Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational Physics 189 (2003) 130–158

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

An incomplete assembly with thresholding algorithm for systems of reaction–diffusion equations in three space dimensions IAT for reaction–diffusion systems ☆

Peter K. Moore *

Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA

Received 11 July 2002; received in revised form 14 March 2003; accepted 21 March 2003

Abstract

Solving systems of reaction–diffusion equations in three space dimensions can be prohibitively expensive both in terms of storage and CPU time. Herein, I present a new incomplete assembly procedure that is designed to reduce storage requirements. Incomplete assembly is analogous to incomplete factorization in that only a fixed number of nonzero entries are stored per row and a drop tolerance is used to discard small values. The algorithm is incorporated in a finite element method-of-lines code and tested on a set of reaction–diffusion systems. The effect of incomplete assembly on CPU time and storage and on the performance of the temporal integrator DASPK, algebraic solver GMRES and preconditioner ILUT is studied.

AMS: 65M20; 65M60; 92E20

1. Introduction

General purpose codes for solving systems of reaction–diffusion equations in three space dimensions using the method-of-lines in time coupled with the finite element method in space (FEMOL) require significant computational resources. Most of these resources are spent in solving large sparse linear systems using iterative methods. These linear systems arise from a Newton or modified-Newton solver within the temporal integrator. The same is true whether uniform or spatially adaptive grids are used. The efficiency of these methods for solving such linear systems can be measured temporally (in terms of CPU time) and spatially (in terms of CPU storage). A trade-off often exists between the two. For many FEMOL codes that use iterative methods such as GMRES [21] or QMR [13] to solve the linear systems, three components play

☆ This research was partially supported by NSF Grant #DMS-0196108.
* Tel.: +214-768-3369; fax: +214-369-2355.
E-mail address: pmoore@mail.smu.edu.

0021-9991/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0021-9991(03)00205-5

a critical role in determining their effectiveness: (i) the assembly and possible storage of a large, sparse Jacobian matrix; (ii) the construction and storage of a preconditioner (e.g., an incomplete factorization of the Jacobian from (i)) and; (iii) the computation or approximation of matrix–vector products in the iterative solver where the matrix is the Jacobian.

Herein, I focus on improving (i), the assembly and storage of the Jacobian matrix, although these changes will impact (ii) and (iii). For iterative methods the Jacobian is used only in computing matrix–vector products and in constructing a preconditioner so only its nonzero entries are needed. This suggests that the standard finite element full assembly (FA) algorithm can be replaced by one that computes an "incomplete" assembly. Incomplete assembly is analogous to incomplete factorization in that assembly criteria mimic criteria used by, for example, the ILUT preconditioner of Saad [20]. Thus, only a fixed number *ifil* of nonzero entries are stored per row and a drop tolerance *itol* is used to discard small values. The resulting procedure is referred to as the incomplete assembly with thresholding (IAT) algorithm (cf. Section 3.2). The preconditioner is then calculated from this incomplete Jacobian.

Efficient storage of the (full or incomplete) Jacobian involves special data structures [22]. The standard FA algorithm computes local Jacobian matrices on each element and adds them into the Jacobian. In a general-purpose code for solving time-dependent reaction–diffusion systems these data structures are based on the mathematical structure of the equations and the spatial discretization. They are therefore created before any entries in the matrix have been computed (cf. Section 3.1). Thus, the data structures depend on the form of the equations, the spatial grid and the order of the elements but not on the matrix entries themselves. This approach may result in the storage of a sizable number of zeros. For example, if the code handles both Neumann and Dirichlet boundary conditions then either extraneous zeros are stored when Dirichlet boundary conditions are present or the assembly algorithm is more complicated. Zero entries may also arise if the grid is spatially adapted [18]. Additionally there may be many small (in absolute value) entries whose locations change as the solution changes. The IAT procedure, with its data structures calculated during assembly, is designed to take advantage of these small entries.

One drawback of this strategy is that, in effect, two matrices must be stored, the Jacobian for the matrix–vector products and the preconditioner. This differs from the approach taken in DASPK. DASPK is a so-called "matrix-free" algorithm [3–5] in that only the preconditioner is stored. Matrix-vector products are approximated by a directional derivative involving additional function evaluations. Of course since preconditioners are often based on incomplete factorizations of the Jacobian some assembly of the Jacobian is typically required. In [17] it was demonstrated that though this approach requires less storage it is significantly slower than a code that uses the Jacobian explicitly to calculate the matrix–vector products. For this reason a version of DASPK that calculates these products using the Jacobian explicitly is used. Numerous other strategies have been suggested to improve the efficiency of method-of-lines algorithms for solving reaction–diffusion systems. These include implicit/explicit methods [1,2,16,19], mass lumping [6,24,25] (note the limitations of lumping for nonlinear problems in the third reference) and alternating direction-finite difference methods [7,8]. Comparison with these alternatives is beyond the scope of this study.

The aim of this study is to determine the impact of the incomplete assembly process on FEMOL codes. Several factors are considered including storage, CPU time and solution accuracy. Performance of temporal integrators is typically measured by the number of time steps, Jacobian assemblies and function evaluations [10,17]. Since the preconditioner may be effected by the incomplete Jacobian, its size and the number of iterations are also examined. In order to focus on the behavior of the IAT procedure attention is limited to the *h*-refinement FEMOL code of Moore [18], herein called FEDAS. In this initial effort the code is run in nonadaptive mode. FEDAS uses the time integrator DASPK [5] which, in turn, uses the iterative solver, GMRES with restarts [21]. I also restrict attention to one preconditioner, ILUT [20]. As noted, I have modified DASPK so that the matrix–vector products are computed directly using the Jacobian. The two user-selected parameters of ILUT, *pfil* (number of nonzero entries per row of L and U) and *ptol* (drop tolerance) are kept fixed (cf. Section 4). The insights gained from this study should prove useful for other time integrators, iterative methods and preconditioners.

The effectiveness of the IAT algorithm is examined for three values of *itol* (*ifil* is kept fixed, cf. Section 4) and six finite element discretizations on a test set of reaction–diffusion systems. The test set includes the Cahn–Allen equation [11], a combustion system [15], the Brusselator model of the Belosouv–Zhabotinsky reaction [14] and a Fitzhugh–Nagumo model of nerve conduction [9]. This set is similar to one proposed by Estep et al. [12]. Finally, I have included a linear heat-conduction problem whose exact solution is known as a benchmark problem.

The reaction–diffusion equations in the test set are of the form

$$\mathbf{u}_t = \nabla \cdot \mathbf{D}(\mathbf{u}) + \mathbf{R}(\mathbf{u}), \tag{1}$$

$$\mathbf{D}(\mathbf{u}) = (\mathbf{D}_1\mathbf{u}_x, \mathbf{D}_2\mathbf{u}_y, \mathbf{D}_3\mathbf{u}_z), \tag{2}$$

$$\mathbf{x} = (x,y,z) \in \Omega \equiv [X_0,X_1] \times [Y_0,Y_1] \times [Z_0,Z_1], \quad t \in (0,t_f], \tag{3}$$

where $\mathbf{u}$ is a vector of length M and $\mathbf{D}_i$, $i = 1,2,3$ are $M \times M$ positive definite, constant, diagonal matrices, together with the initial conditions

$$\mathbf{u}(\mathbf{x},0) = \mathbf{u}_0(\mathbf{x}), \quad \mathbf{x} \in \Omega. \tag{4}$$

Dirichlet or Neumann boundary conditions are applied on each of the six boundary faces $\partial\Omega_j$, $j = 1,2,\ldots,6$, for $\mathbf{x} \in \partial\Omega_j$ and for each unknown $i = 1,2,\ldots,M$. Additional restrictions must be placed on the functions $\mathbf{R}$ to ensure that (1)–(4) is a well-posed parabolic system with locally isolated solutions.

The finite element Galerkin method in FEDAS uses a piecewise polynomial hierarchical spatial basis of degree $p \geqslant 1$ to solve (1)–(4) (cf. Section 2). Dirichlet boundary conditions are treated as algebraic equations. The resulting system of differential-algebraic equations (DAEs) is integrated in time using the BDF code DASPK [5] as outlined in Section 2. In Section 3, I describe the full and incomplete assembly procedures. Detailed descriptions of the problems in the reaction–diffusion test set are given in Section 4. The effects of using incomplete assembly in FEDAS on storage, CPU time, accuracy and on the performance of DASPK and GMRES as functions of the reaction–diffusion system, the finite element discretization and *itol* are presented in Section 4. In Section 5, I conclude with some observations.

2. Discretization

The FEMOL code FEDAS [18] discretizes in space using the finite element method and integrates the resulting differential system in time using DASPK. DASPK uses a modified-Newton method with GMRES as the linear system solver. As a preconditioner I am using ILUT.

The Galerkin form of (1)–(4) together with, for simplicity, Dirichlet boundary conditions, consists of determining $\mathbf{u}(\mathbf{x},t) \in H_E^1(\Omega) \times (t > 0)$ such that

$$\begin{aligned} &(\mathbf{u}_t,\mathbf{v}) - (\mathbf{R},\mathbf{v}) + (\mathbf{D},\nabla\mathbf{v}) = 0 \quad \forall \mathbf{v} \in H_0^1, \ t > 0, \\ &(\mathbf{u},\mathbf{v}) = (\mathbf{u}_0,\mathbf{v}) \quad \forall \mathbf{v} \in H_0^1, \ t = 0, \end{aligned} \tag{5}$$

where

$$(\mathbf{u},\mathbf{v}) = \int_\Omega \mathbf{u}^{\mathrm{T}}\mathbf{v}\,\mathrm{d}\mathbf{x}. \tag{6}$$

As usual, the Sobolev space $H^1(\Omega)$ consists of functions having square integrable first derivatives. The subscripts E and 0 further restrict functions to satisfy the Dirichlet boundary conditions and homogeneous versions thereof, respectively.

Introduce partitions

$$\begin{aligned}&\Omega_x \equiv \{X_0 = x_0 < x_1 < \cdots < x_N = X_1\}, \quad \Omega_y \equiv \{Y_0 = y_0 < y_1 < \cdots < y_N = Y_1\},\\ &\Omega_z \equiv \{Z_0 = z_0 < z_1 < \cdots < z_N = Z_1\},\end{aligned} \tag{7}$$

to divide Ω into $N_{\text{elem}} \equiv N^3$ elements thus, forming a grid $\Delta_\Omega = \Omega_x \times \Omega_y \times \Omega_z$ and approximate $H^1(\Omega)$ by a finite-dimensional subspace $S^{\Delta_\Omega,p}$ of piecewise polynomials of degree p, $1 \leqslant p \leqslant 6$. Consider the finite element approximation $\mathbf{U}(\mathbf{x},t) \in S_E^{\Delta_\Omega,p}$ of $\mathbf{u}(\mathbf{x},t) \in H_E^1$ having the form

$$\mathbf{U}(\mathbf{x},t) = \sum_{i=1}^{N_{\text{node}}} N_i(\mathbf{x})\mathbf{U}_i^N(t) + \sum_{i=1}^{N_{\text{edge}}}\sum_{j=2}^{p} E_{i,j}(\mathbf{x})\mathbf{U}_{i,j}^E(t) + \sum_{i=1}^{N_{\text{face}}}\sum_{j=4}^{p}\sum_{k=2}^{j-2} F_{i,j,k}(\mathbf{x})\mathbf{U}_{i,j,k}^F(t), \tag{8}$$

where N_{node}, N_{edge} and N_{face} are the number of nodes, edges and faces, respectively. The nodes, edges, faces and elements of Δ_Ω are referred to as modes. The nodal basis functions $N_i(\mathbf{x})$ are the standard trilinear basis functions. The edge- and face-based basis functions, $E_{i,j}(\mathbf{x})$ and $F_{i,j,k}(\mathbf{x})$, respectively, are given in [23]. Together with the nodal basis functions they comprise a hierarchical basis for $S^{\Delta_\Omega,p}$.

I approximate $\mathbf{v}$ by $\mathbf{V} \in S_0^{\Delta_\Omega,p}$ in a similar manner. Then replacing $\mathbf{u}$ and $\mathbf{v}$ in (5) by $\mathbf{U}$ and $\mathbf{V}$, $\mathbf{U}(\mathbf{x},t)$ is the solution of the differential system

$$\begin{aligned}&(\mathbf{U}_t,\mathbf{V}) - (\mathbf{R},\mathbf{V}) + (\mathbf{D},\nabla\mathbf{V}) = 0 \quad \forall\mathbf{V} \in S_0^{\Delta_\Omega,p}, \ t > 0,\\ &(\mathbf{U},\mathbf{V}) = (\mathbf{u}_0,\mathbf{V}) \quad \forall\mathbf{V} \in S_0^{\Delta_\Omega,p}, \ t = 0,\end{aligned} \tag{9}$$

together with the algebraic equations arising from interpolating the Dirichlet boundary conditions. Integrals in (9) are approximated by using tensor-product formulas of one-dimensional Gauss-quadrature rules.

Explicit utilization of (8) reveals that (9) has the form

$$\mathbf{g}(\dot{\mathscr{U}},\mathscr{U},t) = 0, \tag{10}$$

together with initial conditions where $\mathscr{U}$ is a vector of the Galerkin coordinates $\mathbf{U}_i^N$, $\mathbf{U}_{i,j}^E$ and $\mathbf{U}_{i,j,k}^F$ (in x–y–z-order from (X_0,Y_0,Z_0) to (X_1,Y_1,Z_1)). I integrate (10) using DASPK [5]. DASPK uses GMRES [21] to solve the linear systems arising from the modified-Newton method. A typical modified-Newton step in DASPK has the form

$$\mathbf{J}\Delta\mathscr{U}^m = -c\mathbf{g}(\alpha\mathscr{U}^m + \mathscr{H},\mathscr{U}^m,t), \tag{11}$$

where

$$\mathbf{J} = \alpha\frac{\partial\mathbf{g}}{\partial\dot{\mathscr{U}}} + \frac{\partial\mathbf{g}}{\partial\mathscr{U}} \tag{12}$$

is the Jacobian matrix, $\mathscr{H}$ is the history vector, and α is a parameter that depends on the step size and order of the method. The parameter c allows the same Jacobian to be used over several time steps of different sizes provided the difference is not too large [3].

