Journal of Computational Physics **182,** 478–495 (2002)
doi:10.1006/jcph.2002.7179

Computational Solution of Two-Dimensional Unsteady PDEs Using Moving Mesh Methods

G. Beckett,[1] J. A. Mackenzie, A. Ramage, and D. M. Sloan

Department of Mathematics, University of Strathclyde, Livingstone Tower, 26 Richmond Street, Glasgow G1 1XH, Scotland
E-mail: d.sloan@strath.ac.uk

Received November 15, 2001; revised April 17, 2002

Numerical experiments are described which illustrate some important features of the performance of moving mesh methods for solving two-dimensional partial differential equations (PDEs). Here we are concerned with algorithms based on moving mesh methods proposed by W. Huang and R. D. Russell [*SIAM J. Sci. Comput.* **20**, 998 (1999)]. We show that the accuracy of the computations is strongly influenced by the choice of monitor function, and we present a monitor function which yields a higher rate of convergence than those that are commonly used. In an earlier paper [G. Beckett, J. A. Mackenzie, A. Ramage, and D. M. Sloan, *J. Comput. Phys.* **167**, 372 (2001)], we demonstrated a robust and efficient algorithm for problems in one space dimension in which the mesh equation is decoupled from the physical PDE and the time step is controlled automatically. The present work extends this algorithm to deal with problems in two space dimensions.

Key Words: adaptivity; equidistribution; moving meshes.

1. INTRODUCTION

Many unsteady problems governed by partial differential equations (PDEs) have solutions with regions of high variation, such as boundary layers or moving wave fronts. Problems of this type present significant challenges to computational scientists who wish to design numerical algorithms which will yield accurate approximations in a computationally efficient manner. Since the early 1990s a great deal of research effort has been focused on the development of adaptive moving mesh methods for solving problems with steep solutions. These methods generally use a fixed number of mesh points, and the points are continuously

[1] Supported by EPSRC Grant GR/M26459.

0021-9991/02 $35.00

relocated as time evolves so that, at any instant in time, the spatial density of the mesh points is proportional in some sense to the solution variation. In two space dimensions, several methods have been developed to determine mesh movement. The moving finite element method of Miller and Miller [17], for example, determines the mesh movement by minimising the residual for the governing PDEs. One disadvantage of this method is that certain penalty functions have to be introduced to prevent singularity of the mass matrix. Subsequent work by Baines [3] has provided a better understanding of the moving finite element method. Here, we are concerned with the moving mesh strategy developed by Huang and Russell [13, 14]. This is based on the solution of a system of MMPDEs that is derived from the gradient flow equation of a functional that takes account of mesh adaptation, quality control, and smoothness. It has been used successfully for generating both structured and unstructured meshes for a number of problems [9]. Moving mesh methods for problems in three space dimensions have been proposed recently by Li *et al.* [15]. The approach adopted there is based on the theory of harmonic maps that has been exploited by Dvinsky [10] for grid generation.

No general convergence analysis has been produced for moving mesh methods, and insight into the behaviour of the methods has to be obtained by means of numerical experiments. Despite the satisfactory degree of success that has already been attained, further developments are desirable in terms of formulation and implementation before the methods can be widely recognised as robust tools in the computational solution of PDEs. Several developmental steps have been taken since the MMPDEs were introduced: Cao *et al.* [8] have given some insight into the role played by the monitor function, and they have given some guidelines concerning the choice of monitor function. A recent paper by Huang [12] focuses on several practical aspects of formulating and solving MMPDEs. Among other things, he considers spatial balance, scaling invariance, and an algorithm for solving the system of discretised equations.

The objective of this paper is threefold. First, we give a convincing demonstration that the accuracy of the computations is strongly dependent on the choice of monitor function, and we present a monitor function which yields a higher rate of convergence than those which are currently commonly used. Second, we describe a robust and efficient algorithm for solving the system of discretised equations in which the mesh equations are decoupled from the physical PDE. Finally, an efficient time-step control mechanism is incorporated into the algorithm. In an earlier paper [7], we considered these formulation and implementation matters for unsteady PDEs in one space dimension, and the current work may be regarded as an extension of the one-dimensional results. It is hoped that our findings will be a useful addition to the valuable guidelines offered by Huang [12] for problems in two space dimensions.

The structure of the paper is as follows. In Section 2, we present the two-dimensional Burgers equation which will be adopted as a model problem for the computational experiments. We also describe the moving mesh method, including the design of the monitor function and the structure of the decoupled solution algorithm. Section 3 deals with the discretisation of the model problem and the MMPDE and includes details of the time-step control mechanism and the preconditioned iterative method used for solving the linear systems which arise in the solution of the moving mesh equations. The results of the numerical experiments are presented in Section 4, with conclusions and comments following in Section 5.

2. OVERVIEW OF ADAPTIVE ALGORITHM

2.1. *Model Problem*

In this paper, numerical computations are performed on the two-dimensional viscous Burgers equation

$$\frac{\partial u}{\partial t} + \lambda u \frac{\partial u}{\partial x_1} + \mu u \frac{\partial u}{\partial x_2} = \varepsilon \Delta u, \quad (x_1, x_2) \in \Omega = (0, 1)^2, \quad t > 0, \tag{2.1}$$

with initial and boundary conditions chosen such that

$$u(x_1, x_2, t) = (1 + \exp(\phi))^{-1},$$

where

$$\phi = \frac{1}{2\varepsilon}(\lambda x_1 + \mu x_2 - t - t_0).$$

We are interested in the singularly perturbed regime for which $0 < \varepsilon \ll \max(|\lambda|, |\mu|)$. Then the solution has a steep interior planar layer of thickness $O(\varepsilon)$ which propagates across the domain with constant velocity. The constants λ and μ determine the angle θ between the normal to the direction of propagation and the line $x_1 = x_2$ such that

$$\lambda = \cos(\theta) + \sin(\theta) \text{ and } \mu = \cos(\theta) - \sin(\theta).$$

For ease of notation we will often refer to the physical coordinates as $x = x_1$ and $y = x_2$.

2.2. *The Moving Mesh PDE*

In multiple dimensions, a moving mesh method may be considered as a way of constructing an invertible time-dependent grid mapping $\boldsymbol{\xi}(\boldsymbol{x}, t) : \Omega \to \Omega_C$ between the physical domain Ω and a prescribed computational domain Ω_C. In order to avoid potential mesh crossings or foldings we use the map $\boldsymbol{\xi}(\boldsymbol{x}, t)$ rather than the inverse map $\boldsymbol{x}(\boldsymbol{\xi}, t)$ in the subsequent analysis (see, for example, the discussion in [10]). A mesh Π on Ω is then generated as the preimage of a fixed grid Π_C on Ω_C. For the grid mapping to be useful, it must satisfy a number of properties. First, it must be inexpensive to construct, relative to the cost of solving the physical problem: to this end, it is desirable that the mapping $\boldsymbol{\xi}(\boldsymbol{x}, t)$ be obtainable using a standard numerical discretisation on the grid Π_C. Second, Ω_C should be convex and have a simple geometric structure, for example a regular polyhedron. Third, at time t the mesh Π should have nodes clustered in order to resolve steep layers in the physical solution: one would expect to be able to solve (2.1) to a prescribed tolerance with significantly fewer grid nodes for a solution-adaptive grid than for a uniform grid.

