IMPLICIT AND SEMI-IMPLICIT SCHEMES: ALGORITHMS

R. KEPPENS[a,*], G. TÓTH[b,1], M.A. BOTCHEV[c,d,2] AND A. VAN DER PLOEG[d,3]

[a] *FOM-Institute for Plasma-Physics Rijnhuizen, PO Box 1207, 3430 BE Nieuwegein, Netherlands*
[b] *Astronomical Institute, Utrecht University, PO Box 80000, 3508 TA Utrecht, Netherlands*
[c] *Mathematical Institute, Utrecht University, PO Box 80010, 3508 TA Utrecht, Netherlands*
[d] *Centrum voor Wiskunde en Informatica, PO Box 94079, 1090 GB Amsterdam, Netherlands*

SUMMARY

This study formulates general guidelines to extend an explicit code with a great variety of implicit and semi-implicit time integration schemes. The discussion is based on their specific implementation in the Versatile Advection Code, which is a general purpose software package for solving systems of non-linear hyperbolic (and/or parabolic) partial differential equations, using standard high resolution shock capturing schemes. For all combinations of explicit high resolution schemes with implicit and semi-implicit treatments, it is shown how second-order spatial and temporal accuracy for the smooth part of the solutions can be maintained. Strategies to obtain steady state and time accurate solutions implicitly are discussed. The implicit and semi-implicit schemes require the solution of large linear systems containing the Jacobian matrix. The Jacobian matrix itself is calculated numerically to ensure the generality of this implementation. Three options are discussed in terms of applicability, storage requirements and computational efficiency. One option is the easily implemented matrix-free approach, but the Jacobian matrix can also be calculated by using a general grid masking algorithm, or by an efficient implementation for a specific Lax–Friedrich-type total variation diminishing (TVD) spatial discretization. The choice of the linear solver depends on the dimensionality of the problem. In one dimension, a direct block tridiagonal solver can be applied, while in more than one spatial dimension, a conjugate gradient (CG)-type iterative solver is used. For advection-dominated problems, preconditioning is needed to accelerate the convergence of the iterative schemes. The modified block incomplete LU-preconditioner is implemented, which performs very well. Examples from two-dimensional hydrodynamic and magnetohydrodynamic computations are given. They model transonic stellar outflow and recover the complex magnetohydrodynamic bow shock flow in the switch-on regime found in De Sterck *et al.* [*Phys. Plasmas*, **5**, 4015 (1998)]. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: conservation laws; numerical approximation; PDEs; initial value; time-dependent initial boundary value problems; magnetohydrodynamics

1. INTRODUCTION

Many physical and astrophysical phenomena can be modeled by a conservative system of hyperbolic partial differential equations with additional non-hyperbolic source terms. Generally, we write such a system as

* Correspondence to: FOM-Institute for Plasma-Physics Rijnhuizen, PO Box 1207, 3430 BE Nieuwegein, Netherlands. Tel.: +31 30 6096941; fax: +31 30 6031204; e-mail: keppens@rijnh.nl
[1] Current address: Department of Atomic Physics, Eötvös University, Pázmány Péter sétány 1, Budapest H-1117, Hungary. E-mail: gtoth@hermes.elte.hu
[2] E-mail: botchev@cwi.nl
[3] E-mail: A.v.d.Ploeg@marin.nl

CCC 0271–2091/99/110335–18$17.50
Copyright © 1999 John Wiley & Sons, Ltd.
Received January 1998
Revised December 1998

$$\partial_t \vec{U} = \vec{R}(\vec{U}) = -\sum_i \partial_i \vec{F}_i(\vec{U}) + \vec{S}(\vec{U}, \partial_i \vec{U}, \partial_i \partial_j \vec{U}, \mathrm{x}, t), \qquad (1)$$

where t is the time co-ordinate, i and j run over one, two or three components of the spatial co-ordinate $\mathbf{x}$, $\vec{U}$ denotes the vector of conservative variables, and the right-hand-side $\vec{R}$ contains the source terms $\vec{S}$ and the derivatives of the fluxes $\vec{F}_i$. An example of such a system is given by the equations of magnetohydrodynamics (MHD), where the conservative variables are mass, momentum, total energy density and the magnetic field components. Source terms may then represent external gravitational forces, or dissipative effects like viscosity, resistivity and thermal conduction, among others.

In order to solve (1) numerically, one usually combines a sophisticated spatial discretization with explicit time stepping, e.g. a predictor–corrector scheme. However, in such explicit schemes, the numerical stability conditions put an upper limit $\Delta t_{\max}$ on the allowed time step. The explicit approach can be very inefficient when we use time marching towards a steady state solution and only the final solution is of interest. If the convergence towards the steady state stagnates, the physical time corresponding to the final solution is huge relative to $\Delta t_{\max}$, therefore, an excessive number of explicit time steps are needed.

Even in time-accurate calculations, it may occur that the variables $\vec{U}(t)$ evolve on a time-scale much longer than $\Delta t_{\max}$. For example, even if the fastest wave of the hyperbolic system is not induced during the simulation, it still restricts the time step. Another possibility is that a strongly elliptic source term imposes a limiting diffusion time $\Delta t_{\max}$, while in reality it is balanced by another term and $\vec{U}(\mathrm{x}, t)$ evolves slowly.

An implicit treatment of some or all of the variables in system (1) can lift the most severe stability restrictions on the time step, or can even allow arbitrarily large time steps be taken to reach a steady state. The right strategy of temporal discretization for the most efficient numerical solution clearly depends on the problem at hand. To allow for a problem-specific temporal discretization, we implemented general (semi-)implicit time integration schemes in the Versatile Advection Code (VAC, see http://www.phys.uu.nl/ ~ toth/) [1,2], which is a general purpose software package for solving equations of type (1). The newly implemented (semi-)implicit time integration schemes now allow the most optimal combination of the explicit high resolution schemes (see Section 2) with an implicit treatment of part or all of the variables, fluxes and source terms involved. Although we will discuss implementation aspects specifically for VAC, our approach may serve as a general guideline to extend an explicit code with an optional implicit time integration scheme. We differentiate between steady state and time-accurate problems, and show how for time-accurate problems, second-order temporal accuracy is maintained in various semi-implicit discretizations where only some of the variables or only some terms are treated implicitly. This is done in Section 3.

If we denote the implicitly treated variables by $\vec{U}_{\mathrm{impl}}$ and the implicitly treated part of $\vec{R}$ by $\vec{R}_{\mathrm{impl}}$, the essential building block for the implicit and semi-implicit schemes resides in the accurate evaluation of the Jacobian matrix $\partial \vec{R}_{\mathrm{impl}}/\partial \vec{U}_{\mathrm{impl}}$ (Section 4), and the ability to solve the (large) linear systems, in which this matrix appears, efficiently (Section 5). We note that it is clearly advantageous to have the option to treat only part of the variables and equations implicitly. The stability problems encountered by explicit schemes are lifted by choosing $\vec{R}_{\mathrm{impl}}$ to coincide with the part causing them, and the linear system can be solved faster when it is smaller in size and/or the matrix elements are simple to calculate.

In Section 4, we show how we use the different spatial discretizations already available in the explicit code (Section 2) to calculate the Jacobian matrix numerically. The simplest approach is to approximate the action of $\partial \vec{R}_{\mathrm{impl}}/\partial \vec{U}_{\mathrm{impl}}$ on a vector by the directional difference, every time the iterative scheme requires a matrix–vector multiplication. This matrix-free method can be easily implemented, it is independent of the spatial discretization, and requires very little storage. Unfortunately, the matrix–vector multiplication can become computationally expensive, direct linear solvers cannot be used, and possibilities for preconditioning are very restricted. Moreover, the iterative solvers may fail to converge due to the fact that the matrix is effectively perturbed at each iteration by the error in the numerical evaluation of the matrix–vector product. However, the matrix-free approach can be applied successfully for treating implicitly elliptic-type source terms (e.g. resistivity in MHD) or in combination with the minimal residual approximated implicit (MRAI) strategy, recently introduced in [3].

Another option is to calculate and store the matrix elements by perturbing $\vec{U}$ and applying the explicit scheme, so that the matrix can be used by any direct or iterative scheme. This requires a lot of computation and storage for the matrix, thus we restrict ourselves to compact spatial discretizations involving the nearest neighbouring cells only. We describe a general and simple algorithm to obtain the matrix by perturbing the variables in appropriately chosen spatial patterns, and a less general, more complicated, but also more efficient method, where we exploit a specific form of the spatial discretization. The various implementations are discussed in Section 4.

