic initial-boundary value problems. Similar techniques can be used for the preconditioning of elliptic difference equations. Such techniques are analysed in this paper. It is shown that the spectral radius of the Jacobian matrix associated with the system of equations can be reduced considerably by this type of preconditioners, without much computational effort. Theoretically, this results in a much more rapid convergence of function iteration methods like the Jacobi type methods. The use of smoothing techniques is illustrated for a few one-dimensional and two-dimensional problems, both of linear and nonlinear type. The numerical results show that the use of rather simple smoothing matrices reduce the number of iterations by at least a factor 10.

1980 Mathematics Subject Classification: Primary 65N10.
Key Words & Phrases: numerical analysis, elliptic boundary value problems, preconditioning, smoothing.
Note: These investigations were partly supported by the Netherlands Foundation for Technical Research (STW), future Technical Science Branch Division of the Netherlands Organization for the Advancement of Pure Research (ZWO).

1. Introduction

In a number of papers smoothing techniques have been used for stabilizing explicit integration methods in order to solve efficiently *parabolic* and *hyperbolic* initial-boundary-value problems (cf.[2,3,4,6,8]). In this paper, we shall analyse similar smoothing techniques for accelerating iteration methods in order to solve *elliptic* boundary-value problems. The resulting iteration methods can be interpreted as *residue smoothing methods,* and belong, in this respect, to the same class of methods as the well-known multigrid methods and the unigrid method of McCormick and Ruge [1]. However, unlike these methods, the smoothed iteration methods discussed here are extremely simple to implement on a computer and turn out to be rather effective.

Our starting point is the system of (nonlinear) equations

$$\mathbf{f}(\mathbf{u}) = \mathbf{0} \tag{1.1}$$

obtained by discretizing the elliptic boundary-value problem. A number of iteration methods for solving (1.1) express the $(n+1)$st iterate $\mathbf{u}_{n+1}$ *explicitly* in terms of one or more preceding iterates and the corresponding residue vectors $\mathbf{f}(\mathbf{u}_n),\ldots$. For instance, the Jacobi-type iteration methods such as

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \omega \mathbf{f}(\mathbf{u}_n), \tag{1.2}$$

ω being a relaxation parameter, and many "time-stepping" methods are of such a form (cf. [5,p.221] for other types of Jacobi methods). The convergence of these explicit iteration methods may be rather slow if the condition number of the Jacobian matrix $\partial \mathbf{f}/\partial \mathbf{u}$ is large, i.e. the value of ρ/δ, ρ and δ denoting the magnitude of the largest and smallest eigenvalue of $\partial \mathbf{f}/\partial \mathbf{u}$, is much bigger than 1. It is the purpose of this paper to analyse smoothing techniques for "preconditioning" the system (1.1) such that the condition number associated with the preconditioned system is much smaller than that of the original system. The smoothing operators analysed here are of the form

Report NM-R8705
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

$$S := P_k(D), \tag{1.3}$$

where P_k is a polynomial of degree k satisfying $P_k(0) = 1$ and D is a difference matrix with eigenvalues on the unit disk.

Thus, instead of (1.1) we shall solve the preconditioned equation

$$\mathbf{f}^*(\mathbf{u}) := S\mathbf{f}(\mathbf{u}) = \mathbf{0}. \tag{1.4}$$

The difference operator D should be such that for any test vector $\mathbf{v} = (v_j) := (w(\mathbf{x}_j))$, $w(\mathbf{x})$ being a sufficiently smooth function of $\mathbf{x}$, we have $D\mathbf{v} \to \mathbf{0}$ as the grid is refined. Then the smoothing matrix S will converge to the identity matrix I by virtue of our condition $P_k(0) = 1$.

EXAMPLE 1.1. Consider a one-dimensional problem (two-point boundary-value problem)

$$(e^u)_{xx} + g(u) = 0,\ 0 \leqslant x \leqslant 1$$
$$u(0) = 0,\ u(1) = 1.$$

Standard symmetric discretization yields the system

$$\begin{cases} f_0(\mathbf{u}) := u_0 = 0, \\ f_j(\mathbf{u}) := \dfrac{1}{\Delta^2}(e^{u_{j-1}} - 2e^{u_j} + e^{u_{j+1}}) + g(u_j) = 0,\ j = 1,\ldots,M, \\ f_{M+1}(\mathbf{u}) := u_{M+1} - 1 = 0, \end{cases} \tag{1.1'}$$

where $\Delta := 1/(M+1)$. As we shall show in Section 3 (cf. Table 3.4) the Jacobi process (1.2) converges extremely slow for this problem. Acceleration of convergence can be obtained by solving (1.4) with

$$D := \frac{1}{4}\begin{bmatrix} 0 & & & & 0 \\ 1 & -2 & 1 & & \\ & & & & \\ & & 1 & -2 & 1 \\ 0 & & & & 0 \end{bmatrix}_{(M+2)\times(M+2)}, \quad P_k(z) := 1 + z. \tag{1.5}$$

With this choice the "preconditioning" matrix $S := P_k(D) = I + D$ assumes the form of an "averaging" matrix:

$$S = \frac{1}{4}\begin{bmatrix} 4 & 0 & & & 0 \\ 1 & 2 & 1 & & \\ & & & & \\ & & 1 & 2 & 1 \\ 0 & & & 0 & 4 \end{bmatrix}_{(M+2)\times(M+2)}. \quad \square \tag{1.6}$$

In Section 2 we derive optimal polynomials $P_k(z)$ for the *model situation:*

$$D := \frac{1}{\rho}\frac{\partial \mathbf{f}}{\partial \mathbf{u}},\ \rho := \rho\left(\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right) \tag{1.7}$$

where $\partial\mathbf{f}/\partial\mathbf{u}$ has its eigenvalues in the negative interval $[-\rho, 0)$. We emphasize, however, that in actual application we shall not use the difference matrix D defined by (1.7), because it is much too expensive in a general nonlinear case. Instead, we shall employ the "optimal" polynomials $P_k(z)$ together with a "cheap" matrix D possessing the same type of spectrum as $\rho\partial\mathbf{f}/\partial\mathbf{u}$. In fact, we want to use matrices D that are to a large extent *independent of the problem to be solved.* For instance, in all one-dimensional problems with Dirichlet boundary conditions, we employ the matrix (1.5), and in all two-dimensional elliptic problems with Dirichlet conditions we employ the two-dimensional analogue of (1.5), i.e. a

matrix with zero rows if the row correspond to boundary points and with rows of the form

$$\frac{1}{8}(0,...,0,1,0,...,0,1,-4,1,0,...,0,1,0,...,0) \tag{1.8}$$

in nonboundary points. Smoothing matrices $S = P_k(D)$ using matrices D of this form leave all components of the residue vector $\mathbf{f}(\mathbf{u}_n)$ corresponding to boundary points unchanged. Hence, the "boundary" components of $f(\mathbf{u}_n)$ are fixed during the smoothing process. This approach is satisfactory in the case of Dirichlet conditions (cf. Section 3; we observe that in this case $\mathbf{f}$ has zero-boundary components, hence the diagonal elements of the corresponding rows in D can be replaced by nonzero values so that it becomes a nonsingular matrix).

2. Construction of smoothing matrices

2.1. The model situation

We start with the analysis of smoothing matrices of the form $S = P_k(D)$ where D is defined by (1.7). From now on we will assume that $\partial\mathbf{f}/\partial\mathbf{u}$ has *negative* eigenvalues. We want a polynomial $P_k(z)$ such that $P_k(D)\partial\mathbf{f}/\partial\mathbf{u}$ has also *negative* eigenvalues and the smallest possible condition number.

THEOREM 2.1. *Let $\partial\mathbf{f}/\partial\mathbf{u}$ have its eigenvalues in the negative interval $[-\rho,-\delta]$ and let D be defined by (1.7). Then, of all polynomials $P_k(z)$ with $P_k(0) = 1$ and $P_k(z) \geqslant 0$ on $[-1,0]$, the polynomial*

$$P_k(z) = \frac{T_{k+1}(w_0+w_1z)-T_{k+1}(w_0)}{w_1T'_{k+1}(w_0)z}, \quad w_0 := \frac{\rho+\delta}{\rho-\delta}, \; w_1 = w_0+1 \tag{2.2}$$

generates a smoothing matrix such that the condition number of $S\partial\mathbf{f}/\partial\mathbf{u}$ is minimal. This condition number is given by ρ^*/δ^*, where

$$\rho^* = (\rho-\delta)\frac{1+T_{k+1}(w_0)}{2T'_{k+1}(w_0)}, \; \delta^* = (\rho-\delta)\frac{-1+T_{k+1}(w_0)}{2T'_{k+1}(w_0)}. \; \square \tag{2.3}$$

The proof of this theorem can straightforwardly be given by using well-known properties of the Chebyshev polynomial $T_{k+1}(x)$. In fact, the polynomial (2.2) resembles the polynomials employed in Chebyshev iteration (cf. [7]).

For elliptic problems the condition number ρ/δ of $\partial\mathbf{f}/\partial\mathbf{u}$ is usually very large. In such cases, the smoothing matrix defined by (2.1) and (2.2) is rather effective, because the condition number of $S\partial\mathbf{f}/\partial\mathbf{u}$ can be made as small as we want by increasing k:

$$\frac{\rho^*}{\delta^*} = \frac{1+T_{k+1}(w_0)}{-1+T_{k+1}(w_0)} \approx \frac{2+2\frac{\delta}{\rho}T'_{k+1}(1)}{2\frac{\delta}{\rho}T'_{k+1}(1)} \approx \frac{1}{(k+1)^2}\,\frac{\rho}{\delta}. \tag{2.4}$$

2.2. The nonmodel situation

As we already remarked in the Introduction we do not want to define D by (1.7). In this section, we assume that D is a matrix with *negative* eigenvalues in the interval $[-1,0]$. Let these eigenvalues be denoted by μ and suppose that the residue vector $\mathbf{f}(\mathbf{u}_n)$ is expanded in the eigenvectors of D. Then, by applying the smoothing matrix $S = P_k(D)$ to $\mathbf{f}$, these eigenvectors are multiplied by

$$P_k(\mu) = \frac{T_{k+1}(w_0+w_1\mu)-T_{k+1}(w_0)}{w_1T'_{k+1}(w_0)\mu}. \tag{2.5}$$

In Figure 2.1 the polynomial $P_k(\mu)$ is plotted for $k = 5$ and $\delta/\rho \ll 1$. From this picture we see that the smoothing matrix has a strong damping effect on all eigenvector components of $\mathbf{f}(\mathbf{u}_n)$ with eigenvalues close to -1. Since, usually, these eigenvector components represent the high frequencies of $\mathbf{f}(\mathbf{u}_n)$ we conclude that S reduces the high frequencies of the residue vector.

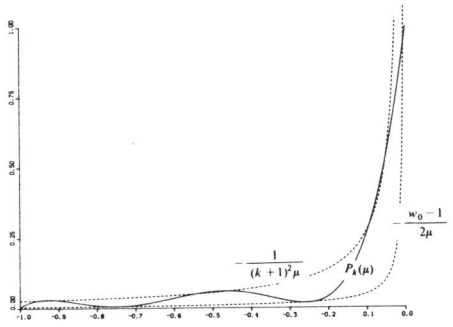


FIGURE 2.1. Behaviour of $P_k(\mu)$ for $\delta/\rho << 1$.

2.3. Generation of smoothed residues by recursion

In order to generate the smoothed residue $\mathbf{f}^* = P_k(D)\mathbf{f}$ we may employ the following recursion:

$$\begin{cases} \mathbf{f}_0 = \mathbf{f},\ \mathbf{f}_1 = 2(2w_0 + w_1 D)\mathbf{f}, \\ \mathbf{f}_{j+1} = 2(w_0 + w_1 D)\mathbf{f}_j - \mathbf{f}_{j-1} + 2T_{j+1}(w_0)\mathbf{f},\ j = 1,\ldots,k-1, \\ \mathbf{f}^* = \mathbf{f}_k / T'_{k+1}(w_0). \end{cases} \tag{2.6}$$

This recursion is easily derived from the three-terms Chebyshev recursion. It is numerically stable if D has its eigenvalues in the interval $[-1,(w_0-1)/(w_0+1)]$.

In our experiments we have always set $w_0 = 1$ and $w_1 = 2$. The matrix D defined by (1.5) (or its two-dimensional analogue) can then be used without danger of instability. However, a consequence of this choice is that $P_k(D)$ may have zero-eigenvalues (see Figure 2.1). This may give problems in the convergence when using an iteration method on $\mathbf{f}^*(\mathbf{u}) = \mathbf{0}$. In Section 3 we will discuss this aspect for the Jacobi method.

The recursion (2.6) requires k matrix-vector multiplications and is extremely simple to implement for problems in one or more dimensions with irregular boundaries. Moreover, the storage requirements are rather modest.

2.4. Generation of smoothed residues by factorization

The smoothed residue $\mathbf{f}^*$ can be obtained by far less than k matrix-vector multiplications if $k = 2^q - 1$ for some positive integer q and if $w_0 = w_1 - 1 = 1$. Let the matrices F_j be defined by

$$F_1 = I + D,\ F_{j+1} = (I - 2F_j)^2,\ j \geqslant 0. \tag{2.7a}$$

Then for all matrices D the smoothing matrix $S = P_k(D)$ can be factorized according to

$$S = F_q \cdot F_{q-1} \cdot \ldots \cdot F_1. \tag{2.7b}$$

Thus, only $q \approx^2 \log(k+1)$ matrix-vector multiplications are required to generate $\mathbf{f}^* = S\mathbf{f}$. The proof of this factorization property follows from a similar property of Chebyshev polynomials (cf. [2,

Lemma 3.2]).

