Adding Implicit and Semi-Implicit Schemes to the Versatile Advection Code

R. Keppens, FOM-Institute for Plasma-Physics Rijnhuizen,
P.O. Box 1207, 3430 BE Nieuwegein, The Netherlands,
keppens@rijnh.nl

G. Tóth, Astronomical Institute, Utrecht University,
P.O. Box 80000, 3508 TA Utrecht, The Netherlands,
toth@fys.ruu.nl

M. A. Botchev, Mathematical Institute, Utrecht University,
P.O. Box 80010, 3508 TA Utrecht, The Netherlands,
botchev@math.ruu.nl

A. van der Ploeg, Centrum voor Wiskunde en Informatica,
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands,
Auke.van.der.Ploeg@cwi.nl

May 30, 1997

Subject Classification and Key Words:

35L65-Conservation Laws
65D99-Numerical approximation
65M99-PDEs, initial value and time-dependent initial-boundary value problems
77A05-Magnetohydrodynamics

Proposed Running Head: Implicit and Semi-Implicit Schemes

Proofs to: R. Keppens
Mailing Address: FOM-Institute for Plasma-Physics Rijnhuizen,
P.O. Box 1207, 3430 BE Nieuwegein, The Netherlands
e-mail: keppens@rijnh.nl
Tel.: (+31) 30-6096941
Fax.: (+31) 30-6031204

Abstract

We formulate 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, we show how second order spatial and temporal accuracy for the smooth part of the solutions can be maintained. We discuss strategies to obtain steady state and time accurate solutions implicitly.

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 our implementation. We discuss three options available in VAC 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 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-type iterative solver is used. For advection dominated problems, preconditioning is needed to accelerate the convergence of the iterative schemes. We implemented the Modified Block Incomplete LU-preconditioner which performs very well. As an alternative approach, we describe the Minimal Residual Predictor-Corrector time stepping scheme, which can be combined with the matrix-free method.

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

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

where t is the time coordinate, i and j run over 1, 2 or 3 components of the spatial coordinate $\mathbf{x}$, $\vec{U}$ 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 Δt_{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 Δt_{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 Δt_{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 Δt_{max}, while, in reality, it is balanced by another term and $\vec{U}(\mathbf{x}, t)$ 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 to take arbitrarily large time steps 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) [19, 20], which is a general purpose software package for solving equations of type (1). The newly implemented (semi-)implicit time integration schemes now allow the user to select the most optimal combination of the explicit high-resolution schemes which were already available (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 $\vec{U}_{impl}$ and the implicitly treated part of $\vec{R}$ with $\vec{R}_{impl}$, the essential building block for the implicit and semi-implicit schemes resides in the accurate evaluation of the Jacobian matrix $\partial\vec{R}_{impl}/\partial\vec{U}_{impl}$ (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 $\vec{R}_{impl}$ 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 $\partial\vec{R}_{impl}/\partial\vec{U}_{impl}$ 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 Predictor-Corrector (MR-PC) strategy, recently introduced in [3]. For completeness, we include a brief summary of the main ideas of this new time stepping strategy in Section 6.

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-type iterative schemes (a recent overview of these schemes is given in [7]). Implemented in VAC are Conjugate Gradient (CG) [11] for symmetric matrices, while for nonsymmetric systems, we can use the restarted Generalized Minimal Residual Scheme (GMRES) [16] and the Stabilized Bi-Conjugate Gradient method (Bi-CGSTAB) [24], and their generalizations GMRESR [23] and BiCGstab(ℓ) [17]. For advection dominated problems, the Modified Block Incomplete *LU* (MBILU) preconditioner [22] is of vital importance to accelerate the convergence of the iterative schemes.

The motivation behind the development and implementation of all these algorithms in VAC lies in their ultimate applicability to solve the equations of hydrodynamics and magnetohydrodynamics. In an accompanying paper [12], we present astrophysically oriented test problems demonstrating the advantages of many different options for the time discretization. A problem-specific approach to eliminate the most restrictive constraint on the time step greatly accelerates the solution procedure.