Section 5 describes the linear solvers. In one spatial dimension, we use a direct solver for the block tridiagonal systems. In more than one dimension, we solve the block penta- and heptadiagonal systems with conjugate gradient (CG)-type iterative schemes (a recent overview of these schemes is given in [4]). Conjugate gradient [5] is used for symmetric matrices, while for non-symmetric systems, we can use the restarted generalized minimal residual scheme (GMRES) [6] and the stabilized bi-conjugate gradient method (Bi-CGSTAB) [7], and their generalizations GMRESR [8] and BiCGstab(ℓ) [9]. For advection-dominated problems, the modified block incomplete LU (MBILU) preconditioner [10] is of vital importance to accelerate the convergence of the iterative schemes.

The motivation behind the development of all these algorithms lies in their ultimate applicability to solve, e.g. the equations of hydrodynamics and magnetohydrodynamics. In Section 6, we illustrate their effectiveness when calculating polytropic hydrodynamic outflow from a rotating star and for magnetohydrodynamic bow shock calculations containing multiple shock fronts. These steady state, two-dimensional problems cover smooth and discontinuous flow problems respectively. In particular, we recover the complex topology of a magnetohydrodynamic bow shock flow in the switch-on regime recently presented by De Sterck *et al.* [11]. Other examples, including time-accurate computations, are found in [12–14]. We compliment the suite of astrophysically oriented test problems given in an accompanying paper [14], by providing all algorithmic and implementation details.

We conclude in Section 7 by summarizing the advantages of having optional implicit and semi-implicit time discretization schemes available in a general code for hydro- and magnetohydrodynamic problems. A problem-specific approach to eliminate the most restrictive constraint on the time step greatly accelerates the solution procedure.

2. EXPLICIT SCHEMES

Several equation modules of type (1) are available in VAC, all written in a dimension-independent notation, with $1 \le D \le 3$ spatial dimensions and C vector components with $D \le C \le 3$. The $D = 1$, $C = 2$ or 3, and the $D = 2$, $C = 3$ cases are usually referred to as 1.5D and 2.5D. Presently available equation modules are the equations of adiabatic hydrodynamics, the hydrodynamic equations, the equations of isothermal MHD, and the MHD equations.

The conservative variables $\vec{U}(\mathrm{x}, t)$ are spatially discretized on a structured grid in a finite volume sense: for each cell k of volume V_k, we store the average values $\vec{U}_k(t) \equiv (1/V_k) \int_{V_k} \vec{U}(\mathrm{x}, t)$. Until now, VAC was designed to advance $\vec{U}_k$ in time using one out of four spatially and temporally second-order-accurate explicit schemes: the flux corrected transport scheme (FCT) [15] and three total variation diminishing (TVD) schemes [16], two of them with a Roe-type Riemann solver [17,18] (TVD and TVD-MUSCL), and a Lax–Friedrichs-type scheme (TVDLF) [19,20]. All of these methods calculate smooth flows to second-order accuracy, and are able to capture discontinuities, but they differ in their diffusive and dispersive character near such discontinuities. We briefly summarize the main properties of these explicit schemes, for details see [20].

The FCT scheme is a conservative two-step method, where each step involves a transport and diffusion stage ensuring stability, corrected wherever possible with an antidiffusion stage to recover second-order accuracy. The FCT schemes can be rather dispersive close to discontinuities, introducing spurious oscillations.

It is known from the mathematics of one-dimensional hyperbolic scalar equations that the total variation does not increase with time. The TVD methods are constructed in such a way that this property is carried over to the numerical total variation:

$$\sum_k |\Delta U^{n+1}_{k+1/2}| \leq \sum_k |\Delta U^n_{k+1/2}|, \tag{2}$$

where $\Delta U_{k+1/2} = U_{k+1} - U_k$ with the sum running over all cells, and the superscripts n and $n+1$ denote the discrete time levels. Generalizations of TVD schemes to multidimensional systems of equations yield practically oscillation-free solutions.

The one-step TVD method can sharply resolve discontinuities, and does so by using an approximate Riemann solver to advance the solution. The Riemann solver considers the solution from time t_n as a piecewise constant initial value problem (Riemann problem) at each cell interface to get to time $t_{n+1} = t_n + \Delta t_n$: provided the time step Δt_n is small enough, we can solve for the cell-averaged solution at time t_{n+1} by making a decomposition in the characteristic wave fields at each interface. The approximation results from linearizing the system (1) to obtain the characteristic wave fields. The spatial and temporal second-order accuracy is achieved by second-order correction terms corresponding to a piecewise linear approximation. The TVD property (2) is ensured by applying limiters on the jumps allowed in each of the characteristic wave fields.

Two-step TVD methods implemented are of MUSCL-type (monotonic upwind schemes for conservation laws). A Hancock predictor step ensures the temporal second-order accuracy. Instead of applying limiters on the jumps in characteristic wave fields as in the one-step TVD method, one can now limit the slopes obtained from the conservative variables ΔU themselves, or from the primitive variables, to achieve second-order spatial accuracy where the solution is smooth and the TVD property. TVD-MUSCL also makes use of the Riemann solver-based decomposition in characteristic wave fields.

A computationally less expensive, slightly more diffusive, but very robust variant of this scheme uses the fastest wave speed only so that no Riemann solver is required. This Lax–Friedrich-type TVD scheme is called TVDLF, and is described in detail in [20]. In the implicit schemes we will only need the first-order-accurate version, which updates the $\vec{U}$ variables by

$$\vec{U}_i^{n+1} = \vec{U}_i^n - \frac{\Delta t}{2\Delta x}[\vec{F}_{i+1}^n - \vec{F}_{i-1}^n] + \Delta t\vec{S}_i + \frac{\Delta t}{2\Delta x}[c_{i+1/2}^{\max}(\vec{U}_{i+1}^n - \vec{U}_i^n) - c_{i-1/2}^{\max}(\vec{U}_i^n - \vec{U}_{i-1}^n)], \tag{3}$$

where $c_{i+1/2}^{\max}$ denotes the maximum physical propagation speed corresponding to the interface-averaged state of $\vec{U}_i$ and $\vec{U}_{i+1}$. In the original Lax–Friedrichs scheme, $c^{\max}$ is replaced by $\Delta x/\Delta t$, which makes that scheme more diffusive.

Boundary conditions are implemented by the use of ghost cells surrounding the computational domain. Before each step in the multi-step time integration schemes, the ghost cells are updated based on $\vec{U}$ in the computational domain and on the type of the boundary condition represented by the ghost cell. Typically, the variables in the ghost cell are either independent of $\vec{U}$ (e.g. inflow), or functions of the neighbouring cell inside the computational domain (e.g. solid wall or outflow). A special case arises for periodic boundary conditions that relate the ghost cell to the opposite side of the computational domain. The implicit schemes are designed to be compatible with any of these boundary conditions.

3. IMPLICIT AND SEMI-IMPLICIT SCHEMES

We discretize the system of equations (1) using a one parameter semi-implicit scheme to advance from time level n to $n+1$ by

$$\vec{U}^{n+1} = \vec{U}^n + \Delta t[\vec{R}(\vec{U}^n) + \beta\vec{R}_{\text{impl}}(\vec{U}^{n+1}) - \beta\vec{R}_{\text{impl}}(\vec{U}^n)], \tag{4}$$

where $\vec{R}$ is a conservative high-order discretization (TVD, TVD-MUSCL, or TVDLF), $\vec{R}_{\text{impl}}$ contains all the implicitly treated terms, and the β parameter may vary between 0 and 1. We note that, for $\vec{R}_{\text{impl}} = \vec{R}$, the backward Euler scheme corresponds to $\beta = 1$, and the trapezoidal scheme to $\beta = 1/2$.

When $\vec{R}_{\text{impl}}$ is non-linear, (4) can be linearized by using

$$\vec{R}_{\text{impl}}(\vec{U}^{n+1}) = \vec{R}_{\text{impl}}(\vec{U}_{\text{expl}}^{n+1}, \vec{U}_{\text{impl}}^n) + \frac{\partial\vec{R}_{\text{impl}}}{\partial\vec{U}_{\text{impl}}}(\vec{U}_{\text{impl}}^{n+1} - \vec{U}_{\text{impl}}^n) + \mathcal{O}(\Delta t^2), \tag{5}$$

where $\vec{U}_{\text{impl}}$ contains all implicitly treated variables of $\vec{U}$, hence symbolically, we write $\vec{U} = (\vec{U}_{\text{expl}}, \vec{U}_{\text{impl}})$. We always evaluate the Jacobian $\partial\vec{R}_{\text{impl}}/\partial\vec{U}_{\text{impl}}$ at $(\vec{U}_{\text{expl}}^{n+1}, \vec{U}_{\text{impl}}^n)$, and in the case of an explicit time dependence in the source term $\vec{S}$, the evaluation is done at t_{n+1}.

The linearization (5) makes an error $\mathcal{O}(\Delta t^3)$ in (4), so it is always sufficient to prove second-order accuracy for the non-linear scheme (4), which we do in the following subsections for steady state and time-accurate problems.