2.4.1. One-dimensional problems

The factorization (2.7) requires the precomputation of the factor matrices F_j. In one-dimensional problems, this offers no difficulties. We easily find in the case (1.7):

$$F_1 = \frac{1}{4}\begin{bmatrix} 4 & 0 & & & 0 \\ 1 & 2 & 1 & & \\ & & & & \\ & & 1 & 2 & 1 \\ 0 & & & 0 & 4 \end{bmatrix}, \quad F_2 = \frac{1}{4}\begin{bmatrix} 4 & 0 & & & & & & & 0 \\ 2 & 1 & 0 & 1 & & & & & \\ 1 & 0 & 2 & 0 & 1 & & & & \\ & & & & & & & & \\ & & 1 & 0 & 2 & 0 & 1 & & \\ & & & 1 & 0 & 1 & 2 & & \\ 0 & & & & & 0 & 4 & & \end{bmatrix},$$

$$F_j = \frac{1}{4}\begin{bmatrix} 4 & & & & & & & & & & & & & & \\ 2 & 2 & & & & & -1 & 0 & 1 & & & & & & \\ . & & . & & & & & . & & & & & & & \\ 2 & & & 2 & & -1 & & & & & 1 & & & & \\ 2 & & & & 1 & & & & & & & 1 & & & \\ 2 & & & -1 & & 2 & & & & & & & 1 & & \\ . & & & & . & & & & . & & & & & . & \\ 2 & -1 & & & & & 2 & & & & & & & 1 & \\ 1 & & & & & & & 2 & & & & & & & 1 \\ & . & & & & & & & . & & & & & & \\ & & 1 & & & & & & & 2 & & & & & 1 \\ & & & & & . & . & . & & & & & & & \end{bmatrix},$$

where $j \geq 3$ and where the position of the element -1 in the second row is at the 2^{j-1}-th column.

Notice that only a few nonzero elements occur on each row.

2.4.2. Two-dimensional problems

In two or more dimensions the derivation of the matrices F_j defined by (2.7a) is not attractive. Therefore, we consider an alternative which only uses one-dimensional smoothing matrices.

We confine ourselves to two-dimensional problems. Let the residue vector $\mathbf{f}$ be arranged in a two-dimensional array in the natural way. First we compute an intermediate array $\mathbf{f}^*$ by applying to all rows of $\mathbf{f}$ the one-dimensional smoothing matrix S discussed in the preceding subsection. Next, we do the same with all columns of $\mathbf{f}^*$ to obtain the array $\mathbf{f}^{**}$. The preconditioned system of equations is then given by

$$\mathbf{f}^{**}(\mathbf{u}) := \tilde{S}\mathbf{f}(\mathbf{u}) = \mathbf{0}. \tag{2.8}$$

Of course, the corresponding smoothing matrix $\tilde{S}$ is essentially different from S and it is of interest to know the damping effect of S on high frequencies in $\mathbf{f}$.

In order to get some insight into the properties of $\tilde{S}$ we expand $\mathbf{f}$ in a discrete Fourier series:

$$\mathbf{f} = \sum_{\omega} c(\omega)\mathbf{e}(\omega), \quad \mathbf{e}(\omega) := (exp(i\omega \cdot \mathbf{x}_j)) \tag{2.9}$$

where ω represents the frequency vector and $\mathbf{x}_j$ runs through the grid points on which $\mathbf{f}$ is defined. Let $\{\mathbf{x}_j\}$ be a uniform grid $\{j\Delta, l\Delta\}$ with square meshes. Applying the smoothing matrix $\tilde{S}$ to $\mathbf{f}$ has the effect that the components $c(\omega)\mathbf{e}(\omega)$ of $\mathbf{f}$ are essentially multiplied by the factor $P_k(\mu_x)P_k(\mu_y)$, where μ_x and μ_y are the eigenvalues of the difference matrices D_x and D_y, respectively used in the row-smoothing of $\mathbf{f}$ and the column-smoothing of $\mathbf{f}^*$. For (μ_x, μ_y) away from the origin the estimate

$$P_k(\mu_x)P_k(\mu_y) \leqslant \frac{1}{(k+1)^4 \mu_x \mu_y}, \quad -1 \leqslant \mu_x, \mu_y < 0 \tag{2.10}$$

gives an indication of the increased damping power of the smoothing matrix $\tilde{S}$ (cf. Figure 2.2). It should be remarked, however, that this does not automatically imply an increased damping of the iteration error when $\tilde{S}$ is combined with, e.g., Jacobi iteration (cf. Section 3).

We conclude our discussion of the matrix S by deriving an estimate for the spectral radius ρ^{**} of $\partial \mathbf{f}^* / \partial \mathbf{u}$ in the model situation where

$$\frac{1}{2}(D_x + D_y) = \frac{1}{\rho}\frac{\partial \mathbf{f}}{\partial \mathbf{u}}, \quad \rho := \rho\left(\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right). \tag{2.11}$$

Let $w_0 = w_1 - 1 = 1$ in (2.2), then the eigenvalues of $\tilde{S}\partial \mathbf{f}/\partial \mathbf{u}$ are given by

$$\lambda^{**} := P_k(\mu_x)P_k(\mu_y)\frac{\mu_x + \mu_y}{2}\rho = \frac{T_{k+1}(1+2\mu_x)-1}{2(k+1)^2\mu_x}\,\frac{T_{k+1}(1+2\mu_y)-1}{2(k+1)^2\mu_y}\,\frac{\mu_x+\mu_y}{2}\rho, \tag{2.12}$$

where $-1 \leqslant \mu_x, \mu_y \leqslant 0$ and $\mu_x + \mu_y \leqslant -2\delta/\rho$.

For small values of k the value of $\rho^{**} := \max|\lambda^{**}|$ can straightforwardly be determined. For instance,

$$k=1\text{: } \rho^{**} = \frac{4\rho}{27} \text{ assumed at } \mu_x = \mu_y = -\frac{1}{3} \tag{2.13a}$$

$$k=2\text{: } \rho^{**} = \frac{192\rho}{3125} \text{ assumed at } \mu_x = \mu_y = -\frac{3}{20}.$$

For larger values of k we investigated λ^{**} numerically: we found

$$k \geqslant 3\text{: } \rho^{**} \approx .55\frac{\rho}{(k+1)^2} \text{ assumed at } \mu_x = \mu_y \approx -\frac{1.35}{(k+1)^2}. \tag{2.13b}$$

The behaviour of $\lambda^{**}(\mu_x, \mu_y)$ with $\mu_x = \mu_y$ is plotted in Figure 2.2.

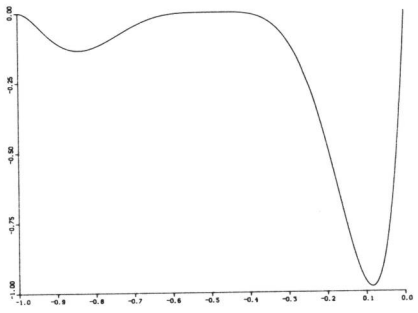


FIGURE 2.2. Eigenvalues λ^{**} along $\mu_x = \mu_y$ for $k = 3$ and $\rho = (k+1)^2 / .55$

3. Smoothed Jacobi iteration

We shall discuss the application of the Jacobi-type iteration method (1.2) to the systems (1.4) and (2.8) in the following two subsections.

3.1. *The case* $\mathbf{f}(\mathbf{u})=\mathbf{0}$

In first approximation, the error equation associated with the iteration process

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \omega S\mathbf{f}(\mathbf{u}),\ S = P_k(D),\ n \geqslant 0 \tag{3.1}$$

reads

$$\mathbf{u}_{n+1} - \mathbf{u} = A_n(\mathbf{u}_n - \mathbf{u}),\ A_n := I + \omega P_k(D)\frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{u}_n), \tag{3.2}$$

where $\mathbf{u}$ denotes the exact solution of (1.1).

In order to get an indication how the relaxation parameters ω should be chosen, we consider the model situation (1.7). Setting $w_0 = w_1 - 1 = 1$ in the expression (2.2) for $P_k(z)$, we find that the eigenvalues $\alpha_n(\mu)$ of A_n are given by

$$\alpha_n(\mu) = 1 + \omega\rho\frac{T_{k+1}(1+2\mu)-1}{2(k+1)^2}, -1 \leqslant \mu \leqslant -\frac{\delta}{\rho}, \tag{3.3}$$

where μ denotes the eigenvalues of $\partial\mathbf{f}/\rho\partial\mathbf{u}$. From this expression it follows that $\alpha_n(\mu)$ equals 1 in all points $\mu \in [-1, -\delta/\rho]$ where $T_{k+1}(1+2\mu)$ equals 1 irrespective of the value of ω. This implies that we should not iterate with a fixed value of k. Therefore, we consider cyclic methods where $\omega = \omega(n)$ and $k = k(n)$, $\omega(n)$ and $k(n)$ being periodic functions of n: $\omega(n) = \omega(N+n)$, $k(n) = k(N+n)$ with N fixed. Instead of α_n we consider the "average" amplification factor

$$\alpha(\mu) = |\prod_{n=0}^{N-1} \alpha_n(\mu)|^{1/N}. \tag{3.4}$$

During a cycle of N iterations we shall impose the condition

$$\omega = \frac{2C(k+1)^2}{\rho} \tag{3.5}$$

where C is a constant. Furthermore, we shall require that $|\alpha_n(\mu)|$ is bounded by 1 for all n, i.e. we require $0 < C \leqslant 1$. Thus, the iteration process assumes the form

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \frac{C}{\rho}D^{-1}[T_{k+1}(I+2D) - I]\mathbf{f}(\mathbf{u}_n). \tag{3.1'}$$

It is easily verified that $\alpha'(0)$ is maximized for $C = 1$. Hence, for $C = 1$ we have a maximal damping in the neighbourhood of $\mu = 0$, however, at the same time, we have $\alpha(-1) = 1$. Furthermore, we found numerically that $C = \frac{1}{2}$ yields a maximal "overall" damping. In Figure 3.1 we have plotted these two extreme cases for $k(n) = 2^n - 1$, $n = 0,...,4$ (notice that $C = 1$ yields an $\alpha(\mu)$ function which is symmetric w.r.t. $\mu = -\frac{1}{2}$. Part a of this figure clearly indicates that, except for a small region near $\mu = 0$, $C = \frac{1}{2}$ indeed leads to a substantially stronger damping than the $C = 1$ value. Part b of Figure 3.1 shows that only eigenvector components of the iteration error which correspond to eigenvalues λ of $\partial\mathbf{f}/\partial\mathbf{u}$ lying in the interval $\approx[-\cdot 004\rho, 0)$ are stronger damped by choosing $C = 1$.

Next, we consider the average damping factor $\alpha(\mu)$ in the case where $k(n) = n$, $n = 0,1,...,15$. Figure 3.2 presents the analogue of Figure 3.1 and shows roughly the same picture.

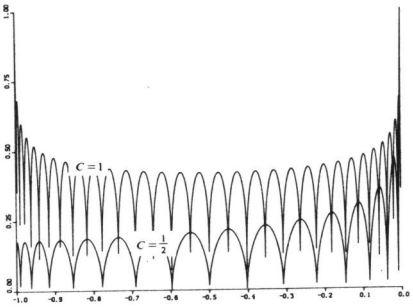


FIGURE 3.1a. Behaviour of $\alpha(\mu)$ on the interval $-1 \leqslant \mu \leqslant 0$ in the case $k(n) = 2^n - 1, n = 0,...,4$

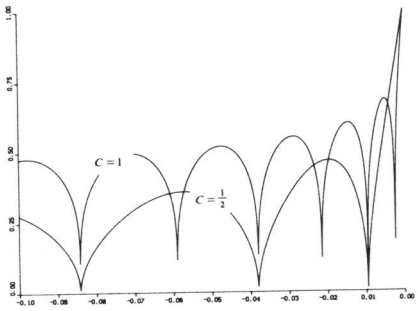


FIGURE 3.1b. Behaviour of $\alpha(\mu)$ on the interval $-\cdot 1 \leqslant \mu \leqslant 0$ in the case $k(n) = 2^n - 1, n = 0,...,4$

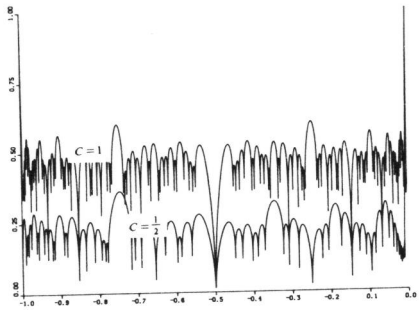


FIGURE 3.2a. Behaviour of $\alpha(\mu)$ on the interval $-1 \leqslant \mu \leqslant 0$ in the case $k(n) = n$, $n = 0,1,...,15$

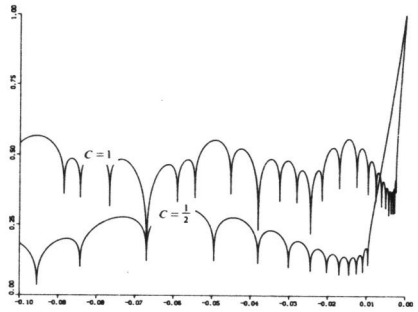


FIGURE 3.2b. Behaviour of $\alpha(\mu)$ on the interval $-1 \leqslant \mu \leqslant 0$ in the case $k(n) = n$, $n = 0,...,15$

We illustrate the smoothed Jacobi method (3.1′) by two examples: a model problem and a nonmodel problem. We applied both the recursive smoothing process (2.6) with $k(n) = n$, $n = 0,...,N-1$, and the factorized smoothing process (2.7) with $k(n) = 2^n - 1$, $n = 0,...,N-1$. The resulting methods are denoted by RSJ(N,C) and FSJ(N,C), respectively (notice that RSJ(1,C) and FSJ(1,C) both represent the conventional Jacobi method). All methods started with the initial approximation (cf. Example 1.1)

$$\mathbf{u}_0 = ((M+1-j)\Delta \cdot u_0 + j\Delta \cdot u_{M+1})_{j=0}^{M+1}, \tag{3.6}$$

and stopped when the scaled residue

$$r(n) := \frac{\|\mathbf{f}(\mathbf{u}_n)\|_\infty}{\|\mathbf{f}(\mathbf{u}_0)\|_\infty} \tag{3.7}$$

dropped below a value specified in the tables of results.