To accelerate the convergence of GMRES I use the incomplete LU factorization with thresholding (ILUT) algorithm of Saad [20]. This preconditioner is controlled by two parameters, a drop tolerance *ptol* ("small" elements are dropped) and a maximum row fill-in *pfil* for both L and U [20,22].

3. Assembly procedures

The FA and IAT assembly algorithms are incorporated in FEDAS. Both algorithms take advantage of the data structures used by the adaptive code. Three basic data structures are employed in FEDAS, an octree for storing the grid, mode-connectivity trees (recall that a mode is an element, face, edge or node) for storing information about the elements and lists of modes. The grid, Δ_Ω is obtained by recursive bisection in each direction beginning with Ω. Thus Δ_Ω is an octree with Ω as the root. The mode-connectivity trees are a series of one-way pointers from elements (the roots) to their faces, from faces to their edges and finally from edges to their nodes. Separate lists of elements, faces, edges and nodes form the third basic data structure.

3.1. Full finite element assembly

The choice of grid Δ_Ω, basis functions N_i, $E_{i,j}$ and $F_{i,j,k}$ and structural form of $\mathbf{R}$ and $\mathbf{D}$ determine that most of the elements of $\mathbf{J}$ are zero. These zeros entries are referred to as the structural zeros of $\mathbf{J}$ while the remaining entries are said to be structurally nonzero and are stored in compressed sparse row (CSR) format [22]. In addition to the storage for $\mathbf{J}$, two integer arrays, IA and JA, are used to store row and column information, respectively. The arrays IA and JA are computed before any assembly of $\mathbf{J}$ begins. There are several reasons for doing this. First IA and JA can be constructed once before time integration begins. Second, since IA and JA are already computed it is easy to assemble the local element matrices into $\mathbf{J}$. Assembling element-by-element minimizes the number of times certain computations, such as evaluating the solution at the quadrature points on each element, must be performed. Thus, IA and JA are determined by the structure of the equations and the grid, including the basis order on each element and not by the actual values of $\mathbf{J}$. The standard finite element procedure (FA) used herein for assembling $\mathbf{J}$ takes place in two steps. The first step involves calculating the portion of $\mathbf{J}$ associated with each element and assembling them element-by-element. After this step the boundary conditions are taken into account. This two-step process significantly reduces the complexity of the code, especially if spatial adaptivity is implemented (as is done in FEDAS). For higher-order elements some of the structural nonzero entries of $\mathbf{J}$ are small due to interactions between basis functions of different degrees and the partial differential equations being solved. The location of these small entries may change with changes in the solution. Additionally, Dirichlet boundary conditions, imposed in the second step of assembly may also result in a sizable number of zero entries present in the structural nonzero portion of the Jacobian. Since IA and JA are precomputed these entries are stored.

3.2. Incomplete assembly with thresholding

An alternative approach is to use incomplete assembly with thresholding. There are significant differences between the IAT and the FA assembly procedures. Although the same CSR format is used to store the Jacobian, the sparse storage vectors IA and JA are computed dynamically as the matrix is assembled. Each time a new Jacobian is needed IA and JA are regenerated; thus, they change as the solution changes. Dynamic assembly is accomplished by assembling the matrix mode-by-mode rather than element-by-element.

Mode-by-mode assembly requires few changes to the element-by-element code used in FA. Four additional data structures are needed. One is a linked list of the finite element modes in order (the natural ordering described in Section 2 is chosen). The other three are lists that link modes (one each for faces, edges and nodes) to an element to which they belong. These lists are computed before integrating in time. For each mode m, neighbor information from the octree is then used to determine the remaining elements that contain it. Assembly is done over these elements with the test functions restricted to the current mode.

The result is an $r \times c$ matrix $\mathbf{J}^m$ where r is equal to the number of basis functions associated with the mode m times M. After assembling $\mathbf{J}^m$ each of its rows $\mathbf{J}^m(i, 1:c)$, i, $i = 1, 2, \ldots, r$ is processed as follows. The elements of the row are sorted by size as in ILUT [20]. The first *ifil* entries in the sorted row that satisfy

$$\frac{|\mathbf{J}^m(i,j)|}{|\mathbf{J}^m(i,1:c)|_\infty} > itol, \tag{13}$$

are assembled into $\mathbf{J}$, and IA and JA are updated appropriately.

The cost, in CPU time, of an incomplete assembly is more expensive than a full assembly for three reasons. First the incomplete algorithm requires that IA and JA must be recomputed every time a new Jacobian is needed. To determine the appropriate elements to keep every entry in the full assembly Jacobian must be computed. Finally mode-by-mode assembly involves redundant calculations. However, assembly cost for IAT is independent of *itol*. Cost savings come in computing the matrix–vector products, where CPU time decreases as *itol* increases, and, possibly, in obtaining and using the preconditioner.

4. Reaction–diffusion test set and results

Computational results for five problems are used to study the effect of *itol* on the performance of FEDAS. In addition to the FA algorithm, the IAT algorithm with three values of *itol* is considered: 10^{-10} (IAT10); 10^{-7} (IAT7) and 10^{-4} (IAT4). The corresponding codes are referred to as FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4, respectively, while FEDAS–IAT refers more generally to any of the latter three. In all cases *ifil* = 20,000, *pfil* = 100, $ptol = 10^{-4}$. With these parameter values all of the preconditioners in the examples below (not surprisingly) use less than 100% of the space available. In most cases the storage for the preconditioner ranges between 85% and 95% of the storage available depending on the problem and discretization. The absolute and relative error tolerances for DASPK are kept fixed at 10^{-6}. The maximum allowable dimension of the Krylov space for GMRES in DASPK is set at 10. GMRES restarts only occur when $p = 6$ and are only significant in Example 5. Thus, in almost all cases fewer than 10 iterations per Newton step are needed. The examples are solved using uniform grids with $p = 4, 5, 6$ and $N = 8, 16$. All computations are performed in double precision on a Compaq Alphastation 667 MHz DS20 with 4GB RAM.

Determining the impact of IAT on FEDAS is complicated by the relationship between *itol*, GMRES and DASPK. DASPK does not depend continuously on *itol* since small changes in the latter may lead to dramatic changes in the former. This is partly due to the order and time step selection strategies in DASPK, e.g., method orders must be integers and time steps increase only by doubling, and partly on the linear and nonlinear convergence criteria. This, in turn, effects the number of Jacobian assemblies, function evaluations and the number of time steps. Nevertheless, as is demonstrated below, some general trends can be observed.

For each example the size of the preconditioner and the Jacobian is studied as a function of t and as a function of the assembly procedure. The effect of the assembly procedures on the CPU time and the solution accuracy is also considered. Since for all but the first problem the true solution is not known the difference between the solution obtained using the full assembly and each of the incomplete assembly routines is examined. Solutions at a fixed time obtained by FEDAS–FA with $p = 4$ and $N = 16$ serve to highlight the spatial complexity of the nonlinear problems. The impact of IAT on DASPK is investigated in the typical way [10,17] by measuring the number of Jacobian assemblies, function evaluations and time steps used by the temporal integrator. Finally, I examine the total number of GMRES iterations, the number of GMRES iterations per time step and the number of GMRES restarts to determine IAT's effect on the iterative solver.

Example 1. Consider the linear heat conduction problem

$$u_t = \Delta u + R(\mathbf{x}, t), \quad \mathbf{x} \in \Omega \equiv [0,1]^3, \ t > 0, \tag{14}$$

where $R(\mathbf{x}, t)$ and the initial and Dirichlet boundary conditions are chosen so that the solution is

$$u(\mathbf{x}, t) = \tanh(5(x + y + z - t - 0.5)). \tag{15}$$

Although (14) is not a reaction–diffusion equation it serves as a benchmark since the exact solution is known. I solve (14) on $0 < t \leqslant 4$. The solution approximates a planar wave that propagates across Ω from $(0,0,0)$ to $(1,1,1)$ leaving at $t \approx 3$. Thus, from $3 \leqslant t \leqslant 4$, $u \approx -1$ throughout Ω.

In Fig. 1 the CPU time as a function of t is plotted for the discretizations $p = 6$ and $N = 8, 16$ (the other discretizations are comparable). As in several of the examples considerable time is consumed by DASPK in its initial phase as it selects the correct order and stepsize. During this phase the stepsize and order are typically changing at every step. Thus, a new Jacobian and preconditioner must be computed at each step resulting in a significant cost in CPU time, especially when using IAT. After the initial phase the CPU time grows slowly except when a new Jacobian and preconditioner are needed when there is a jump in the CPU time. Clearly in this example the cost of assembly and factorization of the Jacobian represents a sizable portion of the total CPU time. For each discretization the number of Jacobian assemblies is the same regardless of the assembly procedure (the performance of DASPK is essentially independent of assembly method for this example). The difference in factorization time between FEDAS–FA and FEDAS–IAT7 is less than 0.2% of the total time when $p = 6$ and $N = 16$ suggesting that when the preconditioners are

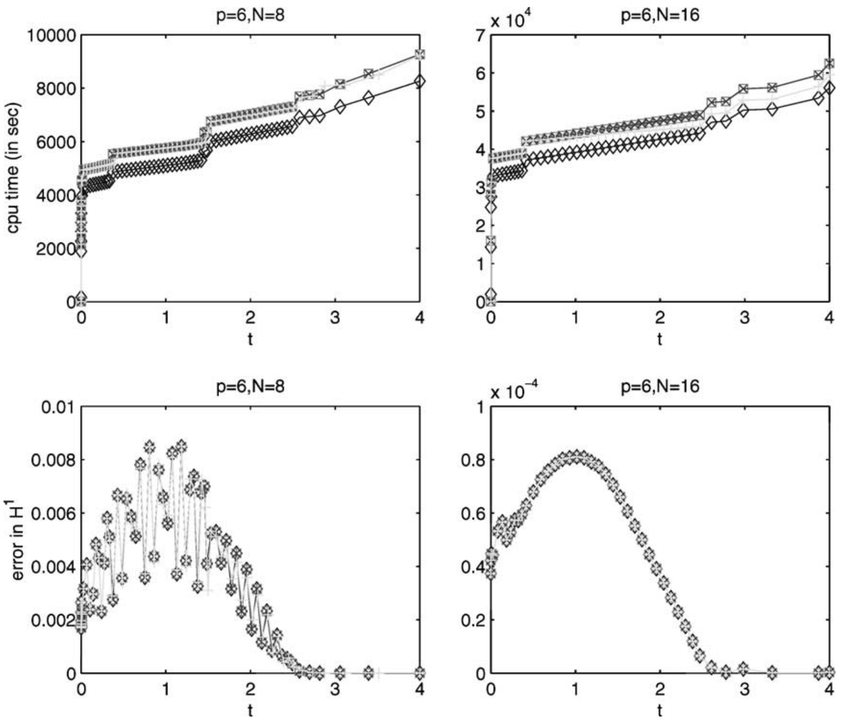


Fig. 1. The CPU time (above) and error in H^1 (below) in solving Example 1 with $p = 6$ and $N = 8, 16$ using FEDAS–FA ($\Diamond$), FEDAS–IAT10 ($\Box$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$).

comparable in size they are also comparable in cost. Thus, since FA is the cheapest, per assembly, FEDAS–FA has the fastest time. The gap in CPU time between FEDAS–FA and FEDAS–IAT closes as p and N increase. On the coarsest grid FEDAS–FA is 26% faster than FEDAS–IAT for all *itol* while on the finest grid FEDAS–FA is 11% faster than FEDAS–IAT10 and FEDAS–IAT7 and only 6% faster than FEDAS–IAT4.

The errors in H^1 are also shown in Fig. 1 for the same discretizations. Once the "wave" has left the region the errors decrease dramatically. For all p when $N = 8$ the errors exhibited the highly oscillatory behavior observed in Fig. 1. This is due to inadequate spatial discretization, i.e., the same behavior is observed in the interpolant error as a function of t (the time step behavior is smooth). What is important is that incomplete assembly has almost no impact on the error.

In Fig. 2 the sizes of the Jacobian and the preconditioner are shown for all discretizations. When $p > 4$ more space is used in storing the FA Jacobian than is used in storing its preconditioner. On the other hand the Jacobians produced by IAT4, IAT7 and IAT10 are smaller than their respective preconditioners. Thus, incomplete assembly leads to a significant improvement in storage efficiency.

With the exception of FEDAS–IAT4 the changing nature of the solution is not reflected in either the size of the preconditioner or the Jacobian, even after the wave has left the domain. The Jacobians produced by FEDAS–IAT4 experience a modest decrease in size when $t > 3$. The reason is as follows. Although the mass and stiffness matrices are fixed throughout the computation the entries in the Jacobian depend on the time step via (12). When $t > 3$ the time step increases making the stiffness matrix a more important component of the Jacobian. Since the stiffness matrix has a larger number of smaller (in absolute value) values than the mass matrix the storage decreases. This decrease shows up first in FEDAS–IAT4.

