

# Acceleration techniques for Newton's non-linear iterative scheme

*Denis Vanderstraeten*

*Report TW 303, March 2000*



Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Acceleration techniques for Newton's non-linear iterative scheme

*Denis Vanderstraeten*

*Report TW 303, March 2000*

Department of Computer Science, K.U.Leuven

## **Abstract**

The solution of non-linear sets of algebraic equations is usually obtained by the Newton's method, exhibiting quadratic convergence. For practical simulations, a significant computational effort consists in the evaluation of the Jacobian matrices. In this paper, we propose and experiment various methods to speed the convergence process either by re-using information from previous iterates or by by-passing the Jacobian evaluations.

These methods are applied to the solution of hyperbolic PDE's arising in CFD problems. A significant improvement is obtained in terms of computation cost compared to the crude Newton's approach.

# Acceleration techniques for Newton's non-linear iterative scheme

DENIS VANDERSTRAETEN<sup>1</sup>

## Abstract

The solution of non-linear sets of algebraic equations is usually obtained by the Newton's method, exhibiting quadratic convergence. For practical simulations, a significant computational effort consists in the evaluation of the Jacobian matrices. In this paper, we propose and experiment various methods to speed the convergence process either by re-using information from previous iterates or by by-passing the Jacobian evaluations.

These methods are applied to the solution of hyperbolic PDE's arising in CFD problems. A significant improvement is obtained in terms of computation cost compared to the crude Newton's approach.

## 1 Motivation

There exists a wide variety of numerical techniques to approximate the solution of PDE's describing physical phenomena such as finite difference, finite volume, finite element, spectral and residual distribution methods, just to name a few of them. In each of these methods the spatial operator is discretized using different numerical principles. However, after applying the spatial discretization operator on the equations, all of them can be cast in the simple form

$$\frac{dU}{dt} = -R(U), \tag{1}$$

where the vector  $U = U(t)$  represents the spatial discretization of the solution vector and  $R(U)$  is called the *residual* vector. If the transient phenomena are not of interests, (1) can be replaced by a set of non-linear algebraic equations

$$R(U) = 0. \tag{2}$$

However, in the absence of a sufficient initial approximation, Newton's method applied to (2) alone will generally not converge. Other methods like fixed point methods or globalization techniques such as trust region methods or line search algorithms are also likely to fail or to stagnate in a local minimum of  $\|R(U)\|$  [1, 2]. The usual alternative is to keep (1) as a time dependent problem and to march to the steady state by following the physical transient (see [3] and the references therein). More precisely, a backward Euler techniques leads to a set of non-linear algebraic equations

$$\frac{\Delta U^{(k)}}{\Delta t^{(k)}} = -R(U^{(k+1)}). \tag{3}$$

---

<sup>1</sup>Katholieke Universiteit Leuven, Department Computer Science, Celestijnenlaan, 200A, B-3001 Heverlee (Belgium), e-mail: denis@cs.kuleuven.ac.be

The superscript  $k$  denotes the (non-linear) iteration number while  $\Delta t^{(k)}$  is the time step and  $\Delta U^{(k)} = U^{(k+1)} - U^{(k)}$  is the correction to the approximate solution. The right-hand-side is linearized by a Taylor expansion truncated to the first order. Hence, every non-linear iteration  $k$  requires the solution of a linear system of the form

$$\left( \frac{1}{\Delta t^{(k)}} I + J^{(k)} \right) \Delta U^{(k)} = -R(U^{(k)}), \quad (4)$$

where the matrix  $J^{(k)}$  is the Jacobian evaluated at  $U^{(k)}$ . If the transient states are not of interest, a local time stepping technique may be used. The numerical stability of the scheme requires a maximum time step controlled by a so-called CFL number. Equation (4) is generalized in

$$\left( \frac{1}{\gamma^{(k)}} D^{(k)} + J^{(k)} \right) \Delta U^{(k)} = -R(U^{(k)}), \quad (5)$$

where the diagonal matrix  $D^{(k)}$  contains the local time steps and  $\gamma^{(k)}$  is the CFL number. In the following, the linear system (5) is also written by