TABLE 3.1. $n/(r(n))^{1/n}$ - values for the problem

$u_{xx} - 20x^3 = 0$, $0 \leqslant x \leqslant 1$

$u(0) = 0$, $u(1) = 1$

$\rho = 4/(\Delta x)^2$, $r(n) \leqslant 10^{-4}$

Method	$\Delta x = 1/20$	$\Delta x = 1/40$	$\Delta x = 1/80$
RSJ(1,.95)	678/.986	n>2000	
RSJ(16,.95)	14/.50	29/.72	112/.92
FSJ(5,.95)	25/.68	30/.73	150/.94
RSJ(1,.5)	1290/.992	n>5000	
RSJ(16,.5)	15/.50	59/.85	221/.96
FSJ(5,.5)	15/.52	74/.88	295/.97

TABLE 3.2. $n/(r(n))^{1/n}$ values for the problem

$$(e^u)_{xx} - 5x^3e^u(4+5u) = 0,\ 0 \leq x \leq 1$$

$$u(0) = 0,\ u(1) = 1$$

$$\rho = 4e/(\Delta x)^2,\ r(n) \leq 10^{-4}$$

Method	$\Delta x = 1/20$	$\Delta x = 1/40$	$\Delta x = 1/80$
RSJ(1,.95)	865/.989	n>3000	
RSJ(16,.95)	19/.61	41/.80	147/.94
FSJ(5,.95)	24/.67	49/.83	195/.95
RSJ(1,.5)	1645/.994	n>6000	
RSJ(16,.5)	36/.77	77/.89	283/.97
FSJ(5,.5)	44/.81	100/.91	380/.98

3.2. The case $\mathbf{f}^{**}(\mathbf{u}) = \mathbf{0}$

The error equation for the iteration process

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \omega\tilde{S}\mathbf{f}(\mathbf{u}) \tag{3.8}$$

(cf.(3.1)) is of the form (3.2) with $A_n := I + \omega\tilde{S}\partial\mathbf{f}/\partial\mathbf{u}(\mathbf{u}_n)$. Again considering the model situation, we find that the eigenvalues $\alpha_n(\mu_x,\mu_y)$ of A_n are given by

$$\alpha_n(\mu_x,\mu_y) = 1 + \omega\lambda^{**}(\mu_x,\mu_y), \tag{3.9}$$

where λ^{**} is defined by (2.12). As in the preceding subsection we shall require that $|\alpha_n| \leq 1$ for all n. This leads us to the condition $\omega \leq 2/\rho^{**}$. From (2.13) it follows that $\rho^{**} = c(k)\rho/(k+1)^2$, hence $\omega \leq 2(k+1)^2/c(k)\rho$; here, $c(k)$ assumes the values

$$c(0) = 1,\ c(1) = 16/27,\ c(2) = 1728/3125,\ c(k) \approx .55 \text{ for } k > 2.$$

In analogy with (3.5) we shall impose the condition

$$\omega = \frac{2C(k+1)^2}{c(k)\rho}, \tag{3.10}$$

where $k = k(n)$ is a periodic function of n and C is a constant in the interval $(0,1]$.

Let us define the average damping factor (cf. (3.4))

$$\alpha(\mu) = \Big(\operatorname*{Max}_{2\tilde{\mu}(\mu) \leq \mu_x \leq \mu} \prod_{n=0}^{N-1} |\alpha_n(\mu_x, 2\mu - \mu_x)|\Big)^{1/N},\ -1 \leq \mu \leq 0, \tag{3.11}$$

where $\tilde{\mu} = \mu$ if $\mu \geq -\frac{1}{2}$ and $\tilde{\mu} = -\frac{1}{2}$ if $\mu \leq -\frac{1}{2}$. This function was investigated numerically in the case where $k(n) = 2^n - 1$, $n = 0,\ldots,4$. We found the best "overall" damping for values of C in the neighbourhood of .6. In Figure 3.3 the function $\alpha(\mu)$ is plotted for $C = 1$ and $C = .6$.

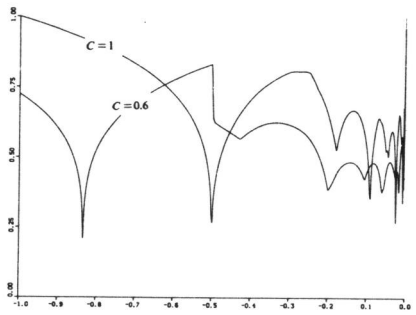


FIGURE 3.3a. Behaviour of $\alpha(\mu)$ on the interval $-1 \leqslant \mu \leqslant 0$ in the case $k(n) = 2^n - 1, n = 0,...,4$

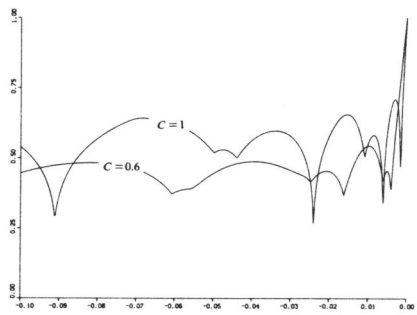


FIGURE 3.3b. Behaviour of $\alpha(\mu)$ on the interval $-\cdot 1 \leqslant \mu \leqslant 0$ in the case $k(n) = 2^n - 1, n = 0,...,4$

The performance of the smoothed Jacobi process ((3.8), (3.10)) is illustrated by a model and a non-model problem. As before, this method is denoted by FSJ(N,C). In addition, we applied the RSJ method employing the two-dimensional version of the matrix D defined in (1.5). The initial approximation $\mathbf{u}_0$ is defined by forming linear interpolations of the boundary values on $x = 0$, $x = 1$ and $y = 0$, $y = 1$, respectively, and by taking the average value of these functions; the iteration process was stopped if the value of $r(n)$ becomes less than a prescribed value which is specified in the tables of results.

TABLE 3.3. $n/(r(n))^{1/n}$-values for the problem

$\Delta u - 6xy(x^2+y^2) = 0,\ 0 \leqslant x,y \leqslant 1$

$u = x^3y^3$ along the boundary

$\rho = 8/(\Delta x)^2,\ \Delta x = \Delta y,\ r(n) \leqslant 10^{-4}$.

method	$\Delta x = 1/20$	$\Delta x = 1/40$	$\Delta x = 1/80$
RSJ(1,.95)	468/.98	n>800	
RSJ(16,.95)	15/.54	16/.54	44/.81
FSJ(5,.95)	31/.74	27/.71	35/.76
RSJ(1,.5)	891/.99	n>800	
RSJ(16,.5)	13/.48	31/.74	
FSJ(5,.6)	16/.54	20/.62	54/.84

TABLE 3.4. $n/(r(n))^{1/n}$-values for the problem

$e^u \Delta u - u^3 = 0,\ 0 \leqslant x,y \leqslant 1$

$u = x^3y^2$ along the boundary

$\rho = 8e/(\Delta x)^2,\ \Delta x = \Delta y, r(n) \leqslant 10^{-3}$.

method	$\Delta x = 1/20$	$\Delta x = 1/40$	$\Delta x = 1/80$
RSJ(1,.95)	252/.97	517/.987	
RSJ(16,.95)	11/.52	12/.54	14/.60
FSJ(5,.95)	18/.68	17/.66	17/.66
RSJ(1,.5)	481/.986	>800	
RSJ(16,.5)	26/.76	26/.76	27/.77
FSJ(5,.6)	36/.82	32/.80	32/.80

REFERENCES

1. S.F. MCCORMICK and J.W. RUGE (1983). Unigrid for Multigrid Simulation, *Mathematics of Computation, 41,* pp. 43-62.
2. 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.
3. A. JAMESON (1983). The Evolution of Computational Methods in Aerodynamics, *J. Appl. Mech., 50,* pp. 1052-1076.
4. A. LERAT (1979). Une Classe de Schémas aux Différences Implicites pour les Systémes Hyperboliques de Lois de Conservation, *C.R. Acad. Sc. Paris t. 288 (18 juin 1979) Série A,* pp. 1033-1036.
5. J.M. ORTEGA and W.C. RHEINBOLDT (1970). *Iterative Solution of Nonlinear Equations in Several Variables.* New York: Academic Press.
6. 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.
7. E.L. WACHSPRESS (1966). *Iterative Solution of Elliptic Systems.* Prentice-Hall International, IUC, London.

8. F.W. Wubs (1986). Stabilization of Explicit Methods for Hyperbolic Partial Differential Equations, *International Journal for Numerical Methods in Fluids, 6,* pp. 641-657.

Explicit-Implicit Methods for Time-Dependent Partial Differential Equations

E.D. de Goede, F.W. Wubs
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

For the integration of partial differential equations we distinguish explicit and implicit integrators. Implicit methods allow large integration steps, but require more storage and are more difficult to implement than explicit methods. However, explicit methods are subject to a restriction on the integration step. In this paper, we introduce explicit-implicit methods, which form a combination of explicit and implicit calculations. For these methods, the impact of varying the explicitness, and thus the implicitness, on the stability is examined.

1980 Mathematics Subject Classification: 65M05, 65M10, 65M20.
Key Words and Phrases : Partial differential equations, explicit-implicit methods, method of lines, tridiagonal equations, incomplete cyclic reduction, stability.
Note : These investigations were sponsored in part by the Netherlands Foundation for Technical Research (STW), future Technical Science Branch Division of the Netherlands Organization for the Advancement of Pure Scientific Research (ZWO).
Note : This report will be submitted for publication elsewhere.

1. Introduction

For the integration of partial differential equations, we distinguish explicit and implicit methods. In general, implicit methods allow large integration steps, but require more storage and are more difficult to implement than explicit methods. However, explicit methods are subject to a restriction on the integration step, because of stability considerations. Implicit methods are in most cases stable for any integration step. This property may be redundant. Here, we concentrate on explicit-implicit methods, which form a combination of explicit and implicit calculations. The objective of such a combination is always to reduce the computational effort to an acceptable level in such a way that the resulting combination still offers attractive stability properties. For the methods presented in this paper, the stability properties vary with the explicitness, and thus the implicitness, of the calculations.

In this paper, we consider one-dimensional problems only. The resulting methods can also be used in alternating direction methods for multi-dimensional cases. We will construct explicit-implicit methods for partial differential equations of the form

$$\frac{\partial \mathbf{u}}{\partial t}(t,\mathbf{x}) = \mathbf{f}(\mathbf{u},\mathbf{u}_x,\mathbf{u}_{xx},\mathbf{x},t), \quad \mathbf{x} \in \Omega \subset \mathbb{R}, \quad t > 0, \tag{1.1}$$

with appropriate boundary conditions. After replacing, on a uniform grid, the spatial derivatives by discrete approximations, we apply an implicit time integrator for the resulting system of ordinary differential equations. We assume that the time integration leads to a linear tridiagonal system of the

Report NM-R8703
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

form

$$A\,\mathbf{Z} = \mathbf{B}\,, \tag{1.2}$$

where A is a square tridiagonal matrix, $\mathbf{Z}$ denotes the unknowns at the advanced time level $n+1$ and $\mathbf{B}$ is a column vector. Next, we separate the uniform grid into two sets of grid points. This choice is determined by stability considerations. Let us assume that the unknowns at the advanced time level on the two sets of grid points are $\mathbf{V}$ and $\mathbf{W}$. The system is now reduced by elimination of $\mathbf{W}$ to a system which involves only the unknowns $\mathbf{V}$. This will be called the reduced system. Next, we approximate the solution for the reduced system by an explicit expression. Once $\mathbf{V}$ is solved, $\mathbf{W}$ is solved by back substitution. By this approach the constructed method is essentially explicit.

A reduced system appears naturally when a few steps of a cyclic reduction method[7] are performed on system (1.2). For the cyclic reduction algorithm, Heller[6] proved the following property : If A in (1.2) satisfies certain diagonal dominance conditions, then the ratio of the off-diagonal elements to the diagonal elements decreases quadratically with each cyclic reduction step. This property is the basis of our approach. It was Hockney's observation that in case of constant diagonals the reduction algorithm could be stopped when the ratio of the off-diagonal elements to the diagonal elements fell below machine precision. Then, the tridiagonal system was essentially diagonal and could be solved without damage to the solution. In many cases this process can be stopped before the mentioned ratio falls below machine precision[6]. The constructed solution method is called incomplete cyclic reduction. It is our approach, to approximate the solution of the reduced system by an explicit expression. Using this approach it is possible to stop the reduction process when the ratio of the off-diagonal elements to the diagonal elements is about a factor 1/6. For example in Table 5.1, it will be shown that our approach requires less cyclic reduction steps in order to obtain accurate results than using incomplete cyclic reduction without any adaptation.

The main purpose of this paper is to construct explicit-implicit methods which have an acceptable stability behaviour. Only for model problems we were able to derive stability conditions. Using our approach, it appeared that the maximum allowed time step increases linearly with the size of the numerical influence domain for hyperbolic equations. For parabolic equations the maximal time step increases quadratically with the size of the influence domain.

