Applied Numerical Mathematics 8 (1991) 65–90
North-Holland

A static-regridding method for two-dimensional parabolic partial differential equations

R.A. Trompert and J.G. Verwer

Centre for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, Netherlands

Abstract

Trompert, R.A. and J.G. Verwer, A static-regridding method for two-dimensional parabolic partial differential equations, Applied Numerical Mathematics 8 (1991) 65–90.

The subject of this paper belongs to the field of numerical solution of time-dependent partial differential equations. Attention is focussed on parabolic problems in two space dimensions the solutions of which possess sharp moving transitions in space and time, such as steep moving fronts and emerging and disappearing layers. For such problems, a space grid held fixed throughout the entire calculation can be computationally inefficient, since, to afford an accurate approximation, such a grid would easily have to contain a very large number of nodes. We consider a so-called static-regridding method that adapts the space grid using nested, locally uniform grids. The notion of locally uniform grid refinement is an example of "domain decomposition", the general idea of which is to decompose the original physical or computational domain into smaller subdomains and to solve the original problem on these subdomains. Hence, our computational subdomains are nested, locally uniform space grids with nonphysical boundaries which are generated up to a level of refinement good enough to resolve the anticipated fine scale structures. This way a fine grid covering the entire physical domain can be avoided. We discuss several aspects occurring in a static-regridding algorithm like ours, such as the data structure, the regridding strategy and the determination of initial and boundary conditions at coarse–fine grid interfaces. We also present two numerical examples to demonstrate the performance of the method. The first example is hypothetical and is used to illustrate the convergence behaviour of the method. The second example originates from practice and describes a model combustion process.

Keywords. Partial differential equations, numerical mathematics, time-dependent problems, static regridding, domain decomposition, method of lines.

1. Introduction

Many evolution problems involving linear or nonlinear partial differential equations (PDEs) have solutions with sharp moving transitions such as steep wave fronts and emerging or disappearing layers. In such situations, a grid held fixed throughout the entire calculation can be computationally inefficient, since, to afford an accurate approximation, such a grid would easily have to contain an unacceptably large number of nodes. Adaptive grid methods strive to resolve these sharp transitions to acceptable degrees of accuracy while avoiding the use of excessive

numbers of grid points. Such methods use, in some way or another, nonuniform or locally uniform grids and, as time proceeds, automatically concentrate the grid in spatial regions of high activity. It thus is attempted to keep the number of nodes at an acceptable level.

For time-dependent problems one may distinguish two main categories of methods, viz., dynamic-regridding and static-regridding methods. In the first category the grid moves continuously in the space–time domain, like in classical Lagrangian methods, while usually the discretization of the PDE and the grid selection are intrinsically coupled. A well-known example is provided by the moving finite-element and related methods (see, e.g., [8]). In the second category the grid is moved only at discrete time levels and no intrinsic coupling exists between the discretization of the PDE and the grid selection. This category comprises a large number of different methods and the method developed in this paper is of the static-regridding type.

Our method is closely related to the methods of Berger and Oliger [4], Gropp [9–11] and Arney and Flaherty [2] and thus an important characteristic of it is locally uniform grid refinement. The notion of locally uniform grid refinement is an example of "domain decomposition", the general idea of which is to decompose the original physical or computational domain into small subdomains and to solve the original problem on these subdomains. The idea of our static-regridding approach can be briefly described as follows. Given a coarse base space grid and a variable base temporal stepsize, a "decomposition" into locally uniform subgrids is performed recursively. Hence, our computational subdomains are nested, locally uniform space grids with nonphysical boundaries which are generated up to a level of refinement well enough to resolve the anticipated fine scale structures. The stepsizes in time and space during this refinement process are chosen automatically by comparing estimates or indicators of local temporal and spatial errors to prescribed tolerances, while the refinement is carried through until all tolerances are met. The process is then continued to the next base space/time mesh, while all fine grid results computed at forward time levels are kept in storage as these are needed for step continuation.