With these applications in mind, 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. The user-friendly Versatile Advection Code has been extended with all schemes discussed in this paper, and is intended as a freely distributed software package (http://www.fys.ruu.nl/~toth/ and http://www.fys.ruu.nl/~mpr/).

2 Explicit Schemes in VAC

Several equation modules of type (1) are available in VAC, all written in a dimension independent notation, with $1 \leq D \leq 3$ spatial dimensions and C vector components with $D \leq C \leq 3$. 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.

In VAC, the conservative variables $\vec{U}(\mathbf{x}, t)$ 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 $\vec{U}_k(t) \equiv (1/V_k) \int_{V_k} \vec{U}(\mathbf{x}, t)$. 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) [2] and three Total Variation Diminishing schemes [10], two of them with a Roe-type Riemann solver [15, 13] (TVD and TVD-MUSCL), and a Lax-Friedrichs type scheme (TVDLF) [25, 18]. All of these methods calculate smooth flows to second order accuracy, and are able to resolve 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 [18].

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:

$$\sum_k \mid \Delta U_{k+1/2}^{n+1} \mid \leq \sum_k \mid \Delta U_{k+1/2}^{n} \mid, \tag{2}$$

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 piece-wise 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 in VAC 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 now limits the slopes obtained from the conservative variables ΔU themselves, 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 [18]. In the implicit schemes we will only need the first order accurate version, which updates the $\vec{U}$ variables by

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

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 in VAC 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 which relates 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. We note that for explicit time integration, VAC allows arbitrary procedures for updating the ghost cells through user written subroutines, and it also allows for boundary regions inside the computational domain.

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

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

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

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

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

where $\vec{U}_{impl}$ contains all implicitly treated variables of $\vec{U}$, hence symbolically, we write $\vec{U} = (\vec{U}_{expl}, \vec{U}_{impl})$. We always evaluate the Jacobian $\partial\vec{R}_{impl}/\partial\vec{U}_{impl}$ at $(\vec{U}^{n+1}_{expl}, \vec{U}^n_{impl})$, and in 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 $\vec{U}_{impl} = \vec{U}$, $\vec{R}_{impl} =$

$\vec{R}$ 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 $\partial\vec{R}_{low}/\partial\vec{U}$ for the Jacobian matrix $\partial\vec{R}_{impl}/\partial\vec{U}_{impl} = \partial\vec{R}/\partial\vec{U}$ 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

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

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). From numerical experiments as those described in the accompanying paper [12], we conclude that for steady state problems, the linearized pseudo time stepping approach is often more efficient than the non-linear Newton-Raphson iteration.

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

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

where we assumed that $R_{impl} = R$ 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. [6], [9]). 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

$$\begin{aligned}\vec{U}^{n+1} &= \vec{U}^n + \Delta t_n \vec{R}(\vec{U}^n) \\ &+\Delta t_n \left[\alpha \frac{\vec{U}^n - \vec{U}^{n-1}}{\Delta t_{n-1}} - \alpha \vec{R}(\vec{U}^n) + \beta \vec{R}_{impl}(\vec{U}^{n+1}) - \beta \vec{R}_{impl}(\vec{U}^n) \right] \end{aligned} \quad (8)$$

where $\vec{R}_{impl}(\vec{U}^{n+1})$ can again be linearized by (5). The scheme is three-level whenever the parameter $\alpha \neq 0$, while for $\alpha = 0$, it is identical with (4). Note that when $\vec{R}_{impl} = \vec{R}$, $\alpha = 1/3$, and $\beta = 2/3$, we get the second order Backward Differentiation Formula (BDF2) [8] 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 $(U^n - U^{n-1})/\Delta t_{n-1} = R(U^n) - (\Delta t_{n-1}/2) dR/dt + \mathcal{O}(\Delta t^2)$. Substituting this into (8) we obtain

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

The coefficient of $d\vec{R}/dt$ 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 $R_{impl} = R$ 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 $\vec{U}_{impl} = \vec{U}$ and $\vec{R}_{impl} = \vec{R}$ 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 $\vec{R}_{impl}$, 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