Our method can be applied directly to both hyperbolic and parabolic equations. For a given time step, the reduced system for $\mathbf{V}$ can be chosen in such a way that the method is stable. The method can also be used to solve elliptic equations when a time-stepping approach is used (cf. [13],pp. 148-154).

We think that an important application of these techniques are problems which cannot be stored in the central memory of the computer. In such a case one has to evaluate the solution at a new time level block by block. The size of such a block is limited by the size of the central memory. By the given technique one can use the maximal time step which is possible on such a block. Furthermore, with respect to parallel computing one can partition the domain in a number of blocks which can be spread over the available processors.

With respect to vectorization of the solution process, we propose two possibilities which are known in the literature. The first is a modification of the incomplete cyclic reduction method (see [6]) and the second is a modification of a solution method given by Wang[21]. Both have good vectorizing properties (see [8,12,20,21]). In our approach a slight decrease of computation time can be obtained compared with the complete cyclic reduction method and the method of Wang. This decrease depends on the time step (see Remark 4.5).

In Section 2, we show how a system of equations arising from an implicit scheme can be separated in two subsystems which correspond to the values **V** and **W**, respectively. In Section 3, we derive a method for approximating the implicit scheme for **V** by an explicit scheme. In Section 4, the stability condition for this explicit-implicit method is derived. In Section 5, we show by a number of numerical experiments, the impact of varying the explicitness, and thus the implicitness, on the stability. In our numerical experiments we applied the methods to both hyperbolic and parabolic differential equations.

2. Construction of the reduced system

Consider the one-dimensional partial differential equation

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u},\mathbf{u}_x,\mathbf{u}_{xx},\mathbf{x},t), \quad \mathbf{x} \in \Omega \subset \mathbb{R}, \quad t > 0, \tag{2.1}$$

with appropriate boundary conditions. Using the method of lines, (2.1) is space discretized on a uniform grid $\Omega_\Delta := \{ j\Delta x \}_j$. This gives a system of ordinary differential equations [11]

$$\frac{d}{dt}\mathbf{U} = \mathbf{F}(\mathbf{U},t), \quad t > 0, \tag{2.2}$$

where $U_j(t)$ approximates $\mathbf{u}(j\Delta x,t)$ and $\mathbf{F}(\mathbf{U},t)$ is a vector function approximating the right-hand side function. Thereafter, a time integrator is applied to (2.2). We confine ourselves to difference formulae, which involve only two adjacent time levels. For the time integration of (2.2) explicit or implicit time integrators can be used. If the solution of (2.2) varies only slowly in time, then usually implicit time integrators are used, which in most cases are stable for any time step. Hence, for the time discretization of (2.2) we consider the θ-method [13]

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t \{\theta\, \mathbf{F}(\mathbf{U}^{n+1},t^{n+1}) + (1-\theta)\, \mathbf{F}(\mathbf{U}^n,t^n)\}, \quad \tfrac{1}{2} \leqslant \theta \leqslant 1, \tag{2.3}$$

where we have the second-order Trapezoidal Rule for $\theta = \frac{1}{2}$ and the Backward Euler method for $\theta = 1$ (see [13]).

The equations in system (2.3) may be nonlinear. In order to obtain a linear system of equations, we introduce the so-called splitting function $\mathbf{G}(\mathbf{Z},\tilde{\mathbf{Z}},t)$ [10]. We choose **G** in such a way that it is linear in its second variable, i.e.

$$\mathbf{G}(\mathbf{Z},\tilde{\mathbf{Z}},t) = J(\mathbf{Z},t)\tilde{\mathbf{Z}} + \mathbf{g}(\mathbf{Z},t). \tag{2.4}$$

J and **g** are chosen so that the splitting condition

$$\mathbf{G}(\mathbf{Z},\mathbf{Z},t) = \mathbf{F}(\mathbf{Z},t)$$

is satisfied. Equation (2.3) is now approximately solved by the iteration process

$$\begin{aligned} &\mathbf{Z}^{(0)} = \mathbf{U}^n \\ &\mathbf{Z}^{(q)} = \mathbf{U}^n + \Delta t \{\theta\mathbf{G}(\mathbf{Z}^{(q-1)},\mathbf{Z}^{(q)},t^{n+1}) + (1-\theta)\mathbf{G}(\mathbf{U}^n,\mathbf{U}^n,t^n)\} \quad , \; q = 1,\ldots,Q \\ &\mathbf{U}^{n+1} = \mathbf{Z}^{(Q)}. \end{aligned} \tag{2.5}$$

In this equation, the iterate $\mathbf{Z}^{(q)}$ has to be solved from a linear system of equations. In order to approximate (2.3) accurately by (2.5), Q has to be chosen large. However, to obtain a second-order accurate scheme, convergence is not needed. For a linear problem (**G** is independent of its first variable) the scheme (2.5) is second-order accurate for $Q = 1$, whereas for a nonlinear problem (2.5) is second-order accurate for $Q \geqslant 2$. For the latter, a system of equations has to be solved at least twice at each time step. In Section 5.4, we will introduce a variant of (2.5) for which only one system of equations has to be solved at each time step.

In order to apply scheme (2.5), we have to solve for all q, a linear system of equations of the form

$$(I - \theta \Delta t J(\mathbf{Z}^{(q-1)}))\mathbf{Z}^{(q)} = \mathbf{B}, \tag{2.6}$$

where

$$\mathbf{B} = \mathbf{Z}^{(0)} + \Delta t \{\theta \mathbf{g}(\mathbf{Z}^{(q-1)}, t^{n+1}) + (1-\theta)\mathbf{G}(\mathbf{U}^n, \mathbf{U}^n, t^n)\}.$$

In the following, we will simply write J instead of $J(\mathbf{Z}^{(q-1)})$. Furthermore, we assume that J is a tridiagonal matrix and of order m. Now we choose l elements from the column vector $\mathbf{Z}^{(q)}$, which we denote by $\mathbf{V}^{(q)}$. This choice is determined by stability considerations (see Section 4). Let $\mathbf{W}^{(q)}$ be the remaining $(m-l)$ elements. Then system (2.6) can be reordered to

$$M \begin{bmatrix} \mathbf{V}^{(q)} \\ \mathbf{W}^{(q)} \end{bmatrix} = P\,\mathbf{B}, \tag{2.7}$$

where

$$\begin{bmatrix} \mathbf{V}^{(q)} \\ \mathbf{W}^{(q)} \end{bmatrix} = P\,\mathbf{Z}^{(q)} \text{ and } M = P(I - \theta\,\Delta t\, J)P^T, \tag{2.8}$$

with P a permutation matrix and P^T the transposed matrix P. Now we reduce system (2.7), by eliminating $\mathbf{W}^{(q)}$, to a system which only involves $\mathbf{V}^{(q)}$. This can be described by a premultiplication of a matrix R. Let M be partitioned according to the separation of $\mathbf{Z}^{(q)}$, i.e.

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix},$$

where M_{11} and M_{22} are square matrices. Then R is of the form

$$R = \begin{bmatrix} I & -M_{12}M_{22}^{-1} \\ O & R_{22} \end{bmatrix}, \tag{2.9}$$

where O is a nil matrix. An obvious choice for R_{22} is $R_{22} = M_{22}^{-1}$. This choice is used in the solution method in Appendix B. However, for the incomplete cyclic reduction algorithm (see Appendix A) R_{22} is such that the submatrix L occurring below is a lower triangular matrix :

$$RM = \begin{bmatrix} T & O \\ E & L \end{bmatrix}, \tag{2.10}$$

where

$$T = M_{11} - M_{12}M_{22}^{-1}M_{21}, \; E = R_{22}M_{21}, \; L = R_{22}M_{22}.$$

Here, T is a tridiagonal matrix. Now, system (2.7) becomes of the form

$$\begin{bmatrix} T & O \\ E & L \end{bmatrix} \begin{bmatrix} \mathbf{V}^{(q)} \\ \mathbf{W}^{(q)} \end{bmatrix} = RP\mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix}. \tag{2.11}$$

The subsystem

$$T\,\mathbf{V}^{(q)} = \mathbf{B}_1 \tag{2.12}$$

is called the reduced system of equations. Because L is a lower triangular matrix, the elements of $\mathbf{W}^{(q)}$ can be solved straightforwardly once $\mathbf{V}^{(q)}$ is known. In Section 3, we approximate the solution of the reduced system (2.12) by an explicit expression.

EXAMPLE 2.1. Let the indices of the grid points corresponding to the reduced system be given by the set

$$\{j \mid j = l \cdot 2^k, l = 1, \ldots, 2^{p-k} - 1\}\,, \tag{2.13}$$

where the number of grid points is $2^p - 1$. Then, as said in the introduction, the reduced system appears naturally when k steps of the cyclic reduction algorithm (see Appendix A) are performed on (2.6).

3. Approximation of the solution for the reduced system

Let us again consider the difference scheme (2.6), which can be written in the form

$$(I - \theta \Delta t\, J)\mathbf{Z}^{(q)} = (I - \theta \Delta t\, J)\mathbf{Z}^{(0)} + \mathbf{A}\,, \tag{3.1}$$

where

$$\mathbf{A} = \Delta t\,\{\theta J \mathbf{Z}^{(0)} + \theta \mathbf{g}(\mathbf{Z}^{(q-1)}, t^{n+1}) + (1-\theta)\mathbf{G}(\mathbf{U}^n, \mathbf{U}^n, t^n)\}\,,$$

and $J = J(\mathbf{Z}^{(q-1)})$. Application of the reduction technique of the previous section yields

$$R\,M\,P\,\mathbf{Z}^{(q)} = R\,M\,P\,\mathbf{Z}^{(0)} + R\,\mathbf{A}\,.$$

Using (2.8) and (2.10), we may write

$$\begin{aligned} T\,\mathbf{V}^{(q)} &= T\,\mathbf{V}^{(0)} + \tilde{\mathbf{B}}_1 \\ E\,\mathbf{V}^{(q)} + L\,\mathbf{W}^{(q)} &= E\,\mathbf{V}^{(0)} + L\,\mathbf{W}^{(0)} + \tilde{\mathbf{B}}_2 \end{aligned} \tag{3.2}$$

where

$$\begin{bmatrix} \tilde{\mathbf{B}}_1 \\ \tilde{\mathbf{B}}_2 \end{bmatrix} = R\,\mathbf{A}\,.$$

Now the reduced system (3.2), by which $\mathbf{V}^{(q)}$ is implicitly given, will be approximated by an explicit expression. Therefore we assume that

$$T = D + C\,. \tag{3.3}$$

The precise form of C (and consequently of D) will be given later. For this moment we assume that

$$\rho(D) > \rho(C)\,,$$

where ρ denotes the spectral radius (maximal modulus of the eigenvalues) and that D^{-1} exists and can be computed at low costs. Splittings such as in (3.3) are commonly applied for the construction of iterative methods. Rewriting (3.2), gives

$$\mathbf{V}^{(q)} - \mathbf{V}^{(0)} = (I + D^{-1}C)^{-1} D^{-1} \tilde{\mathbf{B}}_1\,. \tag{3.4}$$

Using the truncated Neumann series

$$(1 + D^{-1}C)^{-1} \approx 1 - D^{-1}C\,,$$

we obtain

$$\mathbf{V}^{(q)} - \mathbf{V}^{(0)} \approx (I - D^{-1}C) D^{-1} \tilde{\mathbf{B}}_1\,.$$

Now, the formula

$$\tilde{\mathbf{V}}^{(q)} = \mathbf{V}^{(0)} + (I - D^{-1}C) D^{-1} \tilde{\mathbf{B}}_1\,, \tag{3.5}$$

can be used to compute an approximation for the solution of the reduced system.
Approximating the inverse of a matrix, truncated Neumann series are commonly applied with respect to iterative algorithms on vector computers (see [1,2,4,19]).

$$\tilde{\mathbf{V}}^{(q)} = (I - D^{-1}C)D^{-1}\mathbf{B}_1 + (D-1C)^2\mathbf{V}^{(0)} . \tag{3.6}$$

This expression follows by combining (2.12), (3.2) and (3.5). The expression (3.6) is actually used in our numerical experiments.

For the error due to the approximation, we have

$$\|\mathbf{V}^{(q)} - \tilde{\mathbf{V}}^{(q)}\| = \|((I + D^{-1}C)^{-1} - (I - D^{-1}C))\, D^{-1}\tilde{\mathbf{B}}_1\|$$

$$= \Delta t \,\| \frac{(D^{-1}C)^2}{(I + D^{-1}C)}\, D^{-1} R\,\mathbf{A}\| \leqslant \Delta t \,\| D^{-1}C\|^2 \,\|(I + D^{-1}C)^{-1}\, D^{-1} R\,\mathbf{A}\| .$$

This error is small when $\|D^{-1}C\|$ is small.