An attractive feature of moving the points only at discrete time levels is the possibility of dividing the whole solution process into the following computational procedures: spatial discretization, temporal integration, error estimation, regridding and interpolation. Depending on the application, these individual procedures may range from simple or straightforward to very sophisticated. This flexibility is attractive since it makes it possible to treat different types of PDE problems with almost one and the same code, assuming hereby that the grid structure and the associated data structure remain unchanged. In this connection we wish to note that a major part of the development and coding of any static-regridding method, including ours, lies in the grid and data structure. The choice of data structure is important for keeping the unavoidable overhead at an acceptable level, because at each time step grids may be created or removed while also communication between grids of adjacent levels of refinement frequently takes place.

When contrasted with the dynamic approach, an inherent drawback of static regridding is that during the time stepping temporal variations are not minimized because grid points do not move. More specifically, when a steep front passes a fixed grid point, smaller integration steps are required to maintain accuracy than when the grid point travels along with the front in the proper direction. In this connection, interpolation at internal grid interfaces should also be used judiciously for generating sharp solution profiles. Dynamic-regridding methods using a fixed number of moving points obviously do not require interpolation at internal interfaces which may be considered as an advantage. On the other hand, a well-known major disadvantage of dynamic

regridding is that methods of this type often have difficulty in controlling grid skewness and grid tangling. This disadvantage is usually less in hybrid methods where in some way or another static- and dynamic-regridding concepts are combined. Examples of such methods can be found in, e.g., Arney et al. [2], Gropp [11] and, for one-dimensional problems, in Petzold [15] and Verwer, Blom and Sanz-Serna [21]. Finally, an attractive feature of local mesh refinement is that it enables prescribed tolerances to be satisfied by using finer-and-finer grids in regions where greater resolution is needed.

In this paper we concentrate on (systems of) parabolic equations of the reaction-diffusion type,

$$\begin{aligned} &u_t = L(x, y, t, u) := d\Delta u + f(x, y, t, u), \\ &u = u(x, y, t), \quad (x, y, t) \in \Omega \times \{t > 0\}, \end{aligned} \tag{1.1}$$

subjected to appropriate initial and boundary conditions. For simplicity and convenience of presentation, our procedures are described and implemented on rectangular Ω, but they would also apply if Ω is a union of rectangles or can be transformed this way. Throughout the development of the method, the actual form of the operator L and its boundary conditions play no essential role. In fact, since for spatial discretization the use of standard finite differences is supposed, a much wider class of differential operators L is allowed.

The contents of the paper is as follows. We start with an outline of our locally uniform mesh refinement algorithm in Section 2. The various components of the solution algorithm are discussed in greater detail in the following sections. This discussion includes the actual refinement strategy (Section 3), the data structure (Section 4) and the grid interface conditions (Section 5). For the temporal integration we advocate one-step Runge–Kutta methods, since linear multistep methods like BDF are less appropriate due to the startup problem. Section 6 is devoted to a particular Runge–Kutta method, viz., the explicit Runge–Kutta–Chebyshev (RKC) method of van der Houwen and Sommeijer [13]. The temporal integrator we have used in the present investigation is based on this method, but it should be stressed that other choices of Runge–Kutta methods are possible too. The error indicators that govern the selection of the stepsizes in time and space are discussed in Section 7. In Section 8 we present two examples of reaction-diffusion problems (1.1) that were solved with our static-regridding method using the explicit RKC scheme for time integration. One of these two example problems is nonlinear and originates from combustion theory. Our future plans are summarized in Section 9.