$$M^{(k)} \Delta U^{(k)} = -R^{(k)}. \quad (6)$$

Usually, the matrix  $M^{(k)}$  is large, sparse and non-symmetric. The GMRES method is the method of choice to solve (6), making use of a preconditioner to speed-up convergence. Also, it is not necessary to solve (6) exactly and a mild convergence criterion  $\|M^{(k)} \Delta U^{(k)} + R^{(k)}\|_2 \leq \tau^{(k)} \|R^{(k)}\|_2$  is sufficient. If the threshold  $\tau^{(k)}$  remains constant, linear convergence is obtained. On the other hand, for  $\tau^{(k)} \rightarrow 0$ , the non-linear iteration converges super-linearly. Quadratic convergence is obtained only if  $\tau^{(k)} = 0$  for all  $k \geq K$  where  $K$  is a constant. We refer the reader to [2, 4, 5] and to the numerous references therein for a global view of (non-) linear algebra for scientific computing.

Finally, the Newton algorithm can be cast in the following form:

---

```

Choose  $U^{(0)}$ , set  $k = 0$ .
While  $\|R^{(k)}\|_2$  is not small enough
    Evaluate  $J^{(k)}, \gamma^{(k)}, \dots$ 
    Solve iteratively  $M^{(k)} \Delta U^{(k)} = -R^{(k)}$ 
    Set  $U^{(k+1)} = U^{(k)} + \Delta U^{(k)}$ 
     $k = k + 1$ 
End

```

---

In the sequel, we try to understand the convergence of the outer (non-linear) and inner (linear) iterations to derive the computational requirements of the method. We then focus on some specific techniques to improve the performance. Accelerations are illustrated with two examples arising in MHD computations.

## 2 Mathematical Framework

The ultimate goal of acceleration techniques is to provide a gain in computation time. To understand where performance can be improved, we decompose a simulation in “computational tasks” and try quantify them. Every task can be measured in flops by code instrumentation or simply in seconds. Flops counts alone can give little information since the specificity of the computer (BLAS, cache effects, ...) is not considered. Real timings provide better accuracy (it takes the quality of the coding into account) but are not machine independent.

In every non-linear iteration, the cost can roughly be divided in the *set-up* cost and the *solution* cost. The set-up cost consists in the construction of the matrix  $M^{(k)}$  and of the preconditioner. The solution cost is proportional to the number of linear iterations where every iteration requires the application of the preconditioner, a matrix–vector product and some other vector updates<sup>2</sup>. Let the function  $C_{xx}$  denote the cost of a particular operation. We write

$$C_{non-lin} = C_{set-up} + n_i C_{sol},$$

where  $n_i$  represents the number of linear iterations. The right-hand-side can be further decomposed by

$$\begin{aligned} C_{set-up} &= C_{jac} + C_{prec}, \\ C_{sol} &= C_{appl} + C_{mult} + C_{other}. \end{aligned}$$

If the simulation requires  $n_{nl}$  non-linear iterations, the total cost is given by

$$C_{total} = n_{nl} C_{non-lin}.$$

The above expressions are only approximations. For clarity, we have deliberately omitted some tasks such as I/O, evaluation of  $R^{(k)}$ , ... However, these tasks represent very little compared to the other ones.

The costs  $C_{xx}$  can be considered for the entire simulation or as an average over all  $n_{nl}$  non-linear iterations of the simulations. When we need to refer to a particular non-linear iteration, we will use the superscript ( $k$ ). Hence, we may also write

$$n_i = \frac{1}{n_{nl}} \sum_{k=1}^{n_{nl}} n_i^{(k)}.$$

Timings differ with the method that is used to solve the problem. When it is necessary to specify the method, a second superscript is used. For example,  $C_{sol}^{(k,or)}$  refers to the solution time of the linear system at the  $k$ th non-linear iteration, using the *original* (or Newton’s) method. Alternatively, the superscripts (*jf*), (*sham*), (*cfl*), and (*bg*) refer to the Jacobian free, Shamanskii, CFL and bi-grid methods.

---

<sup>2</sup>To be exact, the orthogonalization step in GMRES is also proportional to  $n_i$  which means that the solution cost is proportional to  $n_i^2$ . We neglect this fact.