The choice of C (and consequently of D) is determined by the following considerations :
1. D should be easily invertible, e.g. a diagonal matrix.
2. The replacement of (3.4) by (3.5) should not disturb a possible numerical conservation property of (3.1).
For a discussion of conservation properties of numerical schemes, we refer to [13,16]. In our case, the second consideration is similar to the requirement that the difference

$$\sum_{i=1}^{m}(Z_i^{(q)} - U_i^n) = \mathbf{e}^T(\mathbf{Z}^{(q)} - \mathbf{U}^n) , \quad \text{where} \quad \mathbf{e}^T = [1,1,...,1] , \tag{3.7}$$

is not changed when (3.4) is replaced by (3.5). The difference (3.7) can be evaluated using (3.1). This requirement can be satisfied by choosing

$$D_{ii} = \sum_{j=1}^{l} T_{ij}, \quad D_{ij} = 0 \text{ for } j \neq i , \tag{3.8}$$

and consequently $C = T - D$. By this choice, we have that

$$\mathbf{e}^T C = \mathbf{0}^T , \tag{3.9}$$

where $\mathbf{0}$ is a zero vector. This property does not necessarily imply that a possible conservation property of (3.1) is not disturbed. Hence, we have to calculate the perturbation of (3.1), introduced by the replacement of (3.4) by (3.5), explicitly. Comparison of (3.4) and (3.5), it follows that instead of (3.2) we have solved

$$(T + H)\,\mathbf{V}^{(q)} = (T + H)\,\mathbf{V}^{(0)} + \tilde{\mathbf{B}}_1 , \tag{3.10}$$

where

$$H = C(D^{-1}C)(I - D^{-1}C)^{-1} . \tag{3.11}$$

Hence, system (3.10 and 3.11) is identical to (3.5). Since $\mathbf{e}^T C = \mathbf{0}^T$ (see 3.9), we have $\mathbf{e}^T H = \mathbf{0}^T$. Furthermore, from the definition of R in (2.9), it is easily verified that

$$R^{-1}\begin{bmatrix} H & O \\ O & O \end{bmatrix} = \begin{bmatrix} H & O \\ O & O \end{bmatrix} ,$$

where O denotes a nil matrix. Hence, the modification of (3.1) is given by

$$P^T \begin{bmatrix} H & O \\ O & O \end{bmatrix} P\,(\mathbf{Z}^{(q)} - \mathbf{Z}^{(0)}) .$$

As $\mathbf{e}^T P^T = \mathbf{e}^T$, we have that

$$\mathbf{e}^T P^T \begin{bmatrix} H & O \\ O & O \end{bmatrix} P\,(\mathbf{Z}^{(q)} - \mathbf{Z}^{(0)}) = 0 ,$$

which proves our assertion.

The matrix D can also be seen to originate from a lumping process on the columns of T. Lumping is often used in finite element methods (see [18,14]) in order to obtain a diagonal matrix. Furthermore, it is used in the context of multigrid methods[3].

Summarizing, the method proceeds as follows :
(a) The system of equations (2.6) is reduced to system (2.11).
(b) D and C are constructed as denoted by (3.3) and (3.8).
(c) The explicit expression (see 3.6) is used to approximate the solution for the reduced system.
(d) $\mathbf{W}^{(q)}$ is solved by back substitution.

REMARK 3.1. In terms of iterative methods for tridiagonal systems, the approximation (3.5) can be considered as one step of the point Jacobi method (see [13],p.138).
This can be explained as follows : If we multiply formula (3.4) with $(I - D^{-1}C)$, we may write

$$(I - (D^{-1}C)^2)\,\mathbf{V}^{(q)} = (I - (D^{-1}C)^2)\,\mathbf{V}^{(0)} + (I - D^{-1}C)D^{-1}\tilde{\mathbf{B}}_1 ,$$

Now applying one step of the point Jacobi method, where we use $\mathbf{V}^{(0)}$ as an initial approximation for $\mathbf{V}^{(q)}$, gives

$$\begin{aligned}\mathbf{V}^{(q)} &= (I - (D^{-1}C)^2)\,\mathbf{V}^{(0)} + (I - D^{-1}C)D^{-1}\tilde{\mathbf{B}}_1 + (D^{-1}C)^2\,\mathbf{V}^{(0)} . \\ &= \mathbf{V}^{(0)} + (I - D^{-1}C)D^{-1}\tilde{\mathbf{B}}_1 ,\end{aligned}$$

which corresponds with formula (3.5).

1. STABILITY
In this section, a stability condition will be derived for the system of equations (2.11), where the reduced system of equations (2.12) is approximated by scheme (3.5). We only consider linear systems. For the treatment of linear stability theory we refer to [16]. Here, we require that $\|\mathbf{U}^{n+1}\| \leq \|\mathbf{U}^n\|$ for the homogeneous problem, i.e. (2.11) without forcing terms.

The next theorem is used to derive a stability condition for system (2.11).

THEOREM 4.1. *Let S and J be matrices, where λ_S and λ_J are the corresponding eigenvalues. Then necessary conditions for stability of the scheme*

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t\, S\,(I - \theta \Delta t\, J)^{-1} J\, \mathbf{U}^n , \quad \tfrac{1}{2} \leq \theta \leq 1 , \tag{4.1}$$

are :
(a) $\mathrm{Re}(\lambda_J) \leq 0$,
(b) $\lambda_S \in [0,1]$ *and real.*
Sufficient conditions for stability of this scheme are the conditions (a) and (b), and
(c) S and J are normal matrices and commute with each other.

PROOF. Being commutative, S and J have the same eigensystem. Thus, we arrive at the stability condition

$$|1 + \Delta t\, \lambda_S (1 - \theta \Delta t\, \lambda_J)^{-1} \lambda_J| \leq 1 . \tag{4.2}$$

This condition means that $\lambda_S (1 - \theta \Delta t \lambda_J)^{-1} \lambda_J$ should be in a circle with centre (-1,0) and radius 1. Due to condition (b), (4.2) is satisfied if

$$|1 + \Delta t (1 - \theta \Delta t \lambda_J)^{-1} \lambda_J| = |\frac{1 + (1 - \theta)\Delta t \lambda_J}{1 - \theta \Delta t \lambda_J}| \leqslant 1 .$$

Since (a) and $\frac{1}{2} \leqslant \theta \leqslant 1$, this condition is satisfied. □

Now, the system of equations (2.11) will be written in a form as denoted by (4.1). In the linear case (2.8), we may write for system (2.11)

$$\begin{bmatrix} T & O \\ E & L \end{bmatrix} P \mathbf{U}^{n+1} = \begin{bmatrix} T & O \\ E & L \end{bmatrix} P \mathbf{U}^n + \begin{bmatrix} T & O \\ E & L \end{bmatrix} P (I - \theta \Delta t J)^{-1} \Delta t J \mathbf{U}^n, \tag{4.3}$$

where the forcing terms are omitted. This equation is solved by premultiplication of

$$\begin{bmatrix} T & O \\ E & L \end{bmatrix}^{-1} = \begin{bmatrix} I & O \\ E & L \end{bmatrix}^{-1} \begin{bmatrix} T^{-1} & O \\ O & I \end{bmatrix} . \tag{4.4}$$

In our case, we approximate T^{-1} in (4.4) by $K = (I - D^{-1}C)D^{-1}$ (see (3.5)). Now, first (4.3) is premultiplied with

$$\begin{bmatrix} K & O \\ O & I \end{bmatrix} ,$$

which gives

$$\begin{bmatrix} KT & O \\ E & L \end{bmatrix} P \mathbf{U}^{n+1} = \begin{bmatrix} KT & O \\ E & L \end{bmatrix} P \mathbf{U}^n + \Delta t \begin{bmatrix} K & O \\ O & I \end{bmatrix} \begin{bmatrix} T & O \\ E & L \end{bmatrix} P (I - \theta \Delta t J)^{-1} J \mathbf{U}^n . \tag{4.5}$$

Thereafter, KT is replaced by I and (4.5) is premultiplied by (cf. Remark 3.1)

$$\begin{bmatrix} I & O \\ E & L \end{bmatrix}^{-1} ,$$

which gives the explicit expression

$$P \mathbf{U}^{n+1} = P \mathbf{U}^n + \Delta t \begin{bmatrix} I & O \\ E & L \end{bmatrix}^{-1} \begin{bmatrix} K & O \\ O & I \end{bmatrix} \begin{bmatrix} T & O \\ O & I \end{bmatrix} \begin{bmatrix} I & O \\ E & L \end{bmatrix} P (I - \theta \Delta t J)^{-1} J \mathbf{U}^n .$$

Finally, this leads to

$$\mathbf{U}^{n+1} = \mathbf{U}^{n+1} + \Delta t P^T \begin{bmatrix} I & O \\ E & L \end{bmatrix}^{-1} \begin{bmatrix} KT & O \\ O & I \end{bmatrix} \begin{bmatrix} I & O \\ E & L \end{bmatrix} P (I - \theta \Delta t J)^{-1} J \mathbf{U}^n ,$$

Hence S is of the form

$$S = P^T \begin{bmatrix} I & O \\ E & L \end{bmatrix}^{-1} \begin{bmatrix} KT & O \\ O & I \end{bmatrix} \begin{bmatrix} I & O \\ E & L \end{bmatrix} P . \tag{4.6}$$

The non-trivial eigenvalues of S are the eigenvalues of KT. Then by virtue of Theorem 4.1., the system (2.11) is stable if

$$\lambda_{KT} = \lambda_{(I - (D^{-1}C)^2)} \in [0,1] \text{ and real} , \tag{4.7}$$

where the conditions (a) and (b) should be satisfied.

However, condition (c) is not satisfied for S given in (4.6). Since

$$\begin{bmatrix} I & O \\ E & L \end{bmatrix}^{-1} = \begin{bmatrix} I & O \\ -F & L^{-1} \end{bmatrix},$$

where $F = L^{-1}E$, we find for (4.6)

$$S = P^T \begin{bmatrix} KT & O \\ F(I-KT) & I \end{bmatrix} P .$$

The matrix S should commute with P^TMP. It is straightforward that among others this leads to the condition

$$KTM_{12} = M_{12} .$$

As in general M_{12} is not a nil matrix, KT should be the identity matrix. This is in general not true. Despite of the fact that sufficient condition (c) is not satisfied, we found in the experiments that the stability condition (4.7) was valid.

REMARK 4.1. The eigenvalues λ_{KT} are determined by the choice of the grid points of the reduced system. For a particular choice of Δt the magnitude of the norm of $D^{-1}C$ will rapidly tend to zero when the distance between the two nearest points of the explicitly solved system increases. This follows from the fact that the influence of the solution at one point on the solution at other points decreases when the distance between these two points increases. This holds for hyperbolic as well as parabolic problems. For model problems, using the cyclic reduction process, we can derive stability conditions (see (5.4) and (5.7)). The stability conditions are of the form

$$\frac{\Delta t}{(2^k \Delta x)} \leqslant c_{\text{hyp}} ,$$

$$\frac{\Delta t}{(2^k \Delta x)^2} \leqslant c_{\text{parab}} ,$$

where c_{hyp} and c_{parab} are constants (the so-called stability boundaries), for hyperbolic and parabolic problems, respectively. Furthermore, k denotes the number of steps in the cyclic reduction process.

REMARK 4.2. Due to the choice of C (see 3.3 and 3.8), we have that $C = 0$ for problems of the form

$$\frac{d}{dt}\mathbf{U} = \Lambda\mathbf{U} ,$$

where Λ is a diagonal matrix. Hence for such problems the constructed method is unconditionally stable. This result also holds when T is a diagonal matrix.

REMARK 4.3. The requirement that KT should have real eigenvalues does in general only hold if one starts with central differences for hyperbolic as well as for parabolic problems (see examples). However, in practice often one-sided differences are used. In Section 5.1 we have tested a one-sided difference in the space discretization. again the method gave stable results, although (4.7) is not satisfied in this case.

REMARK 4.4. In the case of a symmetric or antisymmetric Jacobian matrix J with constant coefficients (as below in (5.2) and (5.6)) the matrix T (see (2.10)) is symmetric after one or more steps of the cyclic reduction process. This can be established by performing some steps of the process by hand. Furthermore, the diagonal elements are positive, say b ($b > 0$) and the off-diagonal elements are negative, say a ($a < 0$). Application of the approximation described in Section 3 yields a method which satisfies stability condition (4.7) when $a / b < 1/6$.

REMARK 4.5. From the stability conditions given in Remark 4.1, we observe that the minimal value

of k, in order to satisfy the stability condition involved, decreases with Δt for a constant Δx. Hence, Δt determines the number of reduction steps and thus the needed implicitness. Therefore, for a given time step Δt, the computation time is minimal, when we use the minimal value of k such that the stability condition involved, is satisfied.

5. Numerical illustration

To illustrate the performance of the method described in Sections 2 and 3, we present some experiments, both for linear and nonlinear problems. In the experiments the cyclic reduction algorithm is used to solve the equations. By varying the set of points which are solved explicitly (see (2.8)), we vary the stability property of the method. Our choice will be the regular set of grid points as denoted by (2.13), where k denotes the number of cyclic reduction steps. In this case, we have

$$\mathbf{V} = [\, U_{2^k}, U_{2\cdot 2^k}, U_{3\cdot 2^k}, \ldots, U_{2^p - 2^k} \,]^T ,$$

where the superscripts are omitted. To make optimal use of the cyclic reduction algorithm, we have chosen a uniform grid such that the number of grid points is $N = 2^p - 1$, although this is not essential, with mesh size $\Delta x = L / 2^p$. The aim of our experiments is to show the relation between the number of reduction steps, which is a measure for the implicitness, and the stability behaviour of the applied method. Furthermore, we are interested in the accuracy behaviour when the number of reduction steps varies. To measure the obtained accuracy we define

$$\text{cd} = -{}^{10}\log(\,|\text{ maximal global error at the endpoint } t = T\,|\,) ,$$

denoting the number of correct digits in the numerical approximation at the endpoint. The calculations were performed on the CDC Cyber 170-750 which has a 48-bit mantissa, i.e. a machine precision of about 14 decimal digits.

5.1. A linear hyperbolic problem

As a first example, consider the linear test problem

$$u_t = u_x , \quad 0<t<T, \quad 0<x<L , \tag{5.1}$$

with initial condition

$$u(x, 0) = \sin(2\pi x / L) ,$$

and boundary condition

$$u(L,t) = \sin(2\pi(L+t) / L) .$$

The exact solution is given by

$$u(x,t) = \sin(2\pi(x+t) / L) ,$$

where $L = 64$.