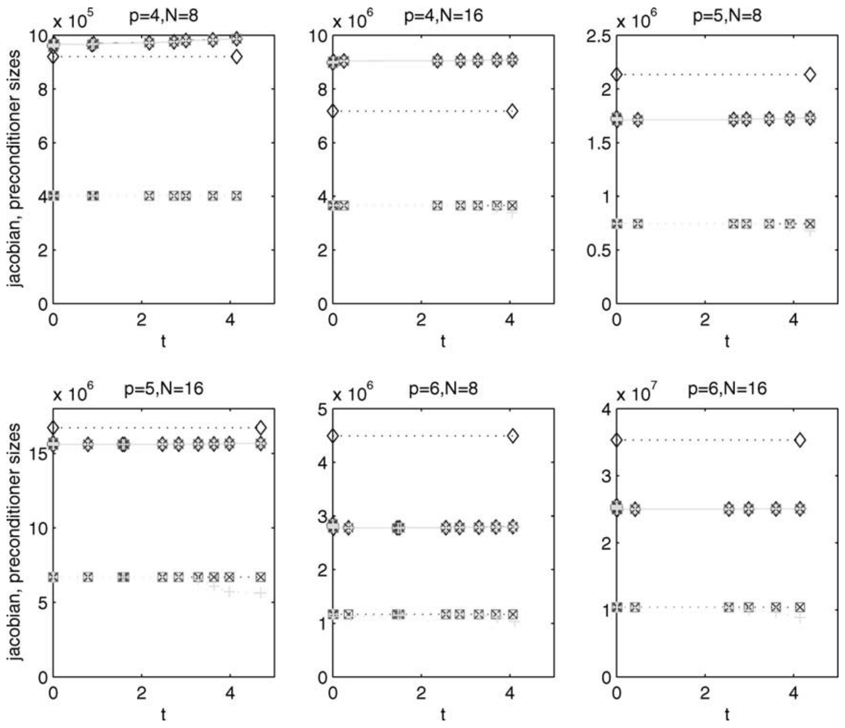


Fig. 2. The size of the arrays for storing the Jacobian (dotted line) and preconditioner (solid line) using FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 (×) and FEDAS–IAT4 (+) for $p = 4, 5, 6$ and $N = 8, 16$ in solving Example 1.

The total number of GMRES iterations is identical for FEDAS–IAT10 and FEDAS–IAT7. FEDAS–FA differs from them by at most 1%. FEDAS–IAT4 differs by less than 1% for all but the finest discretizations where it uses 9% less (cf. Fig. 3). Plots of the number of GMRES iterations per time step are displayed in Fig. 3 for $p = 5, 6$ and $N = 16$. The other discretizations yield similar results. There appears to be no correlation between the iteration behavior, the discretization and the performance of DASPK (see above) except for $t > 3.5$ when the number of iterations per step decreases. No GMRES restarts are used on any runs.

Example 2. Consider the Cahn–Allen equation [11]

$$\begin{aligned} &u_t = \epsilon_1 \Delta u + u - u^3, \quad \mathbf{x} \in \Omega \equiv [0,1]^3, \ t > 0 \\ &u(\mathbf{x},0) = (x(1-x)y(1-y)z(1-z))^2 - 0.00014\mathbf{x} \in \Omega, \\ &\nabla u \cdot \mathbf{n}(\mathbf{x},t) = 0, \quad \mathbf{x} \in \partial\Omega, \end{aligned} \tag{16}$$

where $\epsilon = 1.0 \times 10^{-4}$. The equation has three constant steady states, $u = 0, \pm 1$. Solution behavior can be characterized by two time frames. During the first the solution quickly generates regions where the solution is either 1 or -1 separated by sharp interfaces (cf. Fig. 5). The interfaces move transcendentally slowly and annihilate each other during the second time frame leading to either $u = -1$ or $u = 1$ throughout [11]. With the initial conditions in (16) after the first time frame $u = 1$ inside a small sphere centered at $(0.5, 0.5, 0.5)$, $u = -1$ in the rest of the domain with a sharp interface between the two. At a significantly later time the solution undergoes a rapid transition to $u = -1$ throughout Ω.

I solve (16) on $0 < t \leqslant 10^6$. Solution profiles of u as a function of t at $\mathbf{x} = (0.5, 0.5, 0.5)$, obtained by FEDAS–FA, are shown in the upper portion of Fig. 4 when $p = 6$ and $N = 8, 16$ (the results for the other discretizations are comparable). The corresponding errors e obtained by taking the difference between the full assembly solution and each of the incomplete assembly solutions are displayed at the bottom of Fig. 4. Solution accuracy is independent of the assembly procedure used, except at one time. The difference is larger for the finer discretization and for FEDAS–IAT7 and FEDAS–IAT4. For all discretizations the computed solution has an overshoot before reaching steady state, although the size of the overshoot decreases as the grid is refined. The solution at $t = 94.4$ obtained by FEDAS–FA with $p = 4$ and $N = 16$ is shown in Fig. 5. The sharp interface between the spheres can clearly be seen.

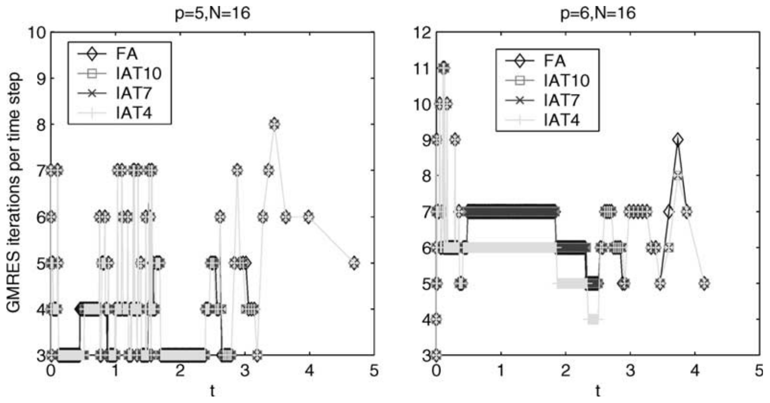


Fig. 3. The number of GMRES iterations per time step for FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 (×) and FEDAS–IAT4 (+) with $p = 5, 6$ and $N = 16$ in solving Example 1.

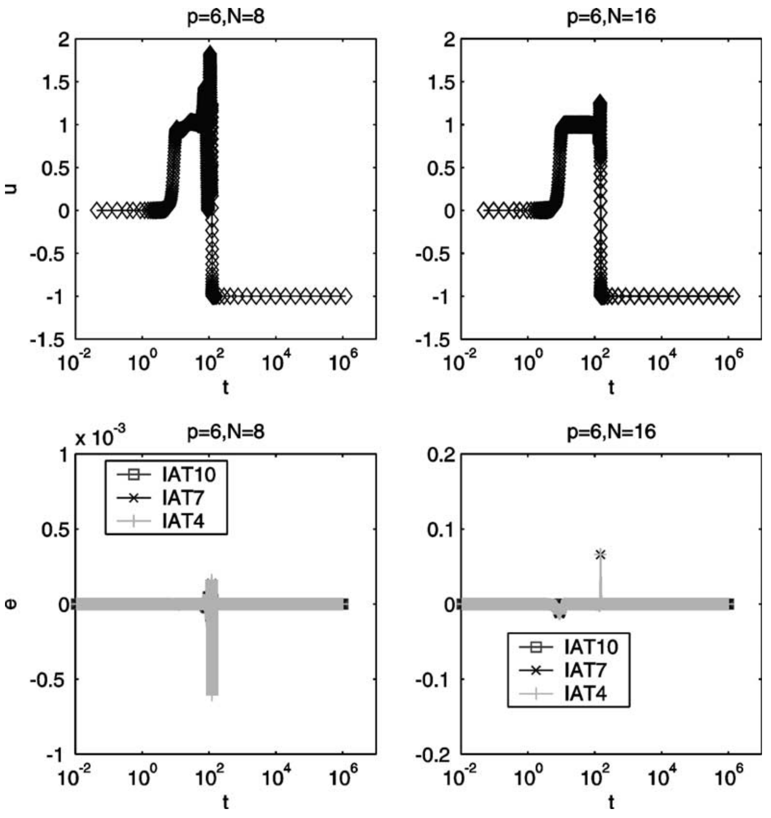


Fig. 4. The solution u (above) at (0.5,0.5,0.5) when using FEDAS–FA ($\Diamond$), and the corresponding differences e (below) between FEDAS–FA and FEDAS–IAT10 ($\Box$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) with $p = 6$ and $N = 8, 16$ in solving Example 2.

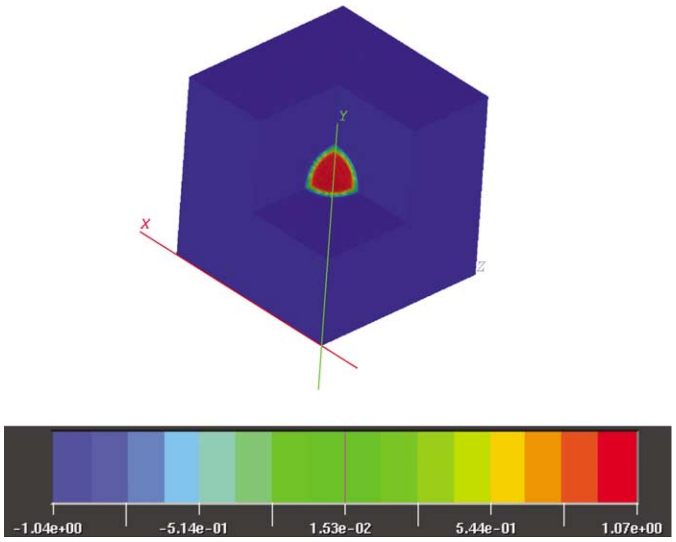


Fig. 5. The solution u at $t = 94.4$ when using FEDAS–FA with $p = 4$ and $N = 16$ in solving Example 2.

Fig. 6 shows the CPU time for all discretizations and assembly procedures. Most of the CPU time is consumed between the time the interface is formed and the steady state, $u = -1$, is reached. In all but the finest discretization FEDAS–FA has the fastest run times. However, the difference between the FEDAS–FA and FEDAS–IAT is modest (a maximum of 26% faster on the coarsest grid) and decreases as p increases. When $p = 6$ and $N = 16$ FEDAS–IAT4 runs 1% faster than FEDAS–FA. From the data in Tables 1 and 2 FEDAS–IAT4 uses more function evaluations and GMRES iterations but fewer time steps and Jacobian assemblies than FEDAS–FA at this discretization. Since the results in Example 1 show that Jacobian assembly is a sizable contributor to the overall CPU time and that the difference in time in obtaining the preconditioner is insignificant (when the preconditioners are comparable in size, cf. Fig. 7) the difference in the number of Jacobian assemblies is the likely explanation for the superior CPU time of FEDAS–IAT4. This is also borne out by noting that FEDAS–IAT10 is at least 11% slower than FEDAS–FA for all discretizations. As seen in Tables 1 and 2 FEDAS–FA and FEDAS–IAT10 run almost identically. From Fig. 7 it follows that the Jacobians computed by FEDAS–IAT10 are noticeably smaller than those computed by FEDAS–FA over much of the time interval while the preconditioners are comparable in size. Therefore the overall cost of matrix–vector products in FEDAS–IAT10 should be less than FEDAS–FA (cf. Example 5). Thus, slower assembly time for IAT10 is the likely explanation for the time difference.

Storage usage for each Jacobian and its preconditioner is presented in Fig. 7. The FA Jacobian requires less storage than its preconditioner when $p < 6$ and more when $p = 6$. Assembly procedures and changes in the solution have only a marginal impact on the size of the preconditioners. On the other hand the amount of storage for the IAT Jacobians does depend on the solution behavior, increasing, sometimes dramatically, during times when the the solution is changing rapidly. In the case of FEDAS–IAT10 Jacobian storage approaches that of FEDAS–FA between the formation of the interface and convergence to steady state.

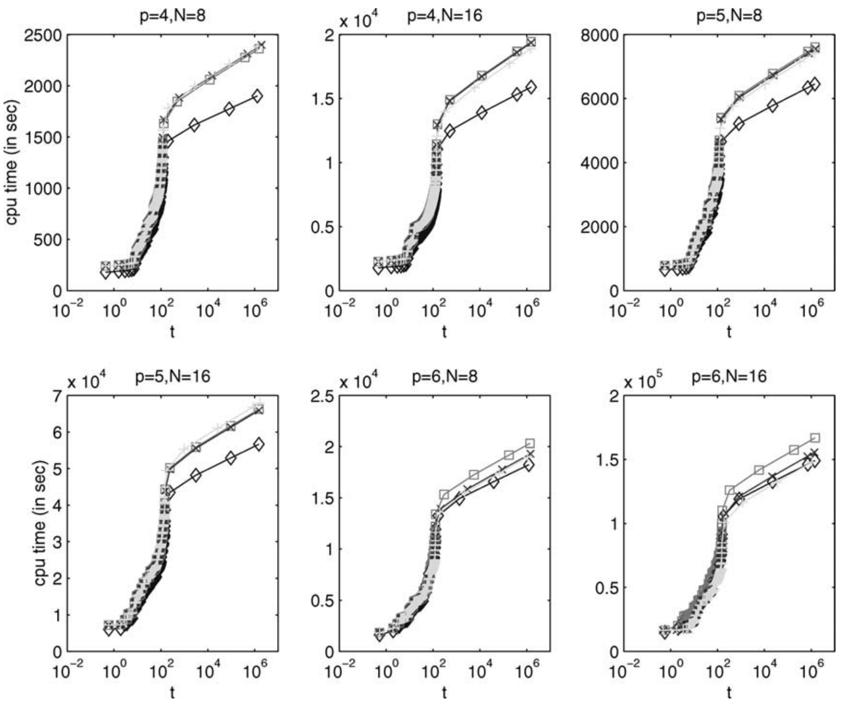


Fig. 6. The CPU time in solving Example 2 with $p = 4, 5, 6$ and $N = 8, 16$ using FEDAS–FA ($\Diamond$), FEDAS–IAT10 ($\Box$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$).