In this paper we choose Ω_C to be the unit square $(0, 1) \times (0, 1)$ and Π_C to be a uniform grid with element size $h = N^{-1}$ in both the ξ_1 and ξ_2 directions. For ease of notation, we will often refer to the computational coordinates as $\xi = \xi_1$ and $\eta = \xi_2$.

As introduced in [14], in the moving mesh PDE approach, the mapping $\boldsymbol{\xi}(\boldsymbol{x})$ corresponding to a fixed value of t is chosen in order to minimise the functional

$$I[\boldsymbol{\xi}] = \frac{1}{2} \int_\Omega \sum_{i=1}^{2} (\nabla \xi_i)^T G^{-1} (\nabla \xi_i) \, d\boldsymbol{x}, \tag{2.2}$$

where G is a 2×2 symmetric positive definite matrix referred to as a monitor matrix and ∇ is the gradient operator with respect to $\boldsymbol{x}$. The Euler–Lagrange equations associated with (2.2) are

$$\nabla.(G^{-1}\nabla\xi_i) = 0, \quad i = 1, 2. \tag{2.3}$$

The moving mesh PDEs are then defined to be the modified gradient flow equations

$$\frac{\partial \xi_i}{\partial t} = \frac{P}{\tau}\nabla.(G^{-1}\nabla\xi_i), \quad i = 1, 2, \tag{2.4}$$

derived from (2.3). In Eq. (2.4), $\tau > 0$ is a user-prescribed temporal smoothing parameter, which modifies the time scale. The spatial balance operator P is a positive function of $(\boldsymbol{x}, t)$ chosen such that the mesh movement has a uniform time scale throughout Ω.

In practice, it is convenient to switch the roles of the dependent and independent variables in (2.4) to give

$$\tau\frac{\partial \boldsymbol{x}}{\partial t} = P\left(\sum_{i,j=1}^{2}(\boldsymbol{a}^i.G^{-1}\boldsymbol{a}^j)\frac{\partial^2 \boldsymbol{x}}{\partial\xi_i\partial\xi_j} - \sum_{i,j=1}^{2}\left(\boldsymbol{a}^i.\frac{\partial G^{-1}}{\partial\xi_j}\boldsymbol{a}^j\right)\frac{\partial \boldsymbol{x}}{\partial\xi_i}\right), \tag{2.5}$$

where $\boldsymbol{a}^i = \nabla\xi_i$.

To complete the specification of (2.5), appropriate and meaningful boundary conditions $\boldsymbol{g}(\boldsymbol{\xi}, t)$, $\boldsymbol{\xi} \in \partial\Omega_C$ must be specified. These boundary conditions are obtained using a one-dimensional moving mesh equation, as described in [14]. We briefly summarise the approach here. Let Γ be a segment of the boundary $\partial\Omega$ and let Γ_C be the corresponding segment of $\partial\Omega_C$. Let $s \in (0, l)$ and $\zeta \in (0, l_C)$ denote arclength along Γ and Γ_C, respectively. Then the mapping $s : \Gamma_C \to \Gamma$ is determined by solving the one-dimensional equation

$$\begin{gathered}\tau\frac{\partial s}{\partial t} = P\frac{\partial}{\partial\zeta}\left(M\frac{\partial s}{\partial\zeta}\right), \quad \zeta \in (0, l_C), \\ s(0) = 0, \quad s(l_C) = l.\end{gathered} \tag{2.6}$$

The monitor function M is derived by projecting the two-dimensional monitor matrix G along the boundary. Specifically, if $\hat{\boldsymbol{s}}(\zeta)$ denotes the unit tangent vector along the boundary, then

$$M(\zeta, t) = \hat{\boldsymbol{s}}^T G\hat{\boldsymbol{s}}.$$

The choice of spatial balance operator P in (2.5) and (2.6) is crucial to the reliability of the moving mesh method. In [14], the operator is chosen as $P = 1/\sqrt[D]{\det(G)}$, where D is the dimension of the spatial domain Ω. This choice is motivated by the work of Dvinsky [10] on generating unique grid functions using the theory of harmonic mappings. More recently, Huang [12] proposed choosing P in order to limit the variation over the domain of the coefficients in (2.5) and (2.6). This is done by setting

$$P = \left(\sum_{i=1}^{2} a_{i,i}^2 + b_i^2\right)^{-\frac{1}{2}}, \quad a_{i,j} = \boldsymbol{a}^i.G^{-1}\boldsymbol{a}^j, \quad b_i = -\sum_{j=1}^{2}\left(\boldsymbol{a}^i.\frac{\partial G^{-1}}{\partial\xi_j}\boldsymbol{a}^j\right) \tag{2.7}$$

in (2.5), which simplifies to

$$P = (M^2 + (M_\xi)^2)^{-\frac{1}{2}} \tag{2.8}$$

in (2.6). With this choice of P, the coefficients of (2.5) and (2.6) are all of size O(1). In this paper, we choose the spatial balance parameter to be as in (2.7) and (2.8).

2.3. *Choice of Monitor Matrix*

The selection of an appropriate monitor matrix is fundamental to the success of mesh adaptation. In this paper, we shall consider the monitor matrix proposed by Winslow [19],

$$G = \begin{bmatrix} w & 0 \\ 0 & w \end{bmatrix}, \tag{2.9}$$

where w is a strictly positive, integrable function called a monitor function. We will perform numerical experiments for two distinct choices: the popular arclength (AL) monitor function

$$w(\boldsymbol{x}, t) = \sqrt{1 + |\boldsymbol{\nabla} u(\boldsymbol{x}, t)|^2}, \tag{2.10}$$

and a generalisation of the BM monitor function proposed by Beckett and Mackenzie [4–6],

$$w(\boldsymbol{x}, t) = \alpha(t) + |\boldsymbol{\nabla} u(\boldsymbol{x}, t)|^{\frac{1}{m}}, \tag{2.11}$$

where m is a positive constant which acts as a smoothing factor on the solution gradient—the analysis presented in [4–6] suggests that $m = 2$ is an appropriate value for a second-order approximation and, as such, will be used for all numerical experiments in this paper. The function $\alpha(t)$ is strictly positive and regulates the grid, ensuring that mesh starvation does not occur in regions of the domain where there is little or no spatial variation in the solution. In [4–6] the authors show that for steady boundary value problems in one dimension, α may be chosen in order to maintain a constant ratio of points inside boundary layers to points in the rest of the domain. The validity of this result was demonstrated for a more general class of singular perturbation problems in Beckett *et al.* [7]. In this paper we generalise the construction of α to problems in more than one space dimension by defining

$$\alpha(t) = \frac{1}{\text{meas}(\Omega)} \int_\Omega |\boldsymbol{\nabla} u(\boldsymbol{x}, t)|^{\frac{1}{m}} \, \mathrm{d}\boldsymbol{x}.$$

It is shown in [7] that in one space dimension this choice of $\alpha(t)$ ensures that approximately half of the grid points are located outside the steep layers. This obvious extension to two space dimensions is designed to control the intensity of mesh concentration. With α selected in this way, the floor on the BM monitor matrix is adjusted automatically in proportion to the measure of the (smoothed) solution gradient. Thus for problem (2.1), as the solution front propagates across the domain for $t \in (0, 0.75)$, increasing in width, the floor α also increases to maintain the node density outwith the layer region. Conversely, for $t > 0.75$, α decreases as the front width decreases.

```
pass = 0; x = x^n; u = u^n;
while (pass < MAXPASS),
   ox = x;
   Evaluate monitor matrix for (x, u);
   Integrate MMPDE forward to obtain new
      grid x at time t_{n+1};
   Integrate PPDE forward to obtain new solution u at time t_{n+1};
   if (||x − ox||_{l∞} < MTOL), break;
   pass = pass + 1;
end while;
x^{n+1} = x, u^{n+1} = u;
```

FIG. 1. Decoupled algorithm for multipass moving mesh method.