Central differences are used at all points except for the first point where a commonly used one-sided difference is applied. In the notation of the split function $\mathbf{G}$ (see 2.4) the discretization is given by

$$\begin{aligned}
(J\mathbf{U})_1 &= \frac{(U_2 - U_1)}{\Delta x} , \quad \mathbf{g}_1 = 0 , \\
(J\mathbf{U})_j &= \frac{(U_{j+1} - U_{j-1})}{2\Delta x} , \quad \mathbf{g}_j = 0 \text{ for } j = 2,\ldots,N-1 , \\
(J\mathbf{U})_N &= \frac{-U_{N-1}}{2\Delta x} , \quad \mathbf{g}_N(t) = \sin(2\pi(L+t) / L) / (2\Delta x) .
\end{aligned} \tag{5.2}$$

Notice that for linear hyperbolic systems, the Jacobian matrix J has almost purely imaginary eigenvalues. For the time integration, we used the Trapezoidal Rule, $\theta = \frac{1}{2}$ ($Q = 1$ due to linearity).

Only for linear test problems we compare our approach with incomplete cyclic reduction without any adaptation (see Introduction). For nonlinear test problems we expect the same behaviour. In the case of incomplete cyclic reduction without any adaptation, the reduced system of equations is solved by (cf. 3.5)

$$\tilde{\mathbf{V}}^{(q)} = (I + D^{-1} C)\mathbf{V}^{(0)} + D^{-1}\tilde{\mathbf{B}}_1 , \tag{5.3}$$

where

$$D_{ii} = T_{ii} \ , \ D_{ij} = 0 \ \text{for} \ j \neq i \ .$$

In Table 5.1 we give the cd-values of the method, obtained at the endpoint $T = 320$. In the last column we listed the values for the incomplete cyclic reduction method.

	scheme (3.5)						scheme (5.3)
Δt	$p = 5$ $\Delta x = 2$	$p = 6$ $\Delta x = 1$	$p = 7$ $\Delta x = 0.5$	$p = 8$ $\Delta x = 0.25$	$p = 9$ $\Delta x = 0.125$		$p = 8$ $\Delta x = 0.25$
1	1.30(1) 1.30(2)	1.80(1) 1.80(2)	1.87(1) 2.11(2)	***(1) 2.24(2) 2.25(3)	***(2) 2.27(3)		***(2) 0.34(3) 2.03(4) 2.25(5)
2	1.19(1) 1.19(2)	1.24(1) 1.51(2) 1.51(3)	***(1) 1.63(2) 1.64(3)	***(2) 1.67(3)	***(3) 1.68(4)		***(3) 0.64(4) 1.68(5)
4	0.78(1) 0.91(2) 0.91(3)	***(1) 1.01(2) 1.04(3) 1.04(4)	***(2) 1.05(3) 1.08(4)	***(3) 1.06(4)			***(4) 0.88(5) 1.08(6)
8	***(1) 0.36(2) 0.43(3) 0.43(4)	***(2) 0.40(3) 0.46(4)	***(3) 0.41(4)				

Table 5.1. Number of correct digits for the linear hyperbolic problem (5.1) with $T = 320$.

In Table 5.1 the number of cyclic reduction steps is given in parenthesis. An unstable behaviour of the integration process is dented by ***. The number of grid points is equal to 2^p.

The results clearly show the effect of varying the number of reduction steps:
(a) The error hardly depends on the number of reduction steps, as long as the computation is stable.
(b) If the mesh size is decreased by a factor two then one extra reduction step is needed to maintain the same stability boundary on Δt.

For scheme (5.3) at least two extra reduction steps are needed to obtain accuracy which is comparable with scheme (3.5). It can be proved that scheme (5.3) is unstable. This scheme is only useful when the off-diagonal elements are neglectable with respect to the diagonal elements.

From (4.7) we can derive, by performing some reduction steps explicitly, the stability condition

$$\frac{\Delta t}{(2^k \Delta x)} \leqslant c_k \,, \quad k = 1,...,p-1 \,, \tag{5.4}$$

where k denotes the number of cyclic reduction steps and c_k is a constant depending on k. If k equals p then the method is purely implicit and unconditionally stable. In Table 5.2 we have listed the values c_k, for $k = 1,...,6$.

k	1	2	3	4	5	6
c_k	1	1.09868	1.12546	1.13230	1.13402	1.13445

Table 5.2. Stability coefficients for hyperbolic problem (5.1).

It appeared that

$$\lim_{k \to \infty} c_k \uparrow 1.134593 \,.$$

REMARK 5.1. From the results in Table 5.1 it is easily verified that condition (5.4) is satisfied. As the number of grid points, at the old time level, involved in the computation of the solution at the new time level is of $O(2^k)$ after k reduction steps, we have from the stability condition (5.4) that the time step increases almost linearly with the size of the influence domain.

REMARK 5.2. Notice that we did not test the method with zero cyclic reduction steps. In this case, it can be shown that $D^{-1} C = 0.5 \Delta t J$ with respect to the internal points. Applying immediately the explicit expression (3.6) we obtain a second-order method which has similar properties as the two-stage second-order Runge-Kutta method[9]. The latter is not stable for problems from which the Jacobian matrix has imaginary eigenvalues. After one step of the cyclic reduction algorithm the eigenvalues of the resulting matrix T (see 2.10)) are real when we start with a Jacobian matrix J with imaginary eigenvalues. In this case, the matrix C given by (3.3) and (3.8) has imaginary eigenvalues and thereby the eigenvalues of KT (see (4.7)) are not real.

We also tested our method for the same linear hyperbolic problem, using one-sided differences for u_x. In this case, J and $\mathbf{g}$ are is given by

$$(J\mathbf{U})_j = \frac{(U_{j+1} - U_j)}{\Delta x} \,, \quad \mathbf{g}_j = 0 \text{ for } j = 1,...,N-1 \,,$$

$$(J\mathbf{U})_N = \frac{-U_N}{\Delta x} \,, \quad \mathbf{g}_N(t) = \sin(2\pi(L+t)/L)/\Delta x \,.$$

In this case the eigenvalues of KT (see (4.7)) are not real. Notwithstanding, this scheme shows a comparable stability behaviour as in the case of central differences (see 5.2). The results are given in the same form as in Table 5.1.

Δt	$p = 5$ $\Delta x = 2$	$p = 6$ $\Delta x = 1$	$p = 7$ $\Delta x = 0.5$	$p = 8$ $\Delta x = 0.25$	$p = 9$ $\Delta x = 0.125$
1	0.43(0) 0.44(1) 0.44(2)	0.66(0) 0.68(1) 0.68(2)	***(0) 0.95(1) 0.95(2)	***(1) 1.23(2) 1.23(3)	***(2) 1.46(3)
2	0.42(0) 0.43(1) 0.43(2)	***(0) 0.67(1) 0.68(2) 0.68(3)	***(1) 0.94(2) 0.94(3)	***(2) 1.20(3) 1.21(4)	***(3) 1.45(4)
4	0.41(1) 0.42(2) 0.42(3)	***(1) 0.64(2) 0.65(3) 0.65(4)	***(2) 0.86(3) 0.86(4)	***(3) 1.03(4)	
8	***(1) 0.34(2) 0.35(3) 0.35(4)	***(2) 0.46(3) 0.46(4)	***(3) 0.53(4)		

Table 5.3. Number of correct digits for the linear hyperbolic problem (5.1) with one-sided differences and $T = 320$.

5.2. A linear parabolic problem

As a second example, consider the linear test problem

$$\begin{aligned} &u_t = u_{xx}\,, \quad 0<t<T, \quad 0<x<L\,, \\ &u_x(0,t) = \frac{2\pi}{L}\, e^{-(\frac{2\pi}{L})^2 t} \\ &u(L,t) = 0\,. \end{aligned} \tag{5.5}$$

The exact solution is given by

$$u(x,t) = e^{-(\frac{2\pi}{L})^2 t} \sin(2\pi x / L)\,,$$

where $L = 32$.

For the space-discretization of (5.5), central differences are used which yields for J and $\mathbf{g}$ (see 2.4)

$$
\begin{aligned}
(J\mathbf{U})_1 &= \frac{(U_2 - U_1)}{(\Delta x)^2}, \quad \mathbf{g}_1(t) = -\left(\frac{2\pi}{L}\, e^{-\left(\frac{2\pi}{L}\right)^2 t}\right)/\Delta x, \\
(J\mathbf{U})_j &= \frac{(U_{j-1} - 2U_j + U_{j+1})}{(\Delta x)^2}, \quad \mathbf{g}_j(t) = 0 \text{ for } j = 2,\ldots,N-1, \\
(J\mathbf{U})_N &= \frac{(U_{N-1} - 2U_N)}{(\Delta x)^2}, \quad \mathbf{g}_N(t) = 0.
\end{aligned}
\tag{5.6}
$$

Here, $x_j = x_0 + j\Delta x$ with $x_0 = -\frac{1}{2}\Delta x$. Furthermore, Δx should be such that $x_{N+1} = L$.
The Jacobian matrix J, given by (5.6), has real eigenvalues. For the time integration we used the Trapezoidal Rule, i.e. $\theta = 0.5$ ($Q = 1$ due to linearity).

The results are given in the same form as in Table 5.1.

	scheme (3.5)					scheme (5.3)
Δt	$p=4$ $\Delta x=2$	$p=5$ $\Delta x=1$	$p=6$ $\Delta x=0.5$	$p=7$ $\Delta x=0.25$		$p=7$ $\Delta x=0.25$
2	2.00(0) 2.13(1)	***(0) 2.64(1) 2.69(2)	***(1) 3.00(2) 3.41(3) 3.43(4)	***(2) 3.40(3) 4.07(4)		***(4) 3.97(5)
4	2.12(1) 2.16(2)	***(1) 2.75(2) 2.85(3)	***(2) 3.49(3) 3.37(4)	***(3) 3.31(4)		***(4) 3.12(5)
8	2.30(2)	2.28(2) 2.78(3) 2.76(4)	***(2) 2.38(3) 2.56(4)	***(3) 2.40(4)		***(4) 2.50(5)
16	2.24(2) 2.16(3)	***(2) 2.01(3) 1.97(4)	***(3) 1.94(4) 1.94(5)	***(4) 1.91(5)		***(4) 1.90(5)

Table 5.4. Number of correct digits for the linear parabolic problem (5.5) with $T = 32$.

Globally, we observe the same effect for this parabolic problem as for the hyperbolic problem (5.3). If the mesh size is decreased by a factor four, then one extra reduction step is needed to maintain the same stability boundary on Δt. Here the numerical error is not only determined by the time

integration. For small time steps, compared with the space mesh, the space discretization error becomes visible.

As for the hyperbolic problem, some extra reduction steps are needed for scheme (5.3) in order to obtain accuracy which is comparable with scheme (3.5).

From (4.7) we obtained the stability condition (cf. (5.4))

$$\frac{\Delta t}{(2^k \Delta x)^2} \leqslant c_k \ , \quad k = 0,...,p-1 \ . \tag{5.7}$$

In Table 5.5 we have listed some values for c_k.

k	0	1	2	3	4
c_k	0.5	0.60355	0.63334	0.64105	0.64299

Table 5.5. Stability coefficients for parabolic problem (5.5).

Here, we have

$$\lim_{k\to\infty} c_k \uparrow 0.643651 \ .$$

It is easily verified that condition (5.7) is in agreement with the results in Table 5.4. The results for this parabolic problem show a similar behaviour as for the hyperbolic problem (5.1). For every applied reduction step the maximal time step increases with about a factor four. Thus the maximal time step increases almost quadratically with the size of the influence domain for the difference equation.

5.3. A nonlinear parabolic problem

Consider the nonlinear one-dimensional heat equation (see [15])

$$\frac{\partial u}{\partial t} = \frac{1}{\rho c(u)} \frac{\partial}{\partial x} \left(K(u) \frac{\partial u}{\partial x}\right) , \tag{5.8}$$

with u the temperature, ρc the heat capacity and K the thermal conductivity. The thermal conductivity and the heat capacity are given by

$$K(u) = 1 + 0.5u$$

$$\rho c(u) = 1 + 0.5u \ .$$

We consider a finite bar, with boundary conditions

$$u_x(0,t) = -\frac{1}{K(u(0,t))}$$

$$u_x(L,t) = 0 \ ,$$

where $L = 2$. So the bar is isolated at the endpoint L and at $x = 0$ a constant heat input $q = -K(u)\frac{\partial u}{\partial x} = 1$ is assumed. Due to the nonlinear nature of equation (5.8), we have

$$\mathbf{G}(\tilde{\mathbf{U}},\mathbf{U},t) = A(\mathbf{K}(\tilde{\mathbf{U}}))\mathbf{U} + \mathbf{g}(\tilde{\mathbf{U}},t) \ ,$$

where

$$(A(\mathbf{K}(\tilde{\mathbf{U}}))\mathbf{U})_j = \frac{1}{(\rho c(\tilde{\mathbf{U}}))_j} \frac{((\mathbf{K}(\tilde{\mathbf{U}})\, D_x\mathbf{U})_{j+\frac{1}{2}} - (\mathbf{K}(\tilde{\mathbf{U}})\, D_x\mathbf{U})_{j-\frac{1}{2}})}{\Delta x} . \tag{5.9}$$

In (5.9) $\mathbf{K}(\tilde{\mathbf{U}})$ is given by

$$(\mathbf{K}(\tilde{\mathbf{U}}))_{j+\frac{1}{2}} = 1 + 0.5 \frac{(\tilde{U}_{j+1} + \tilde{U}_j)}{2} ,$$

and

$$(\rho c(\tilde{\mathbf{U}}))_j = 1 + 0.5\, \tilde{U}_j .$$

Furthermore, $D_x\mathbf{U}$ and $\mathbf{g}$ are given by

$$(D_x\mathbf{U})_{\frac{1}{2}} = 0, \quad \mathbf{g}_1(\tilde{\mathbf{U}},t) = \frac{-1}{(\rho c(\tilde{\mathbf{U}}))_1} / \Delta x ,$$

$$(D_x\mathbf{U})_{j+\frac{1}{2}} = \frac{(U_{j+1} - U_j)}{\Delta x}, \quad \mathbf{g}_j(\tilde{\mathbf{U}},t) = 0 \text{ for } j = 2,...,N-1,$$

$$(D_x\mathbf{U})_{N+\frac{1}{2}} = 0, \quad \mathbf{g}_N(\tilde{\mathbf{U}},t) = 0.$$

Hence, the matrix A is tridiagonal. The grid points are chosen $x_j = j\Delta x + x_0$ with $x_0 = -\frac{1}{2}\Delta x$. The mesh width Δx should be such that $x_{N+\frac{1}{2}} = L$.