3.1. Steady state problems

For steady state calculations, we are only interested in the solution of the non-linear equation $\vec{R}(\vec{U}) = 0$ and the obtained spatial accuracy. To allow for arbitrarily large time steps when time marching towards the steady state, one can use a fully implicit backward Euler scheme, where $\vec{U}_{\text{impl}} = \vec{U}$, $\vec{R}_{\text{impl}} = \vec{R}$ and $\beta = 1$. However, since only the final solution, where $\vec{U}^{n+1}$ equals $\vec{U}^n$, is physically meaningful, we can use a spatially low-order discretization $\partial\vec{R}_{\text{low}}/\partial\vec{U}$ for the Jacobian matrix $\partial\vec{R}_{\text{impl}}/\partial\vec{U}_{\text{impl}} = \partial\vec{R}/\partial\vec{U}$ in (5). Hence, we set $\beta = 1$ in (4), linearize according to (5), replace the Jacobian by its low-order counterpart, and obtain the steady state solution by performing a 'pseudo-time stepping'. This amounts to a sequence of solutions to linear systems

$$\left[\frac{\hat{I}}{\Delta t}-\frac{\partial \vec{R}_{\text{low}}}{\partial \vec{U}}\right](\vec{U}^{n+1}-\vec{U})=\vec{R}(\vec{U}^{n}), \tag{6}$$

where Δt can, in principle, be arbitrarily large. The use of a high-order spatial discretization for $\vec{R}$ ensures that the converged steady state solution is accurate.

We also implemented an alternative approach, where the non-linear system (4) is solved by Newton–Raphson iterations. Then, large linear systems similar to (6) have to be solved per Newton–Raphson iterate. As could be expected, the number of Newton–Raphson iterations turns out to be of the same order as the number of pseudo-time steps needed to reach steady state according to (6).

3.2. Time accurate problems

For time-accurate calculations, we can use scheme (4) with $\beta=1/2$ to obtain second-order temporal accuracy, since

$$\vec{U}^{n+1}=\vec{U}^{n}+\Delta t\left[\vec{R}(\vec{U}^{n})+\frac{1}{2}\vec{R}(\vec{U}^{n+1})-\frac{1}{2}\vec{R}(\vec{U}^{n})\right]=\vec{U}^{n}+\Delta t\vec{R}(\vec{U}^{n})+\frac{1}{2}\Delta t^{2}\frac{\mathrm{d}\vec{R}}{\mathrm{d}t}+\mathcal{O}(\Delta t^{3}), \tag{7}$$

where we assumed that $R_{\text{impl}}=R$ for all the implicitly treated variables. Unfortunately, this trapezoidal scheme is only marginally stable for linear advection problems, and in general it is hardly applicable for stiff problems (see e.g. [21,22]). A common practice is to use β slightly above 1/2; however, such a scheme is neither second-order-accurate nor stable enough in strongly non-linear cases.

Another approach to achieve second-order temporal accuracy is possible by using information from time levels t_{n-1} and t_n to arrive at t_{n+1}. Therefore, we implemented a two-parameter three-level time integration scheme,

$$\vec{U}^{n+1}=\vec{U}^{n}+\Delta t_n\vec{R}(\vec{U}^{n})+\Delta t_n\left[\alpha\frac{\vec{U}^{n}-\vec{U}^{n-1}}{\Delta t_{n-1}}-\alpha\vec{R}(\vec{U}^{n})+\beta\vec{R}_{\text{impl}}(\vec{U}^{n+1})-\beta\vec{R}_{\text{impl}}(\vec{U}^{n})\right], \tag{8}$$

where $\vec{R}_{\text{impl}}(\vec{U}^{n+1})$ can again be linearized by (5). The scheme is three-level whenever the parameter $\alpha\neq 0$, while for $\alpha=0$, it is identical to (4). Note that when $\vec{R}_{\text{impl}}=\vec{R}$, $\alpha=1/3$ and $\beta=2/3$, we get the second-order backward differentiation formula (BDF2) [23] for constant step size Δt.

We show now how second-order temporal accuracy can be obtained. Let us assume that U^n was calculated from U^{n-1} with second-order temporal accuracy, thus $(U^n-U^{n-1})/\Delta t_{n-1}=R(U^n)-(\Delta t_{n-1}/2)\,\mathrm{d}R/\mathrm{d}t+\mathcal{O}(\Delta t^2)$. Substituting this into (8) we obtain

$$\begin{aligned}\vec{U}^{n+1}&=\vec{U}^{n}+\Delta t_n\left[\vec{R}(\vec{U}^{n})-\alpha\frac{\Delta t_{n-1}}{2}\frac{\mathrm{d}\vec{R}}{\mathrm{d}t}+\beta\Delta t_n\frac{\mathrm{d}\vec{R}}{\mathrm{d}t}+\mathcal{O}(\Delta t^{2})\right]\\&=\vec{U}^{n}+\Delta t_n\vec{R}(\vec{U}^{n})+\Delta t_n\left(\beta\Delta t_n-\frac{\alpha}{2}\Delta t_{n-1}\right)\frac{\mathrm{d}\vec{R}}{\mathrm{d}t}+\mathcal{O}(\Delta t^{3}).\end{aligned} \tag{9}$$

The coefficient of $\mathrm{d}\vec{R}/\mathrm{d}t$ simplifies to $(\Delta t_n)^2/2$, as required by the second-order accuracy, if the condition $2\beta-\alpha\Delta t_{n-1}/\Delta t_n=1$ is fulfilled. Again, we assumed that $R_{\text{impl}}=R$ for all the implicitly treated variables. The BDF2 scheme with an adjustable step size corresponds to the additional restriction $\alpha+\beta=1$, which determines both parameters uniquely. In the first time step, when no $\vec{U}^{n-1}$ is available, one could use the trapezoidal scheme for second-order accuracy, or the more stable backward Euler scheme, which introduces a first-order error in the first time step.

In the following subsections we address the issue of second-order accuracy (i) for a spatially first-order discretization for the Jacobian in the linearized schemes, (ii) for the case when only some of the variables are treated implicitly, and (iii) when only some terms are treated implicitly. The latter two cases are commonly referred to as semi-implicit discretizations. In the semi-implicit approach, the time step must be chosen in accordance with the limitations imposed by the explicitly treated terms. By fully implicit method, on the other hand, we mean $\vec{U}_{\text{impl}} = \vec{U}$ and $\vec{R}_{\text{impl}} = \vec{R}$, disregarding possible differences in the numerical discretization.

3.2.1. First-order Jacobian. Like in Section 3.1, it is possible to use a lower-order discretization for $\vec{R}_{\text{impl}}$, which substantially reduces the computational and storage costs for calculating the Jacobian.

Since $\vec{R}$ is always evaluated to second-order spatial accuracy, we can show that (4) is second-order-accurate in space and time for $\beta = 1/2$ from

$$\begin{aligned}\vec{U}^{n+1} &= \vec{U}^n + \Delta t\vec{R}_{\text{high}}(\vec{U}^n) + \frac{\Delta t}{2}[\vec{R}_{\text{low}}(\vec{U}^{n+1}) - \vec{R}_{\text{low}}(\vec{U}^n)] \\ &= \vec{U}^n + \Delta t\vec{R}_{\text{high}}(\vec{U}^n) + \frac{\Delta t}{2}\left[\frac{\mathrm{d}\vec{R}_{\text{low}}}{\mathrm{d}t}\Delta t + \mathcal{O}(\Delta t^2)\right] \\ &= \vec{U}^n + \Delta t\vec{R}_{\text{high}}(\vec{U}^n) + \frac{\Delta t^2}{2}\frac{\mathrm{d}\vec{R}_{\text{high}}}{\mathrm{d}t} + \mathcal{O}(\Delta x\Delta t^2) + \mathcal{O}(\Delta t^3),\end{aligned} \tag{10}$$

where we used $\vec{R}_{\text{high}} \equiv \vec{R}$ and $\vec{R}_{\text{low}} = \vec{R}_{\text{high}} + \mathcal{O}(\Delta x)$. Rigorous proof uses the mean value theorem to show that the local error between the computed U^{n+1} and the exact solution is then also of order $\mathcal{O}(\Delta x\Delta t^2) + \mathcal{O}(\Delta t^3)$. The same argument holds for the three-level scheme (8) as well.