Table 1
The number of Jacobian assemblies (JAC), function evaluations (FNC) and time steps (TS) used by FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4 in solving Example 2

p	N	FEDAS–FA			FEDAS–IAT10			FEDAS–IAT7			FEDAS–IAT4		
		JAC	FNC	TS	JAC	FNC	TS	JAC	FNC	TS	JAC	FNC	TS
4	8	37	929	634	37	939	632	39	885	617	39	888	624
4	16	35	901	492	35	901	492	35	901	492	34	898	493
5	8	42	817	506	42	827	506	42	816	506	42	780	507
5	16	44	845	494	44	845	494	44	845	494	46	837	471
6	8	41	892	595	41	845	563	38	897	614	38	896	604
6	16	42	841	501	42	843	503	39	833	486	36	922	475

Table 2
The total number of GMRES iterations (restarts) used by FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4 in solving Example 2

p	N	FEDAS–FA	FEDAS–IAT10	FEDAS–IAT7	FEDAS–IAT4
4	8	2695 (0)	2727 (0)	2547 (0)	2562 (0)
4	16	2697 (0)	2700 (0)	2698 (0)	2694 (0)
5	8	3877 (0)	3956 (0)	3897 (0)	3742 (0)
5	16	3499 (0)	3487(0)	3494 (0)	3515 (0)
6	8	7373 (20)	7031 (105)	7594 (100)	7548 (95)
6	16	4859 (11)	4826 (10)	4765 (10)	5334 (1)

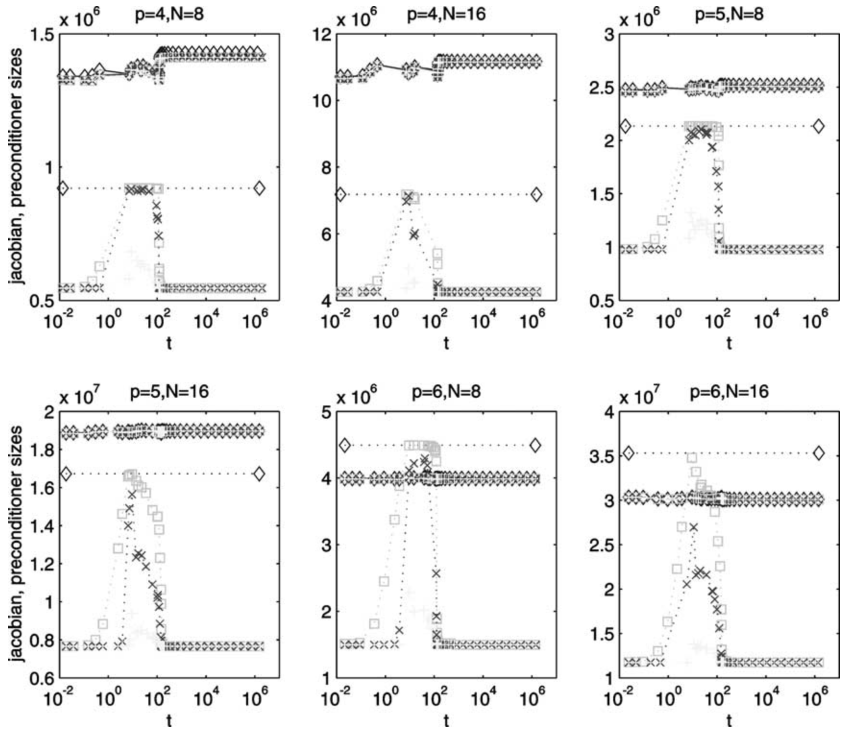


Fig. 7. The size of the arrays for storing the Jacobian (dotted line) and preconditioner (solid line) using FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 (×) and FEDAS–IAT4 (+) for $p = 4, 5, 6$ and $N = 8, 16$ in solving Example 2.

FEDAS–IAT4 uses the least storage throughout and is least effected by the changes in the solution. For most of the computation the IAT Jacobians use significantly less storage than the FA Jacobians and all preconditioners.

The effects on DASPK and GMRES of using the IAT algorithms are presented in Tables 1 and 2. DASPK and GMRES operate almost identically in FEDAS–FA and FEDAS–IAT10. Comparable performance is observed in FEDAS–IAT7 when $p < 6$ but fewer Jacobian assemblies are needed when $p = 6$. As N increases the number of time steps decreases and, if $p > 4$, so do the number of GMRES iterations. As p increases the number of GMRES iterations increases (the maximum number of iterations per step also increases with increasing p). GMRES restarts play a role only when $p = 6$ and are only significant when $N = 8$. In this case the FEDAS–IAT algorithms require more.

As a representative case the number of GMRES iterations per time step for all assembly routines and $p = 6$ and $N = 16$ is displayed in Fig. 8. The top graph plots the data over the whole time interval while the graph on the bottom highlights $[5, 300]$. The number of iterations oscillates rapidly as the interface is forming and during the transition to the single steady state. Relative maxima occur at these two critical times. Once the steady state is reached the number of iterations per step drops to one. The time step history shown in Fig. 9 for FEDAS–FA with $p = 6$ and $N = 16$ demonstrates that though the number of GMRES iterations per step has periods of rapid oscillation the time steps do not (results for other discretizations and assembly routines are comparable).

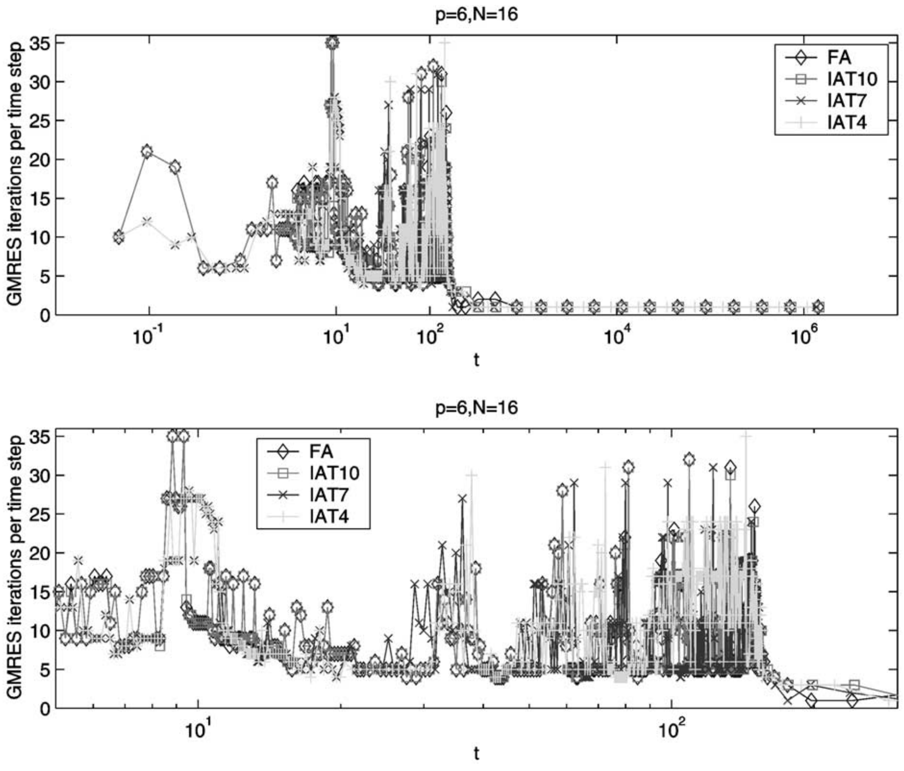


Fig. 8. The number of GMRES iterations per time step for FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 (×) and FEDAS–IAT4 (+) with $p = 6$ and $N = 16$ in solving Example 2 on $(0, 10^6]$ (top) and on $[5, 300]$ (bottom).

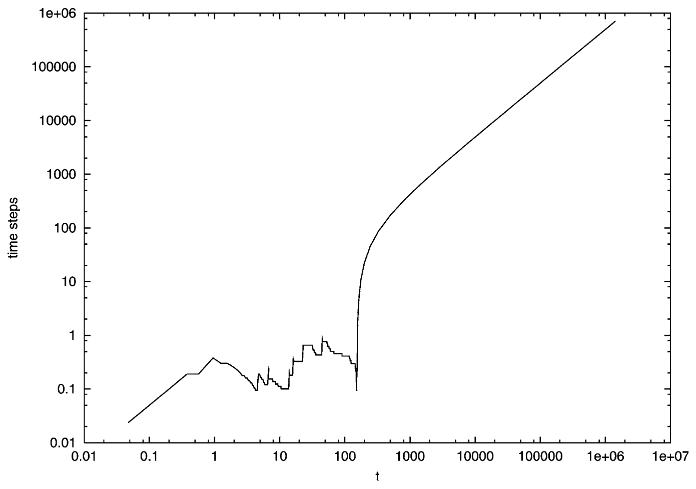


Fig. 9. The time steps for FEDAS–FA with $p = 6$ and $N = 16$ for Example 2.

Example 3. Consider the combustion problem [15]

$$\begin{aligned}
&c_t = \Delta c - Dce^{-\delta/T},\\
&LT_t = \Delta T + \alpha Dce^{-\delta/T}, \quad \mathbf{x} \in \Omega \equiv [0,1]^3,\ t > 0,\\
&c(\mathbf{x},0) = T(\mathbf{x},0) = 1, \quad \mathbf{x} \in \Omega,\\
&\nabla c \cdot \mathbf{n}(\mathbf{x},t) = \nabla T \cdot \mathbf{n}(\mathbf{x},t) = 0, \quad x = 0,\ y = 0 \text{ and } z = 0,\ t > 0,\\
&c(\mathbf{x},t) = T(\mathbf{x},t) = 1, \quad x = 1,\ y = 1, \text{ and } z = 1.
\end{aligned} \tag{17}$$

The variables c and T represent the concentration and temperature of a reacting mixture, respectively, α is the heat release, L is the Lewis number, $D = Re^{\delta}/\alpha\delta$ is the Damkohler number, δ is the activation energy and R is the reaction rate. System (17) is solved with $\alpha = 1$, $L = 0.9$, $\delta = 20$ and $R = 5$. The temperature initially increases slowly with a hot spot forming at the origin. At a finite time, ignition occurs and the temperature at the origin jumps from near unity to approximately $1 + \alpha$ while the concentration goes to 0. A sharp reaction front (cf. Fig. 11) forms and propagates rapidly towards the boundary faces $x = 1$, $y = 1$ and $z = 1$ where boundary layers develop. The solution T, obtained by FEDAS–FA, as a function of t at $\mathbf{x} = (0.25, 0.25, 0.25)$ and $(0.75, 0.75, 0.75)$ is shown in the upper half of Fig. 10 for $p = 4$ and $N = 8, 16$. The corresponding errors e found by taking the difference between the solutions obtained by FEDAS–FA and the FEDAS–IAT algorithms are displayed at the bottom of Fig. 10. The differences are small with the largest difference occurring between the results from FEDAS–FA and FEDAS–IAT4 at the transition. In Fig. 11 the solution T obtained by FEDAS–FA with $p = 4$ and $N = 16$ at $t = 0.317$ is shown.

The CPU times for all discretizations and assembly algorithms are displayed in Fig. 12. Most of the CPU time is consumed during ignition and propagation of the flame front while startup costs are modest. The difference in running times is small with FEDAS–FA at most 12% faster than the others. The data in the final rows of Tables 3 and 4 show that FEDAS–IAT4 uses more function evaluations, time steps and GMRES iterations and restarts than FEDAS–FA when $p = 6$ and $N = 16$. Nevertheless it is 6% faster. As in Example 2, the likely source of these savings is the smaller number of Jacobian assemblies.

In Fig. 13 the memory usage is presented. Preconditioner storage remains relatively constant over the time interval with no noticeable difference among the assembly routines. For all but FEDAS–IAT4 the preconditioner takes less storage than its respective Jacobian. Significant differences in the amount of

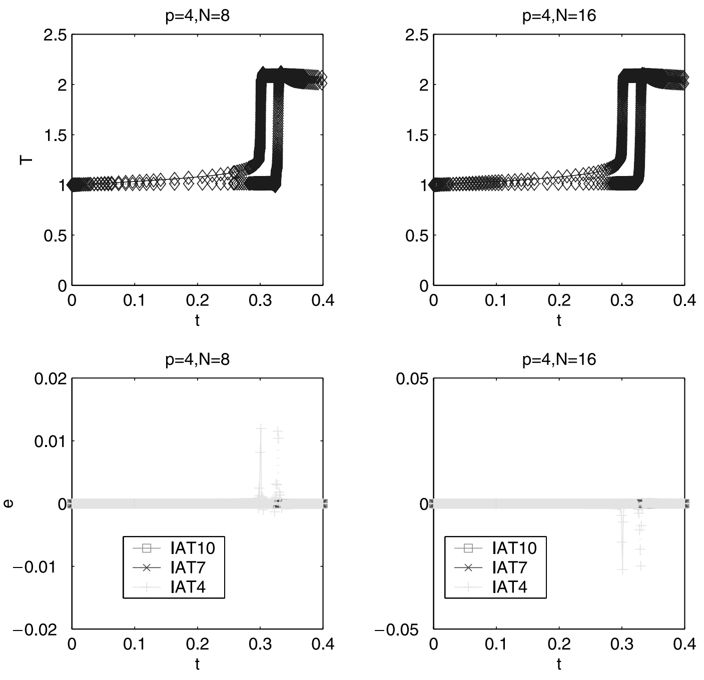


Fig. 10. The solution T (above) at $(0.25, 0.25, 0.25)$ (first wave) and $(0.75, 0.75, 0.75)$ (second wave) when using FEDAS–FA ($\diamondsuit$) and the corresponding differences e (below) between FEDAS–FA and FEDAS–IAT10 ($\square$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) with $p = 4$ and $N = 8$ and 16 in solving Example 3.

storage needed for the Jacobian exist with FEDAS–FA requiring the most and FEDAS–IAT4 the least. Near ignition and the beginning of flame front propagation the size of the Jacobian surprisingly decreases then increases dramatically for FEDAS–IAT7 and slightly for FEDAS–IAT4. With FEDAS–IAT10 a

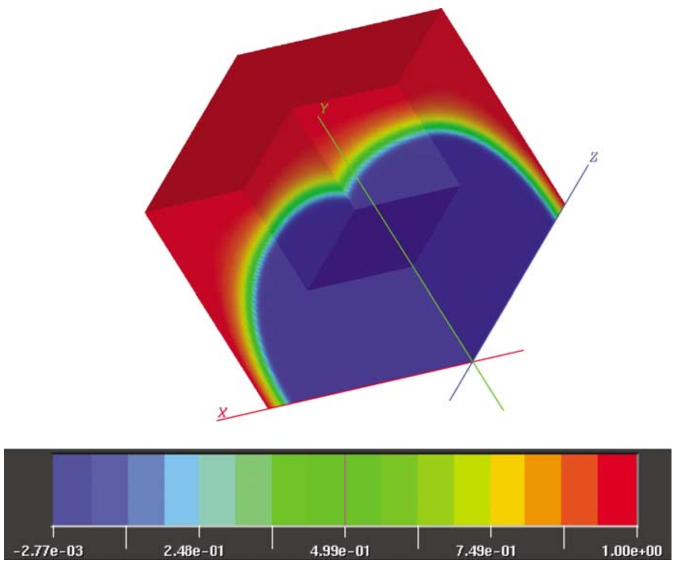


Fig. 11. The solution T at $t = 0.317$ when using FEDAS–FA with $p = 4$ and $N = 16$ in solving Example 3.