For the time integration we used (2.6) with $Q = 2$ (θ will be given later).
Furthermore, for $q = 1$ $\mathbf{Z}^{(q)}$ is solved using $k-1$ reduction steps in the cyclic reduction algorithm, and for $q = 2$ $\mathbf{Z}^{(q)}$ is solved using k reduction steps.
As the first step is a prediction only, we have used there one reduction step less than in the second step. Other choices are, of course, possible.

In the nonlinear experiments, we determined a reference solution using a very small integration step and we only considered the error due to the time integration. We have chosen a space mesh $2/(2^5-1)$. For the time integration we used the Trapezoidal Rule ($\theta = 0.5$) and the Backward Euler method ($\theta = 1$). The results are given in Table 5.6.

Δt	$\theta = 0.5$			$\theta = 1$		
	$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$
0.0125	4.11	4.23		2.33	2.76	2.76
0.025	2.88	3.57	3.59	***	2.47	2.48
0.05	***	2.73	2.73		2.15	2.16
0.1		2.03	2.02		1.79	1.87
0.2		1.34	1.34		***	1.61

Table 5.6. Number of correct digits for the nonlinear parabolic problem (5.8) with $T = 1$ and k the number of cyclic reduction steps.

The results of this example give rise to conclusions similar to the previous examples : when the number of cyclic reductions increases, and thus the implicitness, the stability of the method increases.

The results clearly show the first-order behaviour of the Backward Euler method and the second-order behaviour of the Trapezoidal Rule. The maximal time step for the Trapezoidal Rule is twice the maximal step for the Backward Euler method. This is due to the fact that the magnitude of elements of matrix $A(\mathbf{K}(\mathbf{U}))$ of the Backward Euler method are twice the magnitude of the elements of the Trapezoidal Rule.

5.4. A nonlinear hyperbolic problem

Consider a simplified form of the shallow water equations in one dimension :

$$\begin{aligned} u_t &= -\lambda u - g\zeta_x \,, \\ \zeta_t &= -(hu)_x \,, \end{aligned} \tag{5.10}$$

where u denotes the depth-averaged velocity, λ the bottom friction, ζ the elevation of the water surface, $\bar{h}$ the depth when the water is in rest, h the total depth given by $h = \bar{h} + \zeta$ and g the acceleration due to gravity. The first equation is a momentum equation describing the change in time of the velocity u. The second equation is a continuity equation. The numerical solution of (5.10) is required in the region

$$0 \leqslant x \leqslant L \,, \quad \text{for } 0 \leqslant t \leqslant T \,.$$

The boundary conditions are

$$u(0,t) = -\sin(\omega t) \quad \text{and} \quad \zeta(L,t) = \cos(\omega t) \,.$$

The initial conditions are given by

$$u(x,0) = 0 \quad \text{and} \quad \zeta(x,0) = 1 \,.$$

Let $\mathbf{W} = (\mathbf{U},\mathbf{Z})^T$, where $\mathbf{U}_j(t) \approx u(j\Delta x,t)$ and $\mathbf{Z}_j(t) \approx \zeta(j\Delta x,t)$. In this case $\mathbf{G}(\mathbf{W},\tilde{\mathbf{W}},t)$ is defined by

$$\begin{aligned} \mathbf{G}^U(\mathbf{W},\tilde{\mathbf{W}},t) &= -gD_x\tilde{\mathbf{Z}} - \lambda\tilde{\mathbf{U}} \,. \\ \mathbf{G}^Z(\mathbf{W},\tilde{\mathbf{W}},t) &= -(M\mathbf{H}\, D_x\tilde{\mathbf{U}} + M\mathbf{U}\, D_x\tilde{\mathbf{H}}) \,. \end{aligned} \tag{5.11}$$

where

$$\begin{aligned} (D_x\mathbf{U})_j &= (U_{j+1} - U_{j-1})/\Delta x \,, \\ H_j &= \bar{h} + \frac{(Z_{j-1} + Z_{j+1})}{2} \,, \\ (M\mathbf{H})_j &= \frac{(H_{j-1} + H_{j+1})}{2} \,. \end{aligned}$$

At the grid point on the left boundary we used

$$H_1 = \bar{h} + Z_2 \,.$$

In this case, we will not give the precise form of J and $\mathbf{g}$ (see 2.4), but they can be derived straightforwardly from (5.11). By this choice for the space-discretization it can be shown that two independent sets of equations arise, i.e. a system for (U_{2k},Z_{2k+1}), and a system for $((U_{2k+1},Z_{2k+2}),\ k = 1,2,3,\ldots)$. Hence, by omitting the latter system the number of variables is reduced by a factor two. The variables of the remaining system are now space staggered. For more details on space staggering we refer to [5,17].

The variant of (2.5) which will be used here, is the following method :

$$\begin{aligned} \mathbf{W}^{(0)} &= \mathbf{W}^n \\ \mathbf{W}^{(q)} &= \mathbf{W}^n + \frac{\Delta t}{2}\{\mathbf{G}(\mathbf{W}^{(q-1)},\mathbf{W}^{(q)},t^{n+1}) + \mathbf{G}(\mathbf{W}^n,\mathbf{W}^n,t^n)\} \quad \text{for } q=1,\ldots,Q \\ \mathbf{W}^{n+1} &= \mathbf{W}^n + \frac{\Delta t}{2}\{\mathbf{G}(\mathbf{W}^{(Q)},\mathbf{W}^{(Q)},t^{n+1}) + \mathbf{G}(\mathbf{W}^n,\mathbf{W}^n,t^n)\}\,. \end{aligned} \tag{5.12}$$

The first Q steps give each rise to a linear implicit equation for $\mathbf{W}^{(q)}$, whereas the last step is purely explicit. For $Q \geqslant 1$ this method is second-order accurate in time. Using this method possible conservation properties of the semi-discretization are preserved irrespective of the linearization used in the second equation. In our case, the second equation in (5.12) represents mass conservation. This conservation is only simulated by $\mathbf{G}^Z$ if the first and second argument of $\mathbf{G}^Z$ are equal, because in this case it holds that

$$\mathbf{G}^Z(\mathbf{W},\mathbf{W},t) = -(M\mathbf{H}\,D_x\mathbf{U} + M\mathbf{U}\,D_x\mathbf{H}) = -D_x(\mathbf{HU})\,. \tag{5.13}$$

Hence, the second equation of (5.12) does not simulate this conservation property as long as $\mathbf{W}^{(q)} \neq \mathbf{W}^{(q-1)}$. However, the third equation makes the overall method conservative for all choices of Q.
If the expression for $\mathbf{U}^{(q)}$ is substituted into the equation for $\mathbf{Z}^{(q)}$, then the second equation from (5.12) gives rise to a tridiagonal system for $\mathbf{Z}^{(q)}$. Once $\mathbf{Z}^{(q)}$ is solved, $\mathbf{U}^{(q)}$ can be solved straightforwardly.

In Table 5.7 results are given for the case $Q = 1$. The constants are chosen

$$T = 5120\ ,\ \Delta x = 25\ ,\ L = 3175\ ,\ \omega = \frac{2\pi}{3600}\ ,\ g = 9.81 \text{ and } \bar{h} = 10\,.$$

Δt	$k=2$	$k=3$	$k=4$	$k=5$
10	4.71	4.71	4.71	4.71
20	***	4.00	4.01	4.01
40		***	3.37	3.38
80			***	2.01

Table 5.7. Number of correct digits for the nonlinear hyperbolic problem (5.10) with k the number of cyclic reduction steps.

These results show again that the error does not depend on the number of reduction steps, as long as the computation is stable. As in the linear hyperbolic case, about a factor two is gained when an extra reduction step is applied. When k equals five, the method is unconditionally stable. If we perform the first step (see (5.12)) twice, we obtain comparable results, except when ($\Delta t = 80$ and $k = 5$). In that case, we have a considerable increase of the number of correct digits.

6. Conclusions

In this paper we have constructed explicit-implicit methods starting from a one-step implicit method, which yields a tridiagonal system. These methods were constructed for time dependent partial differential equations. The method makes use of the fact that the interdependence of the solution at two different points at the new time level decreases if the physical distance between these points increases. This fact causes that the magnitude of the off-diagonals decrease rapidly when compared with the main diagonal in each step of the cyclic reduction process. The constructed methods have the following properties :

(a) The accuracy is hardly influenced if we replace the one-step implicit method by an approximating explicit-implicit method as long as the integration is stable.

(b) If the one-step implicit method satisfies a conservation property, then this property is preserved when the implicit method is replaced by the approximating explicit-implicit method.

(c) The maximum allowed time step increases linearly or quadratically with the number of points of the old time level which influence the solution at a point at the new time level for hyperbolic or parabolic equations, respectively.

In this paper, we have restricted ourselves to one-dimensional problems. However, the technique can be applied directly to alternating direction methods often used in multi-dimensional cases. Such methods lead to a succession of one-dimensional problems, each of which can be treated by the described technique.
The approximation of fully implicit methods in the multi-dimensional case by explicit-implicit methods is subject of future research. We expect that the theory for this case will develop along the same lines.

7. References

[1] Adams, L., m-Step preconditioned conjugate gradient methods, *SIAM J. Sci. Stat. Comput.*, vol. 6 (1985), pp. 452-463.

[2] Axelsson, O., A survey of preconditioned iterative methods for linear systems of equations, *BIT*, vol. 25 (1985), pp. 166-187.

[3] Dendy, J.K. jr., Black box multigrid for nonsymmetric problems, *Appl. Math. and Comput.*, vol 13 (1983), pp. 261-283.

[4] Dubois, P.P., Greenbaum, A. and G.H. Rodrigue, Approximating the inverse matrix for use in iterative algorithms on vector computers, *Computing*, vol. 22 (1979), pp. 257-268.

[5] Hansen, W. Theorie zur errechnung das wasserstandes und der stromungen in randmeeren nebst anwendungen, *Tellus*, vol. 8 (1956), pp. 287-300.

[6] Heller, D., Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems, *SIAM J. Numer. Anal.*, vol. 13 (1976), pp. 484-496.

[7] Hockney, R.W., A fast direct solution of Poisson's equation using Fourier analysis, *J. Assoc. Comp. Mach.*, vol. 12 (1965), pp. 95-113.

[8] Hockney, R.W. and C.R. Jesshope, *Parallel computers : architecture, programming and algorithms*, Adam Hilger, Ltd., Bristol, 1981.

[9] Houwen, P.J., *Construction of integration formulas for initial-value problems*, North-Holland, Amsterdam, 1977.

[10] Houwen, P.J. and J.G. Verwer, One-step splitting methods for semi-discrete parabolic equations, *Computing*, vol 22 (1979), pp.291-309.

[11] Lambert, J.D., *Computational methods in ordinary differential equations*, Wiley, London-New York, 1973.

[12] Lambiotte, J.J. and R.G. Voight, The solution of tridiagonal linear systems on the CDC Star-100 computer, *ACM Trans. Math. Software*, vol. 1 (1975), pp. 308-329.

[13] Mitchell, A.R. and D.F. Griffiths, *The finite difference method in partial differential*

equations, Wiley, Chichester, 1980.

[14] MITCHELL, A.R. AND R.W. WAIT, *The finite element analysis and applications*, John Wiley, New York, 1985.

[15] ORIVUORI, S., Efficient method for solution of non-linear heat conduction problems, *Int. J. for Num. Meth. in Eng.*, vol. 14 (1979), pp. 1461-1476.

[16] RICHTMYER, R.D. AND K.W. MORTON, *Difference methods for initial value problems*, Wiley, New York, 1967.

[17] STELLING, G., *On the construction of computational methods for shallow water flow problems*, Thesis TH Delft, 1983.

[18] STRANG, G. AND J. FIX, *An analysis of the finite element method*, Prentice Hall, Englewood Cliffs, 1973.

[19] VORST, H.A. VAN DER, A vectorizable variant of some ICCG methods, *SIAM J. Sci. Stat. Comput.*, vol. 3 (1982), pp.350-356.

[20] VORST, H.A. VAN DER, *Vectorization of linear recurrence relations*, Faculty of Mathematics & Informatics Delft, in preparation.

[21] WANG, H.H., A parallel method for tridiagonal system equations, *ACM Trans. on Math. Softw.*, vol. 7 (1981), pp. 170-183.

APPENDIX A

In Appendix A and B we give two possible solution methods for system (2.7), which yield a system of equations as denoted by (2.11).