2.4. *Iterative Solution Algorithm*

Given the current approximation $(\boldsymbol{x}^n, \boldsymbol{u}^n)$ at time t_n and a time step Δt_n, we now describe the procedure for integrating forward in time to the new time level $t_{n+1} = t_n + \Delta t_n$. One approach would be to couple the discretised systems for the moving mesh equation (MMPDE) and the physical equation (PPDE) together to give a single nonlinear system for $(\boldsymbol{x}^{n+1}, \boldsymbol{u}^{n+1})$. However, there are a number of disadvantages to this approach. First, the size of the resulting system would be large and even for moderate grid densities may be prohibitive. Second, this approach does not easily admit different convergence criteria for the mesh and physical solution. As noted in Babuška and Rheinboldt [2], it is not necessary to compute the mesh with the same level of accuracy as the physical solution. In the interests of reducing computational costs, one would like to exploit this property. Third, as mentioned previously, a user may wish to have control over the discretisation of the physical problem and such flexibility is severely restricted by coupling the unknowns together into one large nonlinear system.

We therefore integrate forward in time in an iterative manner, solving for the grid and physical solution alternately. The algorithm is summarised in Fig. 1 (described in a pseudocode). Specific details of the individual components will be provided in Section 3: here, we only discuss features of the overall algorithm. At the start of each pass the current mesh is stored—in addition to being utilised in the mesh convergence test it will also be required in a simple mesh error indicator in the adaptive time-step selection procedure. Then, after approximating the monitor matrix, the mesh is integrated forward to the new time level. This involves first integrating each boundary mesh equation forward to provide boundary conditions for the interior mesh equation. Once we have the mesh equation at the new time level, we integrate forward the physical PDE. This iteration is repeated until either the distance (measured in the l_∞ norm) between two successive grids falls below a specified tolerance MTOL or the maximum number of iterations MAXPASS is reached. For efficiency reasons it is strongly desirable to perform only a small number of iterations at each time step—in this paper MAXPASS $= 4$.

The approach described in [12] is equivalent to setting MAXPASS $= 1$ in the algorithm, performing a single iteration at each time step. However, as demonstrated in [7], this results in the grid lagging behind the solution of the physical PDE and severely impedes the size of time step which may be taken. Because of this, we advocate the use of a four-pass scheme, that is, setting MAXPASS $= 4$. If the grid converges, the loop will be stopped before four passes have been completed. However, for time-dependent problems this is seldom observed in practice.

3. DETAILS OF NUMERICAL ALGORITHM

3.1. *Discretisation of MMPDE*

For a Winslow-type monitor matrix (2.9), the MMPDE (2.5) simplifies to

$$\begin{aligned} \frac{\partial \boldsymbol{x}}{\partial t} &= P(a\boldsymbol{x}_{\xi\xi} + b\boldsymbol{x}_{\xi\eta} + c\boldsymbol{x}_{\eta\eta} + d\boldsymbol{x}_{\xi} + e\boldsymbol{x}_{\eta}), \quad (\xi,\eta) \in \Omega_C, \\ \boldsymbol{x}(\xi,\eta,t) &= \boldsymbol{g}(\xi,\eta,t), \quad (\xi,\eta) \in \partial\Omega_C, \end{aligned} \tag{3.1}$$

where

$$\begin{aligned} a &= \frac{1}{\tau w}\frac{\left(x_\eta^2 + y_\eta^2\right)}{J^2}, \quad b = -\frac{2}{\tau w}\frac{(x_\xi x_\eta + y_\xi y_\eta)}{J^2}, \quad c = \frac{1}{\tau w}\frac{\left(x_\xi^2 + y_\xi^2\right)}{J^2}, \\ d &= \frac{1}{\tau J^2}\left(-\frac{\partial}{\partial\xi}(w^{-1})\left(x_\eta^2 + y_\eta^2\right) + \frac{\partial}{\partial\eta}(w^{-1})(x_\xi x_\eta + y_\xi y_\eta)\right), \\ e &= \frac{1}{\tau J^2}\left(\frac{\partial}{\partial\xi}(w^{-1})(x_\xi x_\eta + y_\xi y_\eta) - \frac{\partial}{\partial\eta}(w^{-1})\left(x_\xi^2 + y_\xi^2\right)\right), \end{aligned}$$

and $J = x_\xi y_\eta - x_\eta y_\xi$ is the Jacobian of the grid mapping. Equation (3.1) is a coupled system of two equations for the mappings x and y. However, if the system is linearised by freezing the coefficients a, b, c, d, and e, we may decouple (3.1) into two scalar equations for x and y.

We now describe the discretisation of the system (3.1), focusing on the equation for x: the equation for y may be treated similarly. For ease of notation, we start by defining the following difference operators:

$$\begin{aligned} D_{\xi\xi}x_{i,j} &= N^2(x_{i+1,j} - 2x_{i,j} + x_{i-1,j}), \\ D_{\xi\eta}x_{i,j} &= \frac{N^2}{4}(x_{i+1,j+1} - x_{i-1,j+1} - x_{i+1,j-1} + x_{i-1,j-1}), \\ D_{\xi}^0 x_{i,j} &= \frac{N}{2}(x_{i+1,j} - x_{i-1,j}). \end{aligned}$$

Operators in the η direction are defined similarly.

To discretise (3.1) on the uniform computational grid Π_C, we freeze the coefficients and replace the spatial derivatives by appropriate second-order central difference operators. Numerical integration from time t_n to $t_n + \Delta t_n$ is effected using an implicit backward Euler method. This gives rise to the system

$$\begin{aligned} \tau\frac{x_{i,j}^{n+1} - x_{i,j}^n}{\Delta t_n} &= P_{i,j}\left(a_{i,j}^n D_{\xi\xi}x_{i,j}^{n+1} + b_{i,j}^n D_{\xi\eta}x_{i,j}^{n+1} + c_{i,j}^n D_{\eta\eta}x_{i,j}^{n+1}\right. \\ &\qquad \left. + d_{i,j}^n D_{\xi}^0 x_{i,j}^{n+1} + e_{i,j}^n D_{\eta}^0 x_{i,j}^{n+1}\right), \quad 1 < i, j < N, \\ x_{i,j} &= g_1(\xi_i, \eta_j), \quad (\xi_i, \eta_j) \in \partial\Omega_C. \end{aligned} \tag{3.2}$$

In Eq. (3.2), the coefficients a, b, c, d, and e are discretised at the old time level t_n using central differences. The monitor function w is approximated by

$$w_{i,j} = \left(1 + \left(D_x^0 u_{i,j}\right)^2 + \left(D_y^0 u_{i,j}\right)^2\right)^{\frac{1}{2}},$$

for (2.10), and

$$w_{i,j} = \alpha + \left(\left(D_x^0 u_{i,j}\right)^2 + \left(D_y^0 u_{i,j}\right)^2\right)^{\frac{1}{2m}},$$

where

$$\alpha = \frac{1}{\text{area}(\Omega)} \sum_{i,j=0}^{N-1} \left(\left(D_x^0 u_{i,j}\right)^2 + \left(D_y^0 u_{i,j}\right)^2\right)^{\frac{1}{2m}} \text{area}(\Omega_{i,j}),$$

for (2.11), with

$$D_x^0 u_{i,j} := \left(D_\xi^0 x_{i,j}\right)^{-1} D_\xi^0 u_{i,j} + \left(D_\eta^0 x_{i,j}\right)^{-1} D_\eta^0 u_{i,j}$$

and similarly for D_y^0.