3.2.2. Semi-implicit: some variables. When only some of the variables $\vec{U}$ are treated implicitly, we first advance the explicitly treated $\vec{U}_{\text{expl}}$ using a second-order explicit time integration scheme, e.g. a two-stage Runge–Kutta scheme

$$\vec{U}^{n+1}_{\text{expl}} = \vec{U}^n_{\text{expl}} + \Delta t\vec{R}_{\text{expl}}\left(\vec{U}^n + \frac{\Delta t}{2}\vec{R}(\vec{U}^n)\right) = \vec{U}^n_{\text{expl}} + \Delta t\vec{R}_{\text{expl}}(\vec{U}^n) + \Delta t\frac{\partial\vec{R}_{\text{expl}}}{\partial\vec{U}}\frac{\Delta t}{2}\frac{\partial\vec{U}}{\partial t} + \mathcal{O}(\Delta t^3). \tag{11}$$

We subsequently advance the variables $\vec{U}_{\text{impl}}$ by (4) using $\beta = 1/2$ as follows:

$$\begin{aligned}\vec{U}^{n+1}_{\text{impl}} &= \vec{U}^n_{\text{impl}} + \frac{\Delta t}{2}[\vec{R}_{\text{impl}}(\vec{U}^n_{\text{expl}}, \vec{U}^n_{\text{impl}}) + \vec{R}_{\text{impl}}(\vec{U}^{n+1}_{\text{expl}}, \vec{U}^{n+1}_{\text{impl}})] \\ &= \vec{U}^n_{\text{impl}} + \Delta t\vec{R}_{\text{impl}}(\vec{U}^n) + \frac{\Delta t^2}{2}\frac{\mathrm{d}\vec{R}_{\text{impl}}}{\mathrm{d}t} + \mathcal{O}(\Delta t^3).\end{aligned} \tag{12}$$

A similar proof can be carried out for the three-level scheme.

3.2.3. Semi-implicit: some terms. Up to this point we always assumed that for the implicitly treated variables $\vec{U}_{\text{impl}}$ all the fluxes and sources are treated implicitly, thus, for these variables $R_{\text{impl}} = R$ disregarding possible differences in the discretization. We face a new problem when $R_{\text{impl}} \neq R$, for example, when only the source terms S are treated implicitly. This may be useful for solving resistive MHD problems, where the stability constraints on the time step arise from both the fast magnetosonic waves and the resistive diffusion time scale. As the grid resolution increases and Δx decreases, the diffusive time scale becomes more restrictive than the CFL condition, because it scales with Δx^2 rather than Δx. In such situations, a semi-implicit approach that treats the diffusive source terms (and only those) implicitly, can be highly effective.

We can combine an explicit two-step method for the explicitly treated terms $\vec{R}_{\text{expl}}$, where the predictor step involves the total residual $\vec{R}$, and an implicit treatment for $\vec{R}_{\text{impl}}$, as

$$\begin{aligned}\vec{U}^{n+1} &= \vec{U}^n + \Delta t\vec{R}_{\text{expl}}\left(\vec{U}^n + \frac{\Delta t}{2}\vec{R}(\vec{U}^n)\right) + \frac{\Delta t}{2}[\vec{R}_{\text{impl}}(\vec{U}^n) + \vec{R}_{\text{impl}}(\vec{U}^{n+1})]\\ &= \vec{U}^n + \Delta t\vec{R}(\vec{U}^n) + \frac{\Delta t^2}{2}\frac{\mathrm{d}\vec{R}_{\text{expl}} + \mathrm{d}\vec{R}_{\text{impl}}}{\mathrm{d}t} + \mathcal{O}(\Delta t^3),\end{aligned} \tag{13}$$

where $\vec{R} = \vec{R}_{\text{expl}} + \vec{R}_{\text{impl}}$. This scheme is analogous to the trapezoidal scheme. The generalization of the three-level scheme (8) requires a predictor step with a time step $\beta\Delta t$ since the term $\vec{U}^n - \vec{U}^{n-1}$ already contains contributions to $\mathrm{d}\vec{R}_{\text{expl}}/\mathrm{d}t$, thus

$$\begin{aligned}\vec{U}^{n+1} &= \vec{U}^n + \Delta t_n\vec{R}_{\text{expl}}(\vec{U}^n + \beta\Delta t_n\vec{R}(\vec{U}^n)) + \Delta t_n\vec{R}_{\text{impl}}(\vec{U}^n)\\ &\quad + \Delta t_n\left[\alpha\frac{\vec{U}_n - \vec{U}^{n-1}}{\Delta t_{n-1}} - \alpha\,\vec{R}(\vec{U}^n) + \beta\vec{R}_{\text{impl}}(\vec{U}^{n+1}) - \beta\vec{R}_{\text{impl}}(\vec{U}^n)\right]\\ &= \vec{U}^n + \Delta t_n\vec{R}_{\text{expl}}(\vec{U}^n) + \Delta t_n\vec{R}_{\text{impl}}(\vec{U}^n) + \Delta t_n\left(\beta\Delta t_n - \frac{\alpha}{2}\Delta t_{n-1}\right)\frac{\mathrm{d}\vec{R}_{\text{expl}} + \mathrm{d}\vec{R}_{\text{impl}}}{\mathrm{d}t}\ \mathcal{O}(\Delta t^3).\end{aligned} \tag{14}$$

4. JACOBIAN EVALUATION

The linearization (5) leads to large linear systems like (6) containing the Jacobian matrix $\partial\vec{R}_{\text{impl}}/\partial\vec{U}_{\text{impl}}$. The same applies when an extra Newton–Raphson iteration is performed to get a solution of the non-linear system (4). Since we have several spatial discretizations for fluxes, source terms and boundary conditions already available in the explicit schemes (Section 2), we opt for a general numerical evaluation of the Jacobian, where all or most of these discretizations can be used.

The easiest way to extend an explicit code to an implicit one is by using an iterative method to solve the large linear systems containing the Jacobian matrix, and approximate each needed evaluation of a matrix–vector product by a direct calculation of the action of the Jacobian matrix on the vector. This results in a matrix-free method that is independent of the actual discretizations used.

To allow for preconditioning that can significantly accelerate the convergence of the iterative solvers, we also implemented two ways of calculating the Jacobian matrix itself numerically. We describe a general method to calculate the Jacobian matrix using grid masks. Its implementation in VAC is restricted to any spatial discretization involving nearest neighbouring cells. A more efficient way to calculate the Jacobian matrix for a specific spatially first-order discretization, namely the first-order variant of the TVDLF scheme (3), is also provided. By allowing compact spatial discretizations only, storage requirements for the Jacobian matrix are reduced.

We discuss the three implementations for evaluating the Jacobian in the following. To simplify notation, we write from here on $\partial\vec{R}_{\text{impl}}/\partial\vec{U}_{\text{impl}}$ as $\partial\vec{R}/\partial\vec{U}$.

4.1. Matrix-free Jacobian evaluation

In cases where, for example, source terms cannot be formulated compactly, or when a high-order Jacobian evaluation is desirable, we refrain from calculating the Jacobian matrix, since it would require significant storage. We can avoid the actual calculation of the many non-zero matrix elements by using an iterative solver for the linear systems, and when the Jacobian matrix needs to be evaluated in direction $\Delta\vec{U}$, we use

$$\frac{\partial\vec{R}}{\partial\vec{U}}\Delta\vec{U}=\frac{\vec{R}(\vec{U}^n+\epsilon\Delta\vec{U})-\vec{R}(\vec{U}^n)}{\epsilon}. \tag{15}$$

In this expression, ϵ must be small for physical accuracy, but larger than machine precision to bound the effect of rounding errors; thus, we use $\epsilon=10^{-6}\|\Delta\vec{U}\|$, where $\|\cdot\|$ denotes the second norm. The boundary conditions should be applied on $\vec{U}^n+\epsilon\Delta\vec{U}$ before evaluating $\vec{R}$. As the other term $\vec{R}(\vec{U}^n)$ does not depend on $\Delta\vec{U}$, one evaluation of $\vec{R}$ is needed for each matrix–vector multiplication.

Since the Jacobian is obtained by explicitly evaluating the appropriate part of the residual, this method is independent of the discretization used, but may be very expensive computationally. This is not so critical when $\vec{R}$ contains only an elliptic-type source term, where a larger than nearest neighbour stencil is used. For example, when merely resistive source terms for the MHD equations are treated in this way, the linear systems that need to be solved are diagonally dominated and a CG-type method converges fast.

We note that combining a CG-type iterative method with a matrix-free evaluation of the Jacobian is not always foolproof, as the evaluation (15) is prone to numerical error and effectively perturbs the matrix at each matrix–vector product. This may, in turn, destroy the orthogonality properties of the Krylov subspace basis, which is explicitly or implicitly generated in CG-type iterative methods (more on this can be found in [24]).

However, the matrix-free approach can be used successfully when the Krylov dimension is kept moderate or small, like in the MRAI schemes discussed in [3]. In MRAI schemes, the linearized equation of an implicit scheme in which one is interested is solved approximately with a few minimal residual (GMRES) iterations. The convergence of GMRES is not checked, the number of iterations is kept simply fixed. To assure stability, the step size is then adjusted in a special way. The MRAI time stepping is not unconditionally stable, but its stability region is much wider than those of ordinary explicit schemes. The scheme is adaptive in the sense that the solution components corresponding to the stiff part of the spectrum are treated as in an implicit scheme [3]. The attractive features of MRAI are: (i) the possibility to use the matrix-free evaluation (15), which makes the storage requirements of the scheme very low, and (ii) the fact that they are easily parallelized. Ideal, linear scaling on distributed memory architectures for MRAI schemes is demonstrated in [25].