2. Outline of the algorithm

In this section we present a rough outline of our recursive static-regridding algorithm, so as to facilitate the presentation and discussion of algorithmic details in later sections. Our algorithm is based on the principle of locally uniform mesh refinement (LUMR). LUMR may be contrasted with pointwise refinement which leads to truly nonuniform grids. As already noted in the introduction, LUMR is an example of "domain decomposition". When considered this way, our domain is a base space/time grid determined by a base space grid consisting of rectangular quadrilateral cells with sides Δx, Δy for $T \leqslant t \leqslant T + \Delta t$. The temporal stepsize Δt is called the base temporal stepsize and the base space grid covers the physical domain. It is assumed in this

paper that this domain is rectangular, but without essential changes the domain is allowed to be made up of a union of rectangles (polygonal boundary parallel to the coordinate axes). The "domain decomposition" or regridding on locally uniform grids now takes place entirely within this base space/time grid defined on the time interval $T \leqslant t \leqslant T + \Delta t$. Starting at the physical initial time, the complete algorithm is then repeatedly applied until the desired final physical time is reached.

We will outline all computations required for advancing the solution at the base grid at time T to the next base grid at time $T + \Delta t$. This outline is similar as in Gropp [10]. The base grid parameters Δx, Δy are supposed to be prescribed. The base temporal stepsize Δt is supposed to be a trial value. For clarity we discuss first a single level of refinement where the coarse grid coincides with the base grid. The following steps are followed to advance the solution from time T to the next base time level $T + \Delta t$:

(1) Integrate on the coarse grid using one coarse time step of size Δt. Adaptation in space is always preceded by adaptation in time, that is, the time step is first subjected to a temporal local error test and eventually the integration is redone with a smaller Δt until acceptance takes place. Call the accepted values of u on the coarse grid at time $T + \Delta t$ the new coarse u-values and those at time T the old coarse u-values. Both sets of values are saved.

(2) Integration is followed by regridding. Using the new coarse u-values, decide where the fine grid will be for $T \leqslant t \leqslant T + \Delta t$. This is done by invoking a spatial local error indicator and a clustering and buffering algorithm to distribute all untolerable cells over the fine grid. The fine grid may consist of different disjunct fine subgrids. Overlapping fine subgrids are not allowed in our method and fine subgrids need not be a rectangle. The actual refinement is cellular and carried out by bisecting all sides of untolerable cells. At this point the nonrefined part of the coarse grid is complete and not further processed within the current coarse time step.

(3) Regridding is followed by interpolation. Return to time level T and determine initial values for the fine grid. If a cell was refined in the previous coarse time step, then we use the available fine grid u-values and interpolation is not needed. If a cell was not refined before, then we interpolate old coarse u-values. For $T \leqslant t \leqslant T + \Delta t$ we need to specify boundary values at grid interfaces where fine grid cells abut on coarse cells. Using old and new coarse u-values, at these grid interfaces numerical Dirichlet boundary conditions are prescribed via interpolation. If an interface coincides with the physical boundary, then physical boundary conditions are used.

(4) Next the fine grid is integrated over the interval $T \leqslant t \leqslant T + \Delta t$ while fine grid u-values are subjected to the temporal local error test. This adaptation in time may result in a smaller temporal stepsize than Δt. If $T + \Delta t$ is reached, then the new fine grid u-values are injected in the coarse grid points, i.e., the value of u at $T + \Delta t$ on the coarse grid is taken to be the value of u just computed on the fine grid. Further, all fine grid u-values at $T + \Delta t$ are saved for use in the next coarse time step. The solution at time $T + \Delta t$ is now complete.

Multiple levels are handled in a natural, recursive fashion. After each accepted time step of (4), taken on a grid of refinement level l, say, a regridding may take place resulting in a grid of refinement level $l + 1$. In fact, the computational steps follow precisely points (2)–(4) above.

Note that all fine grid results at forward time levels are kept in storage and that for step continuation the most accurate results are used that are available.