The one-dimensional MMPDE (2.6) applied on each boundary segment must also be discretised. This is done in a manner analogous to the above and gives

$$\tau \frac{s_j^{n+1} - s_j^n}{\Delta t_n} = P_j D_\zeta^0 \left(M_j D_\zeta^0 s_j^{n+1}\right). \tag{3.3}$$

3.2. *Discretisation of Physical PDE*

In this section, we shall outline the discretisation of (2.1). There is considerable flexibility in how one solves the physical problem and this flexibility can be attributed to the iterative solution procedure described in Section 2.4, in which the mesh and physical solution are integrated forward in time in a decoupled manner.

Using the grid mapping $\boldsymbol{x}(\boldsymbol{\xi}, t)$, problem (2.1) may be transformed into computational coordinates to give

$$J\dot{v} + \hat{F}_\xi + \hat{H}_\eta = (\hat{K} v_\xi)_\xi + (\hat{L} v_\xi)_\eta + (\hat{L} v_\eta)_\xi + (\hat{M} v_\eta)_\eta, \tag{3.4}$$

where $v(\xi, \eta, t) = u(x(\xi, \eta), y(\xi, \eta), t)$ and the coefficients are defined as

$$\begin{aligned}
\hat{F} &= \frac{\lambda}{2} v^2 (y_\eta - x_\eta) + v(\dot{y} x_\eta - \dot{x} y_\eta), \\
\hat{H} &= \frac{\mu}{2} v^2 (x_\xi - y_\xi) + v(\dot{x} y_\xi - \dot{y} x_\xi), \\
\hat{K} &= \frac{\varepsilon\left(x_\eta^2 + y_\eta^2\right)}{J}, \\
\hat{L} &= -\frac{\varepsilon(x_\xi x_\eta + y_\xi y_\eta)}{J}, \\
\hat{M} &= \frac{\varepsilon\left(x_\xi^2 + y_\xi^2\right)}{J}.
\end{aligned}$$

If the solution-adaptive grid is to be effective, this transformation should smooth out any steep layers and eliminate rapid solution changes in space, thus making problem (3.4) amenable to a classical approximation technique applied on a uniform grid. Thus motivated, we discretise Eq. (3.4) on the uniform computational domain Π_C using second-order central

differences in space and integrate forward in time using a second-order singly diagonally implicit Runge–Kutta method (SDIRK2), with Butcher array

$$\begin{array}{c|c} \boldsymbol{c} & A \\ \hline & \boldsymbol{b}^T \end{array} = \begin{array}{c|cc} \gamma & \gamma & 0 \\ 1 & 1-\gamma & \gamma \\ \hline & 1-\gamma & \gamma \end{array},$$

where $\gamma = (2 - \sqrt{2})/2$.

Application of SDIRK2 to (3.4) gives rise to two nonlinear systems of equations,

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(t_n + \gamma\,\Delta t_n, \boldsymbol{v}^n + \gamma\,\Delta t_n \boldsymbol{k}_1), \\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n + \Delta t_n, \boldsymbol{v}^n + (1-\gamma)\Delta t_n \boldsymbol{k}_1 + \gamma\,\Delta t_n \boldsymbol{k}_2), \end{aligned} \tag{3.5}$$

where $\boldsymbol{f}$ is such that $\dot{\boldsymbol{v}} = \boldsymbol{f}(\boldsymbol{v})$. From (3.5), the approximation at time t_{n+1} is then obtained as

$$\boldsymbol{v}^{n+1} = \boldsymbol{v}^n + \Delta t_n((1-\gamma)\boldsymbol{k}_1 + \gamma \boldsymbol{k}_2).$$

The solution of (3.5) is found using a Newton iteration, the convergence criterion for which is that the size of the residual measured in the l_∞ norm falls below a prescribed tolerance $KTOL = \varepsilon \times 10^{-6}$, where ε is the small parameter in (2.1). In our experience, this method generally converges in fewer than four iterations.

The SDIRK2 time integrator has an embedded scheme which is first-order accurate in time, given by

$$\hat{\boldsymbol{v}}^{n+1} = \boldsymbol{v}^n + \Delta t_n \boldsymbol{k}_1.$$

Since $\boldsymbol{k}_1$ is already known, this embedded solution is obtained at no extra computational cost. A simple error indicator is therefore obtained by measuring the difference between the two approximations,

$$\mathrm{ERR} = \|\boldsymbol{v}^{n+1} - \hat{\boldsymbol{v}}^{n+1}\|_{l_\infty}.$$

This is included in the time-step control mechanism described in Section 3.4.

3.3. *Solution of Discrete Mesh Equations*

At each time step, we must solve the discretised mesh equations (3.2) for x (and a similar set for y). That is, we have to solve the linear systems

$$Q\boldsymbol{x} = \boldsymbol{z}_1, \quad Q\boldsymbol{y} = \boldsymbol{z}_2 \tag{3.6}$$

for the new grid coordinates, where Q is of size $(N-1)^2 \times (N-1)^2$. Using the subscript notation

$$\begin{bmatrix} NW & N & NE \\ W & C & E \\ SW & S & SE \end{bmatrix}$$

to represent the relative positions of nodes in a standard nine-point stencil, the entries of a

row of Q (for nodes away from the boundary) are given by

$$P_c \begin{bmatrix} \frac{q}{4} b_{NW} & -q\left(c_N + \frac{h}{2} e_N\right) & -\frac{q}{4} b_{NE} \\ -q\left(a_W - \frac{h}{2} d_W\right) & 1 + 2q(a_C + c_C) & -q\left(a_E + \frac{h}{2} d_E\right) \\ -\frac{q}{4} b_{SW} & -q\left(c_S - \frac{h}{2} e_S\right) & \frac{q}{4} b_{SE} \end{bmatrix},$$

where $q = N^2 \Delta t_{n+1}$ and P_C is the spatial balancing parameter corresponding to the central node. Because the mesh equations have to be solved up to MAXPASS times at every time step, it is essential to have an efficient solution algorithm for (3.6).