### 3 Test cases

In the following, we consider two test cases, a bow shock and a nozzle flow problem for the solution of the ideal MHD equations:

**Nozzle flow :** We consider the simple test case of a hypersonic flow through a converging diverging nozzle (Fig. 1a). Uniform flow enters the nozzle from the left and leaves it at the right. The flow is fully superfast in the nozzle. The bottom boundary is a symmetric boundary. At the top of the nozzle we impose perfectly conducting inviscid wall boundary conditions. Initially, the following uniform flow field is prescribed: density  $\rho = 1.5$ , momentum vector  $\rho\mathbf{v} = (4, 0, 0)$ , magnetic field vector  $\mathbf{B} = (2, 0, 0)$  and total energy density  $E = 14.9$ . The grid contains 5712 cells and 3015 nodes.

**Bow shock flow :** 2D bow shock flow simulations serve as basic building blocks of hypersonic blunt body simulations in hydrodynamics and of space physics applications in MHD. This includes bow shock simulations over planets, comets or coronal mass ejections. Because of symmetry, we only simulate the flow field over the upper left quadrant of a cylinder, and the solution is symmetric with respect to the horizontal axis (Fig. 1b). Initially we impose the following uniform flow field: density  $\rho = 1$ , momentum vector  $\rho\mathbf{v} = (6, 0, 0)$ , magnetic field vector  $\mathbf{B} = (0.5, 0, 0)$ , total energy density  $E = 19.625$ . The grid contains 6152 cells and 3203 nodes.

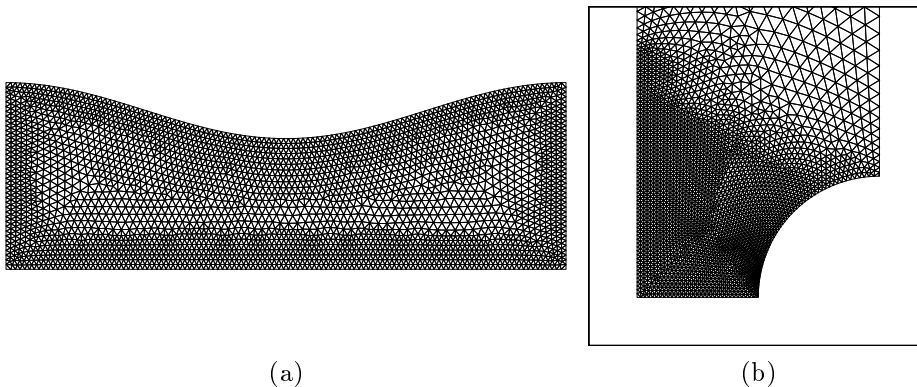


Figure 1: Grids used for the experiments with the acceleration techniques: (a) Nozzle, (b) Bow shock detail.

Simulations have been obtained using the RA code developed by rpad Csik using the residual fluctuation splitting method [6, 7, 8, 9, 10]. The linear systems are solved with a GMRES(30) using a block ILU preconditioner using a threshold  $\tau^{(k)} = 10^{-2}$ . Convergence is obtained when the norm of the residual  $\|R(U)\|_2$  is smaller than  $10^{-15}$  for the Nozzle and  $10^{-7}$  for the Bow shock problem. For both problems, the CFL number is controlled by an exponential strategy. Timing

results and the iteration counts are given in Table 1 for the two problems on a Sun Sparc Ultra. For each test case, the first column corresponds to the *total* CPU time needed, the second column gives the *average* time per non-linear iterations and the third column presents results *per linear iteration* (when they have a meaning).

	Bow shock			Nozzle		
	Total	Non-Lin	Linear	Total	Non-Lin	Linear
$n_{nl}$	137	1		20	1	
$n_i$	792	5.8	1	245	12.2	1
$C_{total}$	4134.5	30.2		697.0	34.9	
$C_{non-lin}$	4111.7	30.0		681.9	34.1	
$C_{set-up}$	3493.6	25.5		547.0	27.4	
$C_{jac}$	3133.6	22.9		490.3	24.5	
$C_{prec}$	241.2	1.8		38.3	1.9	
$C_{sol}$	380.2	2.8	0.48	95.0	4.7	0.38
$C_{mult}$	100.8	0.7	0.13	27.1	1.3	0.11
$C_{appl}$	189.0	1.4	0.24	50.5	2.5	0.21

Table 1: Original timings.