An illustration of the recursive local refinement, in one space dimension for simplicity of presentation, can be found in Fig. 1. Including the base grid there are three levels of spatial refinement. In this figure the temporal stepsize is halved when going to a finer grid. The numbers next to the individual grids indicate the order in which the solution was computed. Herewith we do not distinguish between disjunct subgrids at the same level of refinement. Only the order of the time integrations is indicated. If in this example the base space grid is given refinement level 1, then the order of the corresponding occurring refinement levels is 1, 2, 3, 3, 2, 3, 3. The next occurring refinement level is 1, assuming that the next time step is taken on the next base grid. For clarity, a time history diagram of the occurring refinement levels is given in Fig. 2. The location of the fine grids is not made visible in this time history diagram and thus it may also correspond with a base time step of a two-dimensional computation. With figures like Figs. 1 and 2 one may now easily conceive other possible order of refinements. Finally, Fig. 3 shows a two-dimensional example of a base grid with three fine subgrids.

In conclusion, the principle on which our static-regridding method is based is recursive LUMR and cellular refinement. The building blocks of our algorithm are nested, locally uniform finer-and-finer grids which are adaptively defined by invoking indicators for local spatial and temporal errors. These locally uniform fine grids need not be rectangles and their location in the base space/time domain is automatically governed by the error indicators. Initial and boundary conditions for a refined subgrid are taken from the parent coarse grid or from the given initial function and physical boundary conditions. The recursive refinement takes place repeatedly per coarse time step. Solutions computed on current fine grids are kept in memory for use in the next coarse time step.

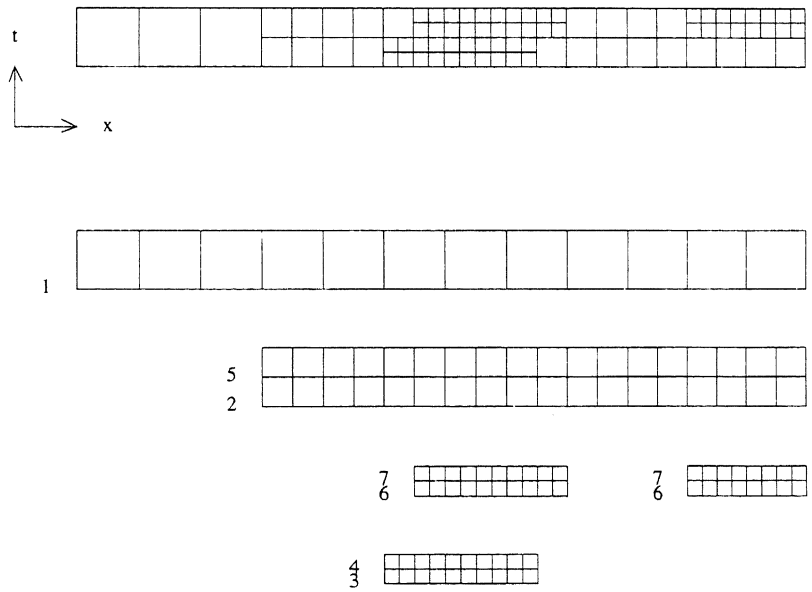


Fig. 1. Typical set of LUMR grids in one space dimension for one base time step. The final composite grid is shown at the top of the figure. Note that here the stepsizes in time are halved. In actual application the new stepsize is determined by the local time error indicator.