As N increases, direct solution of these systems quickly becomes impractical. We therefore adopt an iterative approach, which has the added attraction that the solution at the previous time step can be used to provide a good initial guess at each stage. The coefficient matrix Q in (3.6) is nonsymmetric so the popular Bi-CGSTAB method [18] is an appropriate choice. The convergence rate of this algorithm, which depends on the eigenvalue spectrum of the coefficient matrix, can be improved by introducing the concept of preconditioning. Theoretically, this is equivalent to replacing Q with a preconditioned matrix $\hat{Q}^{-1}Q$ whose eigenvalue spectrum facilitates faster Bi-CGSTAB convergence, by having more "clustered" eigenvalues. The choice $\hat{Q} = Q$ is perfect in terms of clustering the eigenvalues: however, as the Bi-CGSTAB algorithm involves solving a system with $\hat{Q}$ as coefficient matrix at each (half) iteration, this is not a practical idea. Clearly, there is a trade-off between gaining fast Bi-CGSTAB convergence and solving systems involving $\hat{Q}$ cheaply.

Here we will compare the performance of two common preconditioners. The first is diagonal (Jacobi) preconditioning, that is, setting $\hat{Q} = \mathrm{diag}(Q)$, which scales the eigenvalues of Q in a relatively simple way. It is extremely cheap to implement and is sometimes surprisingly effective. A plot of the spectrum of a typical matrix Q (with $N = 32$) is shown in Fig. 2a: the effect of applying diagonal scaling is seen in Fig. 2b. The second preconditioner considered is an incomplete LU (ILU) factorisation (see, for example, [1, 16]), where sparse lower and upper triangular matrices L and U are found such that $LU \simeq Q$. If $\hat{Q} = LU$ is a good approximation to Q, the eigenvalues of the preconditioned system are well clustered around unity. At the same time, forward and backsubstitution offer an effective way of implementing the preconditioner solves required by Bi-CGSTAB. The major expense comes from forming the factors themselves, but we note that this need only be done once per pass, as the x and y systems have the same coefficient matrix. We consider the type of ILU factorisation where the level of approximation is regulated by choosing a drop tolerance δ. After each column of L and U has been calculated, all entries in that column which are smaller in magnitude than δ are "dropped" from L or U. The amount of fill-in depends on the choice of δ; for example, choosing $\delta = 0$ would produce the full LU factorisation of Q. In all of our tests, the method has performed well with a drop tolerance of 1×10^{-4} times the norm of the relevant column of Q. The effect of this preconditioner on the eigenvalues of Q is shown in Fig. 2c. The eigenvalues of the preconditioned system are arranged in a very tight cluster around unity. This is ideal for Bi-CGSTAB, and we would expect the iterative method to converge in a very small number of iterations.

In the following numerical results, the iteration counts and CPU times shown are average values over the first 60 time steps for the given problem. Our numerical tests have shown that the performance of all of the methods is almost independent of many of the parameters

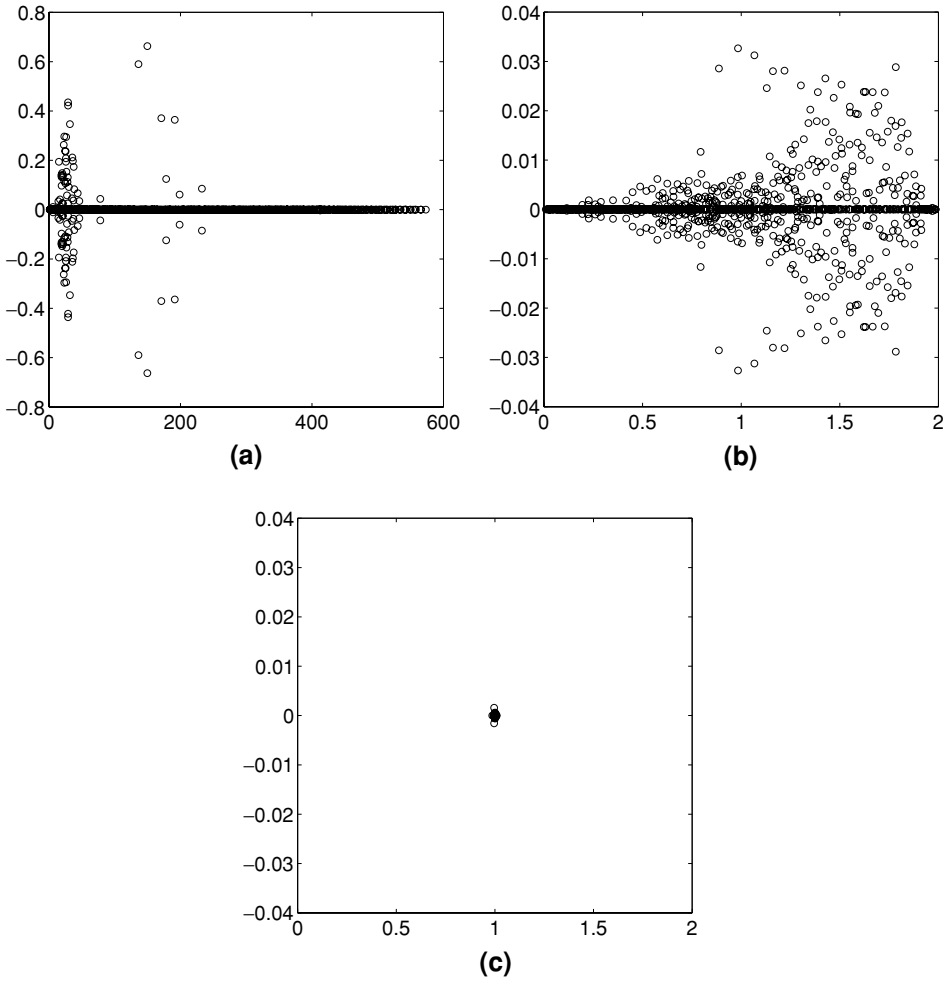

FIG. 2. Eigenvalue spectra of a typical matrix Q. (a) No preconditioning, (b) diagonal scaling, (c) ILU.

in the overall solution algorithm, for example choice of monitor function, spatial balancing parameter, front direction, and number of passes. We therefore only present results for a sample problem with $\varepsilon = 5.0 \times 10^{-3}$, BM monitor matrix, MAXPASS $= 4$, and $\theta = 0$, where θ gives the direction of propagation as defined at (2.1). These are the parameters which give rise to the matrix Q used in Fig. 2 (with $N = 32$). Table I contains information on the average number of Bi-CGSTAB iterations per linear system k, the average CPU time per Bi-CGSTAB iteration t_{iter}, and the average CPU time taken to form the preconditioner t_{pre}. The resulting average total time spent solving the linear systems (3.6) at each pass is $t_{\text{pass}} = t_{\text{pre}} + 2kt_{\text{iter}}$. For completeness, we also include CPU times for using a direct solver. Clearly, ILU provides a relatively cheap and efficient way of solving the MMPDE linear systems at each pass. This is due to its excellent eigenvalue-clustering property, as highlighted above. Finally, we note that as changes in the grid from pass to pass are usually relatively small, we also investigated the possibility of using ILU factors "frozen" from the previous pass rather than recomputing them (as is clear from Table I, calculating the factors is the main expense of ILU). However, at least for the size of problem considered here, the

TABLE I
Iteration Counts and CPU Times (Averaged over the First 60 Passes) for Sample Problem with $\varepsilon = 5.0 \times 10^{-3}$, BM Monitor Matrix, and MAXPASS = 4

	Diagonal scaling				ILU				Direct
N	k	t_{iter}	t_{pre}	t_{pass}	k	t_{iter}	t_{pre}	t_{pass}	t_{pass}
16	16.10	0.0062	0.00	0.20	0.88	0.0227	0.02	0.05	0.09
32	31.29	0.0163	0.00	1.02	0.99	0.0707	0.14	0.28	1.21
64	60.55	0.0639	0.00	7.74	1.06	0.3302	1.08	1.78	9.37
128	103.33	0.3024	0.002	62.52	1.38	1.6667	8.94	13.54	75.26
256	159.61	1.3064	0.07	417.09	2.01	7.2139	68.53	97.53	799.54

increase in the number of Bi-CGSTAB iterates required outweighed the benefits of reducing factorisation times in this way. The timings presented in Section 4 are therefore based on calculating new ILU factors at each pass. This has proved to be a robust and efficient method for all problems tested.