4.2. General Jacobian evaluation using grid masks

It is also possible to calculate the individual matrix elements numerically in a fairly general way. Since storage and computational demands for the Jacobian matrix become excessive when the spatial discretization uses a larger than nearest neighbour stencil, we limit the discussion of this method to compact stencils. However, the method is more generally applicable whenever a structured grid is used.

Again, we want to devise a method where the calculation of the Jacobian matrix is independent of the way in which $\vec{R}$ is evaluated. This then offers the possibility to, for example, use a first-order upwind evaluation for the fluxes, without the effort to linearize the (approximate) Riemann solver in $\vec{U}$ analytically.

When the grid is structured, and when only neighbouring cells influence the evaluation of the fluxes and source terms at a certain grid cell, we can easily label the grid cells such that the stencil patterns corresponding to the same label do not overlap. In D dimensions, we subdivide the grid into blocks of size $(2 \times D + 1)^D$ within which a specific labelling of the grid points is repeated. The obtained *grid masks* providing the labels are shown in Figure 1. In 1D all grid points are grouped in consecutive blocks of three points, in 2D we split the domain up in 5×5 squares, and in 3D the domain is split in $7 \times 7 \times 7$ cubes.

We perturb a certain component w of the implicitly treated $\vec{U}$ by ϵ in all grid cells j that carry the same label in the masked grid, apply the boundary conditions, and subsequently evaluate $\vec{R}$ for the perturbed state $\vec{U} + \epsilon\delta_j^w$ over the whole grid, with $\epsilon = 10^{-6}\|U^w\|$. The matrix elements are obtained from

$$\frac{\partial R_i^u}{\partial U_j^w} = \frac{R_i^u(\vec{U} + \epsilon\delta_j^w) - R_i^u(\vec{U})}{\epsilon}. \tag{16}$$

In this expression, the superscript u denotes the component of $\vec{R}$ considered, while the grid cell i must be inside the stencil corresponding to the perturbed cell j. In 1D, for instance, when all cells labelled with 1 are perturbed, we can immediately read off the main diagonal contributions to the Jacobian matrix for all cells labelled with 1, simultaneously with lower and upper diagonal contributions for cells labelled with 2 and 3 respectively. As we loop over the variable index w and the grid labels from 1 to $N_{\text{stencil}} = 2 \times D + 1$, we can build up the whole matrix. Overall, when we have N_{impl} implicitly treated variables, we can build up the full Jacobian matrix in this way at the expense of $1 = N_{\text{stencil}} \times N_{\text{impl}}$ explicit evaluations of $\vec{R}$ on the full grid. Note also that this method builds up the Jacobian matrix in a data parallel fashion.

Since we use numerical differentiation to calculate the matrix elements, the algorithm (16) is independent of the actual discretization. Therefore, we can easily exploit the first-order variant of the TVDLF scheme given in (3), but also the first-order upwind method, i.e. the unlimited and one-step variant of the TVD-MUSCL scheme that uses the Roe-type approximate Riemann solver. Moreover, whenever source terms can be calculated using a compact (nearest neighbour) stencil, they can be included directly in the Jacobian calculation. This is useful for compactly formulated elliptic-type source terms that impose a too restrictive time step limitation. To demonstrate this, we implemented compactly formulated resistive source terms for the MHD equations in the Jacobian calculation, and use them in the accompanying paper [14] to solve a time-dependent resistive MHD test problem.

We note that other grid masks could be defined to deal with larger stencils, but the perfect tiling of the grid by the stencil pattern may not always be possible. Our algorithm may serve as a general guideline to build up the Jacobian for implicit time integration schemes using an existing explicit code and a moderate amount of extra coding. Finally, we note that special

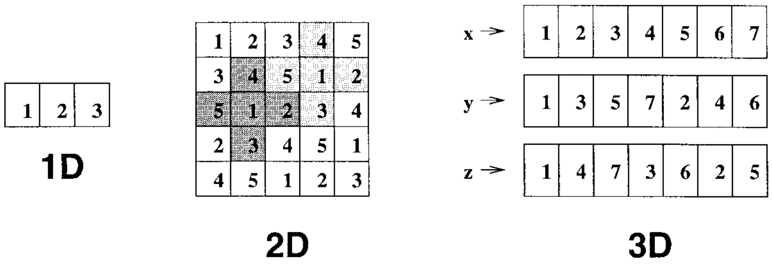

Figure 1. Grid masks used for the calculation of the Jacobian matrix.

care is taken to be able to deal with periodic boundaries, which introduce extra blocks in the Jacobian matrix. In our implementation, the number of grid points in the direction of the periodicity must be a multiple of N_{stencil}.

4.3. *Efficient Jacobian evaluation for TVDLF*

A third option available to calculate the Jacobian matrix is specifically designed for obtaining the Jacobian for the first-order TVDLF-variant given by (3) more efficiently. Indeed, when we write

$$\frac{\partial \vec{R}}{\partial \vec{U}} = -\sum_i \partial_i \frac{\partial \vec{F}}{\partial \vec{U}} + \frac{\partial \vec{S}}{\partial \vec{U}}, \tag{17}$$

and subsequently use the spatial discretization (3) to evaluate the residual, we obtain the matrix elements (in 1D, and similarly in more than one spatial dimension) from

$$\begin{aligned}
\frac{\partial R_i^u}{\partial U_i^w} &= \frac{\partial S_i^u}{\partial U_i^w} - \frac{1}{2\Delta x}\left(c_{i+1/2}^{\max} + c_{i-1/2}^{\max}\right)\frac{\partial U_i^u}{\partial U_i^w},\\
\frac{\partial R_i^u}{\partial U_{i+1}^w} &= -\frac{1}{2\Delta x}\frac{\partial F_{i+1}^u}{\partial U_{i+1}^w} + \frac{1}{2\Delta x}c_{i+1/2}^{\max}\frac{\partial U_{i+1}^u}{\partial U_{i+1}^w},\\
\frac{\partial R_i^u}{\partial U_{i-1}^w} &= +\frac{1}{2\Delta x}\frac{\partial F_{i-1}^u}{\partial U_{i-1}^w} + \frac{1}{2\Delta x}c_{i-1/2}^{\max}\frac{\partial U_{i-1}^u}{\partial U_{i-1}^w},
\end{aligned} \tag{18}$$

where u and w refer to specific components of $\vec{R}$ and $\vec{U}$, and i is a grid cell index. Numerical differentiation is used to take the partial derivatives, hence, for any function f we have:

$$\frac{\partial f}{\partial U^w} = \frac{f(\vec{U} + \epsilon\delta^w) - f(\vec{U})}{\epsilon}. \tag{19}$$

Since $\vec{F}$ and $\vec{S}$ need to be evaluated for each $w = 1, \ldots, N_{\text{impl}}$ only once over the full grid, the total computational expense is approximately N_{stencil} times less than for the general algorithm described in the previous section. On the other hand, the evaluation of (18) itself involves all the matrix elements of the Jacobian, thus it has to be carefully optimized to achieve the expected performance.

The terms $\partial U^u/\partial U^w$ are Kronecker deltas δ_u^w except in the ghost cells, where the boundary conditions may couple the different components of $\vec{U}$ together. Therefore, the boundary conditions are applied both on $\vec{U}$ and $\vec{U} + \epsilon\delta^w$, and a numerical derivative is taken for the resulting $\partial U^u/\partial U^w$ in the ghost cells. The partial derivatives of the fluxes $\partial F^u/\partial U^w$ are also calculated in the ghost cells, and eventually the matrix elements $\partial R_i^u/\partial U_{\text{ghost}}^w$ are added to the matrix element $\partial R_i^w/\partial U_j^w$, where j is the index of the physical cell on which the ghost cell depends.

Note that in (18), we assume that the source terms $\vec{S}$ are local. An example for such a local source term is an external gravitational field. Also, we do not differentiate the maximal physical propagation speed $c^{\max}$, which causes a subtle difference between the matrix obtained from (18) and the Jacobian matrix calculated by the general method (16), even when the same spatial discretization (first-order TVDLF) is used. Neglecting the dependence of $c^{\max}$ on U^w is identical to the 'dropping the time level' approach suggested by Yee [19]. We note that our implementation of this Jacobian calculating method works for 1D, 2D and 3D Cartesian, polar and general structured grids, for slab and axial symmetry, and for normal and periodic boundaries.

5. LINEAR SYSTEM SOLVERS

The solution strategy that we adopt to solve the large linear systems containing the Jacobian matrix efficiently, depends on the dimensionality and the type of problem at hand. If the Jacobian matrix is available, iterative solvers can exploit suitable preconditioners, or the linear systems can even be solved by a direct method.