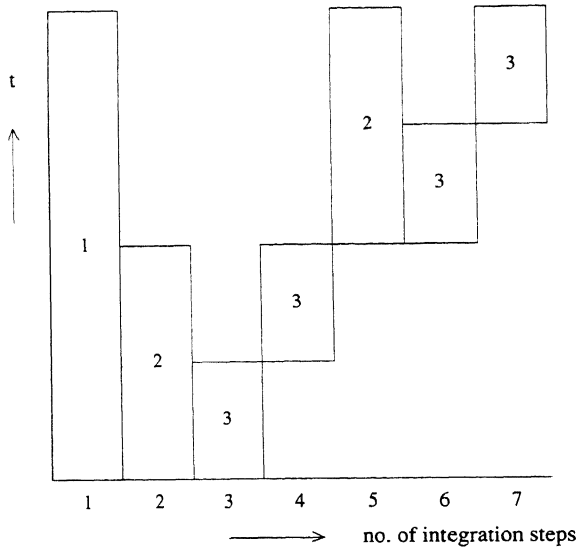


Fig. 2. Time history diagram of refinement levels occurring in Fig. 1. Each bar corresponds with an integration step. The levels are indicated within the bars. The lower and upper line of a bar correspond with the old and new time value of the integration step. The actual location of the refined grids is not shown.

Inherent in the approach we have adopted is that grid information is not passed to the next coarse time step. This necessarily is a bit wasteful in situations where the sharp transitions move very slowly, e.g., when approaching steady state. On the other hand, the computational effort for the coarser grids normally shall not be large. Further, uniform subgrids allow an efficient use of vector-based algorithms and finite-difference or finite-element expressions on uniform grids are more accurate and cheaper to process than on nonuniform grids. In this respect the current LUMR approach should be contrasted with pointwise refinement where arbitrary levels of refinement around any point are allowed. This pointwise refinement leads to truly nonuniform grids on which usually less points are needed than on an LUMR grid. However, an inherent

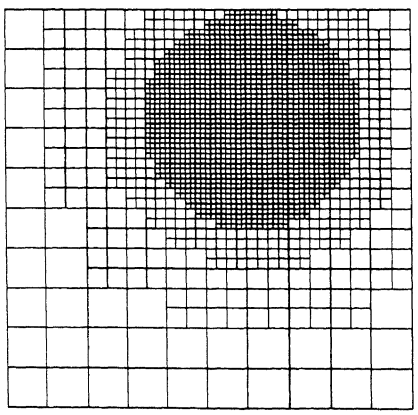


Fig. 3. Example of 2D grid structure. It is emphasized that fine grids are not patched into coarse ones, but overlays them. The time dependency is not shown here.

drawback of a truly nonuniform grid is a more complex and expensive data structure. We refer to Gropp [9–11], Berger [3] and Ewing [7] for some further discussions on various advantages and/or disadvantages when comparing the two refinement techniques. We also note that the LUMR methods of Berger and Oliger [3,4] (see also Arney and Flaherty [2]) are based on noncellular refinement and truly rectangular subgrids which may overlap and rotate to align with an evolving dynamic structure.

3. Refinement strategy and grid structure

When a new refinement level is to be created, the new fine grid has to meet two demands. First the new fine grid should be as efficient as possible, that is, no cells are unnecessarily refined. This implies that the new fine grid is allowed to have an irregular shape and also may consist of different disjunct subgrids. For clarity, it is noted once more that in the discussion we do not distinguish between subgrids belonging to the same level of refinement. In other words, when we write "new fine grid", we mean the complete grid associated to the next higher refinement level and this grid may consist of different disjunct subgrids. The second demand is that the accuracy at interior nodes of a new fine grid should not be diminished by low accuracy nodes on its boundary. Therefore, the boundary of the new fine grid must either coincide with a physical boundary or lie in part of the physical domain where the accuracy is sufficiently high, relative to the error measurement used. This second demand is most important because when at a certain level a cell is not further refined, we never return to this cell within the current base time step. In this section we describe a local refinement strategy that conforms to both demands.

The refinement strategy decides where a new fine grid will be placed (cf. point (2) of Section 2). The refinement is governed by a so-called local spatial error indicator, ests, and a corresponding tolerance value, tols, that has to be specified. Ideally, ests estimates the genuine local spatial truncation error that has been committed on the grid currently in use. We discuss the actual choice of the indicator in Section 7. For the discussion of the present section it suffices to suppose that we are given values of ests at the current forward time level at all nodal points of the grid currently in use.