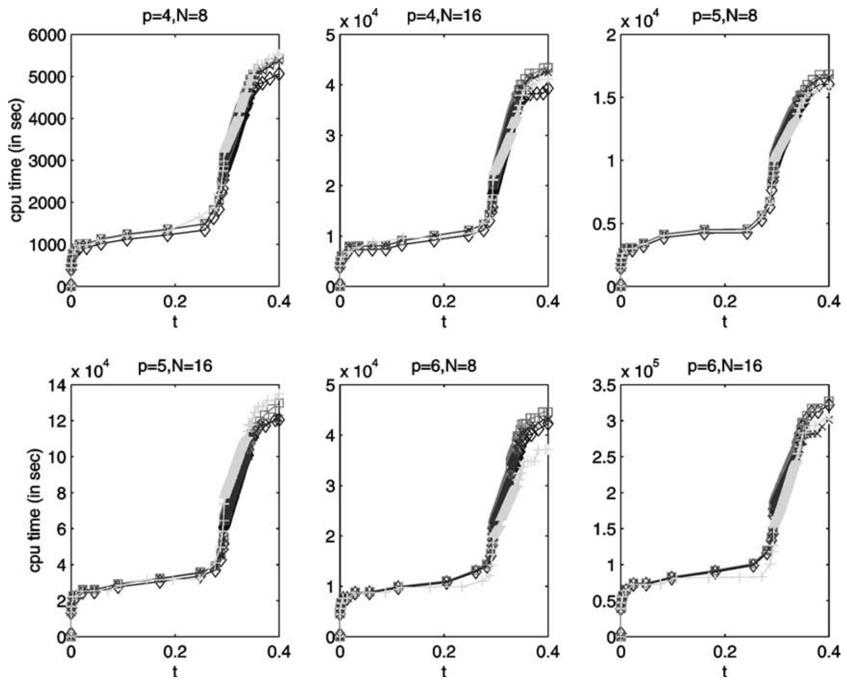


Fig. 12. The CPU time in solving Example 3 with $p = 4, 5, 6$ and $N = 8, 16$ using FEDAS–FA ($\diamondsuit$), FEDAS–IAT10 ($\square$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$).

Table 3
The number of Jacobian assemblies (JAC), function evaluations (FNC) and time steps (TS) used by FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4 in solving Example 3

p	N	FEDAS–FA			FEDAS–IAT10			FEDAS–IAT7			FEDAS–IAT4		
		JAC	FNC	TS	JAC	FNC	TS	JAC	FNC	TS	JAC	FNC	TS
4	8	34	1477	1180	34	1477	1180	34	1477	1180	37	1444	1126
4	16	24	1825	984	26	1823	987	26	1835	982	31	1378	1119
5	8	33	1583	1160	33	1583	1160	33	1583	1160	33	1502	1155
5	16	26	1826	977	27	1828	984	26	1781	971	32	1771	1047
6	8	27	1846	1275	28	1867	1258	28	1867	1258	25	1693	1081
6	16	27	1597	945	27	1597	945	25	1642	965	23	2218	1027

Table 4
The total number of GMRES iterations (restarts) used by FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4 in solving Example 3

p	N	FEDAS–FA	FEDAS–IAT10	FEDAS–IAT7	FEDAS–IAT4
4	8	3520 (0)	3523 (0)	3519 (0)	3416 (0)
4	16	5196 (0)	5191 (0)	5155 (0)	3645 (0)
5	8	5559 (0)	5544 (0)	5555 (0)	5289 (0)
5	16	6507 (0)	6556 (0)	6406 (0)	6786 (0)
6	8	10,671 (31)	11,534 (10)	11,560 (10)	10,772 (8)
6	16	7632 (17)	7631 (17)	7914 (17)	10,551 (29)

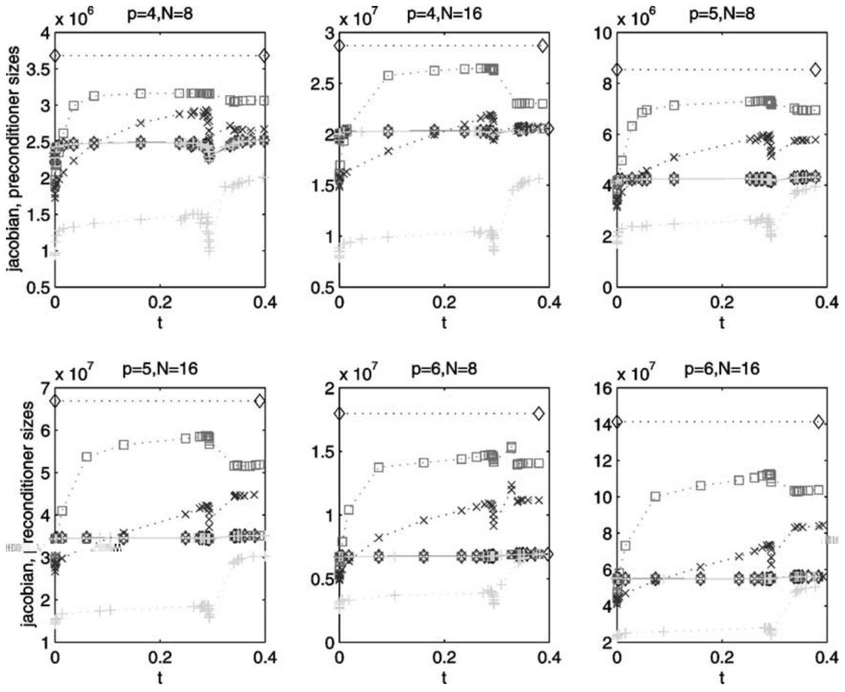


Fig. 13. The size of the arrays for storing the Jacobian (dotted line) and preconditioner (solid line) using FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 (×) and FEDAS–IAT4 (+) for $p = 4, 5, 6$ and $N = 8, 16$ in solving Example 3.

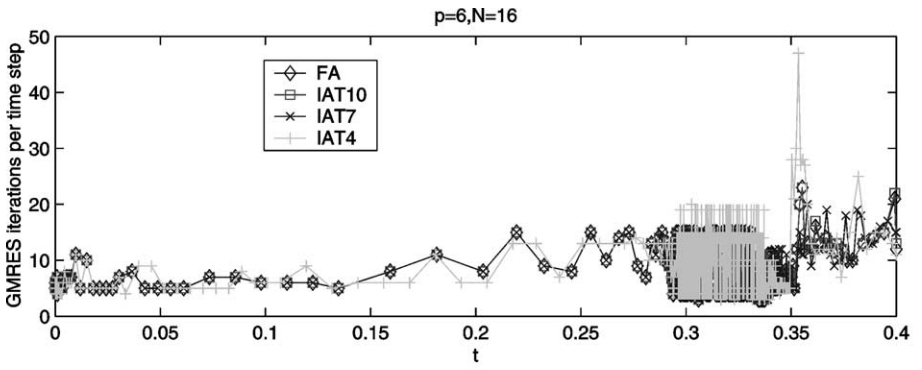
gradual and permanent decrease in Jacobian size beginning at that time is observed. Currently, I have no explanation for this behavior.

From Tables 3 and 4 it follows that FA, IAT10 and IAT7 lead to essentially the same performance of DASPK and GMRES while IAT4 produces notable differences, especially when $p = 6$. As N increases the number of time steps and Jacobian assemblies decreases. As p increases the number of GMRES iterations increases. As in Example 2 there are no GMRES restarts if $p < 6$. More occur when $N = 8$ but even in this case their occurrence is less than (with one small exception) 10% of the number of time steps.

The number of GMRES iterations per time step for $p = 6$, $N = 16$, is displayed in Fig. 14. The top graph plots the data over the whole time interval while the graph on the bottom focuses [0.28, 0.35]. For all assembly methods the number of iterations oscillates most rapidly during flame front propagation and the amplitude of the oscillations is relatively large. The oscillations smooth out as the boundary layers are formed. As in Example 2, the time step behavior is much smoother. These general trends are also observed for the other discretizations.

Example 4. Consider the Brusselator problem with diffusion [14]

$$\begin{aligned}
&u_t = \epsilon\Delta u + 1 + u^2 v - 4.4u,\\
&v_t = \epsilon\Delta v + 3.4u - u^2 v, \quad \mathbf{x} \in \Omega \equiv [0,1]^3, \;\; t > 0,\\
&u(\mathbf{x},0) = 0.5 + y + 0.4z + 0.5g(y) + 0.02g(z),\\
&v(\mathbf{x},0) = 1.0 + 5x + 0.25g(x), \quad \mathbf{x} \in \Omega,\\
&\nabla u \cdot \mathbf{n}(\mathbf{x},t) = \nabla v \cdot \mathbf{n}(\mathbf{x},t) = 0, \quad \mathbf{x} \in \partial\Omega.
\end{aligned} \tag{18}$$



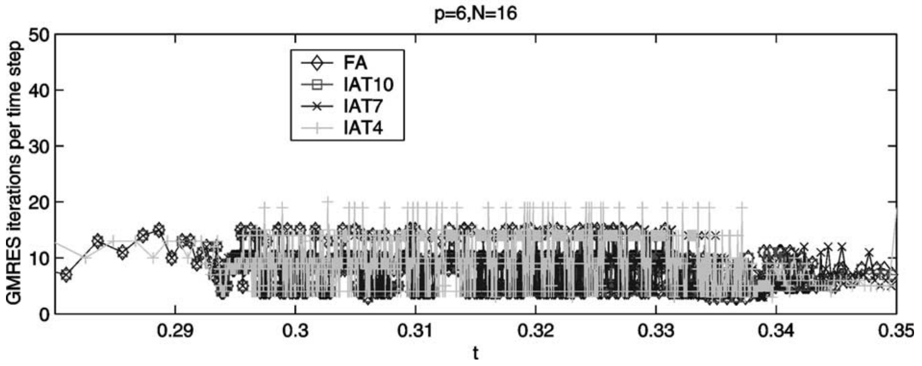


Fig. 14. The number of GMRES iterations per time step for FEDAS–FA ($\diamond$), FEDAS–IAT10 ($\square$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) with $p = 6$ and $N = 16$ in solving Example 3 on $(0, 0.4]$ (top) and $[0.28, 0.35]$ (bottom).

The function $g(\xi) = \tanh 20(\xi - 1) - \tanh 20(\xi)$ serves to enforce continuity between the initial and boundary conditions. For t large and ϵ small the solution exhibits a time-periodic wave-like solution with steep fronts rapidly propagating across Ω. Between the waves are longer periods of quiescence. I solve (18) on $0 < t \leqslant 30$. After an initial transient three waves pass through the domain. In the upper portion of Figs. 15 and 16 u, obtained by FEDAS–FA, is plotted as a function of t at $\mathbf{x} = (0.25, 0.25, 0.25)$ and $(0.75, 0.75, 0.75)$, for $p = 5$ and $N = 8, 16$, respectively. The corresponding errors e found by taking the difference between the solutions obtained by FEDAS–FA and the FEDAS–IAT algorithms are displayed at the bottom of Figs. 15 and 16. The differences are smaller than in the previous two examples. As in Examples 2 and 3 the errors are larger for the finer discretization and for IAT4. Finally the errors are due to phase (and not amplitude) differences. This is also true for the other discretizations. The solution u at $t = 23.2$, obtained by FEDAS–FA with $p = 4$ and $N = 16$, when the wave is in the domain, is shown in Fig. 17.

The CPU times for the all discretizations and codes are shown in Fig. 18. The alternate active and rest states of the solution (cf. Figs. 15 and 16) can be observed in the stair-step behavior of the CPU time. As in Example 1 significant time is spent during the initial phase in choosing the optimal temporal order and time step. The differences in CPU time among the assembly procedures are negligible (under 6% when $p > 4$). FEDAS–FA is fastest except for $p = 6$ and $N = 16$ when FEDAS–IAT4 is 4% faster. A comparison of FEDAS–FA and FEDAS–IAT4 in Tables 5 and 6 for this discretization supports the contention that the difference in CPU time is due to the difference in the number of Jacobian assemblies. This is in line with the results in Examples 2 and 3.

