Time integration of three-dimensional numerical transport models

B.P. Sommeijer, P.J. van der Houwen, J. Kok

Department of Numerical Mathematics

Report NM-R9316 October 1993

Time Integration of Three-Dimensional Numerical Transport Models

B.P. Sommeijer, P.J. van der Houwen & J. Kok
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract
We analyse three-dimensional models for computing transport of salinity, pollutants, suspended material (such as sediment or mud), etc. The main purpose of this paper is to present an overview of the various possibilities for the time discretization that can take advantage of the parallelization and vectorization facilities offered by CRAY type computers. Among the suitable time integration techniques, we have both explicit and implicit methods. In explicit methods, the parallelization is straightforward, but these methods are hampered by a severe time step restriction due to stability. This can be avoided by selecting an implicit method; however, such a choice necessitates the frequent solution of sets of tridiagonal equations. Since the solution of these equations requires the greater part of the total solution time, this part of the algorithm needs special attention in order to get good performance on parallel/vector architectures. Following a suggestion of Golub and Van Loan, we have experimented with several implementations on a CRAY YMP4, which will be reported.

AMS Subject Classification (1991): 65M12, 65M20
C.R. Subject Classification (1991): G.1.7, G.1.8
Keywords and Phrases: Transport models, 3D advection-diffusion equations, numerical time integration, parallelism.
Note The investigations reported in this paper were supported by the NCF (National Computer Facility) who provided the authors with a grant (CRG 93.01) from Cray's University Research and Development Grant Program.

1. Introduction

The mathematical model describing transport processes of salinity, pollutants, suspended material (such as sediment or mud), etc., is defined by a system of 3D advection-diffusion-reaction equations

$$\text{(1.1a)}\qquad \frac{\partial c_i}{\partial t} = -\frac{\partial}{\partial x}(uc_i) - \frac{\partial}{\partial y}(vc_i) - \frac{\partial}{\partial z}((w - w_f)c_i) + \frac{\partial}{\partial x}(\varepsilon_x \frac{\partial c_i}{\partial x}) + \frac{\partial}{\partial y}(\varepsilon_y \frac{\partial c_i}{\partial y}) + \frac{\partial}{\partial z}(\varepsilon_z \frac{\partial c_i}{\partial z}) + g_i,$$

where

c_i	concentrations of the contaminants,
u, v, w	local fluid velocities in x, y, z directions,
w_f	fall velocity, only nonvanishing in the case of suspended material,
ε_x, ε_y, ε_z	diffusion coefficients in x, y, z directions,
g_i	source and reaction terms.

The velocities u, v, w, and the diffusion coefficients ε_x, ε_y, ε_z are assumed to be known in advance. The fall velocity w_f is only relevant in the case of modelling transport of suspended material, and may be a nonlinear function of the concentration (cf. Toro et al. [12]). The terms g_i describe chemical reactions, emissions from sources, etc., and therefore depend on the concentrations c_j. Thus, the mutual coupling of the equations in the system (1.1a) is only due to the functions g_i. Because of this weak coupling, we shall concentrate on the numerical modelling of a single transport equation. The extension to systems is relatively easy and will be subject of future research. In the following, we omit the index i occurring in (1.1a).

The physical domain in space is bounded by the vertical, closed boundary plane p(x,y) = 0, the water elevation surface $z = \zeta(t,x,y)$, and the bottom profile $z = -d(x,y)$. The functions p, ζ and d are also assumed to be known in advance. The boundary conditions along these boundary planes depend on the particular application at hand. Most considerations in this paper apply to the case where they are of the general, linear form:

Report NM-R9316
ISSN 0169-0388
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

$$\text{(1.1b)}\quad \begin{array}{lll} \text{Vertical boundaries} & p(x,y) = 0: & a_v(t,x,y,z)c + b_v \frac{\partial c}{\partial n} = c_v(t,x,y,z), \\ \text{Water surface boundary} & z = \zeta(t,x,y): & a_s(t,x,y)c + b_s \frac{\partial c}{\partial z} = 0, \\ \text{Bottom boundary} & z = -d(x,y): & a_d(t,x,y)c + b_d \frac{\partial c}{\partial z} = c_d(t,x,y). \end{array}$$

Here $\partial/\partial n$ denotes differentiation along the inward normal, a_v, a_s, a_d, c_v, and c_d are given functions and b_v, b_s and b_d are given constants. If the constants b_v, b_s or b_d vanish, then the boundary condition will be called a Dirichlet-type condition.

Given the initial concentration $c(t_0,x,y,z)$, the concentration c can be computed in space and time by solving the initial-boundary value problem (1.1). The numerical solution of this 3D problem requires powerful computing facilities such as offered by the CRAY supercomputers. In order to exploit these facilities, the spatial computational domain should be as simple as possible. There are two obvious approaches to simplify the spatial computational domain, viz. (i) coordinate transformations and (ii) the 'dummy-point' approach. In both cases, the spatial computational domain becomes a rectangular box which can be discretized by means of a uniform rectangular grid leading to efficient implementations on vector computers.

In the case of coordinate transformations, the 3D physical domain itself is mapped on a rectangular box. This approach has the additional advantage that the physical boundaries are exactly represented. However, the price we pay is firstly, a much more complicated mathematical model and secondly, as a consequence of the transformation, the possibility of introducing coefficients functions of large magnitude. The transformation of only one space variable already leads to quite unattractive formulas. We illustrate this by working out the transformation of the z-variable, the so-called *σ-transformation*. In the σ-transformation, the variables c and z are replaced by the new variables c* and σ which are related according to

$$\text{(1.2)}\qquad c(t,x,y,z) = c^*(t,x,y,\sigma) = c^*(t,x,y,\theta(t,x,y,z)),\ \ \sigma = \theta(t,x,y,z) := \frac{z - \zeta(t,x,y)}{h(t,x,y)},$$

where h is the water height defined by $h := d(x,y) + \zeta(t,x,y)$. Evidently, σ takes values in the unit interval [−1,0]. From (1.2) the transformed equation in terms of c*, t, x, y, and σ is easily obtained by substitutions like

$$\frac{\partial c}{\partial t} = \frac{\partial c^*}{\partial t} + \frac{\partial c^*}{\partial \sigma}\frac{\partial \theta}{\partial t},\quad \frac{\partial c}{\partial x} = \frac{\partial c^*}{\partial x} + \frac{\partial c^*}{\partial \sigma}\frac{\partial \theta}{\partial x},\ \ldots,\ \frac{\partial c}{\partial z} = \frac{\partial c^*}{\partial \sigma}\frac{\partial \theta}{\partial z} = \frac{\partial c^*}{h\partial \sigma}.$$

Since the function $\theta(t,x,y,z)$ depends both on t and all spatial variables, the transformed equation is much more complicated, and therefore more expensive, than the original transport equation (1.1a). Furthermore, these expressions show the introduction of large coefficients $\partial\theta/\partial x$ and $\partial\theta/\partial y$ into the model in the case of rapidly changing bottom profile d(x,y). Similarly, transformation of the (x,y)-variable introduces large coefficients in the case of rapidly changing coast geometry. As a consequence, the numerical solution process should be based on an implicit time integration in order to avoid restrictive time step conditions.

The second, 'dummy-point' approach encloses the whole physical domain in a rectangular box and considers this box as the computational domain. As a consequence, there may be a lot of meaningless, dummy grid points. However, in spite of performing calculations at these dummy points, the regular grid facilitates efficient implementation which compensates the additional computational effort. Compared with the coordinate transformation approach, the advantage is the simple mathematical model, the disadvantage is a less accurate representation of the physical boundaries.

The two approaches just outlined can be combined. For example, in De Goede [1], we find such a combined approach for the shallow water equations. In the present paper, we shall follow the dummy-point approach.

2. Semidiscretization

A widely used approach for the discretization of (1.1) is to apply the method of lines (MOL). This semidiscretization process transforms the partial differential equation into a system of ordinary differential equations (ODEs) by discretizing only the spatial derivatives and leaving time continuous. Another approach is the so-called *direct* discretization, in which the derivatives in space and time are *simultaneously* replaced by discrete analogues. For a comparison of both techniques we refer to [10]. Since we shall concentrate in this paper on the time integration aspects, we follow the MOL approach.

Before applying the semidiscretization process, it is convenient to rewrite equation (1.1a) by taking into account the particular applications we have in mind. In this paper, we shall only consider the *incompressible* case without sources, that is, we assume $u_x + v_y + w_z = 0$, and furthermore, the diffusion coefficients ε_x, ε_y, ε_z will be assumed to be constant. Recalling that we concentrate on a single transport equation, (1.1a) simplifies to

$$\text{(2.1)} \qquad \frac{\partial c}{\partial t} = - u \frac{\partial}{\partial x} c - v \frac{\partial}{\partial y} c - w \frac{\partial}{\partial z} c + \frac{\partial}{\partial z}(w_f c) + \varepsilon \left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} + \frac{\partial^2 c}{\partial z^2} \right) + g.$$

The first step in the semidiscretization process is to let the physical domain be enclosed by a rectangular box, containing the grid points

$$\text{(2.2a)} \qquad P_{j,k,m} := (x_0 + j\Delta x,\ y_0 + k\Delta y,\ z_0 + m\Delta z), \quad j = 0,\dots, d_x+1; \quad k = 0,\dots, d_y+1; \quad m = 0,\dots, d_z+1,$$

where (x_0, y_0, z_0) corresponds with the south-west corner point at the bottom of the rectangular box. It will be assumed that the mesh parameters Δx, Δy and Δz are such that the physical boundaries can be approximated by a subset of grid points with sufficient accuracy. This subset of boundary grid points will be denoted by $\partial\mathbb{B}$. Boundary points approximating the vertical boundary plane are assumed to lie in vertical polygonal planes (i.e., each horizontal cross-section produces the same horizontal polygon). Moreover, the polygon just mentioned must have angles larger than 90 degrees, that is, three neighbouring grid points with the same z-coordinate are not allowed to lie on a rectangular triangle. For the water surface points, we assume that no two points are located on the same vertical line. The same assumption is made for the bottom points. The boundary grid points divide the remaining grid points in outer and inner ones lying 'outside' and 'inside' $\partial\mathbb{B}$. These sets of points are denoted by $\mathbb{B}_{out}$ and $\mathbb{B}_{in}$. Furthermore, we assume that the enclosing box is such that no grid points of $\partial\mathbb{B}$ are on the boundary planes of the box (i.e., the indices j, k, m of the boundary points satisfy $1 \le j \le d_x$, $1 \le k \le d_y$, and $1 \le m \le d_z$).