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

where we used $\vec{R}_{high} \equiv \vec{R}$ and $\vec{R}_{low} = \vec{R}_{high} + \mathcal{O}(\Delta x)$. 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 $\vec{U}_{expl}$ using a second order explicit time integration scheme, e.g. a two-stage Runge-Kutta scheme

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

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

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

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 $\vec{U}_{impl}$ all the fluxes and sources are treated implicitly, thus for these variables $R_{impl} = R$ disregarding possible differences in the discretization. We face a new problem when $R_{impl} \neq R$, for example, when only the source terms $\vec{S}$ 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 $\vec{R}_{expl}$, where the predictor step involves the total residual $\vec{R}$, and an implicit treatment for $\vec{R}_{impl}$, as

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

where $\vec{R} = \vec{R}_{expl} + \vec{R}_{impl}$. This scheme is analoguous with 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 $d\vec{R}_{expl}/dt$, thus

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

We also implemented a simple scheme, which combines a *semi-implicit predictor step* with an explicit corrector step

$$\begin{aligned}\vec{U}^{n+1/2} &= \vec{U}^n + \frac{\Delta t}{2}\left[\vec{R}(\vec{U}^n) + \vec{R}_{impl}(\vec{U}^{n+1/2}) - \vec{R}_{impl}(\vec{U}^n)\right], \\ \vec{U}^{n+1} &= \vec{U}^n + \Delta t \vec{R}(\vec{U}^{n+1/2}). \end{aligned} \tag{15}$$

For applications involving implicitly treated elliptic-type source terms, scheme (15) can be sufficiently stable.

4 Jacobian Evaluation

The linearization (5) leads to large linear systems like (6) containing the Jacobian matrix $\partial\vec{R}_{impl}/\partial\vec{U}_{impl}$. 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 which is independent of the actual discretization used. Its implementation, advantages and disadvantages are discussed below.

To allow for preconditioning which 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 $\partial\vec{R}_{impl}/\partial\vec{U}_{impl}$ as $\partial\vec{R}/\partial\vec{U}$.

4.1 Matrix-Free Jacobian Evaluation

In cases where e.g. source terms can not 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

$$\frac{\partial\vec{R}}{\partial\vec{U}}\Delta\vec{U} = \frac{\vec{R}(\vec{U}^n + \epsilon\Delta\vec{U}) - \vec{R}(\vec{U}^n)}{\epsilon}. \tag{16}$$

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 $||.||$ 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 Conjugate Gradient-type method converges fast.

We note that combining a Conjugate Gradient-type iterative method with a matrix-free evaluation of the Jacobian is not always fool-proof, as the evaluation (16) 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 Conjugate Gradient-type iterative methods (more on this can be found in [4]).

However, the matrix-free approach can be used succesfully when the Krylov dimension is kept moderate or small, like in the MR-PC schemes outlined below (Section 6). One may also combine an iterative method of low Krylov dimension and a matrix-free approach with an extra outer Newton-Raphson iteration on the non-linear system, to improve accuracy and convergence.

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, e.g., use a first order upwind evaluation for the fluxes, without the effort to linearize the (approximate) Riemann solver in $\vec{U}$ analytically.

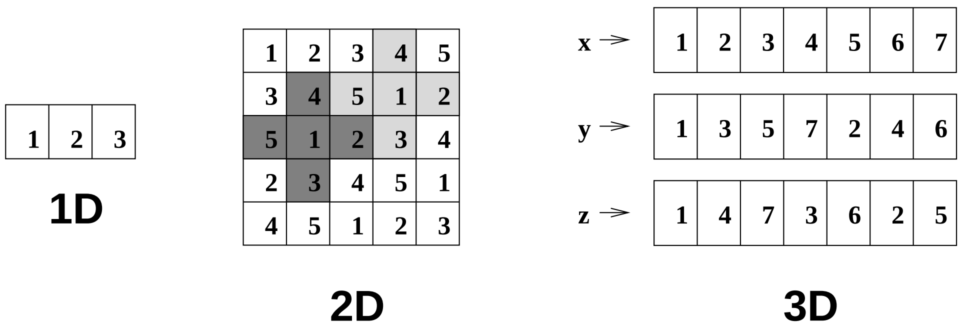


Figure 1: Grid masks used for the calculation of the Jacobian matrix.