The Bow shock problem is solved in 4134s while the Nozzle problem uses 697s. It takes roughly 30s (resp. 34s) per non-linear iteration among which 25 (resp. 27s) is spent in the set-up time. Average timings for the set-up are equivalent for both problems since they have approximately the same number of unknowns. On the other hand, the average solution time per non-linear iteration is different. Indeed, the Nozzle problem requires twice as many linear iterations (12.2 versus 5.8) because the linear systems are more difficult to solve.

## 4 Acceleration Techniques

Let us assume that the Jacobian matrix is computed numerically by finite differences. For a carefully chosen step  $h$ , the  $i$ th column of  $J$  is given by

$$J(:, i) \simeq \frac{R(U + he_i) - R(U)}{h}, \quad (7)$$

where the vector  $e_i$  is the unit vector. The choice of  $h$  is sensitive. It should be small enough as to provide a meaningful approximate of the derivatives but not too small as to result in cancellation. The optimal value of  $h$  is in  $\mathcal{O}(\sqrt{\epsilon})$  where  $\epsilon$  is the precision with which the function  $R(\cdot)$  can be evaluated. If machine accuracy can be obtained, we have  $\epsilon = 10^{-16}$ . Of course, the value of  $h$  also depends on  $U_i$  and [1] suggests to use

$$h = \text{sign}(U_i) \times \sqrt{\epsilon} \times \max \left\{ |U_i|, \mu \right\},$$

where  $\mu$  is a threshold added to avoid taking values of  $h$  too close to zero.

Using (7), the evaluation of the entire Jacobian matrix of size  $n \times n$  would thus require  $n+1$  function evaluations. Of course, since the solution  $U$  is approximated by a piecewise linear functions with compact supports, it suffices to evaluate local perturbations. For example, in a two-dimensional MHD computation, the cells are defined by three nodes, each of them containing 8 variables. Hence, in every cell,  $3 \times 8 = 24$  local perturbations need to be computed. The assembly of the perturbations is similar to 24 global function evaluations.

In many practical simulations, the evaluation of the Jacobian matrix corresponds to a significant portion of the total CPU time. For the test cases (Table 1), more than 70% are taken in these evaluations. Acceleration techniques must then concentrate in ways to by-pass the Jacobian evaluations or to re-use information from previous time steps. Some of these ideas are detailed below.

## 4.1 Jacobian Free

The basic idea comes from the fact that a matrix–vector product

$$y = Jx \simeq \frac{R(U + hx) - R(U)}{h},$$

can be obtained without evaluating the Jacobian matrix  $J$  (in this case, the value of  $h$  must be scaled by  $\|x\|_\infty^{-1}$ ). Comparing the cost of a Jacobian evaluation (24 function evaluations for MHD) and of a matrix–vector product (1 function evaluation), we can directly conclude that a matrix-free implementation is advantageous if the number of linear iterations  $n_i$  does not exceed 24.

The previous statement is a little crude and deserves several comments. First, in a matrix-free method, the usual preconditioners can not be used since they need the coefficients of the matrix. One can either rely on other type of preconditioners such as multigrid or polynomial preconditioners or one can also discard the preconditioning step. If there is no preconditioner, additional computation time is saved ( $C_{prec}$  and  $C_{appl}$ ). The price to pay is that more linear iterations are needed. Let the superscript (*jf*) denote the Jacobian-free method while (*or*) stands for the original method. For a particular non-linear iteration  $k$ , if

$$C_{set-up}^{(or)} + n_i^{(k,or)} C_{sol}^{(or)} \geq C_{set-up}^{(jf)} + n_i^{(k,jf)} C_{sol}^{(jf)} \simeq n_i^{(k,jf)} C_{mult}^{(jf)} \quad (8)$$

then a matrix-free method is faster. Of course, it is difficult to estimate the number of iterations  $n_i^{(k,or)}$  and  $n_i^{(k,jf)}$  in advance.