Next the spatial differential operators are replaced by finite differences, so that we can approximate (2.1) by a system of ODEs. Since the convection part of the transport model is nondissipative, we shall use (standard) *symmetric* differences for discretizing $\partial/\partial x$, $\partial/\partial y$ and $\partial/\partial z$ (cf. [14, p. 24]). We remark that there is a considerable amount of literature on the spatial discretization of advection dominated partial differential equations. For a recent overview we refer to [15], in which also *unsymmetric* discretizations (such as upwind) are discussed. Furthermore, we denote the numerical approximations at $P_{j,k,m}$ to c, u, ... by capitals $C_{j,k,m}$, $U_{j,k,m}$, ... , and we introduce the spatial shift operators $\mathbb{X}_\pm$, $\mathbb{Y}_\pm$ and $\mathbb{Z}_\pm$ defined by

$$\mathbb{X}_\pm C_{j,k,m} := C_{j\pm1,k,m}\ , \quad \mathbb{Y}_\pm C_{j,k,m} := C_{j,k\pm1,m}\ , \quad \mathbb{Z}_\pm C_{j,k,m} := C_{j,k,m\pm1}\ .$$

Then, on the *computational* set of grid points $\mathbb{S}$ defined by

$$\text{(2.2b)} \qquad \mathbb{S} := \{P_{j,k,m}\colon 1 \le j \le d_x,\ \ 1 \le k \le d_y\ ,\ \ 1 \le m \le d_z\}$$

the associated ODEs take the form

$$\begin{aligned}\text{(2.3a)} \qquad \frac{dC_{j,k,m}}{dt} &= - \left\{ \frac{1}{2\Delta x} U_{j,k,m}[\mathbb{X}_+ - \mathbb{X}_-] + \frac{1}{2\Delta y} V_{j,k,m}[\mathbb{Y}_+ - \mathbb{Y}_-] + \frac{1}{2\Delta z} W_{j,k,m}[\mathbb{Z}_+ - \mathbb{Z}_-] \right\} C_{j,k,m} \\ &\quad + \varepsilon \left\{ \frac{1}{(\Delta x)^2} [\mathbb{X}_+ - 2 + \mathbb{X}_-] + \frac{1}{(\Delta y)^2} [\mathbb{Y}_+ - 2 + \mathbb{Y}_-] + \frac{1}{(\Delta z)^2} [\mathbb{Z}_+ - 2 + \mathbb{Z}_-] \right\} C_{j,k,m} \\ &\quad + \frac{1}{2\Delta z}\, \omega(C)_{j,k,m} [\mathbb{Z}_+ - \mathbb{Z}_-]\, C_{j,k,m} + g_{j,k,m},\end{aligned}$$

where $\omega(c) := w_f(c) + \dfrac{c\partial w_f(c)}{\partial c}$.

The next step is to take into account the boundary conditions. If $P_{j,k,m}$ is a (physical) boundary point where the boundary condition is of Dirichlet-type, then $C_{j,k,m}$ is explicitly given (see (1.1b)), so that by means of (numerical) differentiation with respect to time, we obtain ODEs of the form

$$\text{(2.3b)} \qquad \frac{dC_{j,k,m}}{dt} = d_{j,k,m}(t),$$

which should replace the ODEs occurring in (2.3a) at all Dirichlet-type boundary points (here, $d_{j,k,m}(t)$ is explicitly determined by the boundary conditions).

If $P_{j,k,m}$ is a boundary point of non-Dirichlet-type, then the corresponding ODE in (2.3a) asks for the concentration at one or more outer grid points. We shall call these concentrations the *auxiliary outer concentrations*. By means of the boundary conditions, these auxiliary outer concentrations can explicitly be expressed in terms of concentrations at inner (or boundary) grid points. For example, let $P_{j,k,m}$ be a boundary point on the water surface. Then the boundary condition at $P_{j,k,m}$ yields (cf. 1.1b))

(2.3c) $$(a_s(t))_{j,k,m}C_{j,k,m} + (2\Delta z)^{-1}\, b_s\left[C_{j,k,m+1} - C_{j,k,m-1}\right] = 0,$$

from which the auxiliary outer concentration $C_{j,k,m+1}$ can be obtained. A similar argument applies at bottom points.

In the case of vertical boundary points, we distinguish the situations indicated below.

x	o	+		x	o	+		x	o	+
*	o	+		*	o	+		x	o	+
x	o	+		x	x	o		x	*	o
	(a)				(b)				(c)	

Here, x, o and + denote gridpoints in the (x,y)-plane lying in $\mathbb{B}_{out}$, $\partial\mathbb{B}$ and $\mathbb{B}_{in}$, respectively. The grid point indicated by $*$ lies in $\mathbb{B}_{out}$ and is the point where the auxiliary outer concentration is needed. Suppose that this point is given by $P_{j,k,m}$. In the situations (a) and (b), the boundary condition in (1.1b) yields for the auxiliary outer concentration $C_{j,k,m}$ the equation

(2.3d) $$(a_v)_{j+1,k,m}C_{j+1,k,m} + b_v(2\Delta x)^{-1}\left(C_{j+2,k,m} - C_{j,k,m}\right) = (c_v)_{j+1,k,m}.$$

Situation (c) is less simple, unless $\Delta x = \Delta y$. In that case, we may write

(2.3e) $$\tfrac{1}{2}\left((a_v)_{j,k+1,m} + (a_v)_{j+1,k,m}\right) + b_v\left(\Delta x\sqrt{2}\right)^{-1}\left(C_{j+1,k+1,m} - C_{j,k,m}\right) = \tfrac{1}{2}\left((c_v)_{j,k+1,m} + (c_v)_{j+1,k,m}\right),$$

again leading to an explicit expression for the auxiliary outer concentration $C_{j,k,m}$. Since we have no particular reason to have non-square meshes in the (x,y)-plane, we shall from now on assume that $\Delta x = \Delta y$.

Finally, we assume vanishing concentrations at all points in the computational set $\mathbb{S}$ which are in $\mathbb{B}_{out}$, as well as on the boundaries of the enclosing box, i.e.,

(2.3f) $$C_{j,k,m} = 0, \quad j = 0, d_x + 1; \quad k = 0, d_y + 1; \quad m = 0, d_z + 1.$$

In conclusion, the equations (2.3) corresponding to the set of grid points $\mathbb{S}$ as defined by (2.2b) define a second-order consistent semidiscretization of the initial-boundary value problem (1.1) of dimension $d := d_x d_y d_z$. Notice that only the concentrations defined by this system of ODEs corresponding to the grid points of $\mathbb{B}_{in}$ and $\partial\mathbb{B}$ are relevant.

In the analysis of time integrators for (2.3), it is more convenient to represent the system (2.3) in the form

(2.4) $$\frac{dC(t)}{dt} = \mathbf{F}(t,C(t)) := A_x(t)C(t) + A_y(t)C(t) + A_z(t)C(t) + N_z(C(t))C(t) + \mathbf{G}(t,C(t)) + \mathbf{B}_x(t) + \mathbf{B}_y(t) + \mathbf{B}_z(t),$$

where $C(t)$ is a vector of dimension d containing all concentrations defined at the points of $\mathbb{S}$, $A_x(t)$, $A_y(t)$ and $A_z(t)$ are d-by-d tridagonal matrices only depending on t, and $N_z(\mathbf{C})$ is a d-by-d tridiagonal matrix which vanishes if the fall velocity w_f occurring in (1.1a) vanishes. The vectors $\mathbf{B}_x(t)$, $\mathbf{B}_y(t)$ and $\mathbf{B}_z(t)$ represent the inhomogeneous contributions of the boundary conditions at the vertical east and west boundaries, at the vertical north and south boundaries, and at the surface and bottom boundaries, respectively.

In order to get some insight into the magnitude of the entries in these arrays, we consider the case where the fluid velocities and the diffusion coefficients can be considered as (locally) constant in space. Then, ignoring boundary conditions and omitting the spatial subscripts (j,k,m), the matrices A_x, A_y, A_z and N_z are defined by the stencils

$$
\begin{array}{llllll}
 & A_x = \frac{1}{(\Delta x)^2} & [\ \varepsilon + \frac{1}{2}\Delta x\, U & -2\varepsilon & \varepsilon - \frac{1}{2}\Delta x\, U &]_{x\text{-direction}}, \\
 & A_y = \frac{1}{(\Delta x)^2} & [\ \varepsilon + \frac{1}{2}\Delta x\, V & -2\varepsilon & \varepsilon - \frac{1}{2}\Delta x\, V &]_{y\text{-direction}}, \\
(2.5) & A_z = \frac{1}{(\Delta z)^2} & [\ \varepsilon + \frac{1}{2}\Delta z\, W & -2\varepsilon & \varepsilon - \frac{1}{2}\Delta z\, W &]_{z\text{-direction}}, \\
 & N_z = \frac{1}{2\Delta z} & [\ -\omega(C) & 0 & \omega(C) &]_{z\text{-direction}}.
\end{array}
$$

These approximate values will be used in our stability analysis of the various time integrators.