Matrix decomposition I : Cyclic reduction

The cyclic reduction algorithm was originally developed by Hockney[7], for the discrete version of Poisson's equation. The cyclic reduction algorithm is well-suited for use on a parallel or vector computer, as many of the quantities involved may be computed independently of the others. This case has been studied by Lambiotte and Voight [12], with attention to a vector computer.

We assume that the system of linear algebraic equations arising from implicit difference formula (2.3), which must be solved at each time step is a special case of the tridiagonal system

$$\alpha_j x_{j-1} + \beta_j x_j + \gamma_j x_{j+1} = b_j ,$$

for $1 \leq j \leq m$, where $\alpha_1 = 0$ and $\gamma_m = 0$.
Also, we assume that $m = 2^p - 1$, although this is not essential, where p is some positive integer. In matrix form, we obtain

$$\begin{bmatrix} \beta_1 & \gamma_1 & & & & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & & & \\ & . & . & . & & \\ & & \alpha_{m-1} & \beta_{m-1} & \gamma_{m-1} & \\ 0 & & & \alpha_m & \beta_m & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ b_{m-1} \\ b_m \end{bmatrix} . \tag{A.1}$$

The cyclic reduction algorithm separates the system in two subsystems, which involve respectively the rows with even indices and the rows with odd indices. Let us rewrite (A.1) as follows :

$$\begin{aligned} \alpha_{j-1} x_{j-2} + \beta_{j-1} x_{j-1} + \gamma_{j-1} x_j &= b_{j-1} \\ \alpha_j x_{j-1} + \beta_j x_j + \gamma_j x_{j+1} &= b_j \\ \alpha_{j+1} x_j + \beta_{j+1} x_{j+1} + \gamma_{j+1} x_{j+2} &= b_{j+1} \end{aligned} .$$

Multiplying the first equation by $-\alpha_j / \beta_{j-1}$, the third by $-\gamma_j / \beta_{j+1}$ and adding to the second equation, we obtain

$$(\frac{-\alpha_{j-1}\alpha_j}{\beta_{j-1}})\, x_{j-2} + (\beta_j - \frac{\gamma_{j-1}\alpha_j}{\beta_{j-1}} - \frac{\alpha_{j+1}\gamma_j}{\beta_{j+1}})\, x_j + (\frac{-\gamma_{j+1}\gamma_j}{\beta_{j+1}})\, x_{j+2} = \tag{A.2}$$
$$\frac{-\alpha_j}{\beta_{j-1}}\, b_{j-1} + b_j + \frac{-\gamma_j}{\beta_{j+1}}\, b_{j+1} .$$

In order to simplify the notation, we introduce

$$\kappa_j = (\frac{-\alpha_{j-1}\alpha_j}{\beta_{j-1}}) , \lambda_j = (\beta_j - \frac{\gamma_{j-1}\alpha_j}{\beta_{j-1}} - \frac{\alpha_{j+1}\gamma_j}{\beta_{j+1}}) , \mu_j = (\frac{-\gamma_{j+1}\gamma_j}{\beta_{j+1}}) \text{ and}$$
$$\frac{-\alpha_j}{\beta_{j-1}}\, b_{j-1} + b_j + \frac{-\gamma_j}{\beta_{j+1}}\, b_{j+1} = B_j .$$

Then (A.2) is equal to

$$\kappa_j\, x_{j-2} + \lambda_j\, x_j + \mu_j\, x_{j+2} = B_j .$$

Thus, if j is even, the new system of equations involves x_j's with even indices. Similar equations hold for x_2 and x_{m-1}. The process of reducing the equations in this fashion is known as cyclic reduction. Then (A.1) may be written as the following equivalent system :

$$\begin{bmatrix} \lambda_2 & \mu_2 & & & 0 \\ \kappa_4 & \lambda_4 & \mu_4 & & \\ & \cdot & \cdot & \cdot & \\ & & \kappa_{m-3} & \lambda_{m-3} & \mu_{m-3} \\ 0 & & & \kappa_{m-1} & \lambda_{m-1} \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ \cdot \\ x_{m-3} \\ x_{m-1} \end{bmatrix} = \begin{bmatrix} B_2 \\ B_4 \\ \cdot \\ B_{m-3} \\ B_{m-1} \end{bmatrix} . \tag{A.3}$$

and

$$\begin{bmatrix} \beta_1 & 0 & & & 0 \\ 0 & \beta_3 & 0 & & \\ & \cdot & \cdot & \cdot & \\ & & 0 & \beta_{m-2} & 0 \\ 0 & & & 0 & \beta_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ \cdot \\ x_{m-2} \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_3 \\ \cdot \\ b_{m-2} \\ b_m \end{bmatrix} - \begin{bmatrix} \gamma_1 & & & 0 \\ \alpha_3 & \gamma_3 & & \\ & \cdot & \cdot & \\ & & \alpha_{m-2} & \gamma_{m-2} \\ 0 & & & \alpha_m \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ \cdot \\ x_{m-3} \\ x_{m-1} \end{bmatrix} . \tag{A.4}$$

Since $m = 2^p - 1$ and the new system (A.3) involves only x_j's with even indices, the dimension of the new system is $2^{p-1} - 1$. Note that once (A.3) is solved, it is easy to solve for the x_j's with odd indices, as evidenced by (A.4). The system (A.4) is known as the eliminated equations.

Since system (A.3) is tridiagonal and in the form of (A.1), we can apply the reduction algorithm repeatedly until we have one equation. However, we can stop the process after any step and use another method to solve the reduced system of equations. After renumbering, we obtain a system of equations as denoted by (2.11).

Appendix B

Matrix decomposition II : A parallel method.
Here, we use a variant on Wang's algorithm [21]. Let us assume that the system of linear equations given in (A.1), is of the form

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{d}_1 & & & 0 \\ \mathbf{a}_k & \beta_k & \mathbf{c}_k & & \\ & \mathbf{e}_2 & \mathbf{A}_2 & \mathbf{d}_2 & \\ & & \mathbf{a}_l & \beta_l & \mathbf{c}_l \\ 0 & & & \mathbf{e}_3 & \mathbf{A}_3 \end{bmatrix} \mathbf{x} = \mathbf{b} , \tag{B.1}$$

where $\mathbf{A}_1, \mathbf{A}_2$ and $\mathbf{A}_3$ are tridiagonal matrices and

$$\mathbf{a}_i = [0,...,0,\alpha_i] \ , \ \mathbf{c}_i = [\gamma_i,0,...,0] \ , \ \mathbf{d}_1 = [0,...,0,\gamma_{k-1}]^T \ ,$$
$$\mathbf{d}_2 = [0,...,0,\gamma_{l-1}]^T \ , \ \mathbf{e}_2 = [\alpha_{k+1},0,...,0]^T \ , \ \mathbf{e}_3 = [\alpha_{l+1},0,...,0]^T \ .$$

In this example, we use three block matrices, but this reduction technique can be applied for an arbitrary number of block matrices. For this subdivision x_k and x_l will be the unknowns of the reduced system of equations. If the block matrices are invertible, then system (B.1) may be replaced by

$$\begin{bmatrix} \mathbf{A}_1^{-1} & & & & 0 \\ & 1 & & & \\ & & \mathbf{A}_2^{-1} & & \\ & & & 1 & \\ 0 & & & & \mathbf{A}_3^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{A}_1 & \mathbf{d}_1 & & & 0 \\ \mathbf{a}_k & \beta_k & \mathbf{c}_k & & \\ & \mathbf{e}_2 & \mathbf{A}_2 & \mathbf{d}_2 & \\ & & \mathbf{a}_l & \beta_l & \mathbf{c}_l \\ 0 & & & \mathbf{e}_3 & \mathbf{A}_3 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{A}_1^{-1} & & & & 0 \\ & 1 & & & \\ & & \mathbf{A}_2^{-1} & & \\ & & & 1 & \\ 0 & & & & \mathbf{A}_3^{-1} \end{bmatrix} \mathbf{b} ,$$

which is equivalent to

$$\begin{bmatrix} \mathbf{I} & \mathbf{v}_1 & & & 0 \\ \mathbf{a}_k & \beta_k & \mathbf{c}_k & & \\ & \mathbf{w}_2 & \mathbf{I} & \mathbf{v}_2 & \\ & & \mathbf{a}_l & \beta_l & \mathbf{c}_l \\ 0 & & & \mathbf{w}_3 & \mathbf{I} \end{bmatrix} \mathbf{x} = \mathbf{b}' ,$$

where the $\mathbf{v}$ and $\mathbf{w}$ are column vectors with $\mathbf{v}_i = \mathbf{A}_i^{-1}\mathbf{d}_i$ and $\mathbf{w}_i = \mathbf{A}_i^{-1}\mathbf{e}_i$. So far, this method corresponds with the first steps of Wang's algorithm. Now, we eliminate $\mathbf{a}_k$, $\mathbf{c}_k$, $\mathbf{a}_l$ and $\mathbf{c}_l$, which yields

$$\begin{bmatrix} \mathbf{I} & \mathbf{v}_1 & & 0 \\ & \beta_k' & \nu_k & \\ & \mathbf{w}_2 & \mathbf{I} & \mathbf{v}_2 \\ & \nu_l & \beta_l' & \\ 0 & & \mathbf{w}_3 & \mathbf{I} \end{bmatrix} \mathbf{x} = \mathbf{b}'' .$$

By a simple reordering this system can be brought to a system of the form (2.11). The k^{th} and the l^{th} row, which do not contain elements of the block matrices, form the reduced system of equations. It should be noted that the elimination of the off-diagonal elements of the matrices $\mathbf{A}_i$ can de done independently. Thereby, this approach is well suited for vector and parallel computers (see [20]).

SAMENVATTING

Dit proefschrift bestaat uit twee delen; het eerste gedeelte beschrijft numerieke technieken voor de ondiepwatervergelijkingen die speciaal voor gebruik op de vectorcomputer CYBER 205 ontwikkeld zijn; het tweede gedeelte bestaat uit een aantal artikelen waarin theoretische aspecten van de numerieke integratie van partiële differentiaalvergelijkingen behandeld worden.
Het proefschrift begint met een algemene introductie voor de beide gedeelten, waarin het kader van het onderzoek uiteengezet wordt.

Deel I:
Na een introductie in hoofdstuk 1 volgt in hoofdstuk 2 een beschrijving van de ondiepwatervergelijkingen. Voor het visceuse geval zijn enkele nieuwe randvoorwaarden afgeleid die van belang zijn voor praktijkproblemen.
Verschillende aspecten van de numerieke integratie van de ondiepwatervergelijkingen worden behandeld in hoofdstuk 3. Hieronder vallen de plaats- en tijdsdiscretisatie alsmede de in het kader van dit project ontwikkelde nieuwe stabilisatietechniek voor de expliciete tijdsintegrator. De keuze van een expliciete tijdsintegrator is ingegeven door het 4 jaar geleden in gebruiknemen van de vectorcomputer CYBER 205.
De vectorisatie van de algoritme voor de CYBER 205 komt in hoofdstuk 4 aan de orde. Er bestaan op deze machine verschillende instructies waarmee de formules in de buurt van randen en het bepalen van de positie van de rand ingeval van droogvallen en onderlopen efficient kunnen worden uitgevoerd. Voorts wordt aandacht geschonken aan de rekensnelheid als functie van de schaal van het probleem.
Naast de CYBER 205 algoritme is ook software ontwikkeld voor het specificeren van diverse ondiepwatermodellen en voor het visualiseren van de uitvoergegevens. Een beschrijving vindt men in hoofdstuk 5.
In hoofdstuk 6 worden resultaten van stromingsberekeningen in complexe geometriëen, ontleend aan praktijkproblemen, gegeven.

Deel II:
Van de artikelen in het tweede gedeelte handelen drie (de artikelen 1, 2 en 4) over stabilisatie van expliciete methoden door "smoothing". In het eerste artikel wordt de stabilisatie geïntroduceerd voor hyperbolische partiële differentiaalvergelijkingen. De stabilisatietechniek is verder geanalyseerd in het tweede artikel: optimale smoothing operatoren voor zowel hyperbolische als parabolische vergelijkingen zijn afgeleid. Een analyse van toepassing van smoothing op elliptische problemen is gegeven in het vierde artikel.

Voorts is onderzocht of de nauwkeurigheid van plaatsdiscretisaties verbeterd kan worden wanneer bij benadering bekend is welke frequencies de oplossing bepalen. Analyse en resultaten van dit onderzoek zijn beschreven in het derde artikel.
Het vijfde artikel beschrijft een aantal expliciet-impliciete methoden voor tijdsafhankelijke partiële differentiaalvergelijkingen. Dit type methoden is een goed alternatief voor volledig impliciete methoden wanneer geen onvoorwaardelijke stabiliteit nodig is. Bovendien wordt het met impliciete methoden verbonden algebraprobleem gereduceerd.

Curriculum vitae

De schrijver van dit proefschrift is geboren op 16 november 1957 te Sellingen (Z.O. Groningen). Na het behalen van het VWO-diploma begon hij in 1976 met de studie Toegepaste Wiskunde aan de Universiteit Twente. In 1983 studeerde hij af als wiskundig ingenieur bij Prof. dr. ir. P.J. Zandbergen. Het afstudeerwerk werd uitgevoerd bij het Nationaal Lucht- en Ruimtevaartlaboratorium onder leiding van Dr. ir. J.W. Boerstoel. Van 1983 tot 1987 voerde de auteur het STW-project "Evaluatie en stabilisatie van numerieke methoden voor de ondiepwatervergelijkingen" uit onder leiding van Prof. dr. P.J. van der Houwen, Dr. ir. G.K. Verboom en B.P. Sommeijer. Dit proefschrift geeft een verslag van dit project.