Fortunately, it is still possible to use a matrix-free method with significant savings. To understand why, one must realize that  $n_i^{(k)}$  is an increasing function of  $k$ : A time-stepping strategy is used where the time steps grow with the non-linear iteration number. Hence, at the beginning of the iterations, the matrix  $M^{(k)}$  has large diagonal components and is well conditioned. As a result, the actual value of  $n_i^{(k)}$  is small. As the iteration proceeds, the conditioning of the matrix degrades and more linear iterations are required to solve the system. This characteristic is reinforced if an inexact Newton strategy is used that decreases  $\tau^{(k)}$  as convergence occurs. The typical evolution of  $n_i^{(k)}$  is pictured in Fig. 2 for the Bow shock problem.

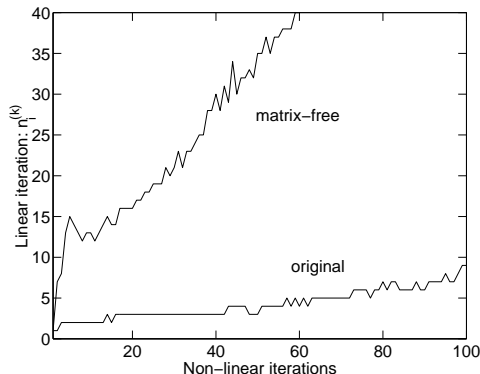


Figure 2: Evolution of  $n_i^{(k)}$  as a function of the non-linear iteration  $k$  for the Newton's method with or without a preconditioning step.

Since  $n_i^{(k)}$  is increasing, it suffices to use the matrix-free alternative as long as a gain is obtained (based on (8)) and to switch to the original method (with Jacobian evaluation). Two algorithms can be defined based on the above ideas:

**Alg 1.** : Iteration  $k = 1$  is executed by evaluating the Jacobian matrix and the set-up cost  $C_{set-up}^{(or)}$  is measured. The next iterations use a matrix-free algorithm and monitor  $C_{sol}^{(jf)}$ . When the cost of the linear system exceeds the measured set-up cost, the algorithm switches to the original method. This corresponds to (8) with  $n_i^{(k,or)}$  set to zero.

**Alg 2.** : If the equation to be solved is known, we can estimate the ratio  $C_{jac}^{(or)}$  over  $C_{mult}^{(jf)}$  (around 24 for the 2D MHD equations). Hence, we decide to stop the matrix-free method when

$$n_i^{(k,jf)} > \frac{C_{jac}^{(or)}}{C_{mult}^{(jf)}} \simeq 24.$$

In practice, a value somewhat larger than 24 may be used to account for the other terms such as  $C_{prec}^{(or)}$ , ... A value of 27 was measured experimentally.

Table 2 presents timings results obtained with the matrix-free method for the two test cases using Alg 1. The Jacobian-free method ends at iteration  $k = 5$  for the Nozzle and at iteration  $k = 40$  for the Bow shock problem. As mentioned earlier, the linear systems of the Nozzle are difficult and the Jacobian-free method can not be used very long. With the Bow shock, the time steps remain small for a long period and the Jacobian free method is effective for many iterations.

A reduction of 10% of  $C_{total}$  is obtained for both test cases. As expected, the set-up cost decreases but the solution time grows. This is due to the additional number of iterations (no preconditioning is used with the matrix-free) and to the larger cost of the matrix-vector products (0.49s versus 0.13s for the Bow shock).

	Bow shock			Nozzle		
	Total	Non-Lin	Linear	Total	Non-Lin	Linear
$n_{nl}$	137	1		19	1	
$n_i$	1313	9.6		296	15.6	1
$C_{total}$	3814.5	27.8		665.0	35.0	
$C_{non-lin}$	3791.5	27.6		649.5	34.5	
$C_{set-up}$	2647.3	19.3		436.3	22.9	
$C_{jac}$	2342.3	17.1		387.8	20.4	
$C_{prec}$	184.7	1.3		28.4	1.5	
$C_{sol}$	903.4	6.6	0.69	173.0	9.1	0.59
$C_{mult}$	641.3	4.7	0.49	109.2	5.7	0.37
$C_{appl}$	164.1	1.3	0.13	44.1	2.3	0.15

Table 2: Jacobian-free timings.