3. Basic time integration methods

In this section we discuss potential time integrators for integrating the space-discretized transport equation (2.4) on CRAY-type computers. In particular, we discuss stability, vectorization, parallelization, and storage aspects. We will consider

Stabilized Runge methods,
Locally one-dimensional methods,
Richardson extrapolation of locally one-dimensional methods,
Fractional stabilized Runge methods,
Operator splitting methods,
Predictor-corrector methods.

3.1. Stabilized Runge methods

Suppose that we apply to (2.4) the explicit method

$$
\begin{aligned}
&\mathbf{C}^{(1)} = \mathbf{C}_n + \alpha_1 \Delta t \mathbf{F}(t_n, \mathbf{C}_n),\ \mathbf{C}^{(2)} = \mathbf{C}_n + \alpha_2 \Delta t \mathbf{F}(t_n, \mathbf{C}^{(1)}),\ \dots,\ \mathbf{C}^{(q-2)} = \mathbf{C}_n + \alpha_{q-2}\Delta t \mathbf{F}(t_n, \mathbf{C}^{(q-3)}), \\
(3.1)\quad & \\
&\mathbf{C}^{(q-1)} = \mathbf{C}_n + \frac{1}{2}\Delta t \mathbf{F}(t_n, \mathbf{C}^{(q-2)}),\quad \mathbf{C}_{n+1} = \mathbf{C}_n + \Delta t \mathbf{F}(t_n + \frac{1}{2}\Delta t, \mathbf{C}^{(q-1)}).
\end{aligned}
$$

This method is a q-stage Runge-Kutta method in which the t-arguments in the first q−1 stages are frozen. Per step, it requires q calls of the righthand side function $\mathbf{F}(t,\mathbf{C})$ with only two different values of the t-argument. Thus, in problems where the t-update of $\mathbf{F}(t,\mathbf{C})$ is costly, the method (3.1) may be attractive, even for larger values of q. In the case of (2.4), the main effort in evaluating $\mathbf{F}(t,\mathbf{C})$ comes from the evaluation of the matrices $A_x(t)$, $A_y(t)$, $A_z(t)$, and $N_z(t,\mathbf{C}(t))$, so that (3.1) is effectively a *two-stage* method. Furthermore, it is second-order accurate in time for any set of fixed, bounded coefficients $\{\alpha_1, \dots, \alpha_{q-2}\}$, the storage requirements are modest and the structure of the system (2.4) allows an extremely efficient implementation on vector computers. Moreover, in the applications we have in mind, the vector loops are so long that the availability of several parallel vector processors can be fully exploited without sacrificing vector speed.

The only drawback is the stability condition on the size of the time step which is of the form

$$
(3.2)\qquad \Delta t \le \frac{\beta(q)}{\rho(\partial \mathbf{F}/\partial \mathbf{C})},
$$

where $\rho(\cdot)$ denotes the spectral radius function and $\beta(q)$ defines the stability boundary of the method. The coefficients $\{\alpha_1, \dots, \alpha_{q-2}\}$ can be employed for relaxing the stability condition (3.2) (cf. [4]). The resulting method may be considered as a stabilized Runge method (if q = 2, then Runge's original method appears). A suitable choice of the coefficients depends on the type of the system (2.4), i.e., the nature of the spectrum of $\partial\mathbf{F}/\partial\mathbf{C}$. We distinguish the real, imaginary and 'left halfplane' stability boundary. These boundaries are relevant in the case where the eigenvalues of $\partial\mathbf{F}/\partial\mathbf{C}$ are on the negative axis, the imaginary axis and in the left halfplane, respectively.

If the diffusion terms are dominating in (2.4), i.e., if

$$
\Delta x \ll 4\varepsilon \min\left\{|U|^{-1}, |V|^{-1}\right\},\quad \Delta z \ll 4\varepsilon \min\left\{|W|^{-1}, |\omega(C)|^{-1}\right\},
$$

then the eigenvalues of $\partial F/\partial C$ are essentially negative with spectral radius

(3.3a) $$\rho(\partial F/\partial C) \approx 4\varepsilon \frac{2(\Delta z)^2 + (\Delta x)^2}{(\Delta x \Delta z)^2}.$$

In this case, the so-called Runge-Kutta-Chebyshev (RKC) method which possesses the nearly optimal real stability boundary $\beta(q) = 2q^2/3$, should be effective [6,13]. Since the computational effort per step increases linearly with q, whereas the stability boundary is quadratic in q, the RKC method is an efficient time integrator for diffusion dominated problems.

If convection is dominating, then, within the family of stabilized Runge methods, the four-stage method with $(q, \alpha_1, \alpha_2) = (4, 1/4, 1/3)$ is recommended. This method has the same stability polynomial as the standard, fourth-order Runge-Kutta method, that is, it satisfies the stability condition (3.2) with imaginary stability boundary $\beta(4) = 2\sqrt{2}$. In convection dominated problems, we have

(3.3b) $$\rho(\partial F/\partial C) \approx \frac{\Delta z(|U| + |V|) + \Delta x(|W| + |\omega|)}{\Delta x \Delta z},$$

showing that the stability condition (3.2) requires Δt to be of $O(\Delta x + \Delta z)$ which is often acceptable taking into account the relatively low costs per step. We shall refer to this second-order, four-stage method as the RK24 method.

The stabilized Runge methods are in a sense special purpose methods because the coefficients can be tuned to a particular problem class. This makes them suitable candidates for use in fractional step methods to be discussed in Section 3.3.

3.2. Locally one-dimensional methods

We shall consider the locally one-dimensional (LOD) method of Yanenko [17] in its original form and a modification which is of interest for a parallel implementation (cf. [8]).

3.2.1. LOD method of Yanenko. The LOD method of Yanenko is based on the idea of splitting the righthand side according to the spatial derivatives and to perform an integration step by integrating the fractional equations sequentially by means of the highly stable backward Euler method. When applied to (2.4) we obtain

(3.4) $$\begin{aligned} &C^{(1)} - \Delta t A_x(t_{n+1})C^{(1)} = C_n + \Delta t B_x(t_{n+1}),\\ &C^{(2)} - \Delta t A_y(t_{n+1})C^{(2)} = C^{(1)} + \Delta t B_y(t_{n+1}),\\ &C^{(3)} - \Delta t[A_z(t_{n+1}) + N_z(C^{(3)})]C^{(3)} = C^{(2)} + \Delta t B_z(t_{n+1}),\\ &C_{n+1} - \Delta t G(t_{n+1}, C_{n+1}) = C^{(3)}. \end{aligned}$$

This method is first-order accurate and will be referred to as the LOD1 method. In addition to evaluating the matrices A_x, A_y, and A_z, the costs per step consist of solving d_yd_z tridiagonal, linear systems of dimension d_x, d_xd_z tridiagonal, linear systems of dimension d_y, d_xd_y tridiagonal, nonlinear systems of dimension d_z, and d nonlinear scalar equations. We remark that, without reducing the order of accuracy, the matrix $N_z(C^{(3)})$ in the third stage may be replaced by $N_z(C_n)$. Furthermore, if G does not depend on C, then the fourth stage can be omitted by replacing $B_z(t_{n+1})$ with $B_z(t_{n+1}) + \Delta t G(t_{n+1})$ and by setting $C_{n+1} = C^{(3)}$.

The overall costs of the method (3.4) seem to be extremely high, both in computational volume per step and with respect to storage. The present (shared memory) CRAY computers offer an amount of storage which is in the order of 1 - 2 Gigabytes, corresponding to 128 - 256 Megawords (double precision). Since this LOD1 method requires 16 arrays (cf. Table 3.1), including the arrays for storing the velocity fields, grids of up to 8 - 16 million grid points can be treated.

With respect to the computational effort per step, we observe that in this particular case where so many systems of equal dimension are involved, the solution process can be implemented rather efficiently on vector computers by executing the substeps of the many tridiagonal solvers vectorwise (cf. [2, p.156] and Section 6 of the present paper). This technique was successfully used by de Goede [1] in solving the three-dimensional shallow water equations, and will be referred to as *vectorization-across-tridiagonal-solvers*. Thus, for LOD methods, the computational effort per step can be substantially reduced on vector computers. Moreover, the favourable stability characteristics of the underlying backward Euler method enables us to take relatively large stepsizes. To be more precise, let us ignore the boundary conditions in (3.4) to obtain

$$\begin{aligned}
C_{n+1} &= [I - \Delta t A_z(t_{n+1}) - \Delta t N_z(C_n)]^{-1} C^{(2)} + \Delta t G(t_{n+1}, C_{n+1}) \\
&= [I - \Delta t A_z(t_{n+1}) - \Delta t N_z(C_n)]^{-1} [I - \Delta t A_y(t_{n+1})]^{-1} C^{(1)} + \Delta t G(t_{n+1}, C_{n+1}) \\
&= [I - \Delta t A_z(t_{n+1}) - \Delta t N_z(C_n)]^{-1} [I - \Delta t A_y(t_{n+1})]^{-1} [I - \Delta t A_x(t_{n+1})]^{-1} C_n + \Delta t G(t_{n+1}, C_{n+1}).
\end{aligned}$$

Hence, perturbations of C_n and C_{n+1} are related according to

(3.5a) $$\Delta C_{n+1} = R(\Delta t)\, \Delta C_n,$$

where the stability matrix R is defined by

(3.5b) $$R(\Delta t) := [I - \Delta t J_G]^{-1} [I - \Delta t A_z - \Delta t N_z]^{-1} [I - \Delta t A_y]^{-1} [I - \Delta t A_x]^{-1},$$

with J_G denoting the Jacobian of $G(t,C)$ with respect to C at (t_n, C_n). In the case of LOD methods, it is justified to analyse the stability matrix by means of the normal mode analysis, that is, by freezing the coefficients, the matrices A_x, A_y, A_z, N_z and J_G can be assumed to share the same eigensystem of the form (see, e.g. [14])

$$V(\xi_x, \xi_y, \xi_z) = \{\exp(j\xi_x + k\xi_y + m\xi_z)i \colon P_{jkm} \in \mathbb{S},\ \ 0 \le \xi_x, \xi_y, \xi_z \le \pi\}.$$

The quantities ξ_x, ξ_y and ξ_z are restricted to the interval $[0,\pi]$ (because the grid cannot 'resolve' higher modes). Evidently, if the corresponding eigenvalues of A_x, A_y, A_z, N_z and J_G are in the left halfplane, then we have unconditional, strong stability. Thus, ignoring boundary conditions and a possibly nonvanishing fall velocity, we conclude from (3.5) that the LOD1 method is unconditionally stable provided that the diffusion coefficient ε is nonnegative and the diagonal matrix $\partial G/\partial C$ has nonpositive diagonal entries. The possibility of taking arbitrarily large stepsizes, together with the technique of vectorization-across-tridiagonal-solvers, implies that the overall costs of the LOD approach are acceptable. However, since the LOD1 method is only first-order accurate, large stepsizes may destroy the accuracy. In order to retain accuracy, we may use Richardson extrapolation, to be discussed in Section 3.2.3.