In the linear systems like (6), we always work with normalized variables independent of the solution strategy, so that we solve for unknowns x related to the updates $\Delta\vec{U} = \vec{U}^{n+1}_{\text{impl}} - \vec{U}^{n}_{\text{impl}}$ according to

$$x^u \equiv \frac{\Delta U^u_{\text{impl}}}{\sqrt{N^{-1}_{\text{grid}} \sum_k (U^{u,n}_k)^2}}, \tag{20}$$

where the total number of grid cells k is denoted by N_{grid}, and u selects a component of the vector of conserved quantities $\vec{U}$. We normalize each component of $\vec{U}$ separately, since the physical units may be very different, which can be a problem for the iterative schemes that handle all unknowns the same way. Similarly, we use normalized right-hand-side vectors b in the linear systems. In this way, we end up solving linear systems $\hat{A}x = b$. Therefore, when we use iterative solvers for the linear systems, we can use an absolute stopping criterion both in steady state and time-accurate calculations. For steady state calculations, we account for the possibility of a very large (pseudo) time step Δt. In cases where a preconditioner is exploited, we use a relative stopping criterion instead.

5.1. Direct linear system solver for 1D problems

A spatially first-order evaluation of the Jacobian matrix on a 1D grid leads to a block tridiagonal matrix structure. For these banded matrices, storage demands are low and a direct solver can be used. We implemented a direct block tridiagonal solver, together with a cyclic version based on the Woodbury formula [26] to deal with periodic boundary conditions. The direct solver for the block tridiagonal systems can be combined with a reordering based on a combination of domain decomposition and cyclic reduction. The resulting parallel version of the direct solver performs well on distributed memory machines [27].

5.2. Iterative solvers for multi-dimensional problems

For multi-dimensional problems, the linear systems can only be solved by iterative means, since direct methods would require too much CPU time and memory. The iterative method used must be chosen in accordance with the algebraic properties of the occurring matrix (symmetric or not, diagonally dominant or other, . . .). The CG method is used for symmetric positive definite problems, while GMRES, GMRESR, Bi-CGSTAB and BiCGstab(ℓ) are used for solving unsymmetric systems. For advection-dominated problems, however, the matrices appearing in the linear systems are strongly non-symmetric and non-diagonally dominant. For such problems, suitable preconditioners are vital to accelerate the convergence behaviour of the iterative methods.

A nearest neighbour discretization of the Jacobian matrix on a structured grid leads to block pentadiagonal systems in 2D and block heptadiagonal systems in 3D. In the following, we briefly describe the MBILU preconditioning technique that we use for these systems. Let us denote the block penta- or heptadiagonal matrix by $\hat{A}$. We first construct a lower triangular matrix $\hat{L}$ and an upper triangular matrix $\hat{U}$ such that $\hat{A} \approx \hat{L}\hat{U}$, and the action of $\hat{L}^{-1}$ and $\hat{U}^{-1}$

can be computed without too much computational effort. In order to solve a linear system $\hat{A}x = b$, the iterative method is applied to the preconditioned system

$$\hat{L}^{-1}\hat{A}\hat{U}^{-1}\tilde{x} = \hat{L}^{-1}b, \qquad x = \hat{U}^{-1}\tilde{x}. \tag{21}$$

The factors $\hat{L}$ and $\hat{U}$ come from a block incomplete LU (BILU-) decomposition, in which the block sparsity pattern of $\hat{L} + \hat{U}$ is taken the same as that of $\hat{A}$. For our applications, both in two and in three dimensions, this implies that during the construction of $\hat{L}$ and $\hat{U}$ only corrections on the main diagonal blocks are performed. For this kind of BILU-decompositions, computer storage demands are very low, because the preconditioner only requires one extra block diagonal.

In [10] it is shown that the matrix–vector multiplication with the preconditioned matrix can be implemented efficiently by using a block form of the Eisenstat implementation [28]. The multiplication with the preconditioned matrix $\hat{L}^{-1}\hat{A}\hat{U}^{-1}$ costs about the same number of floating point operations as the multiplication with $\hat{A}$. We use a block form of the relaxed modified incomplete decomposition [29], hence MBILU. The fill-in blocks that are not allowed in the factors $\hat{L}$ and $\hat{U}$ are first multiplied by a constant factor α_{relax} in the interval [0, 1], before they are added to the main diagonal blocks. In [10] it was pointed out that good convergence is obtained for α_{relax} in the interval [0.3, 0.7].

Again, extra blocks may arise due to periodicity in one or more directions. In the current implementation, this is accounted for in the first spatial direction only. Periodicity in other directions could be taken into account by allowing extra fill-in blocks in the incomplete LU-decomposition. This could also be useful for lowering the number of (GMRES or other) iteration steps. Experiments showed that when preconditioning is needed (unsymmetric systems, as in advection-dominated problems), the actual choice of the iterative scheme is of lesser importance: it is the preconditioner that determines the rate of convergence. Therefore, a deciding factor in the choice of iterative scheme to solve the preconditioned linear systems is the amount of storage they require.

The storage requirement is best measured by the number of temporary vectors required, where each vector contains $N_{\text{impl}} \times N_{\text{grid}}$ double precision numbers. The CG method can only be used for symmetric matrices, and the preconditioned matrix is not symmetric in general. Without preconditioning, however, it can be a very efficient iterative scheme for symmetric matrices obtained from elliptic-type terms. In our implementation, CG requires only two temporary vectors. Bi-CGSTAB works for symmetric and non-symmetric matrices as well, and it requires storage for seven vectors. Its more stable variant BiCGstab(ℓ) needs $5 + 2\ell$ vectors. The GMRES(N_{restart}) scheme requires $N_{\text{restart}} + 1$ temporary vectors. Reducing N_{restart}, i.e. the dimension of the Krylov subspace, will typically decrease the convergence rate, so values between 5 and 20 are usually used. Another possibility to ease storage requirements, without sacrificing the convergence properties, is to use the nested GMRESR(N_{inner}, N_{truncate}) algorithm, where N_{inner} is the Krylov dimension for the inner GMRES iteration, while N_{truncate} denotes the number of Krylov subspace basis vectors that are kept. This method needs $2 + 2N_{\text{truncate}} + N_{\text{inner}}$ vectors. Typical values are $N_{\text{inner}} = 2$, 3 and $N_{\text{truncate}} = 5$. In our experience Bi-CGSTAB is often the optimal choice.

6. EXAMPLES

We illustrate the effectiveness of a fully implicit approach on two steady state problems. For examples of implicit, time-accurate calculations based on the algorithms presented in Section

3.2, we refer the reader to the accompanying paper [14]. We exploit pseudo-time stepping in both problems, and use MBILU preconditioning when iteratively solving the linear systems (5) with Bi-CGSTAB. Both approaches for calculating the low-order Jacobian matrices are compared for a 2.5D hydrodynamic calculation modelling smooth transonic polytropic stellar outflow. A 2D MHD problem uses the efficient TVDLF Jacobian evaluation. Examples of the MRAI strategy in combination with a matrix-free Jacobian evaluation are found in [14].

6.1. Transonic stellar wind from a rotating star

We set forth to calculate an axisymmetric stellar outflow from a star of mass M_* and radius r_*, rotating with constant angular velocity Ω_*. The star is surrounded by a hot corona. As a result, the hot coronal plasma can overcome the gravitational pull of the star by the outward acceleration resulting from thermal and centrifugal forces. A stellar wind ensues, with a continuous acceleration from low, subsonic speeds close to the stellar surface to supersonic speeds at large radial distances. We can model this flow using the Euler equations, with the energy equation replaced by a polytropic constraint where the thermal pressure $p \sim \rho^{\gamma}$, with density ρ and polytropic index γ. The axisymmetry assumes $\partial/\partial\varphi = 0$, in a cylindrical (R, φ, z) co-ordinate system centred on the star with its polar and rotation axis as the z-axis. Hence, we solve for the conserved variables density $\rho(R, z)$ and the three components of the momentum vector $\rho\mathbf{v}(R, z)$ on a radially stretched spherical (r, θ) grid in a poloidal cross-section. We restrict the calculation to a 300×20 grid in (r, θ) modelling a quarter of the full poloidal cross-section. Boundary conditions at the pole and the equator are then simply found from symmetry arguments. The grid ranges in radius $r \in [1, 50]r_*$. At the outer boundary, where the outflow is supersonic, all quantities are extrapolated linearly into the ghost cells. For the boundary conditions at the stellar surface and for the initial conditions, we make use of a partially analytic 1D solution for the equatorial regions only. This approach is explained fully in [12]. In essence, the equatorial solution provides $\rho^{\mathrm{E}}(r)$, $v_r^{\mathrm{E}}(r)$, $v_\varphi^{\mathrm{E}}(r)$, and the 2D calculation starts by setting $\rho(r, \theta; t = 0) = \rho^{\mathrm{E}}(r)$, $v_r(r, \theta; t = 0) = v_r^{\mathrm{E}}(r)$ and $v_\varphi(r, \theta; t = 0) = v_\varphi^{\mathrm{E}}(r) \sin\theta$ so that it vanishes at the pole $\theta = 0$, while $v_\theta(t = 0) = 0$ everywhere. The mass flux $f_{\mathrm{mass}} = \rho^{\mathrm{E}} r^2 v_r^{\mathrm{E}}$ from the equatorial solution is used to impose the mass input at the stellar boundary through $\rho v_R = f_{\mathrm{mass}} R/r^3$ and $\rho v_z = f_{\mathrm{mass}} z/r^3$. The rotation rate prescribes $v_\varphi = \Omega_* R_*$, and the density is linearly extrapolated in the ghost cells at the stellar base.