3.4. *Time-Step-Size Control*

Essential to the success of a solution-adaptive scheme is the implementation of an effective time-step control mechanism. This requires the availability of a reliable error indicator. As described in Section 3.2, the SDIRK2 scheme has an embedded first-order SDIRK time integrator which gives rise to an error indicator ERR with no additional computational effort. The user prescribes an absolute tolerance ERRTOL reflecting the desired level of accuracy. If at time level t_n ERR > ERRTOL, the solution at t_{n+1} is rejected and is recomputed with a smaller time step. If ERR $\leq$ ERRTOL, then the solution is accepted and integration continues to the new time level $t_{n+2} = t_{n+1} + \Delta t_{n+1}$ with

$$\Delta t_{n+1} = \Delta t_n \min\left(\text{MAXFAC}, \max\left(\text{MINFAC}, \eta\left(\frac{\text{ERR}}{\text{ERRTOL}}\right)^{\frac{1}{2}}\right)\right). \tag{3.7}$$

A detailed description of this time-step control mechanism is given in Hairer and Wanner [11]. While ERR gives an indication of the error incurred in the lower-order approximation $\hat{\boldsymbol{v}}$, in practice we carry forward the second-order solution $\boldsymbol{v}$.

The approach outlined above is standard. However, as observed in [7], time-step control based solely on the physical solution does not constitute a reliable mechanism for a moving mesh method. This is because the error indicator ERR is insensitive to small inaccuracies in the grid point locations—this is a further consequence of the observations in [2]. Thus, over a number of time steps, the grid drifts away from features of the solution without degrading the error indicator. This problem is eliminated by including a mesh error indicator in the time-step control mechanism. When MAXPASS > 1, a simple indication of the accuracy of the grid is given by

$$\text{MESHERR} = \|\boldsymbol{x}^{n+1} - \boldsymbol{ox}\|_{l_\infty},$$

where $\boldsymbol{ox}$ denotes the mesh on the previous pass of the decoupled algorithm (see Fig. 1). Again, the user prescribes a parameter MESHTOL which should be linearly dependent on the singular perturbation parameter ε. If at time level t_n MESHERR > MESHTOL, the

solution at time t_{n+1} is rejected and recomputed with a smaller step size. If MESHERR $\leq$ MESHTOL, the solution at time t_{n+1} is accepted and integration proceeds to the new time level $t_{n+2} = t_{n+1} + \Delta t_{n+1}$ with

$$\Delta t_{n+1} = \Delta t_n \min\left(\text{MAXFAC}, \max\left(\text{MINFAC}, \left(\frac{\log(\text{MESHERR})}{\log(\text{MESHBAL})}\right)\right)\right), \tag{3.8}$$

where MESHBAL is a user-prescribed parameter satisfying MESHBAL $<$ MESHTOL. In practice, a value for Δt_{n+1} is computed using both (3.7) and (3.8), and the smaller of the two is adopted.

4. NUMERICAL EXPERIMENTS

In this section, we present numerical results that demonstrate properties of the MMPDE algorithm which have been described earlier.

The first set of results is related to the initial grid, which is computed by integrating the discretised MMPDE (3.2) and (3.3) forward in time to a steady state (MTOL $= 0.1\varepsilon$). The monitor matrix is approximated from the given initial solution using the discretisation of either the AL monitor function (2.10) or the BM monitor function (2.11). We tabulate the error incurred in interpolating the initial data using a piecewise bilinear function on the steady-state grid. In order to appreciate layers in the solution, the error is measured in the L_∞ norm, which we approximate by sampling the error on a 10×10 uniform grid on each mesh cell $(\Omega_C)_{i,j} := [\xi_i, \xi_{i+1}] \times [\eta_j, \eta_{j+1}]$ in computational space. In Table II we present results for the AL and BM monitor functions. We observe that for the BM monitor function, as the grid density N is increased, the error converges at a rate above second order. In contrast, for the AL monitor function the rate of convergence with N is sub-second order. A consequence of this is a difference of an order of magnitude in the interpolation error on the most dense grid, $N = 256$.

The second set of results is for the time-dependent problem (2.1): from the initial grid we integrate forward in time to $T = 0.5$ using the algorithm in Fig. 1. We compare the performance of the BM and AL monitor functions, using both the standard one-pass approach (MAXPASS $= 1$) described in [12] and the multipass approach (MAXPASS $= 4$) introduced for one-dimensional problems in [7]. For each of these four cases, we consider a range of values of N and appropriate values of ETOL. We tabulate the maximum value of the global l_∞ error over all time steps. We also note the computational cost, measured in CPU seconds and normalised by the cheapest run (the BM monitor matrix with MAXPASS $= 1$ on a

TABLE II
Interpolation Error on Initial Grids for $\varepsilon = 5.0 \times 10^{-3}$

	BM monitor		AL monitor	
N	$\|e\|_{L_\infty}$	Conv. rate	$\|e\|_{L_\infty}$	Conv. rate
8	2.936×10^{-1}	—	3.050×10^{-1}	—
16	5.208×10^{-2}	2.50	7.546×10^{-2}	2.02
32	1.041×10^{-2}	2.32	1.987×10^{-2}	1.92
64	1.851×10^{-3}	2.50	6.992×10^{-3}	1.51
128	3.272×10^{-4}	2.50	2.254×10^{-3}	1.63
256	6.806×10^{-5}	2.27	6.806×10^{-4}	1.73

grid of density 16×16). In all runs, the following parameters are used: MINFAC $= 0.1$, MAXFAC $= 2$, $\eta = 0.75$, MESHBAL $= 1.5\varepsilon$, and MESHTOL $= 2\varepsilon$. Our numerical experiments have shown that the performance of the algorithm is insensitive to moderate variation in these parameters.

Tables III and IV show results for the multipass algorithm using the BM and AL monitor functions, respectively. We see that as for the initial grid calculations, the error converges at a rate significantly higher for the BM monitor function than for the AL monitor function. This leads to a difference of more than an order of magnitude in the global error on the finest grid, $N = 256$. In Beckett *et al.* [7] we observed that for a one-dimensional problem, the l_∞ error converged at a rate of $O(N^{-2})$ and $O(N^{-1})$ for the BM and AL monitor functions, respectively. These rates of convergence are close to those observed in Tables III and IV.