3.2.2. Parallel LOD method. Instead of sequential integration of the fractional equations, we may perform parallel integrations to obtain the method

$$\begin{aligned}
&C^{(1)} - \Delta t A_x(t_{n+1}) C^{(1)} = C_n + \Delta t B_x(t_{n+1}), \\
&C^{(2)} - \Delta t A_y(t_{n+1}) C^{(2)} = C_n + \Delta t B_y(t_{n+1}), \\
&C^{(3)} - \Delta t [A_z(t_{n+1}) + N_z(C^{(3)})] C^{(3)} = C_n + \Delta t B_z(t_{n+1}), \\
&C^{(4)} - \Delta t G(t_{n+1}, C_n^{(4)}) = C_n, \\
&C_{n+1} = \tfrac{1}{4}\big(C^{(1)} + C^{(2)} + C^{(3)} + C^{(4)}\big).
\end{aligned} \tag{3.6}$$

This modification of Yanenko's method is again first-order accurate and differs from (3.4) in that all component equations use the same initial value C_n, rather than using the result of the preceding equation as starting value. As a consequence, the vectors $C^{(i)}$, $i = 1, \dots, 4$ can be computed in parallel. This parallel method becomes attractive on computers where not all parallel vector processors can be fully exploited for vector-loop computations.

The stability is governed by the variational equation

$$\begin{aligned}
&\Delta C_{n+1} = R(\Delta t)\, \Delta C_n, \\
&R(\Delta t) := \tfrac{1}{4}\big([I - \Delta t A_x]^{-1} + [I - \Delta t A_y]^{-1} + [I - \Delta t A_z - \Delta t N_z]^{-1} + [I - \Delta t J_G]^{-1}\big).
\end{aligned} \tag{3.7}$$

Unlike the LOD1 method (3.4), the stability characteristics of the parallel LOD method are more difficult to establish.

3.2.3 Richardson extrapolation. By applying Richardson extrapolation, the accuracy behaviour of LOD methods can be improved, and, since Richardson extrapolation is highly parallel, the sequential costs are hardly increased.

Let us represent the LOD method we want to accurize in the compact form

(3.4') $$C_{n+1} = L(\Delta t, C_n),$$

where the operator L defines the particular method under consideration. Assuming that this method is first-order accurate, we define the two-point and three-point, *local* Richardson extrapolation method by the formulas

(3.8a) $$C_{n+1} = -L(\Delta t, C_n) + 2L(\tfrac{1}{2}\Delta t,\ L(\tfrac{1}{2}\Delta t,\ C_n)),$$

(3.8b) $$C_{n+1} = \tfrac{1}{2}[L(\Delta t, C_n) - 8L(\tfrac{1}{2}\Delta t,\ L(\tfrac{1}{2}\Delta t,\ C_n)) + 9L(\tfrac{1}{3}\Delta t,\ L(\tfrac{1}{3}\Delta t,\ L(\tfrac{1}{3}\Delta t,\ C_n)))],$$

respectively. The *sequential* (or, *effective*) computational costs of the two-point and three-point methods are twice and three times the costs of applying the operator L, because all terms occurring in (3.8) can be computed concurrently. Notice that in the case of (3.8b) there is almost perfect load balancing on two processors, because the computational costs of computing the first two terms is roughly equal to that of computing the last term. Although the sequential (effective) costs per step for (3.8b) are a factor 3/2 higher than those for (3.8a), it is likely that, as far as accuracy is concerned, (3.8b) allows us to take stepsizes that are much more than a factor 3/2 larger. We shall refer to {(3.4), (3.8a)} and {(3.4), (3.8b)} as the LOD2 and LOD3 methods, respectively.

We computed numerically the stability region of the LOD2 method and found that the method is, like the LOD1 method (3.4), unconditionally stable provided that the diffusion coefficient ε is nonnegative and the diagonal matrix $\partial G/\partial C$ has nonpositive diagonal entries. The LOD3 method was verified to be 'almost' unconditionally stable in the sense that amplifications by factors 1.0001 occur if the eigenvalues of A_x, A_y, A_z, N_z or J_G approach certain parts of the imaginary axis.

3.3. Fractional stabilized Runge methods

LOD methods are based on the excellent stability behaviour of the backward Euler method. However, we have seen that the RKC method and the RK24 method are potential candidates if, respectively, diffusion and convection are dominating. This suggests replacing the LOD splitting of the righthand side function by diffusion-convection splitting, that is, we write

$$\frac{dC(t)}{dt} = A_1(t)C(t) + A_2(t)C(t) + N_z(C(t))C(t) + G(t,C(t)) + \mathbf{B}_1(t) + \mathbf{B}_2(t),$$

where the matrices A_2 and A_1 are symmetric and skew-symmetric, respectively, and where $\mathbf{B}_1(t)$ and $\mathbf{B}_2(t)$ stem from the contributions of the boundary conditions. Next, we integrate the fractional equations

(3.10) $$\frac{dC(t)}{dt} = A_1(t)C(t) + N_z(C(t))C(t) + \mathbf{B}_1(t), \quad \frac{dC(t)}{dt} = A_2(t)C(t) + G(t,C(t)) + \mathbf{B}_2(t).$$

Again, two versions can be distinguished, viz. the Yanenko-type version and a parallel version.

3.3.1. Yanenko-type method. As in the LOD method of Yanenko, the idea is to integrate the fractional equations (3.9) sequentially in each step. This yields the first-order accurate method

(3.10) $$C^{(1)} = \mathbf{L}_1(\Delta t, C_n), \quad C_{n+1} = \mathbf{L}_2(\Delta t, C^{(1)}),$$

provided that $\mathbf{L}_1$ and $\mathbf{L}_2$ define explicit integration methods that are at least first-order accurate (cf. [8]). An efficient method is obtained by tuning the operator $\mathbf{L}_1$ to the special properties of A_1 and N_z, and the operator $\mathbf{L}_2$ to the special properties of A_2 and G. Since the eigenvalues of A_1 and N_z are purely imaginary, $\mathbf{L}_1$ should define an integration method that is suitable for equations whose Jacobian matrices possess imaginary spectra, that is, we should choose a *hyperbolic* solver. Likewise, the Jacobian associated with the second equation has real eigenvalues, so that this equation should be integrated by a *parabolic* solver.

Suppose that the operators $\mathbf{L}_1$ and $\mathbf{L}_2$ are respectively defined by the RK24 method and the q-stage RKC method of Section 3.1. Then, the stability is determined by the stability conditions

$$(3.11)\qquad \Delta t \le \frac{2\sqrt{2}}{\rho(A_1(t_n) + N_Z(C_n))}, \quad \Delta t \le \frac{2q^2}{3\rho(A_2(t_n) + J_G(t_n,C_n))},$$

where $J_G(t,C)$ is again the Jacobian of G(t,C). By means of the parameter q the step of the RKC method can be tuned to that of RK24.

3.3.2. Parallel fractional methods. Consider the first-order method

$$(3.12a)\qquad C_{n+1} = \frac{1}{2}[\mathbf{L}_1(\Delta t,C_n) + \mathbf{L}_2(\Delta t,C_n)],$$

or the second-order modification

$$(3.12b)\qquad C_{n+1} = \frac{1}{2}[\mathbf{L}_1(\Delta t,\mathbf{L}_2(\Delta t,C_n)) + \mathbf{L}_2(\Delta t,\mathbf{L}_1(\Delta t,C_n))],$$

where $\mathbf{L}_1$ and $\mathbf{L}_2$ are defined as before, for example, by suitable stabilized Runge methods. The method (3.12a) modifies (3.10) in the same way as (3.6) modifies (3.4). Evidently, the expressions $\mathbf{L}_1(\Delta t,C_n)$ and $\mathbf{L}_2(\Delta t,C_n)$ can be computed in parallel. The stability condition for (3.12) is identical with (3.11).