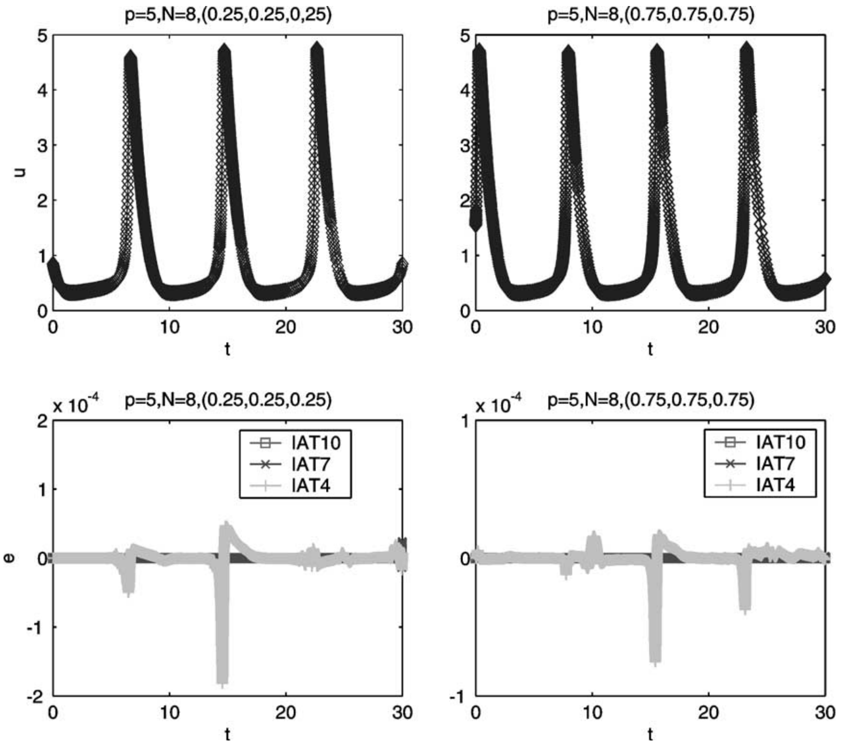


Fig. 15. The solution u (above) at $(0.25, 0.25, 0.25)$ (left) and $(0.75, 0.75, 0.75)$ (right) for $p = 5$ and $N = 8$ when using FEDAS–FA ($\diamond$) and the corresponding differences e (below) between FEDAS–FA and FEDAS–IAT10 ($\square$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) in solving Example 4.

As in the previous examples, Fig. 19 shows that the size of the preconditioner is independent both of the solution activity and of the assembly procedure. For all discretizations the size of the Jacobian assembled using FA, IAT10 and IAT7 is larger than the size of its preconditioner but smaller for FEDAS–IAT4. Jacobians assembled via IAT4, IAT7 and IAT10 vary in size as the solution changes (smaller during quiescent periods, larger when a wave is in the domain) although for IAT10 the changes are modest and only appear when $p > 4$.

As in Examples 1–3 the data in Tables 5 and 6 bear out the similar performance of DASPK and GMRES in FEDAS–FA, FEDAS–IAT10 and FEDAS–IAT7. As N increases the number of function evaluations, time steps and GMRES iterations decreases. The number of Jacobian assemblies also decreases with increasing N for $p = 4$ and 6. The number of GMRES iterations increases with p. As in Examples 2 and 3 there are no GMRES restarts with $p < 6$, and as in Example 3, the number of restarts when $p = 6$ is not significant.

The number of GMRES iterations per time step when $p = 6$ and $N = 16$ for each algorithm is displayed in Fig. 20. During quiescent periods the number of iterations per step remains fixed and close to the minimum over many time steps for the three codes FEDAS–FA, FEDAS–IAT10 and FEDAS–IAT7. When the wave is present in the domain oscillatory behavior is observed. This is also true whenever $p > 4$. The number of iterations per step in FEDAS–IAT4 tends to vary rapidly throughout (though not for all discretizations).

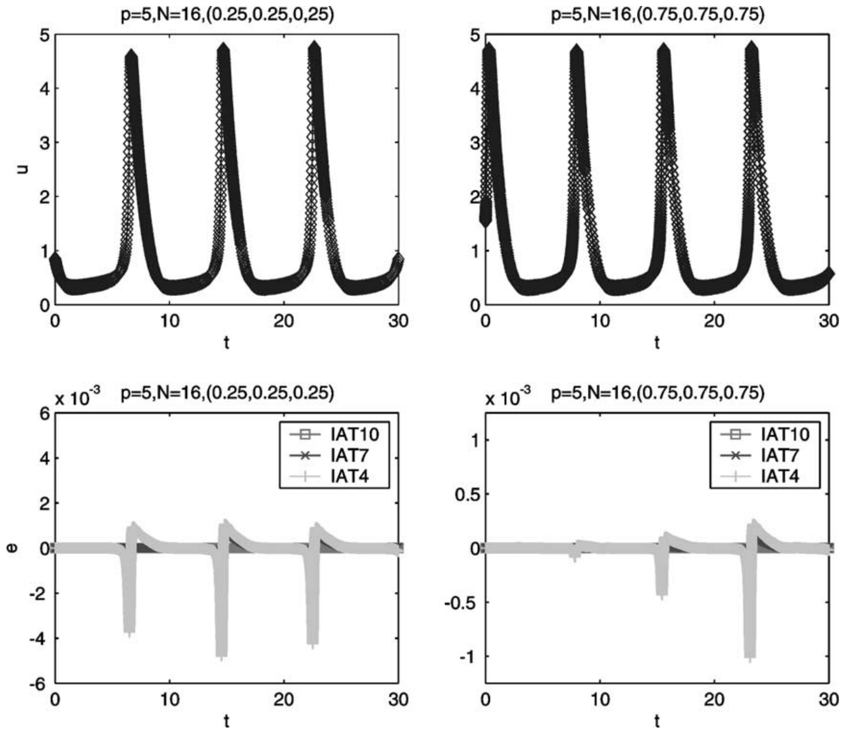


Fig. 16. The solution u (above) at $(0.25, 0.25, 0.25)$ (left) and $(0.75, 0.75, 0.75)$ (right) for $p = 5$ and $N = 16$ when using FEDAS–FA ($\Diamond$) and the corresponding differences e (below) between FEDAS–FA and FEDAS–IAT10 ($\Box$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) in solving Example 4.

Example 5. Consider the Fitzhugh–Nagumo equation [9]

$$\begin{aligned}
&u_t = \epsilon\Delta u + u - u^3/3 - v, \quad \epsilon = 0.1,\\
&v_t = 0.1(u - 1.3v + 0.1), \quad \mathbf{x} \in \Omega \equiv [-50, 50]^3, \ t > 0,\\
&u(\mathbf{x}, 0) = \begin{cases} 1.67, & \mathbf{x} \in R_1,\\ -1.67, & \mathbf{x} \in \Omega \setminus R_1, \end{cases} \quad v(\mathbf{x}, 0) = \begin{cases} 0.3, & \mathbf{x} \in R_2,\\ -0.2, & \mathbf{x} \in \Omega \setminus R_2, \end{cases}\\
&\nabla u \cdot \mathbf{n}(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \partial\Omega,\\
&R_1 = \{(x, y, z)\,|\,x < 0, \ y > 0, \ z/y < -1.11\},\\
&R_2 = \{(x, y, z)\,|\,x < 0, \ y < 0, \ z/y > 1.43\}.
\end{aligned} \tag{19}$$

The solution is characterized by the formation and expansion of a scroll wave that begins forming on the boundary of R_1 and spreads outward [18]. I solve (19) on $0 < t \leqslant 300$. Plots of u, obtained from FEDAS–FA, at $(-25, -25, -25)$ and $(25, 25, 25)$ and corresponding errors e for FEDAS–IAT4, FEDAS–IAT7 and FEDAS–IAT10 with $p = 5, 6$ and $N = 16$ in Figs. 21 and 22 indicate that solution accuracy is not significantly effected by incomplete assembly. In the case of FEDAS–IAT4, as in the earlier examples, the error increases slightly as the grid becomes finer and the error is primarily in phase and not amplitude. The scroll wave at $t = 283.0$ is seen in Fig. 23.

Plots of the CPU time are presented in Fig. 24. Unlike the previous cases FEDAS–FA is not the fastest code, in fact it is always slower than FEDAS–IAT10 and FEDAS–IAT7. The CPU time gap grows as p

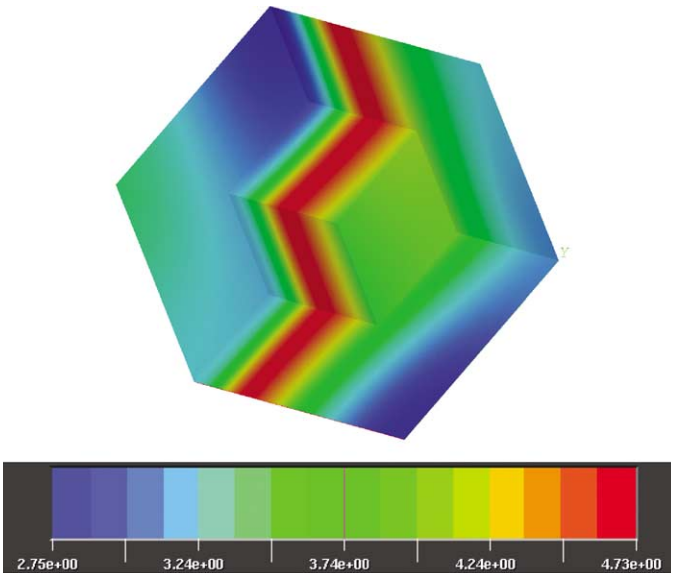


Fig. 17. The solution u at $t = 23.2$ when using FEDAS–FA with $p = 4$ and $N = 16$ in solving Example 4.

increases so that on the finest grid they are 16% faster. This phenomenon cannot be explained by performance differences in DASPK or GMRES since the number of Jacobian assemblies, function evaluations, time steps and GMRES iterations are essentially the same for these three codes (cf. Tables 7 and 8).

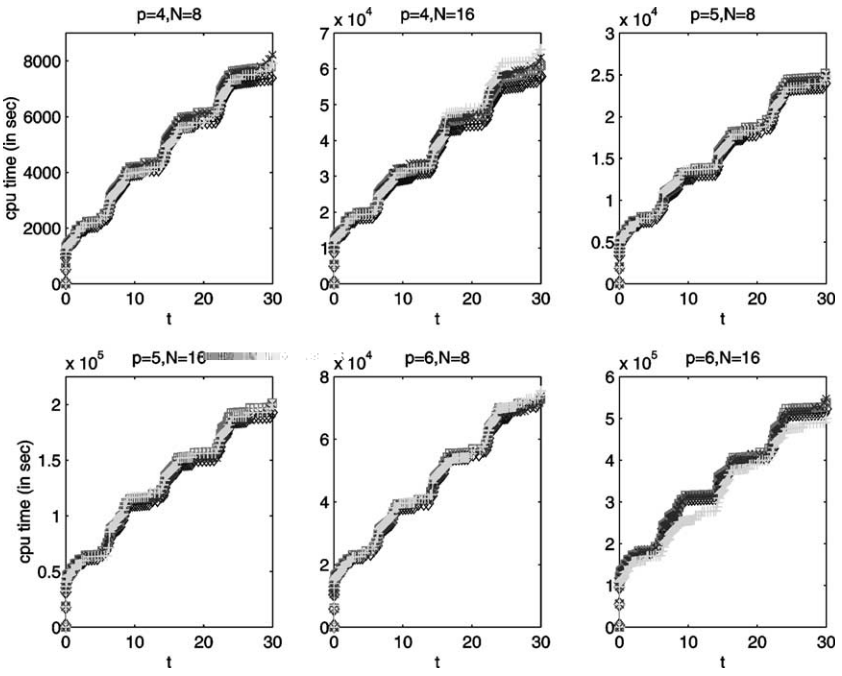


Fig. 18. The CPU time in solving Example 4 with $p = 4, 5, 6$ and $N = 8, 16$ using FEDAS–FA ($\Diamond$), FEDAS–IAT10 ($\Box$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$).