Tables V and VI show results equivalent to those described above but by using the single-pass algorithm. It is immediately apparent that the cost of an individual time step is cheaper for the one-pass approach than for a multipass approach. However, with MAXPASS $= 1$ the grid at the new time level is generated from solution data at the previous level. This causes the grid to lag behind the solution of the physical PDE and severely restricts the time-step size. For both the BM and the AL monitor matrices we see that for each particular parameter pair N and ETOL, the one-pass method is between two and four times cheaper than the corresponding run for the four-pass algorithm. However, in contrast, for each run the global error is larger for the single-pass scheme. Once again, Tables V and VI show that a higher rate of convergence is achieved for the BM monitor function.

The information in Tables III–VI is summarised in Fig. 3, where we show a log–log plot of the l_∞ error against the cost (CPU time) for each of the four cases. The notation system is that AL1 represents the AL monitor function with MAXPASS $= 1$ and so on, in an obvious manner. The thick plain line is a reference of slope -1. For both pass strategies, the cost graph for the BM monitor function is steeper than for the AL monitor function, confirming the improved rate of convergence which has been observed. Furthermore, the BM monitor function gives quantitatively better accuracy. Figure 3 also clearly demonstrates the cost advantage which may be attributed to the multipass algorithm. Although the cost of an individual time step is much cheaper for MAXPASS $= 1$, one is restricted from taking that large time steps because the grid lags behind the solution of the physical PDE. Hence, the overall cost of the one-pass strategy is significantly higher, independent of the choice of monitor function.

Figure 4 shows numerical results for the solution of problem (2.1) on the time interval $t \in [0, 1.5]$ using the BM monitor matrix, with parameters MAXPASS $= 4$, $N = 32$, and

TABLE III
Four-Pass Algorithm with BM Monitor Matrix: $\varepsilon = 5.0 \times 10^{-3}$, $t \in [0, 0.5]$, $\theta = 0.0$, $\tau = 0.1$

N	ETOL	$\|e\|_{l_\infty}$	Conv. rate	Cost
16	8.00×10^{-3}	2.441×10^{-2}	—	4.7
32	2.00×10^{-3}	7.326×10^{-3}	1.74	17.8
64	5.00×10^{-4}	2.384×10^{-3}	1.62	82.6
128	1.25×10^{-4}	5.450×10^{-4}	2.13	936.9
256	3.00×10^{-4}	1.430×10^{-4}	1.93	6026.2

TABLE IV
Four-Pass Algorithm with AL Monitor Matrix: $\varepsilon = 5.0 \times 10^{-3}$, $t \in [0, 0.5]$, $\theta = 0.0$, $\tau = 0.1$

N	ETOL	$\|e\|_{l_\infty}$	Conv. rate	Cost
16	8.00×10^{-3}	3.117×10^{-2}	—	8.1
32	2.00×10^{-3}	2.074×10^{-2}	0.59	26.7
64	5.00×10^{-4}	1.207×10^{-2}	0.78	227.0
128	1.25×10^{-4}	5.726×10^{-3}	1.08	4514.2
256	3.00×10^{-5}	2.138×10^{-3}	1.42	37551.0

TABLE V
One-Pass Algorithm with BM Monitor Matrix: $\varepsilon = 5.0 \times 10^{-3}$, $t \in [0, 0.5]$, $\theta = 0.0$, $\tau = 0.1$

N	ETOL	$\|e\|_{l_\infty}$	Conv. rate	Cost
16	8.00×10^{-3}	4.487×10^{-2}	—	1.0
32	2.00×10^{-3}	1.804×10^{-2}	1.32	4.6
64	5.00×10^{-4}	6.140×10^{-3}	1.55	35.4
128	1.25×10^{-4}	1.809×10^{-3}	1.76	472.4
256	3.00×10^{-5}	4.849×10^{-4}	1.90	4171.6

TABLE VI
One-Pass Algorithm with AL Monitor Matrix: $\varepsilon = 5.0 \times 10^{-3}$, $t \in [0, 0.5]$, $\theta = 0.0$, $\tau = 0.1$

N	ETOL	$\|e\|_{l_\infty}$	Conv. rate	Cost
16	8.00×10^{-3}	8.257×10^{-2}	—	1.5
32	2.00×10^{-3}	6.058×10^{-2}	0.45	8.6
64	5.00×10^{-4}	3.153×10^{-2}	0.94	108.0
128	1.25×10^{-4}	1.387×10^{-2}	1.18	1796.9
256	3.00×10^{-5}	5.301×10^{-3}	1.39	12881.7

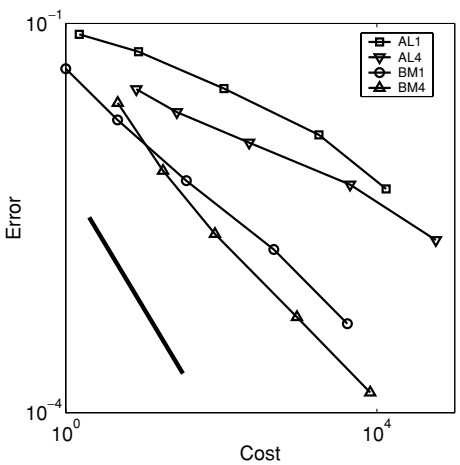

FIG. 3. Cost comparison of BM and AL monitor functions using various pass strategies.