3.4. Operator splitting methods

The disadvantage of the LOD-type and fractional stabilized Runge methods is their low order in time due to the fractioning of the differential equation. An alternative approach is again based on operator splitting, but it does not integrate fractions of equations but the full equation in the subsequent stages. Consider an arbitrary splitting of the righthand side function **F** in (2.4):

$$(3.13)\qquad \mathbf{F}(t,C(t)) = A(t,C(t)) + B(t,C(t)).$$

Then we can define a family of second-order splitting methods by writing

$$(3.14)\qquad \begin{aligned} C_{n+1/2} - \frac{1}{2}\Delta t A(t_{n+1/2},C_{n+1/2}) &= C_n + \frac{1}{2}\Delta t B(t_n,C_n), \\ C_{n+1} - \frac{1}{2}\Delta t B(t_{n+1},C_{n+1}) &= C_{n+1/2} + \frac{1}{2}\Delta t A(t_{n+1/2},C_{n+1/2}). \end{aligned}$$

The stability of these methods can be studied by linearizing the operators A and B to obtain

$$(3.15)\qquad \Delta C_{n+1} = R(\Delta t)\,\Delta C_n, \quad R(\Delta t) := [I - \frac{1}{2}\Delta t B]^{-1}\,[I + \frac{1}{2}\Delta t A]\,[I - \frac{1}{2}\Delta t A]^{-1}\,[I + \frac{1}{2}\Delta t B].$$

Stability requires the eigenvalues of R(Δt) within the unit circle, or equivalently, the eigenvalues of

$$(3.16)\qquad \begin{aligned} &R^*(\Delta t) = [I - \frac{1}{2}\Delta t B]\,R(\Delta t)\,[I - \frac{1}{2}\Delta t B]^{-1} = R_A(\Delta t)\,R_B(\Delta t), \\ &R_A(\Delta t) := [I + \frac{1}{2}\Delta t A][I - \frac{1}{2}\Delta t A]^{-1}, \quad R_B(\Delta t) := [I + \frac{1}{2}\Delta t B][I - \frac{1}{2}\Delta t B]^{-1}. \end{aligned}$$

Thus, the stability is essentially determined by the matrices $R_A(\Delta t)$ and $R_B(\Delta t)$.

The most familiar splitting is the so-called Alternating Direction Implicit (ADI) splitting where the righthand side function is split according to the spatial dimensions. Unfortunately, within the family (3.14), the ADI idea can only be applied to two-dimensional problems. However, three-dimensional splittings are offered by the odd-even hopscotch (OEH) splitting, the odd-even line hopscotch (OELH) splitting and nested operator splitting (for a discussion of Hopscotch techniques we refer to [3]).

3.4.1. Odd-even hopscotch splitting. For any vector $\mathbf{V}$, let $\mathbf{V}_o$ denote the vector with zero components at all grid points $P_{j,k,m}$ where j+k+m assumes *even* values, and likewise, let $\mathbf{V}_x$ denote the vector with zero components at all grid points $P_{j,k,m}$ where j+k+m assumes *odd* values. Then we may define the odd-even hopscotch (OEH) method by

$$A(t,C(t)) := \mathbf{F}_o(t,C(t)), \quad B(t,C(t)) := \mathbf{F}_x(t,C(t)). \tag{3.17}$$

On substitution into (3.14), we find that {(3.14), (3.17)} can be represented in the form

$$\begin{aligned} C_{x,n+1/2} &= C_{x,n} + \tfrac{1}{2}\Delta t\mathbf{F}_x(t_n,C_n) = C_{x,n} + \big(C_{x,n} - C_{x,n-1/2}\big), \\ C_{o,n+1/2} &= C_{o,n} + \tfrac{1}{2}\Delta t\mathbf{F}_o(t_{n+1/2},C_{n+1/2}), \\ C_{o,n+1} &= C_{o,n+1/2} + \tfrac{1}{2}\Delta t\mathbf{F}_o(t_{n+1/2},C_{n+1/2}) = C_{o,n+1/2} + \big(C_{o,n+1/2} - C_{o,n}\big), \\ C_{x,n+1} &= C_{x,n+1/2} + \tfrac{1}{2}\Delta t\mathbf{F}_x(t_{n+1},C_{n+1}). \end{aligned} \tag{3.190}$$

Since only scalarly implicit relations are to be solved, the OEH method is hardly more expensive than a one-stage explicit RK method. However, in this case where A and B have zero rows, the eigenvectors of $R_A(\Delta t)$ and $R_B(\Delta t)$ are not given by the normal modes $\{\exp\big(j\xi_x + k\xi_y + m\xi_z\big)i\}$ as is the case for LOD methods. Hence, we cannot simply require that the eigenvalues of $R_A(\Delta t)$ and $R_B(\Delta t)$ are within the unit circle to obtain the stability condition for the OEH method. A more sophisticated analysis is necessary. For the model problem (3.1a), such an analysis has been been carried out in [11] resulting in the condition $|U|\Delta t \le 2\Delta x/3$ for all $\varepsilon \ge 0$ and indicating a better stability behaviour when diffusion is introduced. Since the costs per step of the OEH method are roughly equivalent to only one function call, the OEH method may offer an alternative to the RK24 method.

3.4.2. Odd-even line hopscotch splitting. Next, let $\mathbf{V}_o$ and $\mathbf{V}_x$ denote the vectors with zero components at grid points $P_{j,k,m}$ where j+k assumes even and odd values, respectively. As before, we define A and B according to (3.17). The resulting OELH method can again be represented in the form (3.18). Evidently, the implicit equations for $C_{o,n+1/2}$ and $C_{x,n+1}$ become sets of tridiagonally implicit relations that can be solved efficiently by using the vectorization-across-the-tridiagonal-solvers technique.

The derivation of the stability condition for the OELH method seems to be possible along the lines of the OEH analysis given in [11].

3.4.3. Nested splitting methods. Suppose that we define the operators A and B in (3.13) by

$$\begin{aligned} A(t,C(t)) &:= A_x(t)C(t) + A_y(t)C(t) + \mathbf{B}_x(t) + \mathbf{B}_y(t), \\ B(t,C(t)) &:= A_z(t)C(t) + N_z(C(t))C(t) + \mathbf{G}(t,C(t)) + \mathbf{B}_z(t). \end{aligned} \tag{3.19}$$

Since this splitting allows the application of the standard normal mode analysis, we conclude that we have unconditional stability if the eigenvalues of $A_x + A_y$ and $A_z + N_z + J_G$ are in the left halfplane. However, this is only true if the two stages of (3.14) are really solved for $C_{n+1/2}$ and C_{n+1}. Let us solve these two stages iteratively by operator splitting to obtain a 'nested' splitting method. In particular, we may apply ADI iteration. For the first stage, this yields

$$\begin{aligned} C^{(j+1/2)} - \tfrac{1}{2}\Delta t A_x C^{(j+1/2)} &= C_n + \tfrac{1}{2}\Delta t\,\big[A_y C^{(j)} + B(t_n,C_n) + \mathbf{B}_x + \mathbf{B}_y\big], \\ C^{(j+1)} - \tfrac{1}{2}\Delta t A_y C^{(j+1)} &= C_n + \tfrac{1}{2}\Delta t\,\big[A_x C^{(j+1/2)} + B(t_n,C_n) + \mathbf{B}_x + \mathbf{B}_y\big], \end{aligned} \tag{3.20a}$$

where the arrays A_x, A_y, $\mathbf{B}_x$ and $\mathbf{B}_y$ are assumed to be evaluated at $t_{n+1/2}$. The iteration error of this inner iteration method satisfies the recursion

$$C^{(j+1)} - C_{n+1/2} = Q(\Delta t)\,[C^{(j)} - C_{n+1/2}], \quad Q(\Delta t) := \tfrac{1}{4}\,(\Delta t)^2\,[I - \tfrac{1}{2}\Delta t A_y]^{-1} A_x\,[I - \tfrac{1}{2}\Delta t A_x]^{-1} A_y, \tag{3.21}$$

so that we have a comparable situation as in the stability recursion (3.15), that is, we have convergence if the eigenvalues of the matrix

(3.22) $$Q^*(\Delta t) := \tfrac{1}{4}\,(\Delta t)^2\, A_x[I - \tfrac{1}{2}\Delta t A_x]^{-1}\, A_y[I - \tfrac{1}{2}\Delta t A_y]^{-1}$$

are within the unit disk. Since it is now justified to apply the normal mode analysis, we conclude that we have unconditional convergence if A_x and A_y have their normal mode eigenvalues in the left halfplane. Furthermore, the low wavenumber components are quickly removed from the iteration error for sufficiently small Δt. However, for the high wavenumber components, we have $Q^*(\Delta t) \approx 1$, so that these components are not damped in the iteration process (in Section 5 we shall discuss techniques for damping of highly oscillatory components).

The second stage of (3.14) can be solved in a similar way by splitting B according to (3.19):

(3.20b)
$$C^{(j+1/2)} - \tfrac{1}{2}\Delta t\,[A_z + N_z]C^{(j+1/2)} = C_{n+1/2} + \tfrac{1}{2}\Delta t\,[A(t_{n+1/2},C_{n+1/2}) + \mathbf{G}(t_{n+1},C^{(j)}) + \mathbf{B}_z],$$
$$C^{(j+1)} - \tfrac{1}{2}\Delta t G(t_{n+1},C^{(j+1)}) = C_{n+1/2} + \tfrac{1}{2}\Delta t\,[A(t_{n+1/2},C_{n+1/2}) + A_zC^{(j+1/2)} + N_zC^{(j+1/2)} + \mathbf{B}_z],$$

where now the arrays A_z, N_z, and $\mathbf{B}_z$ are evaluated at t_{n+1}. We remark that the matrix N_z in the first stage of this iteration process can be replaced by $N_z(C_n)$ without reducing the order of accuracy. Again, we arrive at the convergence condition requiring that the normal mode eigenvalues of $A_z + N_z$ and J_G should be in the left halfplane.

We remark that one iteration in each of the four stages of {(3.20a), (3.20b)} suffices to achieve second-order accuracy. The initial iterate in (3.20a) may be defined by C_n and in (3.20b) by the result of (3.20a). The resulting four-stage nested ADI method (3.20) differs from the four-stage LOD method (3.4) by the righthand sides and by the factor $\Delta t/2$ instead of Δt in the lefthand sides.