For parameter values such that $\sqrt{2GM_*/r_*} = 3.3015c_*$ (c_* denotes the base sound speed and G is the gravitational constant), $\gamma = 1.13$, $\Omega_* r_* = 0.3c_*$, the appropriate mass flux turns out to be $f_{\mathrm{mass}} = 0.01553$ when a scaling is used that sets $r_* = 1$, $\rho_* = 1$ and $c_* = 1$. We solve the equations both explicitly and implicitly using TVDLF with *minmod* limiting applied on the primitive variables. Figure 2 displays the final transonic steady state wind solution obtained: both streamlines and poloidal Mach contours are shown. The transonic transition occurs at the labelled non-circular critical curve. Equatorward centrifugal deflection is seen at the base. Physical discussions of these and other stellar wind solutions can be found in [12].

When we calculate this wind solution implicitly, we can use a Courant number of 1000 and the efficient TVDLF Jacobian evaluation. The MBILU preconditioner is used in the Bi-CGSTAB iterative linear solves. The solution is converged when the difference between subsequent time steps as measured by the relative change from one time level to the next

$$\Delta_2 U \equiv \sqrt{\frac{1}{N_{\mathrm{var}}} \sum_{u=1}^{N_{\mathrm{var}}} \frac{\sum_{\mathrm{grid}} (U_u^{n+1} - U_u^n)^2}{\sum_{\mathrm{grid}} (U_u^n)^2}} \tag{22}$$

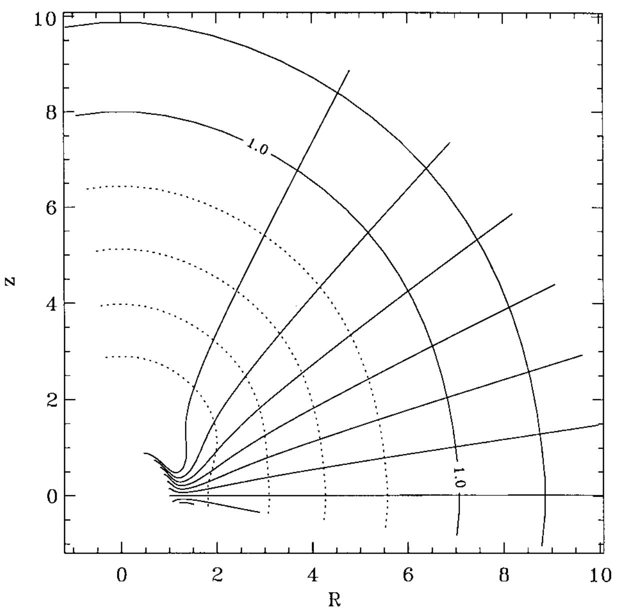

Figure 2. Polytropic transonic stellar wind from a rotating star. The star is in the bottom left corner. We show streamlines and contours (dotted for values below unity) of the poloidal Mach number M_p in the poloidal plane.

drops below 10^{-8}. Note that the normalization is done per component u of the conserved variables $\vec{U}$. Implicitly, this problem took 1382 s on a Cray C90, needing 430 pseudo-time steps with typically 25 Bi-CGSTAB iterations per step. An explicit time integration with the same TVDLF discretization reaches the same accuracy only after 5000 s. When we use the grid masking algorithm to calculate the Jacobian and increase the Courant number to 10000, the implicit solution takes 1920 s needing 354 pseudo-time steps.

6.2. Complex MHD bow shock

A second example is a 2D MHD problem modelling a field-aligned bow shock around a perfectly conducting cylinder. A purely horizontal flow and parallel uniform magnetic field impinge on the cylinder from the left where the inflow conditions are held fixed. The cylinder forms a perfectly conducting impenetrable obstacle, deflecting flow and magnetic field lines. A quarter of the full problem is solved since there is top–bottom symmetry and an open boundary mimics the conditions past the cylinder axis. A similar example is shown in [14], but here we choose the parameters such that the inflow allows for switch-on shocks. It was recently demonstrated [11,30] that in that parameter regime, the bow shock 'dimples' and multiple shock fronts appear in front of the cylinder, containing a bewildering variety of interacting MHD shocks. Figure 3 shows the solution for an inflow Alfvén Mach number $M_A = 1.5$ and plasma beta $\beta = 0.4$, identical in set-up to the result discussed in [11,30]. Magnetic field lines, which are also streamlines, are shown, as well as a grey-scale contour plot of the density. An enlargement of the solution is shown as an inset, where contours depict Alfvén Mach numbers, piling up in shocks. The essential features of the interacting shock fronts are present, but the overall solution is more diffused than the one shown in [11] (their Figures 2, 5 and 6). This is because we used a 120 × 120 spherical grid of radial extent [0.125, 1.4], accumulated towards the cylinder and the horizontal symmetry axis, while De Sterck *et al.* [11] used a specially adapted stretched elliptic-like grid with grid clustering at the shock front positions.

In the implicit calculation, we demanded the residual $\Delta_2 U$ to drop below 10^{-7}, and used the efficient TVDLF Jacobian calculation, MBILU preconditioning and Bi-CGSTAB. The magnetic field divergence is controlled using Powell source terms [18], and minmod limiting is applied to the primitive variables. An efficient way of calculating the steady state solution is to first take 100 time steps explicitly to overcome the initial transient, and subsequently change to fully implicit time integration. We took a Courant number of 100 until $\Delta_2 U < 10^{-5}$, and then raised the Courant number to 1000 to accelerate convergence to below 10^{-7}. This takes 3.5 h to complete on one processor of a Cray C90. An explicit calculation suffers from stagnation and residual fluctuations much akin to those found in the two steady state problems contained in [14]. A similar bow shock flow presented there was calculated seven times faster implicitly than explicitly. That paper also contains a quantitative comparison of using different iterative schemes for the linear system solves.

7. CONCLUSIONS

In this paper, we discussed different implicit and semi-implicit integration schemes. We provided the reader with ample detail, so that our formalism can be taken as a guideline to extend an existing code using explicit high resolution schemes with implicit time integration schemes.

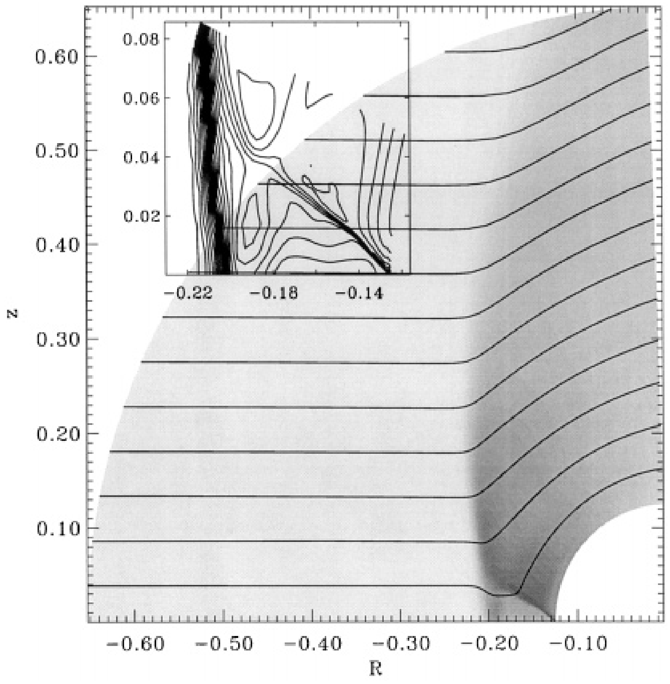

Figure 3. MHD bow shock flow around a perfectly conducting cylinder (at right). Since the inflow parameters are in the switch-on regime, the shock front dimples and multiple shock fronts appear. We show magnetic field lines and density in grey scale. The inset shows the multiple shock fronts ahead of the cylinder as seen in a contour plot of the Alfvén Mach number.

We showed how we always maintain second-order spatial and temporal accuracy, while using a spatially first-order evaluation for the Jacobian matrix. We note that our formalism comprises well-known implicit schemes, like trapezoidal and BDF2, while allowing variable time steps and an implicit treatment of only some terms or variables. VAC is now capable of solving steady state and temporally second-order time-accurate hydrodynamic and magnetohydrodynamic problems implicitly. This is to be done in a problem-dependent fashion, by lifting, if needed, the most restrictive CFL constraint on the time step.