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 labeling 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 which 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

$$\frac{\partial R_i^u}{\partial U_j^w} = \frac{R_i^u(\vec{U} + \epsilon\delta_j^w) - R_i^u(\vec{U})}{\epsilon}. \tag{17}$$

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 labeled with 1 are perturbed, we can immediately read off the main diagonal contributions to the Jacobian matrix for all cells labeled with 1, simultaneously with lower and upper diagonal contributions for cells labeled by 2 and 3, respectively. As we loop over the variable index w and the grid labels from 1 to $N_{stencil} = 2 \times D + 1$, 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 $1 + N_{stencil} \times N_{impl}$ 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 (17) 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 which 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 which 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 [12] 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 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 in VAC 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

$$\frac{\partial \vec{R}}{\partial \vec{U}} = -\sum_i \partial_i \frac{\partial \vec{F}}{\partial \vec{U}} + \frac{\partial \vec{S}}{\partial \vec{U}}, \tag{18}$$

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

$$\begin{aligned}
\frac{\partial R_i^u}{\partial U_i^w} &= \frac{\partial S_i^u}{\partial U_i^w} - \frac{1}{2\Delta x}(c_{i+1/2}^{max} + c_{i-1/2}^{max})\frac{\partial U_i^u}{\partial U_i^w}, \\
\frac{\partial R_i^u}{\partial U_{i+1}^w} &= -\frac{1}{2\Delta x}\frac{\partial F_{i+1}^u}{\partial U_{i+1}^w} + \frac{1}{2\Delta x}c_{i+1/2}^{max}\frac{\partial U_{i+1}^u}{\partial U_{i+1}^w}, \\
\frac{\partial R_i^u}{\partial U_{i-1}^w} &= +\frac{1}{2\Delta x}\frac{\partial F_{i-1}^u}{\partial U_{i-1}^w} + \frac{1}{2\Delta x}c_{i-1/2}^{max}\frac{\partial U_{i-1}^u}{\partial U_{i-1}^w},
\end{aligned} \tag{19}$$

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

$$\frac{\partial f}{\partial U^w} = \frac{f(\vec{U} + \epsilon\delta^w) - f(\vec{U})}{\epsilon}. \tag{20}$$

Since $\vec{F}$ and $\vec{S}$ need to be evaluated for each $w = 1, ..., N_{impl}$ 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 (19) 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 $\partial R_i^u/\partial U_{ghost}^w$ are added to the matrix element $\partial R_i^u / \partial U_j^w$ where j is the index of the physical cell on which the ghost cell depends.

Note that in (19), 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 (19) and the Jacobian matrix calculated by the general method (17), 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 [25].

We also have the option not to include the terms involving c^{max} into (19), which then corresponds to a standard second order centered difference formula for the fluxes. Such a discretization, however, is prone to oscillations close to steep gradients and discontinuities, and the main diagonal blocks are zero, which may cause difficulties for the iterative solvers.

As a final note, we mention that the programming effort and the length of the source code involved in implementing this Jacobian calculating method, which should work for 1, 2, and 3 dimensional Cartesian, polar, and general structured grids, for slab and axial symmetry, for normal and periodic boundaries, is far more than for the more general, although less efficient, *grid masking* algorithm described in the previous section.

5 Linear System Solvers

The solution strategy which 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 unkowns x related to the updates $\Delta\vec{U} = \vec{U}_{impl}^{n+1} - \vec{U}_{impl}^{n}$ according to

$$x^u \equiv \frac{\Delta U_{impl}^u}{\sqrt{N_{grid}^{-1} \sum_k (U_k^{u,n})^2}}, \tag{21}$$

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 one dimensional grid leads to a block tridiagonal matrix structure. For these banded matrices, storage demands are low and a direct solver can be used. In VAC, we implemented a direct block tridiagonal solver, together with a cyclic version based on the Woodbury formula [14] 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 [21].