3.5. Predictor-corrector methods

The methods reviewed so far are either unconditionally, strongly stable but only first-order in time, or second-order in time but at best marginally stable. In this section, we introduce predictor-corrector-type methods that are both second-order in time and unconditionally, strongly stable. For the corrector we choose the second-order backward differentiation formula (BDF):

(3.23a) $$C_{n+1} = \mathbf{L}_n(C_{n+1}),\quad \mathbf{L}_n(C) := -\tfrac{1}{3}C_{n-1} + \tfrac{4}{3}C_n + \tfrac{2}{3}\Delta t\,\mathbf{F}(t_{n+1},C).$$

Let **F** be split according to (3.13). Then, we iterate this BDF corrector by means of the iteration scheme

(3.23b)
$$C^{(j+1/2)} - \tfrac{2}{3}\Delta t A(t_{n+1},C^{(j+1/2)}) = \mathbf{L}_n(C^{(j)}) - \tfrac{2}{3}\Delta t A(t_{n+1},C^{(j)}),$$
$$C^{(j+1)} - \tfrac{2}{3}\Delta t B(t_{n+1},C^{(j+1)}) = \mathbf{L}_n(C^{(j+1/2)}) - \tfrac{2}{3}\Delta t B(t_{n+1},C^{(j+1/2)}),\qquad j = 0, 1, \ldots .$$

Clearly, if this iteration method converges, then it converges to the second-order corrector solution, and at the same time, it generates an unconditionally L-stable solution.

In first approximation, the condition for convergence is obtained by linearization of A and B. Then, the iteration error satisfies

(3.24) $$C^{(j+1)} - C_{n+1} = Q(\Delta t)\,[C^{(j)} - C_{n+1}],\quad Q(\Delta t) := \tfrac{4}{9}\,(\Delta t)^2\,[I - \tfrac{2}{3}\Delta t B]^{-1} A\,[I - \tfrac{2}{3}\Delta t A]^{-1} B.$$

Again assuming that the normal mode analysis can be applied, that is, A and B share the same eigensystem, we conclude that we have unconditional convergence if the normal mode eigenvalues are in the left halfplane (compare the discussion of (3.21)).

The actual solution of the stages in (3.23) again requires an (inner) iteration process. For example, if A and B are defined according to (3.19), then the inner iteration method may be defined as in the nested ADI method described in Section 3.4.3. We remark that the hopscotch splittings will not lead to unconditional convergence.

In actual computation, we hope to restrict the computational effort to only a few iterations. As to the order of accuracy, this is justified, because, assuming that the predictor formula providing the initial iterate $C^{(0)}$ is at least of zero order, all iterates $C^{(j)}$ with $j \geq 1$ are already second-order accurate. However, what about the stability when not iterating to convergence. We shall consider the stability of (3.23) after one full iteration, that is the stability of $C_{n+1} = C^{(1)}$. Assuming that the operators A and B in the splitting (3.13) are linear, we may write

$$\Delta L_n(C) := -\tfrac{1}{3}\Delta C_{n-1} + \tfrac{4}{3}\Delta C_n + \tfrac{2}{3}\Delta t\,[A+B]\,\Delta C,$$

$$\Delta C^{(1/2)} = [I - \tfrac{2}{3}\Delta t A]^{-1}\,[\Delta L_n(C^{(0)}) - \tfrac{2}{3}\Delta t A \Delta C^{(0)}]$$

$$= \tfrac{1}{3}\,[I - \tfrac{2}{3}\Delta t A]^{-1}\,[-\Delta C_{n-1} + 4\Delta C_n + 2\Delta t B \Delta C^{(0)}],$$

$$\Delta C_{n+1} = [I - \tfrac{2}{3}\Delta t B]^{-1}\,[\Delta L_n(C^{(1/2)}) - \tfrac{2}{3}\Delta t B \Delta C^{(1/2)}]$$

$$= \tfrac{1}{3}[I - \tfrac{2}{3}\Delta t B]^{-1}\,[-\Delta C_{n-1} + 4\Delta C_n + 2\Delta t A \Delta C^{(1/2)}].$$

Hence,

(3.25) $$9[I - \tfrac{2}{3}\Delta t A][I - \tfrac{2}{3}\Delta t B]\,\Delta C_{n+1} = 3\,[-\Delta C_{n-1} + 4\Delta C_n] + 4(\Delta t)^2 AB\Delta C^{(0)}.$$

Let us assume that $C^{(0)}$ is computed by a one-step formula satisfying the variational equation

$$\Delta C^{(0)} = P\Delta C_n,$$

where P is a given matrix. Let us assume that A, B and P have a common eigensystem. Then, we may expand C_n in terms of $\zeta^n V$, where V is an eigenvector of A, B and P with eigenvalues $\lambda(A)$, $\lambda(B)$ and $\lambda(P)$, and ζ^n is the corresponding coefficient. On substitution into (3.25) we obtain the characteristic equation

(3.26) $$[3 - 2\Delta t\lambda(A)][3 - 2\Delta t\lambda(B)]\,\zeta^2 - 4[3 + (\Delta t)^2\lambda(ABP)]\,\zeta + 3 = 0.$$

As an example, we consider the splitting (3.13) in which A corresponds to the diffusion part and B to the convection part of the equation. Hence, $\lambda(A)$ and $\lambda(B)$ are assumed to be real and purely imaginary, respectively. Defining $x := \Delta t\lambda(A)$, $i\,y := \Delta t\lambda(B)$ and choosing the 'trivial' prediction $C^{(0)} = C_n$ (i.e., $P = I$), the characteristic equation reads

(3.26') $$[3 - 2x][3 - 2iy]\,\zeta^2 - 4[3 + ixy]\,\zeta + 3 = 0.$$

Applying Schur's theorem (cf. e.g. [4, p. 299]), a straightforward, but tedious, calculation leads to the conclusion that – for all nonpositive x – the roots ζ of (3.26') satisfy $|\zeta| < 1$ (with the exception of the origin, of course, where we have one characteristic root equal to 1). Since the eigenvalues of the discretized diffusion operator are negative indeed, we conclude that this predictor-corrector method with the above splitting is unconditionally stable. Finally, it is of interest to remark that one characteristic root $\zeta \to 1$ if *both* x and y tend to infinity. However, along the axis ($x = 0$ or $y = 0$), both characteristic roots tend to zero. Hence, in the case of purely convection or purely diffusion, this method is L-stable.

3.6. Summary of characteristics of the time integrators. In Table 3.1, we summarized the main properties of the various integrators discussed in the preceding sections.

Table. 3.1. Characteristics of the time integrators.

Method	RK24	OEH	OELH	LOD1	LOD2**	LOD3**	Nested ADI	BDF-ADI
Formula	(3.1)	{(3.14), (3.17)}	{(3.14), (3.17)}	(3.4)	{(3.4), (3.8a)}	{(3.4), (3.8b)}	(3.20)	(3.23)
Δt condition	conditional	conditional	conditional	unconditional	unconditional	unconditional	unconditional	unconditional
Stability	strong	weak	weak	strong	strong	strong	weak	strong
Order of accuracy in time	2	2	2	1	2	3	2	1
Effective **F**(t,C) calls per step	2	1	1	1	2	3	2	≥ 2
Tridiagonal systems per step	none	none	$d_x d_y$	$d_x d_y+d_x d_z+d_y d_z$	$2(d_x d_y+d_x d_z+d_y d_z)$	$3(d_x d_y+d_x d_z+d_y d_z)$	$d_x d_y+d_x d_z+d_y d_z$	$d_x d_y+d_x d_z+d_y d_z$
Vectorization aspects	++	+	+	++	++	++	++	++
Parallelization aspects	++	+	+	++	++	++	+	+
Ease of implementation	++	+	+/–	–	–	–	–	–
Number of arrays*	3	4	6.5	13	26	27	13	14

* excluding the storage needed to store the velocity field (u, v, w)

** For the LOD2 and LOD3 methods it is assumed that the terms in (3.8a) and (3.8b) are computed *concurrently*. Implementing these methods *sequentially* would reduce the required number of arrays to 14 and 15, respectively. However, in that case, the number of effective **F**-calls and tridiagonal systems per step should be multiplied by 1.5 and 2, respectively.

4. Numerical experiments

To test the methods selected in Section 3.6, we take the following example problem

$$\text{(4.1a)}\qquad \frac{\partial c}{\partial t} + \frac{\partial (uc)}{\partial x} + \frac{\partial (vc)}{\partial y} + \frac{\partial (wc)}{\partial z} = \varepsilon \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \right) c + g(t,x,y,z), \quad 0 \le t \le T,$$

with Neumann conditions of the form $\partial c/\partial n = h(t,x,y,z)$ on all boundaries of the physical domain $0 \le x \le L_x$, $0 \le y \le L_y$, $-L_z \le z \le 0$. In our experiments, it is convenient to have the exact solution at our disposal. Therefore, we prescribe the concentration c as

$$\text{(4.1b)}\qquad c(t,x,y,z) = \exp\left\{ \frac{z}{L_z} - \frac{t}{T^\gamma} - \left(1 - \frac{t}{T^\gamma}\right)\left[\left(\frac{x}{L_x} - \frac{1}{2}\right)^2 + \left(\frac{y}{L_y} - \frac{1}{2}\right)^2\right]\right\},$$

where $L_x = L_y = 20\ 000$ [m], $L_z = 100$ [m], and γ is a (dimensionless) parameter which is set to 1.05. The concentration and the diffusion coefficient (to be specified in the tables of results) are assumed to be measured in kg/m^3 and m^2/s, respectively.