We implemented two methods where the (numerical) calculation of the Jacobian matrix is independent of the actual discretization used for the residual. One is the generally applicable, easily coded matrix-free method, which is used in combination with a CG-type iterative solver. This approach works fine for stiff elliptic source terms. For advection dominated problems, one can use the matrix-free method in combination with the MRAI time stepping scheme [3]. Another discretization-independent method is the general grid masking algorithm where the Jacobian matrix elements are calculated. This simple algorithm is described in detail, and allows for upwinded discretizations, without the analytical effort of linearizing the Riemann solver. An efficient Jacobian calculation for the TVDLF scheme is also provided, and differences between the various implementations are pointed out.

For the linear system solvers, the most novel feature is the MBILU preconditioner suitable to deal with advection dominated problems. While more details are given in [10], we summarize in this paper its properties, including storage requirements. Memory issues are also compared for the different CG-type iterative solvers.

We illustrated some aspects of the algorithms discussed on two steady state 2D flow problems, without and with discontinuities (shocks). Other examples, including time-dependent calculations, are found in [12–14]. The algorithmic details given here augment their brief discussion presented in [14], where a selection of test problems demonstrates the efficiency of the implicit and semi-implicit time integration schemes. The test problems described therein model accretion onto a black hole, oscillations of a high density plasma sheet in very low density surroundings, the formation of a bow shock around a perfectly conducting cylinder, and reconnection of magnetic field lines.

Finally, large-scale simulations require suitable parallelization strategies. We pointed out how the Jacobian matrix is calculated in a data parallel way. One can optimally exploit this type of parallelism by distributing the arrays over several processors. The code has been ported to high performance Fortran, and its scaling properties are demonstrated in [31,32].

ACKNOWLEDGMENTS

This work was performed as part of the project on ‘Parallel Computational Magneto-Fluid Dynamics’, funded by the Dutch Science Foundation (NWO) Priority Program on Massively Parallel Computing, and co-ordinated by Professor Dr J.P. Goedbloed. It was sponsored by the Stichting Nationale Computerfaciliteiten (NCF) for the use of supercomputer facilities. RK performed this work as part of the research program of FOM, MAB gratefully acknowledges collaboration with Gerard Sleijpen and Henk van der Vorst. GT currently receives a postdoctoral fellowship (D 25519) from the Hungarian Science Foundation (OTKA), and thanks the Astronomical Institute of Utrecht University for its kind hospitality.

REFERENCES

1. G. Tóth, 'A general code for modeling MHD flows on parallel computers: versatile advection code', *Astrophys. Lett. Comm.*, **34**, 245 (1996).
2. G. Tóth, 'Versatile advection code', in B. Hertzberger and P. Sloot (eds.), *Proceedings of High Performance Computing and Networking Europe 1997*, *Lecture Notes in Computer Science*, Vol. 1225, Springer, Berlin, 1997, p. 253.
3. M.A. Botchev, G.L.G. Sleijpen and H.A. van der Vorst, 'Stability control for approximate implicit time-stepping schemes with minimum residual iterations', *Appl. Num. Math.*, accepted (1998).
4. G.H. Golub and H.A. van der Vorst, 'closer to the solution: iterative linear solvers', in I.S. Duff and G.A. Watson (eds.), *The State of the Art in Numerical Analysis*, Clarendon Press, Oxford, 1997, p. 63.
5. M.R. Hestenes and E. Stiefel, 'Methods of conjugate gradients for solving linear systems', *J. Res. Natl. Burl Stand.*, **49**, 409 (1954).
6. Y. Saad and M.H. Schultz, 'GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems', *SIAM J. Sci. Stat. Comput.*, **7**, 856 (1986).
7. H.A. van der Vorst, 'Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems', *SIAM J. Sci. Statist. Comput.*, **13**, 631 (1992).
8. H.A. van der Vorst and C. Vuik, 'GMRESR: a family of nested GMRES methods', *Numer. Linear Algebra Appl.*, **1**, 369 (1994).
9. G.L.G. Sleijpen and D.R. Fokkema, 'BiCGSTAB(ℓ) for linear equations involving unsymmetric matrices with complex spectrum', *ETNA*, **1**, 11 (1993).
10. A. van der Ploeg, R. Keppens and G. Tóth, 'Block incomplete LU-preconditioners for implicit solution of advection dominated problems', in B. Hertzberger and P. Sloot (eds.), *Proceedings of High Performance Computing and Networking Europe 1997*, *Lecture Notes in Computer Science*, Vol. 1225, Springer, Berlin, 1997, p. 421.
11. H. De Sterck, B.C. Low and S. Poedts, 'Complex magnetohydrodynamic bow shock topology in field-aligned low-β flow around a perfectly conducting cylinder', *Phys. Plasmas*, **5**, 4015 (1998).
12. R. Keppens and J.P. Goedbloed, 'Numerical simulations of stellar winds: polytropic models', *Astron. Astrophys.*, **343**, 251 (1999).
13. D. Molteni, G. Tóth and O.A. Kuznetsov, 'On the azimuthal stability of shock waves around black holes', *Astrophys. J.*, accepted (1998).
14. G. Tóth, R. Keppens and M.A. Botchev, 'Implicit and semi-implicit schemes in the versatile advection code: numerical tests', *Astron. Astrophys.*, **332**, 1159 (1998).
15. J.P. Boris and D.L. Book, 'Flux-corrected transport. I. SHASTA, a fluid transport algorithm that works', *J. Comput. Phys.*, **11**, 38 (1973).
16. A. Harten, 'High resolution schemes for hyperbolic conservation laws', *J. Comput. Phys.* **49**, 357 (1983).
17. P.L. Roe, 'Approximate Riemann solvers, parameter vectors, and difference schemes', *J. Comput. Phys.*, **43**, 357 (1981).
18. K.G. Powell, 'An approximate Riemann solver for magnetohydrodynamics (that works in more than one dimension)', *ICASE Report No 94-24*, Langley, VA, 1994.
19. H.C. Yee, 'A class of high-resolution explicit and implicit shock-capturing methods', *NASA TM-101088*, 1989.
20. G. Tóth and D. Odstrčil, 'Comparison of some flux corrected transport and total variation diminishing numerical schemes for hydrodynamic and magnetohydrodynamic problems', *J. Comput. Phys.*, **128**, 82 (1996).
21. R.P. Fedorenko, 'Stiff systems of ordinary differential equations', in G.I. Marchuk (ed.), *Numerical Methods and Applications*, CRC Press, Boca Raton, 1994, p. 117.
22. E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential–Algebraic Problems*, *Springer Series in Computational Mathematics*, *14*, Springer, Berlin, 1991.
23. E. Hairer, S.P. Nørsett and G. Wanner, *Solving Ordinary Differential Equations I. Non-Stiff Problems*, *Springer Series in Computational Mathematics 8*, Springer, Berlin, 1987.
24. P.N. Brown, 'A local convergence theory for combined inexact-Newton/finite difference projections methods', *SIAM J. Numer. Anal.*, **24**, 407 (1987).
25. M.A. Botchev and H.A. van der Vorst, 'Approximated implicit time stepping schemes in a distributed memory parallel environment', *Preprint 1054*, Department of Mathematics, Utrecht University, March 1998. Available at http://www.math.ruu.ml/publications/.
26. W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in Fortran*, 2nd edn., Cambridge University Press, Cambridge, 1992, p. 68.
27. A. van der Ploeg, 'Reordering strategies and LU-decomposition of block tridiagonal matrices for parallel processing', *Tech. Rep. NM-R9618*, Centrum voor Wiskunde en Informatica, October 1996.
28. S.C. Eisenstat, 'Efficient implementation of a class of preconditioned conjugate gradient methods', *SIAM J. Sci. Statist. Comput.*, **2**, 1 (1981).
29. O. Axelsson and G. Lindskog, 'On the eigenvalue distribution of a class of preconditioning methods', *Numer. Math.*, **48**, 479 (1986).
30. H. De Sterck, B.C. Low and S. Poedts, 'Characteristic analysis of a complex two-dimensional magnetohydrodynamic bow shock flow with steady compound shocks', *Phys. Plasmas*, accepted (1998).
31. G. Tóth and R. Keppens, 'Comparison of different computer platforms for running the versatile advection code', in P. Sloot, M. Bubak and B. Hertzberger (eds.), *Proceedings of High Performance Computing and Networking Europe 1998*, *Lecture Notes in Computer Science*, Vol. 1401, Springer, Berlin, 1998, p. 368.
32. R. Keppens and G. Tóth, 'Simulating magnetized plasmas with the versatile advection code', in *Proc. VECPAR'98, 3rd Int. Meeting on Vector and Parallel Processing*, Part II, Porto, Portugal, June 21–23, 1998, p. 599.