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 occuring matrix (symmetric or not, diagonally dominant or other, ...). The Conjugate Gradient 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 nonsymmetric 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

$$\hat{L}^{-1}\hat{A}\hat{U}^{-1}\tilde{x} = \hat{L}^{-1}b, \qquad x = \hat{U}^{-1}\tilde{x}. \tag{22}$$

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 incomplete block LU-decompositions, computer storage demands are very low, because the preconditioner only requires one extra block diagonal.

In [22] 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 [5]. 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 [1], 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 [22] 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. For more details pertaining to the implementation of the MBILU preconditioner, we refer to [22].

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 $N_{impl} \times N_{grid}$ 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 2 temporary vectors. Bi-CGSTAB works for symmetric and nonsymmetric matrices as well, and it requires storage for 7 vectors. Its more stable variant BiCGstab(ℓ) needs $5 + 2\ell$ vectors. The GMRES($N_{restart}$) scheme requires $N_{restart} + 1$ 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 $2+2N_{truncate}+N_{inner}$ vectors. Typical values are $N_{inner} = 2, 3$ and $N_{truncate} = 5$. In our experience Bi-CGSTAB is often the optimal choice.

The storage required by the iterative scheme is in addition to the memory space occupied by the matrix $\hat{A}$, unless a matrix-free method is used. The matrix has $(2D + 1)N_{impl}^2 \times N_{grid}$ non-zero elements, equivalent with $5N_{impl}$ and $7N_{impl}$ vectors in 2 and 3D, respectively. The MBILU preconditioner needs the size of a block diagonal matrix, which is the same as N_{impl} vectors. This extra memory is only needed in the construction phase of the lower and upper triangular matrices $\hat{L}$ and $\hat{U}$, so that the preconditioner represents no real extra memory cost.

6 MR-PC time stepping

The Minimal Residual Predictor-Corrector (MR-PC) time stepping scheme has been introduced recently in [3] and implemented in VAC. We sketch here briefly the scheme and its basic properties.

One time step by the MR-PC scheme consists of an explicit predictor step (with a time step larger than normally allowed by the CFL-condition) and a corrector step where the linearized system of an implicit scheme is solved by a few *minimal residual*-type (e.g. GMRES) iterations with the initial guess taken from the predictor step.

The MR-PC scheme is implemented in the following way. Any of the linearized (semi-)implicit schemes, like (4), (8) can be taken for the corrector step in MR-PC. Assume for instance that this scheme is (4), then the linearized equation

$$\left[\frac{\hat{I}}{\Delta t} - \beta\frac{\partial \vec{R}_{impl}}{\partial \vec{U}}\right]\left(\vec{U}^{n+1} - \vec{U}^{n}\right) = \vec{R}(\vec{U}^{n}) \tag{23}$$

is solved with only a few ($k \leq 7$) iterative steps. Due to the restricted number of steps, a good initial guess has to be chosen.

For the time accurate calculations the choice of the predictor step is aimed at retaining the desired order of accuracy. For second order accuracy, the initial guess $\vec{U}_0^{n+1}$ for (23) is obtained by a second order explicit scheme, e.g. by

$$\vec{U}_0^{n+1} = \vec{U}^{n} + \Delta t\vec{R}\left(\vec{U}^{n} + \frac{\Delta t}{2}\vec{R}(\vec{U}^{n})\right). \tag{24}$$

If the corrector step is second order in time as well, e.g. (23) with $\beta = 1/2$, the overall MR-PC step will also be second order accurate. See [3] for a proof.

Being an explicit scheme, the MR-PC time stepping is not unconditionally stable, but its stability region is much wider than those of ordinary explicit schemes. For example, already for $k = 1$, the second order MR-PC with Adams2 predictor step and BDF2 corrector has an *effective time step* (time step divided by the number of function evaluations, i.e. taking into account the computational work) typically more than three times larger than that of Adams2 [3].

The order of the local error in MR-PC does not depend on how accurately the linear system (23) is solved. Thus, the corrector step with k minimal residual iterations is mainly used to stabilize the scheme rather than to solve (23). This stabilization is adaptive in the sense that during the corrector iterative steps the solution components corresponding to the stiff part of the spectrum will be treated as in an implicit scheme [3]. This effect accumulates during the time stepping because usually the spectrum of the Jacobian does not vary significantly.