We distinguish two cases, referred to as the *small scale problem* and the *large scale problem.* In the small scale problem, the spatial grid is defined by $d_x = d_y = 41$, $d_z = 11$, and the fluid velocities u, v and w are kept *constant* with values to be specified in the tables of results. This problem will serve to get insight into the stability behaviour and the effect of the diffusion coefficient on the performance of the selected methods. In the large scale problem, the grid parameters are given by $d_x = d_y = 101$ and $d_z = 11$, amounting to 10^5 grid points. The velocity fields are prescribed by the analytical expressions

$$\text{(4.2)}\qquad \begin{aligned} u(t,x,y,z) &= C_1 \sin\left(\frac{x}{L_x} + \frac{y}{L_y}\right) \sin\left(\beta \frac{z}{L_z}\right) f(t), \\ v(t,x,y,z) &= C_2 \cos\left(\frac{x}{L_x} + \frac{y}{L_y}\right) \sin\left(\beta \frac{z}{L_z}\right) f(t), \\ w(t,x,y,z) &= \left[-\frac{C_2}{L_x} \cos\left(\frac{x}{L_x} + \frac{y}{L_y}\right) + \frac{C_1}{L_y} \sin\left(\frac{x}{L_x} + \frac{y}{L_y}\right)\right] \left[-\frac{L_z}{\beta} \cos\left(\beta \frac{z}{L_z}\right)\right] f(t), \end{aligned}$$

with $f(t) = \cos(t/T)$ and $C_1 = 3$, $C_2 = 4$, $\beta = 0.05$. These fluid velocities satisfy the relation for local mass balance, i.e.,

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0.$$

The computational complexity of this test problem is substantial and resembles that of the large scale problems occurring in realistic situations.

In both test problems, the source function g in (4.1a) and the function h defining the boundary conditions follow from the exact solution (4.1b).

4.1. Stability tests in the pure convection case

Since a vanishing diffusion coefficient yields the most stringent case as far as stability is concerned and the most difficult case with respect to accuracy, we will first give results for $\varepsilon = 0$. Table 4.1 lists the global errors (with respect to the PDE solution (4.1b)) for various methods when applied to the small scale problem for different values of the fluid velocities.

From this table we see that all LOD methods behave stably, independent of the values of the fluid velocities and the size of the time step. It is clear however, that the first-order LOD method is not sufficiently accurate.

From the results given in the cases II and III the stability constraints for the OELH method are also clear: since this method is implicit in the vertical, large values for $\Delta t\ w/\Delta z$ do not cause any stability problems. We might say that the OELH method is "unconditionally stable in the vertical" (cf. case III). This is in sharp contrast with its behaviour in the horizontal; here we meet the constraint $\Delta t \max(|u|/\Delta x, |v|/\Delta y) \le 1$ (see case II; an * denotes an unacceptable performance). However, in many practical situations this condition on the time step is not a very severe restriction. Moreover, if the OELH method behaves stably, then it is very accurate; the errors are similar to those of LOD3, but OELH is much cheaper per step.

The stability behaviour of RK24 nicely obeys the theoretical considerations discussed in Section 3.1. The experiments given in case IV show that RK24 behaves unstably as soon as $\Delta t\,(|u|/\Delta x + |v|/\Delta y + |w|/\Delta z)$ gets larger than $2\sqrt{2}$. Note that, with respect to the horizontal spatial discretization this condition is less restrictive than that of OELH;

however, since the most stringent stability condition is usually imposed by the vertical spatial discretization, OELH is to be preferred.

Table 4.1. Global errors for the small scale problem with $\varepsilon = 0$.

Case	u = v	w	T	Δt	$\Delta t\ u/\Delta x$	$\Delta t\ w/\Delta s$	OELH	LOD3	LOD2	LOD1	RK24
I	1	1	1000	100	0.2	10	.011	.011	.084	4.1	*
				60	0.1	5	.0085	.0085	.022	2.1	*
				25	0.04	2	.0077	.0079	.0062	.84	.0071
II	10	1	2000	50	1.0	5	.021	.022	.024	5.7	*
	11	1	2000	50	1.1	5	*	.020	.023	5.6	*
III	1	5	2000	50	0.1	25	.078	.096	.43	*	*
			4000	50	0.1	25	.16	.17	.52	*	*
	1	10	4000	50	0.1	50	.31	.32	2.3	*	*
	1	20	4000	50	0.1	100	.62	.62	9.7	*	*
IV	1	0.5	2000	50	0.1	2.5	.0080				.0076
			4000	50	0.1	2.5	.016				.016
			8000	50	0.1	2.5	.033				.033
	3	0.5	2000	50	0.3	2.5	.0083				.0078
			4000	50	0.3	2.5	.019				1.54
			8000	50	0.3	2.5	.036				*

4.2. The effect of diffusion on stability and accuracy

Next we illustrate the effect of adding diffusion in the model problem. Its influence on the stability is quite small (with the current values of Δx, Δy and Δz), but the effect on the accuracy is significant.

Table 4.2. Global errors for the small scale problem with $\varepsilon = 0.5$.

Case	u = v	w	T	Δt	$\Delta t\ u/\Delta x$	$\Delta t\ w/\Delta s$	OELH	LOD3	LOD2	LOD1	RK24
I	1	1	1000	100	0.2	10	.00086	.0048	.037	.37	*
				50	0.1	5	.00072	.0025	.017	.18	*
				25	0.04	2	.00067	.00074	.0042	.073	.0071
II	10	1	2000	50	1.0	5	.00067	.0026	.018	.18	*
	11	1	2000	50	1.1	5	*	.0026	.018	.18	*
III	1	5	2000	50	0.1	25	.0017	.032	.20	2.5	*
			4000	50	0.1	25	.0016	.032	.20	2.5	*
	1	10	4000	50	0.1	50	.0029	.091	.75	9.2	*
	1	20	4000	50	0.1	100	.0055	.32	3.00	35.1	*
IV	1	0.5	8000	50	0.1	2.5	.00067				.00067
	3	0.5	8000	50	0.3	2.5	.00067				.00067
	5	0.5	8000	50	0.5	2.5	.00070				*

Again, the accuracy of the LOD1 method is not satisfactory, for the LOD2 method it is acceptable, and the extension to the third-order variant is worth the extra effort. The OELH - whenever stable - yields a much better accuracy. The same is true for the RK24 method, but this method is subject to a very stringent stepsize restriction. We remark that, due to stability conditions, the errors obtained by OELH and RK24 are dominated by the *spatial* discretization error, whereas the accuracies produced by the LOD-methods still contain a substantial contribution from the time integration. Furthermore, for RK24 we observe the small stabilizing effect of a nonvanishing diffusion, which is in correspondence with the well known shape of its stability region.

4.3. The large scale problem

In the next table we give the results of a selected number of time integration techniques when applied to the large scale problem {(4.1), (4.2)} with $\varepsilon = 0.5$ and T = 10 000.

Table 4.3. Global errors for the large scale problem {(4.1), (4.2)} with $\varepsilon = 0.5$ and T = 10 000.

# steps	Δt	OELH	LOD3	LOD2	LOD1	RK24
10	1000	0.00089	0.13	0.50	1.81	*
20	500	0.00069	0.030	0.15	0.65	*
40	250	0.00065	0.0062	0.044	0.27	*
80	125	0.00064	0.0013	0.012	0.12	*
160	62.5		0.00063	0.0033	0.056	0.00064
320	31.2		0.00064	0.00093	0.027	0.00064
640	15.6			0.00062	0.013	
1280	7.8				0.0068	

For this problem the spatial discretization error is given by 0.00064. Table 4.3 shows that the errors of all methods converge to this value for $\Delta t \rightarrow 0$. However it is clear that the OELH is the most accurate time integration process, since already for Δt as large as 500, the time integration error is almost negligible compared to the spatial error. Again we observe that the first-order LOD1 method (and to some extent the second-order variant as well) needs very small time steps to let the temporal error be of the same order as the spatial error. If the RK24 method is stable, then its accuracy is sufficient, however a stable behaviour requires a (too) small time step.

In conclusion, for this problem, the OELH method is by far superior, especially if we take into account the required computational effort per step.

5. Smoothing

It is well known that the convection terms in the equation (2.4) may cause high wavenumber oscillations in the numerical solution. By introducing smoothing operators into the numerical scheme, these oscillations can be suppressed. There are various ways of applying smoothing. The most direct way is the smoothing of the numerical solution itself, that is, as soon as a numerical approximation C_{n+1} is obtained, it is replaced by SC_{n+1}, where S denotes a smoothing (or, averaging) matrix. A more sophisticated way of smoothing places smoothing matrices in front of the righthand side terms occurring in the numerical scheme. Naturally, solution smoothing and righthand side smoothing can be combined. In all cases, the smoothing matrices are chosen such that the high wavenumber components from the numerical solution or the righthand side terms are removed. When using smoothers in implicit methods, we should be careful not to destroy the nature of the implicit relations to be solved.

In the stabilized Runge methods, we simply replace all function calls $\mathbf{F}(t,\mathbf{C}(t))$ by $S\mathbf{F}(t,\mathbf{C}(t))$ where S is an explicitly given matrix. This approach was successfully applied by Wubs [16] for the two-dimensional shallow water equations where the standard RK method was actually used as time integrator (a detailed analysis of the smoothing operators of Wubs and its generalizations may be found in [9, 7]).

The LOD methods (3.4) and (3.6) are not expected to benefit from smoothing because strong damping of high wavenumber components is already provided by the method itself.

The explicit fractional Runge methods (3.10) and (3.12) that are based on the righthand side splitting given in (3.9) can be improved by replacing the matrices A_1, A_2 and N_z by S_1A_1, S_2A_2 and S_2N_z, respectively. Here, the same smoothing matrices as used in [9] are suitable.