Table 5
The number of Jacobian assemblies (JAC), function evaluations (FNC) and time steps (TS) used by FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4 in solving Example 4

p	N	FEDAS–FA			FEDAS–IAT10			FEDAS–IAT7			FEDAS–IAT4		
		JAC	FNC	TS	JAC	FNC	TS	JAC	FNC	TS	JAC	FNC	TS
4	8	42	2023	1340	42	2023	1340	45	2020	1336	45	1943	1337
4	16	37	1951	1250	37	1951	1250	41	1864	1236	45	1819	1214
5	8	46	1731	1274	46	1731	1274	46	1724	1273	46	1942	1306
5	16	46	1646	1162	46	1646	1162	46	1716	1172	46	1759	1236
6	8	50	1771	1255	49	1753	1251	50	1751	1248	48	2765	1663
6	16	45	1527	1175	45	1554	1182	47	1596	1171	38	2489	1619

Table 6
The total number of GMRES iterations (restarts) used by FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4 in solving Example 4

p	N	FEDAS–FA	FEDAS–IAT10	FEDAS–IAT7	FEDAS–IAT4
4	8	6407 (0)	6406 (0)	6391 (0)	6128 (0)
4	16	6269 (0)	6269 (0)	6017 (0)	5877 (0)
5	8	8215 (0)	8218 (0)	8197 (0)	9178 (0)
5	16	7279 (0)	7279 (0)	7525 (0)	7709 (0)
6	8	14,084 (129)	13,934 (120)	13,950 (124)	20,302 (85)
6	16	9492 (13)	9859 (15)	10,065 (15)	15,627 (13)

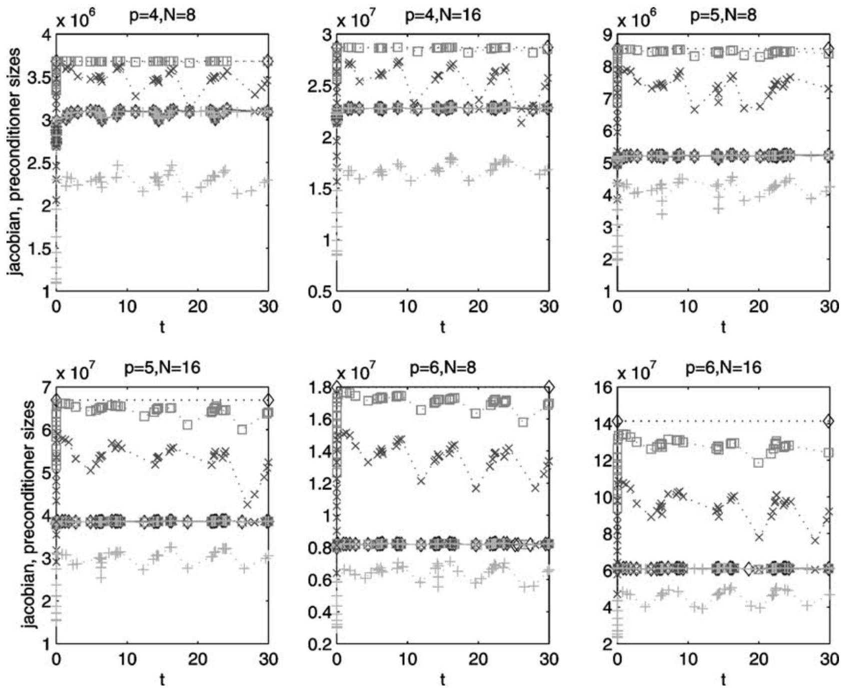


Fig. 19. The size of the arrays for storing the Jacobian (dotted line) and preconditioner (solid line) using FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 (×) and FEDAS–IAT4 (+) for $p = 4, 5, 6$ and $N = 8, 16$ in solving Example 4.

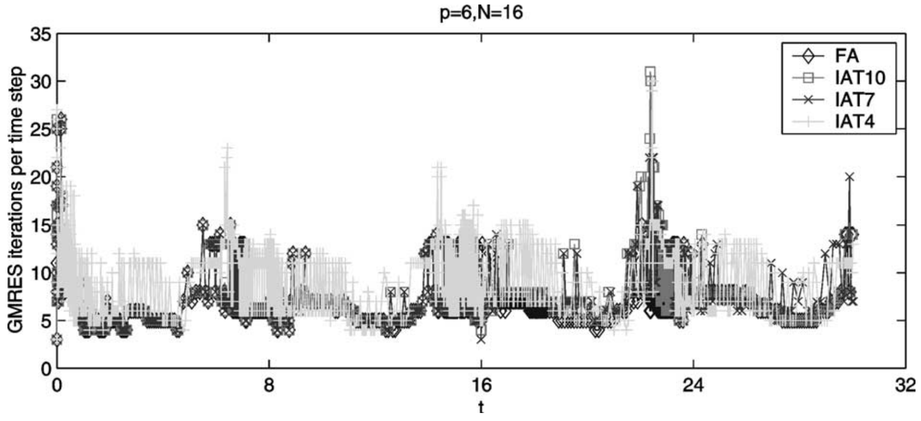


Fig. 20. The number of GMRES iterations per time step using FEDAS–FA ($\Diamond$), FEDAS–IAT10 ($\Box$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) for $p = 6$ and $N = 16$ in solving Example 4.

Two factors contribute to the faster times. From Fig. 25 it follows that the Jacobians produced by FEDAS–IAT10 and FEDAS–IAT7 are significantly smaller than the Jacobians from FEDAS–FA, with the difference increasing as the discretization becomes finer. The size differences are also more pronounced than

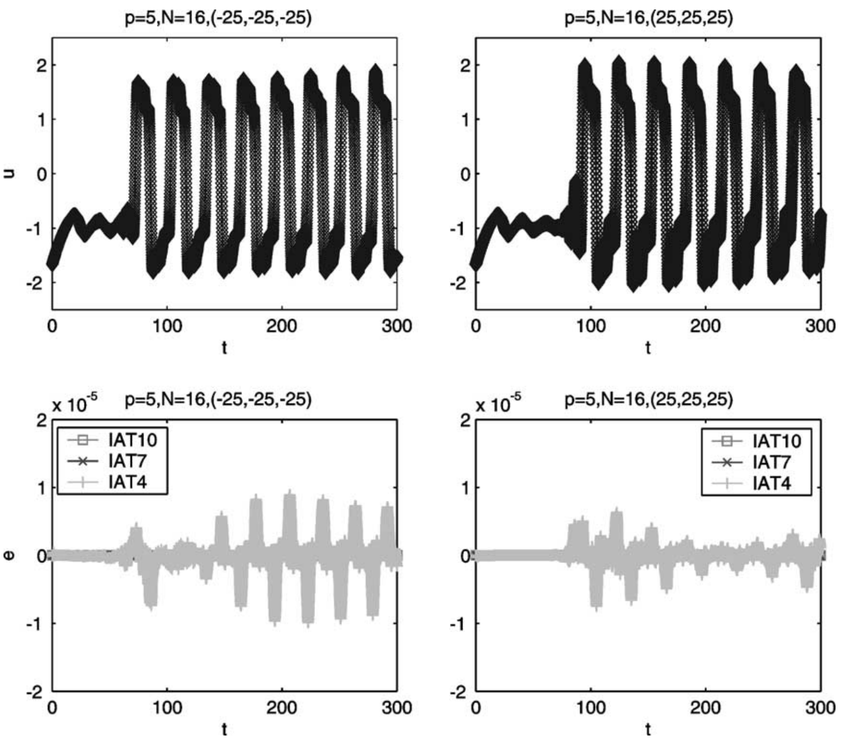


Fig. 21. The solution u (above) at $(-25, -25, -25)$ (left) and $(25, 25, 25)$ (right) for $p = 5$ and $N = 16$ when using FEDAS–FA ($\Diamond$) and the corresponding differences e (below) between FEDAS–FA and FEDAS–IAT10 ($\Box$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) in solving Example 5.

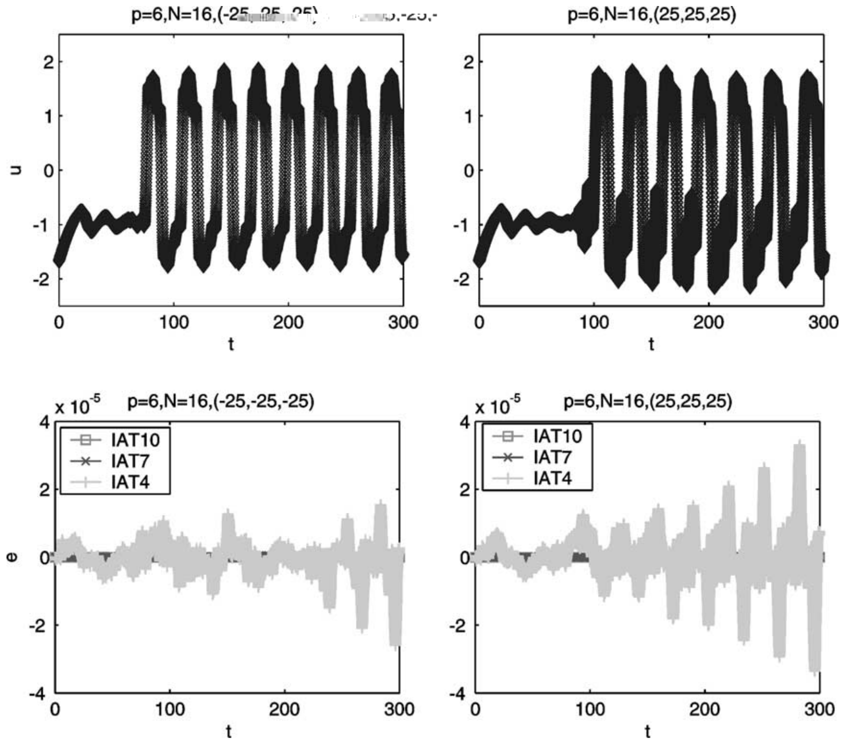


Fig. 22. The solution u (above) at $(-25, -25, -25)$ (left) and $(25, 25, 25)$ (right) for $p = 6$ and $N = 16$ when using FEDAS–FA ($\diamond$) and the corresponding differences e (below) between FEDAS–FA and FEDAS–IAT10 ($\square$), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) in solving Example 5.

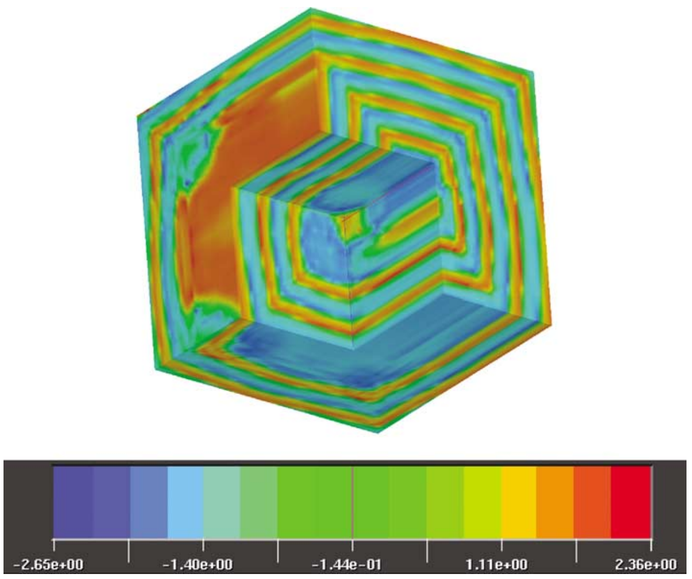


Fig. 23. The solution u at $t = 283.0$ when using FEDAS–FA with $p = 4$ and $N = 16$ in solving Example 5.

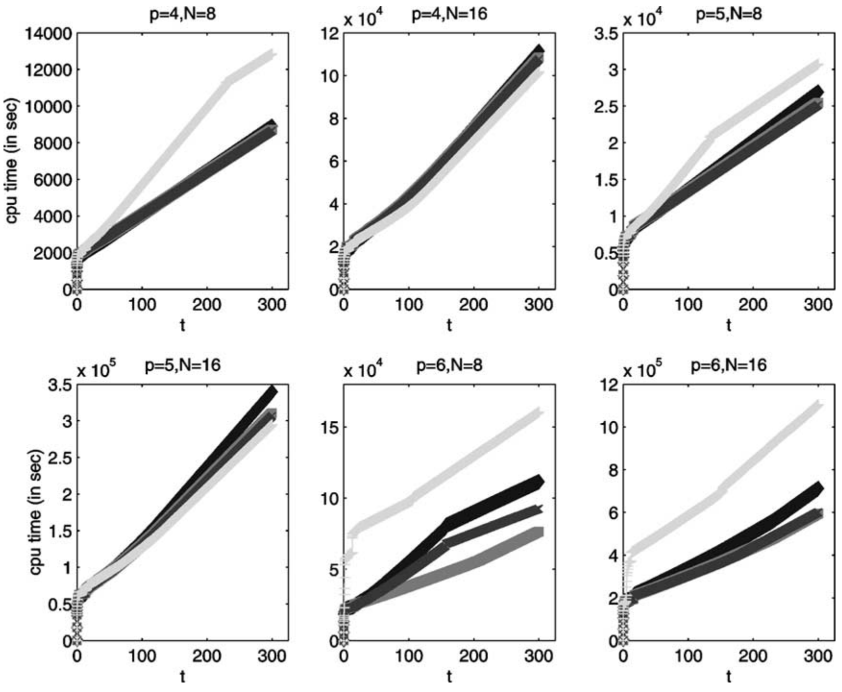


Fig. 24. The CPU time in solving Example 5 with $p = 4, 5, 6$ and $N = 8, 16$ using FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 (×) and FEDAS–IAT4 (+).

those in Examples 3 and 4. As a result the matrix–vector products in FEDAS–IAT10 and FEDAS–IAT7 are more efficient than those in FEDAS–FA. Additionally, since fewer Jacobian assemblies are needed in comparison with the earlier examples (cf. Tables 1, 3 and 5) the difference in assembly times does not dominate the overall time.