For steady state calculations the predictor step is not necessary, since temporal accuracy is not of interest. Moreover, the explicit predictor step puts the main restriction on the stability of MR-PC. Therefore, for the steady state calculations we skip the predictor step to improve stability. The initial guess $\vec{U}_0^{n+1}$ for (23) is taken to be $\vec{U}^n$. The stability region then becomes significantly larger, in a particular steady state problem described in [12], the achievable Courant number for MR-PC(5) (i.e., for MR-PC with $k = 5$ iterations) is about 100.

The attractive feature of MR-PC is the possibility to use the matrix-free evaluation (Section 4.1) of the Jacobian action which makes the storage requirements of the scheme very low. The perturbations induced by the numerical evaluation of the directional differences are not so harmful for the iterative process in MR-PC because the number of iterations k is kept very small. The lack of preconditioning, the other drawback of the matrix-free approach, appears in our experience to be of lesser importance for MR-PC. Besides, as it was noted earlier, the aim of the iterations in MR-PC is to stabilize the scheme rather than to solve (23).

The time step size control for MR-PC is based on the spectral information delivered by the minimal residual iterations and requires a negligible amount of extra work. For details see [3]. Since the MR-PC scheme is explicit, allows to use the matrix free Jacobian evaluation, and does not require preconditioning, it can be easily parallelized. In [12], some numerical experiments done with MR-PC in VAC are presented.

7 Conclusions

In this paper, we discussed different implicit and semi-implicit integration schemes recently implemented in VAC. 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.

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 Conjugate Gradient-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 MR-PC time stepping scheme, of which we outline the main features only (details are provided in [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 implemented in VAC is the MBILU preconditioner suitable to deal with advection dominated problems. While more details are given in [22], we summarize in this paper its properties, including storage requirements. Memory issues are also compared for the different Conjugate Gradient-type iterative solvers implemented in VAC.

A follow-up paper [12] gives a selection of test problems where the efficiency of the implicit and semi-implicit time integration schemes is demonstrated and where the (semi-)implicit schemes clearly outperform an explicit approach. The test problems described therein are hydrodynamic and magnetohydrodynamic problems typical of astrophysical research, modeling 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. In [12], it is shown that the MBILU preconditioning is essential for various HD and MHD problems. We also demonstrate how one can obtain steady state and time accurate solutions containing shocks in a fully or semi-implicit manner.

Finally, large-scale simulations require suitable parallelization strategies. We pointed out how the Jacobian matrix is calculated in a data parallel way. It is our intention to exploit this type of parallelism by optimally distributing the arrays over several processors. Therefore, porting the code to High Performance Fortran has been initiated.

Acknowledgements. This work was performed as part of the project on 'Parallel Computational Magneto-Fluid Dynamics', funded by the NWO Priority Program on Massively Parallel Computing. It was sponsored by the

Stichting Nationale Computerfaciliteiten (NCF) for the use of supercomputer facilities. R.K. thanks FOM, M.A.B. gratefully acknowledges collaboration with Gerard Sleijpen and Henk van der Vorst.

References

[1] O. Axelsson and G. Lindskog, *Numer. Math.* **48**, 479 (1986).

[2] J. P. Boris and D. L. Book, *J. Comput. Phys.* **11**, 38 (1973).

[3] M. A. Botchev, G. L. G. Sleijpen, and H. A. van der Vorst, Low-dimensional Krylov subspace iterations for enhancing stability of time step integration schemes, *Preprint 1004, Dept. of Mathematics, Utrecht University, the Netherlands*, (1997).

[4] P. N. Brown, *SIAM J. Numer. Anal.* **24**(2), 407 (1987).

[5] S. C. Eisenstat, *SIAM J. Sci. Statist. Comput.* **2**, 1 (1981).

[6] R. P. Fedorenko, , in *Numerical Methods and Applications*, edited by G. I. Marchuk (CRC Press, Boca Raton, 1994), p. 117.