With Alg. 1, the first iteration is “wasted” to obtain an estimate of  $C_{set-up}^{(or)}$ . Alg. 2 does not have this drawback but it requires some knowledge of the PDE. Both algorithms provide better performance when the linear threshold is large. In actual implementations, it is effective to use  $\tau^{(k,jf)} \simeq 10^{-1}$ . Another experiments with this latter values showed no degradation of the non-linear convergence while the total time dropped to 3150s for the Bow shock. Indeed, fewer iterations are performed in solution of the linear systems and, therefore, the matrix-free implementation could be used until  $k = 72$ , providing an additional gain.

## 4.2 Shamanskii method

In the Newton’s method, the Jacobian matrix is evaluated at every iteration. On the other hand, the secant method evaluates the Jacobian matrix at the first iteration only and sets  $J^{(k)} = J^{(0)}$ . The Shamanskii method constitutes a trade-off where the evaluation is performed every few iterations. If  $s$  is a typical step, we have

$$J^{(k)} = \begin{cases} \frac{\partial R}{\partial U}(U^{(k)}) & \text{if } \text{mod}(k, s) = 0 \\ J^{(k-1)} & \text{otherwise.} \end{cases}$$

Therefore, the Jacobian is evaluated at iterations 0,  $s$ ,  $2s$ , ... On average we have then

$$C_{jac}^{(sham)} = \frac{1}{s} C_{jac}^{(or)}.$$

An additional advantage with this method is that the linear systems are solved with multiple right-hand-sides. In the context of Krylov subspaces methods, it is thus possible to use spectral information from previous subspaces to solve the current system thus reducing  $C_{sol}$ .

With the Shamanskii method, the problem lies in the determination of  $s$ . Using a large value of  $s$  reduces the Jacobian cost but increases the total number of non-linear iterations since the asymptotic convergence rate is smaller. In other cases, using previous Jacobian matrices does not provide effective descent directions and the iteration may result in a breakdown.

Table 3 presents performance results when the Jacobian matrix is evaluated every two iterations ( $s = 2$ ). For the Nozzle problem, almost the same number of non-linear iterations is needed. This provides a reduction of 40% in the total time. The solution of the Bow shock problem could not be obtained with the current values of the CFL strategy. With a “slower” strategy, the Bow shock problem requires 25% more non-linear iterations. However, since the average set-up time is halved compared to the original method, the Shamanskii method still provides a 20% reduction.

	Bow shock			Nozzle		
	Total	Non-Lin	Linear	Total	Non-Lin	Linear
$n_{nl}$	172	1		21	1	
$n_i$	889	5.2	1	246	11.7	1
$C_{total}$	3251.8	18.9		427.7	27.2	
$C_{non-lin}$	3228.5	18.7		411.6	23.9	
$C_{set-up}$	2427.7	14.1		281.2	17.0	
$C_{jac}$	2106.3	12.2		243.4	14.9	
$C_{prec}$	159.54	0.9		19.9	0.9	
$C_{sol}$	478.0	2.8	0.53	94.7	4.5	0.39
$C_{mult}$	124.05	0.7	0.14	27.5	1.3	0.11
$C_{appl}$	242.04	1.4	0.27	51.0	2.4	0.21

Table 3: Timings obtained with the Shamanskii method ( $s = 2$ ).

The Shamanskii method is a powerful alternative to the pure Newton’s method but it may suffer from slow convergence or even breakdown of the method if the successive Jacobian matrices are “far” from each other. A robust method would thus choose *dynamically* to recompute the Jacobian matrices. For example, if the iterate is close to the solution, it can be advantageous to switch to a Newton’s method (how to measure the distance to a solution is discussed in [11]). Similarly, if the new direction  $\Delta U^{(k, sham)} = M^{(k-1)^{-1}} R^{(k)}$  is not a descent direction, we may choose to discard it and compute  $\Delta U^{(k, or)} = M^{(k)^{-1}} R^{(k)}$ .

### 4.3 Optimal CFL number

In the above test cases, the CFL number is controlled by an exponential law. This choice is quite satisfactory for the Nozzle flow but it is severely sub-optimal for the Bow shock problem. Indeed, small CFL numbers must be taken as long as the shock is not correctly placed which limits the growth of the exponential. Consequently, at the end of the computation, the value of the CFL remains relatively small and super-linear convergence of Newton’s method does not appear.