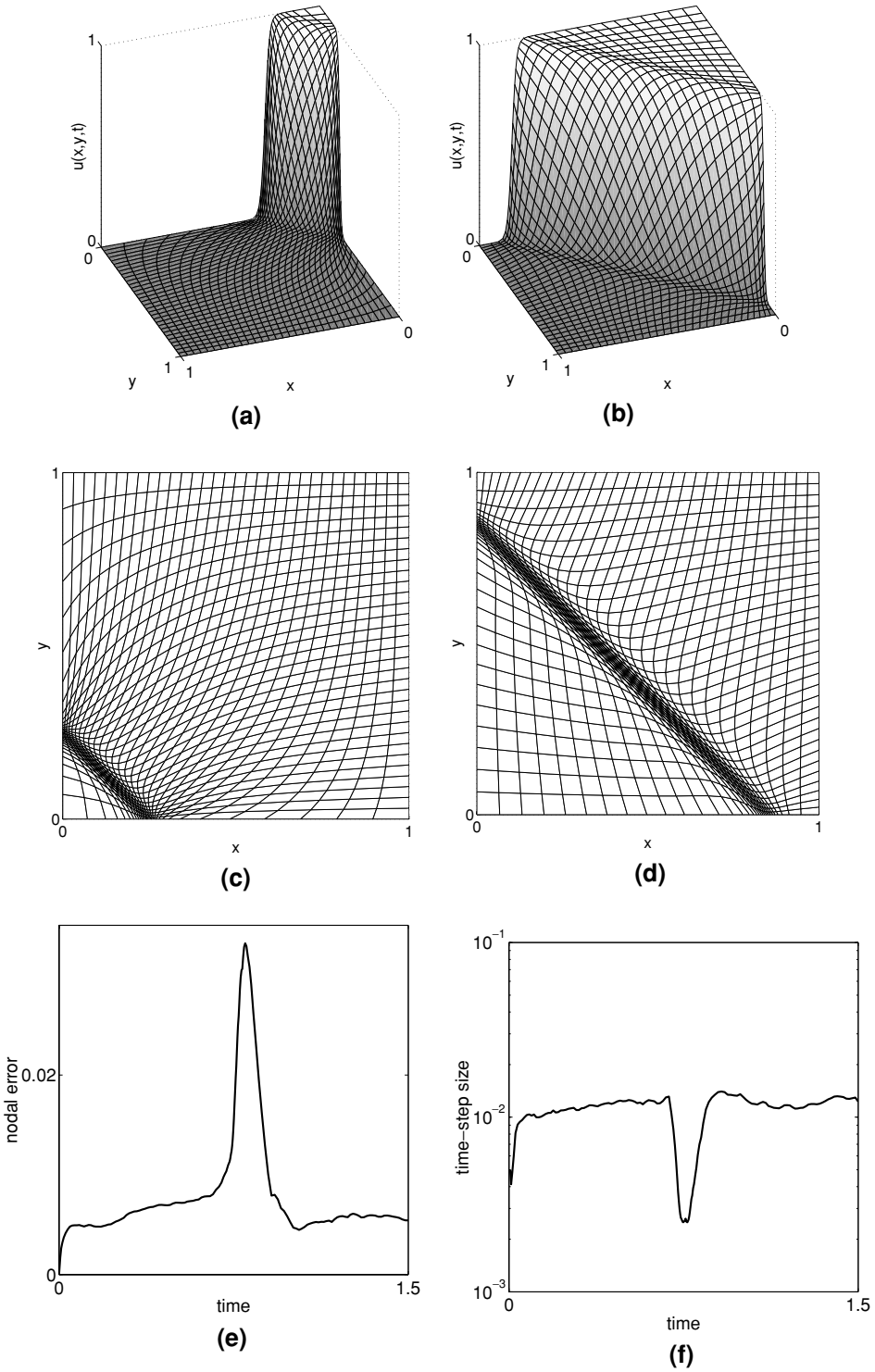

FIG. 4. Solution of the viscous Burgers equation, $\varepsilon = 5.0 \times 10^{-3}$, $t_0 = 0.25$, and $N = 32$, using BM monitor matrix. Solution is plotted at time $t = 0$ and 0.6.

ETOL $= 2.0 \times 10^{-3}$. The top and middle rows of plots show two snapshots of the solution and corresponding grid at $t = 0$ and 0.6. We observe significant adaptation of mesh points into the layer, leading to good resolution of the planar front and no visible oscillations in the solution. The grid is clustered appropriately both in the interior and at the boundary of the domain. The bottom row of plots presents the history of the nodal error and time-step size. As can be seen, away from $t = 0.75$, the nodal error and time-step size vary little as the solution evolves, confirming the reliability of both the BM monitor function and the time-step selection procedure. At time $t = 0.75$, the front lies along the line between coordinates (1, 0) and (0, 1). At this time, additional numerical difficulties are introduced as clustering of the grid points switches from one boundary edge to another. This provides a good test of the robustness of the multipass algorithm. As can be seen from Fig. 4, the time-step size is reduced by a factor of approximately two in the neighbourhood of $t = 0.75$ in response to this localised numerical difficulty. The increase in nodal error is, in part, due to a spatial component which cannot be completely eliminated by a reduction in the time-step size. However, it is clear that the time-step selection is robust and any increase in nodal error is localised around $t = 0.75$.

5. CONCLUSIONS AND COMMENTS

In this study, we have used the two-dimensional Burgers equation as a model for investigating several important aspects of moving mesh partial differential equations. The moving mesh algorithm used in the numerical experiments is based on the strategy developed by Huang and Russell [13, 14] and is derived from the gradient flow equation of a carefully chosen functional. We have shown that the selection of monitor function has a significant effect on the accuracy of the computed results. In particular, we have compared the performance of the commonly used arclength monitor function with one proposed by Beckett and Mackenzie [4–6], showing that the latter achieves a significantly higher rate of convergence.

The moving mesh PDEs are solved by decoupling the mesh equations from the physical PDE, as in our earlier study of problems in one space dimension [7]. The algorithm uses a backward Euler method for the time integration of the linearised mesh equations and a second-order singly diagonally implicit Runge–Kutta method (SDIRK2) for the physical PDE.

A key contribution of this work is to demonstrate that the efficiency, accuracy, and robustness of the moving mesh algorithm all depend on the method adopted for solving the decoupled system. In addition, we have highlighted the importance of using an efficient preconditioned iterative method for the solution of the mesh equations. Finally, the algorithm that we have proposed makes use of effective time-step control mechanisms that employ error indicators for the accuracy of the grid and the accuracy of the solution of the physical PDE.

REFERENCES

1. O. Axelsson, *Iterative Solution Methods* (Cambridge Univ. Press, Cambridge, UK, 1994).
2. I. Babuška and W. C. Rheinboldt, A-posteriori error estimation for the finite element method, *Int. J. Numer. Anal. Eng.* **12**, 1597 (1978).
3. M. J. Baines, *Moving Finite Elements* (Clarendon, Oxford, 1994).
4. G. Beckett and J. A. Mackenzie, Convergence analysis of finite difference approximations on equidistributed grids to a singularly perturbed boundary value problem, *Appl. Numer. Math.* **35**, 87 (2000).

5. G. Beckett and J. A. Mackenzie, On a uniformly accurate finite difference approximation of a singularly perturbed reaction-diffusion problem using grid equidistribution, *J. Comput. Appl. Math.* **131**, 381 (2001).
6. G. Beckett and J. A. Mackenzie, Uniformly convergent high order finite element solutions of a singularly perturbed reaction-diffusion equation using mesh equidistribution, *Appl. Numer. Math.* **39**, 31 (2001).
7. G. Beckett, J. A. Mackenzie, A. Ramage, and D. M. Sloan, On the numerical solution of one-dimensional PDEs using adaptive methods based on equidistribution, *J. Comput. Phys.* **167**, 372 (2001).
8. W. Cao, W. Huang, and R. D. Russell, A study of monitor functions for two-dimensional adaptive mesh generation, *SIAM J. Sci. Comput.* **20**, 1978 (1999).
9. W. Cao, W. Huang, and R. D. Russell, An r-adaptive finite element method based upon moving mesh PDEs, *J. Comput. Phys.* **149**, 221 (1999).
10. A. S. Dvinsky, Adaptive grid generation from harmonic maps on Reimannian manifolds, *J. Comput. Phys.* **95**, 450 (1991).
11. E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II* (Springer-Verlag, New York, 1991).
12. W. Huang, Practical aspects of formulation and solution of moving mesh partial differential equations, *J. Comput. Phys.* **171**, 753 (2001).
13. W. Huang and R. D. Russell, A high dimensional moving mesh strategy, *Appl. Numer. Math.* **26**, 63 (1997).
14. W. Huang and R. D. Russell, Moving mesh strategy based on a gradient flow equation for two-dimensional problems, *SIAM J. Sci. Comput.* **20**, 998 (1999).
15. R. Li, T. Tang, and P.-W. Zhang, A moving mesh finite element algorithm for singular problems in two and three space dimensions, *J. Comput. Phys.* **177**, 365 (2002).
16. J. A. Meijerink and H. A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, *Math. Comput.* **31**, 148 (1977).
17. K. Miller and R. N. Miller, Moving finite elements I, *SIAM J. Numer. Anal.* **18**, 1019 (1981).
18. H. A. van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* **13**, 631 (1992).
19. A. Winslow, Numerical solution of quasi-linear poisson equation in a nonuniform triangle mesh, *J. Comput. Phys.* **1**, 149 (1967).