[7] G. H. Golub and H. A. van der Vorst, Closer to the solution: Iterative linear solvers, to appear in *The State of the Art in Numerical Analysis*, edited by I. S. Duff and G. A. Watson (Oxford University Press), *Preprint at* `http://www.math.ruu.nl/people/vorst/`.

[8] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I. Nonstiff Problems*, Springer Series in Computational Mathematics 8 (Springer-Verlag, 1987).

[9] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems*, Springer Series in Computational Mathematics, Vol. 14 (Springer-Verlag, 1991).

[10] A. Harten, *J. Comput. Phys.* **49**, 357 (1983).

[11] M. R. Hestenes and E. Stiefel, *J. Res. Natl. Bur. Stand.* **49**, 409 (1954).

[12] R. Keppens, G. Tóth, and M. A. Botchev, *Implicit and Semi-Implicit Schemes in the Versatile Advection Code: Numerical Tests*, in preparation.

[13] K. G. Powell, *ICASE Report No 94-24, Langley, VA* (1994).

[14] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in Fortran, 2nd edition* (Cambridge University Press, 1992), p. 68.

[15] P. L. Roe, *J. Comput. Phys.* **43**, 357 (1981).

[16] Y. Saad and M. H. Schultz, *SIAM J. Sci. Stat. Comput.* **7**(3), 856 (1986).

[17] G. L. G. Sleijpen and D. R. Fokkema, *ETNA* **1**, 11 (1993).

[18] G. Tóth and D. Odstrčil, *J. Comput. Phys.* **128**, 82 (1996).

[19] G. Tóth, *Astrophys. Lett. & Comm.* **34**, 245 (1996).

[20] G. Tóth, in *Proceedings of High Performance Computing and Networking Europe 1997* , Lecture Notes in Computer Science, Vol. 1225, edited by B. Hertzberger and P. Sloot (Springer-Verlag, 1997), p. 253.

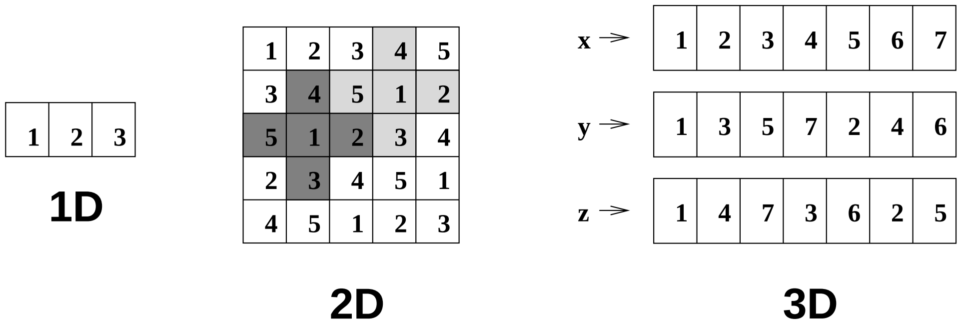
[21] A. van der Ploeg, *Technical Report* **NM-R9618**, Centrum voor Wiskunde en Informatica (October 1996).

[22] A. van der Ploeg, R. Keppens, and G. Tóth, in *Proceedings of High Performance Computing and Networking Europe 1997* , Lecture Notes in Computer Science, Vol. 1225, edited by B. Hertzberger and P. Sloot (Springer-Verlag, 1997), p. 421.

[23] H.A. van der Vorst and C. Vuik, *Technical Report* **R91-80**, Faculty of Technical Mathematics and Informatics Delft (1991).

[24] H. A. van der Vorst, *SIAM J. Sci. Statist. Comput.* **13**, 631 (1992).

[25] H. C. Yee, *NASA TM-101088* (1989).



1 2 3
1D
1 2 3 4 5
3 4 5 1 2
5 1 2 3 4
2 3 4 5 1
4 5 1 2 3
2D
x → 1 2 3 4 5 6 7
y → 1 3 5 7 2 4 6
z → 1 4 7 3 6 2 5
3D

Figure Caption:

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