By using a better strategy for the determination of the CFL number, almost 10% reduction in CPU time can be achieved. We refer the interested reader to [11] for more details and present only the main results in Table 4.

For the Bow shock problem, a 15% improvement of the number of non-linear iterations is achieved. Of course, the computation cost per iteration remains

	Bow shock			Nozzle		
	Total	Non-Linear	Linear	Total	Non-Linear	Linear
$n_{nl}$	118	1		19	1	
$n_i$	669	5.7	1	225	11.8	1
$C_{total}$	3883.7	32.9		772.0	38.7	
$C_{non-lin}$	3860.4	32.7		756.8	39.8	
$C_{set-up}$	3261.5	27.6		620.0	32.6	
$C_{jac}$	2923.2	24.7		561.9	29.6	
$C_{prec}$	226.9	1.9		35.8	1.9	
$C_{sol}$	367.8	3.1	0.55	90.9	4.8	0.40
$C_{mult}$	94.5	0.8	0.14	25.7	1.4	0.12
$C_{appl}$	191.2	1.6	0.29	49.2	2.6	0.22

Table 4: Timings obtained with the optimal control of the CFL number.

identical to Table 1. For the Nozzle problem, no improvement is obtained because the exponential strategy already allows to take large time steps early in the iteration process.

#### 4.4 Bi-grid initial guess

Initial guesses (a uniform flow field) are usually far from the solution and many iterations are then needed before reaching the steady-state flow. If the transient states are not of interest, these iterations constitute a waste of time. Hence, it proved advantageous to choose a better initial guess computed on a coarser grid and projected on the original grid. The cost of computing the initial guess is not trivial but it remains modest for several reasons: (a) the problem size on the coarser grid is much smaller than the size of the original problem, (b) little accuracy of the coarse solution is sufficient. Finally, since  $U^{(0)}$  is only a guess, a simple linear interpolation can be used in the projection step. An additional bonus comes when a good guess is used: the CFL strategy can be adapted to provide faster convergence. Indeed, even if an exponential law is considered, its growth can be much larger since the iterates are already close to the solution.

The test cases have been solved using a better initial guess. The coarse nozzle contains 1912 cells and the coarse bow-shock has 1363 cells. For both problems, a threshold  $\|R^{(k)}\|_2 \leq 10^{-2}\|R^{(0)}\|_2$  was set. It took 9 iterations and 91s. for the Nozzle and 45 iterations (238 s.) for the Bow shock to obtain the initial guess. Table 5 presents the timing results. It can be observed that the length of the initial phase is effectively shortened and that the number of non-linear iterations is reduced by 40%. As a result, the total cost (including the evaluation of the initial guess) decreases by 30% compared to the original performance.

Although this method seems powerful, it is more cumbersome to use it effectively. Two grids are needed, many parameters need to be set. Also, it is not applicable if the transient phenomena need to be captured.

It does not appear that the choice of the coarse grid is very important. Similar experiments have been conducted with other grids, or using recursively the idea

	Bow shock			Nozzle		
	Total	Non-Lin	Linear	Total	Non-Lin	Linear
$n_{nl}$	80	1		14	1	
$n_i$	592	7.4	1	204	14.6	1
$C_{total}$	2744.7	31.8		554.0	38.7	
$C_{non-lin}$	2524.1	31.6		448.8	31.4	
$C_{set-up}$	2114.6	26.4		348.0	24.9	
$C_{jac}$	1884.4	23.5		308.7	22.1	
$C_{prec}$	159.1	2.0		27.5	2.0	
$C_{sol}$	266.8	3.3	0.45	77.1	5.6	0.37
$C_{mult}$	72.5	0.9	0.12	22.6	1.6	0.11
$C_{appl}$	139.4	1.7	0.24	41.5	3.0	0.20

Table 5: Timings obtained using a initial guess projected from a coarse grid.

from a very coarse grid to the finest grid with multiple projection steps. Performance were not significantly better. Our interpretation is that the overall “shape” of the flow field matters and not the very tiny details. To this end, multiple grids do not help while very coarse grids are sufficient.