FEDAS–IAT4 has the slowest times for four of the discretizations and is substantially slower than FEDAS–FA (by more than 40%) when $p = 6$. As can be seen in Tables 7 and 8 FEDAS–IAT4 has a detrimental effect on the performance of DASPK and GMRES when $p = 6$ in terms of both the number of Jacobian assemblies, function evaluations and GMRES iterations. The storage savings from IAT4 as compared with IAT7 and IAT10 are also less substantial in this example, especially on the fine grids. Jacobian storage for FEDAS–IAT7 grows to match the storage used by FEDAS–IAT10 much sooner

Table 7
The number of Jacobian assemblies (JAC), function evaluations (FNC) and time steps (TS) used by FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4 in solving Example 5

p	N	FEDAS–FA			FEDAS–IAT10			FEDAS–IAT7			FEDAS–IAT4		
		JAC	FNC	TS	JAC	FNC	TS	JAC	FNC	TS	JAC	FNC	TS
4	8	16	5160	5060	16	5160	5060	16	5160	5060	19	7642	4548
4	16	18	7945	4260	18	7945	4260	18	7945	4260	18	7520	4224
5	8	19	4729	4647	19	4725	4644	19	4725	4644	20	6134	4541
5	16	17	8127	4413	17	8127	4413	17	8127	4413	18	7743	4343
6	8	19	6633	4702	19	5246	4954	19	6644	4702	57	10,023	6216
6	16	17	4950	4735	17	4950	4735	17	5180	4832	32	9752	6166

Table 8
The total number of GMRES iterations (restarts) used by FEDAS–FA, FEDAS–IAT10, FEDAS–IAT7 and FEDAS–IAT4 in solving Example 5

p	N	FEDAS–FA	FEDAS–IAT10	FEDAS–IAT7	FEDAS–IAT4
4	8	15,424 (0)	15,423 (0)	15,424 (0)	25,859 (0)
4	16	28,714 (0)	28,742 (0)	28,760 (0)	26,762 (0)
5	8	27,446 (0)	27,458 (0)	27,460 (0)	36,880 (0)
5	16	52,868 (0)	52,805 (0)	52,929 (0)	50,112 (0)
6	8	89,046 (6570)	65,029 (4830)	88,970 (6586)	105,355 (4994)
6	16	67,588 (4912)	67,322 (4918)	70,416 (5165)	118,599 (7976)

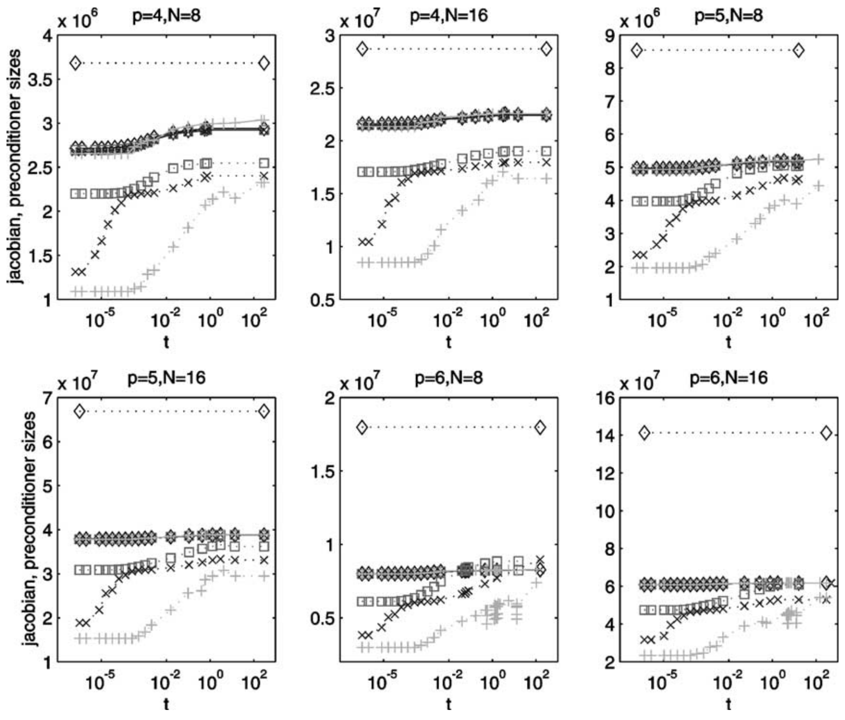


Fig. 25. The size of the arrays for storing the Jacobian (dotted line) and preconditioner (solid line) using FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 (×) and FEDAS–IAT4 (+) with $p = 4, 5, 6$ and $N = 8, 16$ in solving Example 5.

than FEDAS–IAT4. This may explain the similarity in behavior between FEDAS–IAT10 and FEDAS–IAT7 and their difference from FEDAS–IAT4.

Preconditioner storage as seen in Fig. 25 remains nearly constant throughout the computation and is independent of the assembly procedure. More storage is needed to store the Jacobian in FEDAS–FA than is used by its preconditioner while the opposite is true for the FEDAS–IAT algorithms. From Table 8 it follows that restarts play an important role in the performance of GMRES for all assembly routines when $p = 6$. As in Examples 2–4 the number of GMRES iterations increases with increasing p. The number of GMRES iterations per step is displayed in Fig. 26 for $p = 6$ and $N = 16$ on $0 < t \leqslant 300$ (top) and on $[170, 180]$ (bottom). Over much of the domain the number of iterations per step is fairly constant for FEDAS–FA and FEDAS–IAT10. FEDAS–IAT7 varies more at the end while FEDAS–IAT4 has larger oscillations throughout (except for a short initial period).

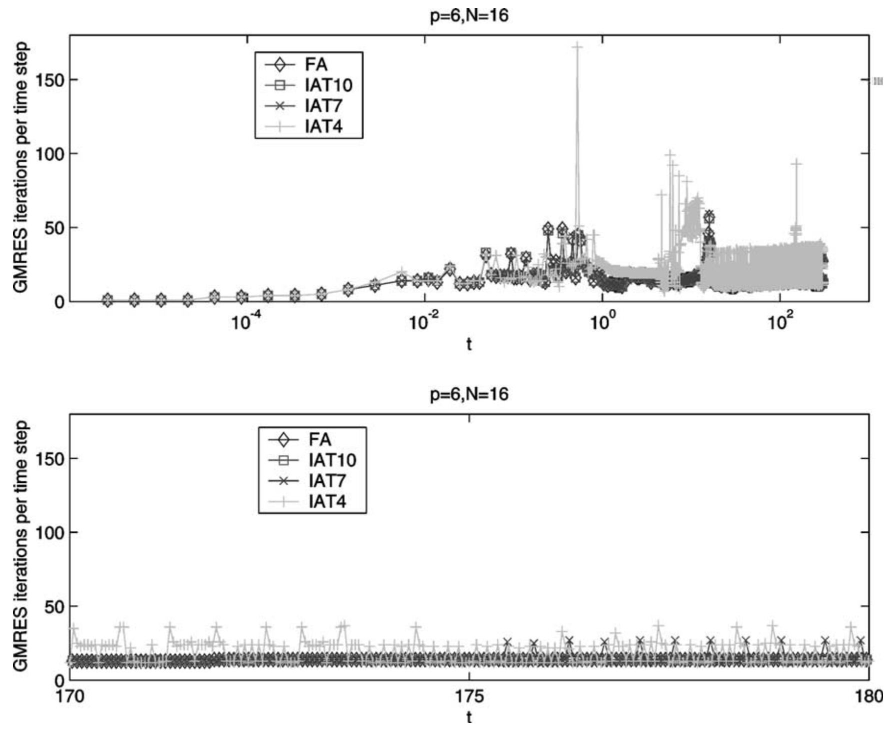


Fig. 26. The number of GMRES iterations per time step using FEDAS–FA (◇), FEDAS–IAT10 (□), FEDAS–IAT7 ($\times$) and FEDAS–IAT4 ($+$) for $p = 6$ and $N = 16$ in solving Example 5 plotted on $(0, 300)$ (top) and $[170, 180]$ (bottom).

5. Conclusions

A new incomplete assembly procedure is presented. The same strategies employed in Saad's ILUT preconditioner [20], maximum row fill-in and a threshold parameter are incorporated into a finite element assembly procedure. The effect of incomplete assembly on a FEMOL code FEDAS [18] is explored for a suite of reaction–diffusion systems. FEDAS uses DASPK for temporal integration and GMRES with ILUT preconditioning to solve large linear systems. The factors examined in this study include: accuracy; CPU time; storage (Jacobian and preconditioner); the number of Jacobian assemblies, function evaluations and time steps used by DASPK; and the number of iterations and restarts in GMRES.

As expected (cf. Section 3) the code using full assembly (FEDAS–FA) is almost always the fastest although as M, p and N increase the CPU time advantage of FEDAS–FA diminishes. With even finer grids this advantage may disappear. In two situations one or more of the FEDAS–IAT codes has faster times than FEDAS–FA: if fewer assemblies are required (Examples 2–4); or if assembly time is not the dominant cost (Example 5). The former is a function of the convergence criteria in DASPK while the latter is a function of the problem. In neither case are they easy to predict. Further reductions in CPU time might be obtained if the band assembly routine of Moore [18] were incorporated into the incomplete assembly algorithm. If the code had been adaptive in space IA and JA would be calculated more often in the full assembly version which would tend to reduce the time differences between the algorithms even further.

The size of the preconditioner is a function of the discretization and the problem but not a function of *itol* or the solution. As a result it would appear that GMRES accounts for changing solution behavior by varying the number of iterations. This is true even when in Example 3 the preconditioner always uses less than 90% of the space available. Further study is needed, however, to determine if increasing *pfil* or decreasing *ptol* would lead to smoother GMRES behavior. For most problems and discretizations the full assembly Jacobian uses more space than the preconditioner. Incomplete assembly leads to savings for almost all problems and those savings increased as *itol* increased and as the grid is refined. Furthermore the size of the incomplete Jacobians often depends on solution behavior. In some cases the incomplete assembly Jacobian use less storage than its preconditioner, especially when $itol = 10^{-4}$. For Example 5 where assembly is not the dominant cost this also leads to faster algorithms when $itol = 10^{-10}$ and 10^{-7}.

In none of the test problems are significant losses of accuracy observed for the values of *itol* considered although the errors tend to grow as the discretization becomes finer with $itol = 10^{-4}$ and to a lesser extent with $itol = 10^{-7}$. Little difference in the operation of DASPK and GMRES exists between FEDAS–FA and FEDAS–IAT for the two smallest values of *itol*. However, when $itol = 10^{-4}$ the performance of both can be quite different, most notably as p increases. Since the Jacobian matrices tend to become more ill-conditioned as p increases it is not surprising to see the number of GMRES iterations and the size of the preconditioner increase with p.

Within the IAT strategy the size of an entry in the Jacobian is a proxy for the size of the contribution that entry makes to the matrix–vector product and to the preconditioner. An alternative thresholding strategy would be to estimate its contribution to the matrix–vector product using the most recent residual vector from DASPK. If the residual is not changing too rapidly over the period the Jacobian is fixed this approach might be more effective.

It is likely that similar results can be obtained on unstructured grids with triangular or tetrahedral elements or on grids with adaptive refinement.

References

[1] G. Akrivis, M. Crouzeix, C. Makridakis, Implicit–explicit multistep finite element methods for nonlinear parabolic problems, Math. Comp. 67 (1998) 457–477.

[2] U.M. Ascher, S.J. Ruuth, B.T.R. Wetton, Implicit–explicit methods for time-dependent partial differential equations, SIAM J. Numer. Anal. 32 (1995) 797–823.

[3] K.E. Brenan, S.L. Campbell, L.R. Petzold, Numerical Solution of Initial Value Problems in Differential-Algebraic Equations, North-Holland, New York, 1989.

[4] P.N. Brown, A.C. Hindmarsh, Matrix-free methods for stiff systems of ode's, SIAM J. Numer. Anal. 23 (1986) 610–638.

[5] P.N. Brown, A.C. Hindmarsh, L.R. Petzold, Using Krylov methods in the solution of large-scale differential-algebraic systems, SIAM J. Sci. Comput. 15 (1994) 1467–1488.

[6] G.F. Carey, J.T. Oden, Finite Elements: Computational Aspects, vol. III, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[7] C. Clavero, J.C. Jorge, F. Lisbona, G.I. Shishkin, An alternating direction scheme on a nonuniform mesh for reaction–diffusion parabolic problems, IMA J. Numer. Anal. 20 (2000) 263–280.

[8] W. Dai, R. Nassar, A second-order ADI scheme for three-dimensional parabolic differential equations, Numer. Method PDEs 14 (1998) 159–168.

[9] R. Dogaru, L.O. Chua, Edge of chaos and local activity domain of Fitzhugh–Nagumo equation, Int. J. Bifur. Chaos 8 (1998) 211–257.

[10] W.H. Enright, T.E. Hull, B. Lindberg, Comparing numerical methods for stiff systems of ODEs, BIT 15 (1975) 10–48.

[11] D.J. Eyre, Pattern formation models for solutions of the Cahn–Allen equation in one dimension, preprint.

[12] D.J. Estep, M.G. Larson, R.D. Williams, Estimating the Error of Numerical Solutions of Systems of Reaction–Diffusion Equations, AMS, Providence, 2000.

[13] R.W. Freund, N.M. Nachtigal, QMR: a quasi-minimal residual method for non-Hermitian linear systems, Numer. Math. 60 (1991) 315–339.

[14] E. Hairer, S.P. Norsett, G. Wanner, Solving Ordinary Differential Equations I: Non-stiff Problems, Spring, Berlin, 1987.

[15] A.K. Kapila, Asymptotic Treatment of Chemically Reacting Systems, Pitman Advanced Publishing Program, Boston, MA, 1983.

[16] R.I. McLachlan, G.R.W. Quispel, Splitting methods, ACTA Numer. 11 (2002) 341–434.

[17] P.K. Moore, R.H. Dillon, A comparison of preconditioners in the solution of parabolic systems in three space dimensions using DASPK and a high order finite element method, Appl. Numer. Math. 20 (1996) 117–128.

[18] P.K. Moore, An adaptive finite element method for parabolic differential systems: some algorithmic considerations for solving in three space dimensions, SIAM J. Sci. Comput 21 (2000) 1567–1586.
[19] S. Ruuth, Implicit–explicit methods for reaction–diffusion problems in pattern formation, J. Math. Biol. 34 (1995) 148–176.
[20] Y. Saad, ILUT: a dual threshold incomplete LU factorization, Numer. Linear Algebra Appl. 1 (1994) 387–402.
[21] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (1986) 856–869.
[22] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing, Boston, MA, 1995.
[23] B. Szabo, I. Babŭska, Finite Element Analysis, Wiley/Interscience, New York, 1991.
[24] V. Thomée, Galerkin Finite Element Methods for Parabolic Problems, Springer, Berlin, 1997.
[25] R. Wait, A.R. Mitchell, Finite Element Analysis and Applications, Wiley, Chichester, 1985.