In the case of the splitting methods (3.14), we introduce the smoothed version

$$(5.1)\qquad \begin{aligned} C_{n+1/2} - \tfrac{1}{2}\Delta t A(t_{n+1/2},C_{n+1/2}) &= C_n + \tfrac{1}{2}\Delta t S_B B(t_n,C_n), \\ C_{n+1} - \tfrac{1}{2}\Delta t B(t_{n+1},C_{n+1}) &= C_{n+1/2} + \tfrac{1}{2}\Delta t S_A A(t_{n+1/2},C_{n+1/2}), \end{aligned}$$

where S_A and S_B are given matrices. The corresponding variational equation takes the form

$$(5.2)\qquad \Delta C_{n+1} = R(\Delta t)\,\Delta C_n,\quad R(\Delta t) = [I - \tfrac{1}{2}\Delta t B]^{-1}\,[I + \tfrac{1}{2}\Delta t S_A A]\,[I - \tfrac{1}{2}\Delta t A]^{-1}\,[I + \tfrac{1}{2}\Delta t S_B B].$$

By an appropriate choice of the smoothing matrices S_A and S_B, that is adapting these matrices to the (type of the) spectrum of A and B, the stability restriction on the time step can be considerably relaxed. Usually, the price to be paid for this gain is a reduced accuracy; however, since the RK type integrators have sufficient accuracy, and moreover the smoothers allow for an extremely cheap implementation on vector computers [7], the overall performance greatly benefits from this technique so that RK methods may become competitive with implicit integrators.

6. Implementational aspects

In Section 4 we have discussed several experiments in which we focused on the numerical properties of the various methods (viz., stability and accuracy). The major aim of these tests was to select an appropriate time integration technique, of course, also taking into account the parallelization/vectorization properties that the particular methods possess (cf. Table 3.1). The actual implementation of the integrator on a CRAY-type machine will be described in a forthcoming report.

From the results presented in Section 4, we draw the conclusion that an *implicit* method is most suitable to perform the time integration. Within this class of methods, the OELH and LOD3 method seem to be very promising candidates. A common feature of both methods is that they spend most of their time in solving the tridiagonal systems. We recall that the LOD method has to solve such systems in all three spatial directions, whereas the OELH method is only implicit in the vertical direction, requiring the solution of $d_x d_y$ systems of dimension d_z. Therefore, an efficient implementation of this part of the solution process is of crucial importance and will be discussed in the next subsections.

6.1. Vectorization across tridiagonal systems

Here we will give a brief outline of the vectorization-across-tridiagonal-systems approach; details will be reported in a forthcoming paper.

Both the LOD and OELH integrator give rise to the frequent solution of tridiagonal systems of the form

(6.1a) $\quad T\,x = b,$

where T is a block-tridiagonal matrix of the special form

(6.1b) $\quad T = I - \Delta t\, J,$

I being the identity matrix and J denoting the Jacobian matrix resulting from solving the implicit relation using Newton's method.

Due to the special properties of both time integraton techniques, (6.1) represents a large number of *uncoupled* tridiagonal systems (cf. Table 3.1 for details). A straightforward approach to solve N uncoupled systems, each of dimension d, would be

```
(ALG1)    for i = 1 until N
              factorize each of the d-by-d blocks into LU
              and solve the corresponding LUx = b.
```

Although this i-loop is perfectly parallelizable, the *recursive* nature of the body of this loop prevents vectorization and hence ALG1 will result in a bad performance on (shared memory) vector machines.

A great improvement can be obtained by interchanging the loops [2, p.156]:

```
(ALG2)    for j = 1 until d
              perform for each of the N systems the required
              calculations in the factorization and
              forward substitution process.

          for j = d until 1 with step −1
              perform for each of the N systems the
              backward substitution process.
```

Now, the bodies of these loops, which involve vectors of length N, are very well suited for vectorization. Since usually $N >> d$, we may expect a good performance. Algorithm ALG2 has been implemented on the CRAY YMP4 and the results (both for scalar mode and vector mode) are given in Table 6.1. In this table we list the CPU time and the associated Mflop rate for a tridiagonal system corresponding to a physical domain with $d_x = d_y = 101$ and $d_z = 11$ (cf. the large scale problem discussed in Section 4).

In our implementation the tridiagonal matrix J is provided by means of 3 arrays, housing the lower-, main- and upper diagonal elements, respectively. Each of these arrays has 3 subscripts, 2 of these correspond to a particular point in the plane and the third one runs along the elements of the particular system. As a consequence, the *ordering* of these subscripts has a serious impact on the performance (recall that LOD-type methods have to solve systems in all three spatial directions). Therefore, special provisions have been made to cope with this difficulty, resulting in an almost constant performance, independent of the particular ordering; full details of these provisions will be discussed in a forthcoming report, describing the performance of the *total* solution process on a CRAY machine.

Table 6.1. CPU times (in milli-seconds) and Mflop rates on a CRAY YMP4, using 1 processor

scalar mode		vector mode		
CPU	Mflops	CPU	Mflops	speedup
102.0	17.1	8.9	197.5	11.5

When comparing the results for scalar and vector mode, we observe a speedup of 11.5, which is in good agreement with what is expected for the CRAY YMP4 (cf. CF77 Compiling System: Fortran Reference Manual). To appreciate the obtained Mflop rates, we remark that the peak performance for this configuration (with one processor) is about 330 Mflops. Note however, that this number is based on the ideal situation: infinite loops in which the arithmetic operations can be combined to so-called 'linked triads' (giving a speedup with a factor 2). However, in the current algorithm this facility can only partly be utilized. Taking this into account, the given Mflop-rates indicate that the vectorization-across-tridiagonal-systems approach performs quite efficiently.

6.2. Further refinements

To reduce storage requirements, our tridiagonal solver TRIDIA overwrites the matrix J with the decomposition of $I - \Delta tJ$. However, in a realistic implementation of a time integration process the time step will change, due to error control. For such situations TRIDIA has an option to reconstruct the matrix J from the stored decomposition of $I - \Delta t_{old}J$, decompose $I - \Delta t_{new}J$, and solve the system. Apparently, this call of TRIDIA (which will be referred to as "$call_{newstep}$") is more expensive than the standard call ("$call_{standard}$") with only decomposition and solution. Therefore, the user should only resort to this possibility when it would be more expensive (or not at all possible) to keep copies of the original coefficients of the matrix J. Note that if J has changed, the only feasible way to call TRIDIA is using the "standard" option. Finally, we provided TRIDIA with a third option, which can be used when both Δt and J remain unchanged. In this case the LU-decomposition part of the algorithm can be skipped and only the forward/backward substitutions are needed to find the solution. This option is denoted by "$call_{save}$". In Table 6.2, we present results of the tests on the CRAY YMP4. This table shows that we obtain a satisfactory speedup, also for the nonstandard options (notice that the results of Table 6.1 are reproduced with the label $call_{standard}$).

Table 6.2. CPU times (in milli-seconds) and Mflop rates on a CRAY YMP4 (using 1 processor) for various options.

	scalar mode		vector mode		
option	CPU	Mflops	CPU	Mflops	speedup
$call_{standard}$	102.0	17.1	8.9	197.5	11.5
$call_{save}$	58.7	9.3	3.7	147.9	15.9
$call_{newstep}$	184.0	16.2	16.5	180.0	11.2

References

[1] De Goede, E.D., *Numerical methods for the three-dimensional shallow water equations on supercomputers*, Thesis, University of Amsterdam, 1992.

[2] Golub, G.H. and Van Loan, C.F., *Matrix Computations*, The Johns Hopkins Press, Baltimore, Maryland, 2nd edition, 1989.

[3] Gourlay, A.R., *Hopscotch: a fast second order partial differential equation solver*, J. Inst. Math. Appl. 6, 1970, 375-390.

[4] Hairer, E. and Wanner, G., *Solving Ordinary Differential Equations II; Stiff and Differential-Algebraic Problems*, Springer Series in Comput. Math., Vol. 14, Springer, Berlin, 1989.

[5] Houwen, P.J. van der, *Construction of Integration Formulas for Initial Value Problems*, North-Holland series in Applied Mathematics and Mechanics, Vol. 19, North-Holland, 1977.

[6] Houwen, P.J. van der and Sommeijer, B.P., *On the internal stability of explicit, m-stage Runge-Kutta methods for large m-values*, Z. Angew. Math. Mech. 60, 1980, 479-485.

[7] Houwen, P.J. van der and Sommeijer, B.P., *Improving the stability of predictor-corrector method by residue smoothing*, IMA J. Numer. Anal. 9, 1990, 371-378.

[8] Houwen, P.J. van der and Sommeijer, B.P., *Fractional Runge-Kutta methods with application to convection-diffusion equations*, Impact Comput. Sc. Engin. 4, 1992, 195-216.

[9] Houwen, P.J. van der, Sommeijer, B.P. and Wubs, F.W., *Analysis of smoothing operators in the solution of partial differential equations by explicit difference schemes*, Appl. Numer. Math. 6, 1989, 501-521.

[10] Hundsdorfer, W. and Trompert, R.A., *Method of lines and direct discretization - a comparison for linear advection*, Report NM-R9314, CWI, Amsterdam, 1993.

[11] Thije Boonkkamp, J.H.M. ten and Verwer, J.G., *On the odd-even hopscotch scheme for the numerical integration of time-dependent partial differential equations*, Appl. Numer. Math. 3, 1988, 183-193.

[12] Toro, M., Rijn, L.C. van and Meijer, K., *Three-dimensional modelling of sand and mud transport in currents and waves*, Technical Report No. H461, Delft Hydraulics, Delft, The Netherlands, 1989.

[13] Verwer, J.G., Hundsdorfer, W.H. and Sommeijer, B.P., *Convergence properties of the Runge-Kutta-Chebyshev method*, Numer. Math. 57, 1990, 157-178.

[14] Vichnevetsky, R. and Bowles, J.B., *Fourier Analysis of Numerical Approximations of Hyperbolic Equations*, SIAM, Philadelphia, 1982.

[15] Vreugdenhil, C.B. and Koren, B. (eds.), *Numerical Methods for Advection-Diffusion Problems*, Notes on Numerical Fluid Mechanics, Vol. 45, Vieweg, Braunschweig, to appear in 1993.

[16] Wubs, F.W., *Numerical solution of the shallow-water equations*, Thesis, University of Amsterdam, 1987.

[17] Yanenko, N.N., *The Method of Fractional Steps*, Springer-Verlag, Berlin, 1971.