## 5 Discussion and Conclusion

In the previous section, we have discussed several ideas to speed-up the computation time of the Newton’s algorithm. Of course, it is possible to *combine* some of the techniques. For example, the CFL strategy is independant of the other methods. On the other hand, the Jacobian free and bi-grid methods are exclusive. Indeed, the matrix-free methods works best when small time steps are taken but starting from a better initial guess enables to take large time steps directly.

For the Nozzle problem, we have combined the bi-grid and the Shamanskii method with an optimal CFL while the initial guess is obtained with a Jabobian free method on the coarser grid. Obtaining the initial guess requires now 76s and the total computation time drops to 340s. This amounts to a 50% reduction of the total time.

The same strategy has been applied to the Bow shock problem. The initial guess is obtained in 172s and 50 iterations are needed to reach the steady-state solution of the fine problem. The total time of 1115s is decreased by a factor 3.7 compared to the Newton’s method.

In summary, we have experimented several methods to speed up the convergence process of Newton’s method. The jacobian free method does not require the computation of the Jacobian but works only when the linear systems are well conditioned. However, if memory is a limiting factor, a full Jacobian-free method can be applied, never requiring to allocate space to store the matrix. Of course, for an effective implementation, specific matrix-free preconditioners must be devised. This is is actually an open question but promising approaches have been conducted towards polynomial and multigrid preconditioners.

If the successive Jacobian matrices are “close” to each other, a Shamanskii method provides a significant and effortless gain. When robustness is an issue, an alternative approach, similar to the *trust region* methods, is to rank-one update the Jacobian from the previous time step instead keeping it identical.

Finally, for steady state flows, an optimal CFL strategy or a good initial guess enable a substantial reduction in the number of non-linear iterations.

*The main conclusion of this work is that although the Newton’s method is appealing from a mathematical point-of-view, actual simulation codes should refrain from using it blindly. Computing a new matrix and solving a linear system are costly operations that should be performed only when necessary.*

## Acknowledgements

The author is grateful to Árpád Csík for providing access to the RA code for the numerical solution of MHD equations as well as for many interesting discussions.

This work is funded by the *Fonds voor Wetenschappelijk Onderzoek* (FWO project G0344.98: “Multidimensional upwinding and parallel implicit solvers for MHD”) and by the UIAP P4/02 Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister’s Office for Science Technology and Culture. The scientific responsibility rests with its author.

## References

- [1] J. E. Dennis and R. B. Schnabel Jr. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall Inc., Englewood Cliffs, New-Jersey, 1983.
- [2] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations*. Frontiers in Applied Mathematics, SIAM, 1995.
- [3] C. T. Kelley and D. E. Keyes. Convergence analysis of pseudo-transient continuation. *SIAM J. Num. Anal.*, Vol. 35, No. 2, pp. 508–523, 1998.
- [4] R. Barrett *et al.* *Templates for the Solution of Linear Systems: Building Blocks for Iterative Solvers*. SIAM, Philadelphia, 1994.
- [5] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, 1998.
- [6] P. L. Roe. Fluctuations and signals – a framework for numerical evolution problems. In *Numerical Methods for Fluid Dynamics*. Academic Press, 1982.
- [7] Á. Csík, H. Deconinck, and S. Poedts. Monotone residual distribution schemes for the ideal 2D magnetohydrodynamic equations on unstructured grids. In *14th AIAA Conference*, pp. 99–3325, 1999.

- [8] Á. Csík, H. Deconinck, and S. Poedts. On the shock capturing properties of the non-conservative symmetrizable form of the ideal magnetohydrodynamic equations. *Journal of Computational Physics*, 2000 (submitted).
- [9] H. Paillère, H. Deconinck, and P. L. Roe. Conservative upwind residual distribution schemes based on the steady characteristics of the Euler equations. *AIAA*, Vol. CP-951700, pp. 592, 1995.
- [10] E. van der Weide, E. Issman, H. Deconinck, and G. Degrez. Parallel, implicit, multi-dimensional upwind, residual distribution method for the Navier-Stokes equations on unstructured grids. *Computational Mechanics*, Vol. 23, No. 2, pp. 199–208, 1999.
- [11] D. Vanderstraeten and Á. Csík. An expert system to control the CFL number of implicit upwind methods. *Int. J. Num. Meth. Fluids*, 2000 (submitted).