Let l be the level index of this grid. Let estsm be the maximum of all ests values. If

$$\text{estsm} > \text{tols}, \tag{3.1}$$

then it is decided to create a new fine grid of level $l+1$. For this purpose, a second spatial tolerance value is introduced. It is this second tolerance, denoted by tolspc and derived from tols, that is used to decide which particular level l cells need to be refined. The second tolerance tolspc is defined as follows. Let p be the order of consistency of the spatial discretization and of the local error indicator. In our case $p = 2$ since we here work with standard finite differences. Since the mesh width of level $l+1$ is half the mesh width of level l, etc., we may invoke the asymptotic order relation

$$\text{estsm}(k) = 2^{-p(k-l)}\text{estsm}(l), \quad k \geqslant l+1, \tag{3.2}$$

where estsm(l) represents estsm of the level-l grid, etc., and k is a grid level index as yet unknown. We here anticipate that on top of level l, another $k-l$ refinement levels will be needed to satisfy the condition

$$\text{estsm}(k) \leqslant \text{tols}. \tag{3.3}$$

From this inequality the unknown integer k is now computed, that is,

$$k = l + 1 + \text{entier}[(\log(\text{estsm}(l)) - \log(\text{tols}))/(p \log 2)]. \tag{3.4}$$

The idea is now to impose, at all nodal values of level l, the refinement condition

$$\text{ests} > \text{tolspc} := \text{estsm}(k), \tag{3.5}$$

hereby introducing the second spatial tolerance value. However, because the calculation of a spatial error indicator comes down to the calculation of higher derivatives by means of finite differences (numerical differencing), it is conceivable that these are underestimated in regions where the solution is steep. Therefore, by way of safety, we suppose that one extra level would be preferable, which means that in (3.5) a safety factor of 2^{-p} must be built in. Thus we finally arrive at the refinement condition

$$\text{ests} > \text{tolspc} := 2^{-p}\text{estsm}(k). \tag{3.6}$$

This condition is used to decide which level-l cells need refinement. The rationale behind this second tolerance value is that we wish to refine all level-l cells, except those for which ests does not exceed the expected maximum of the error indicator values at the anticipated highest refinement level k. This local refinement strategy takes into account the method strategy that when at a certain level a cell is not refined, we never return to this cell within the current base time step. Another natural justification is that, when using local grid refinement, this strategy attempts to have the maximum norm of the spatial error over the complete physical domain equal to or smaller than the maximum norm of the spatial error over local grids.

The actual cell refinement goes as follows. Any level-l node satisfying (3.6) is flagged together with its eight neighbouring nodes. Next, to create an extra buffer, all sides of cells with at least one flagged corner node are bisected. This means that a buffer zone of two coarse or four fine mesh widths is used around any untolerable node. Hence, the minimal number of nodes in a column and row of any subgrid is nine. Herewith it is tacitly assumed that the minimal number of internal points in a row and column of the coarsest base grid is three. Near boundaries, physical and internal ones, the buffering of course slightly differs. Finally, a cluster algorithm groups all untolerable cells together to form the newly defined level-$(l+1)$ grid. It is noted that subgrids in the new fine grid do not overlap and that we do not connect subgrids which are lying close together since this leads to substantial bookkeeping. For the same reason, the local refinement does not distinguish between coordinate directions. This necessarily leads to some waste of points if a high gradient region aligns with a coordinate direction.

The use of tolspc in combination with the buffer zone is rather conservative. However, to our experience, it nicely conforms to the second demand stated above that accuracy at interior nodes of finer-and-finer grids should not be diminished by a too low accuracy at nodes on a previously selected interior boundary.

4. Data structure and memory use

In this section we briefly discuss the data structure we have implemented. Our data structure has close similarities with that of sparse matrix storage schemes, in particular concerning the storage of a sparse matrix as a collection of sparse vectors. The interested reader is referred to [6,

Chapter 2], for the various technical details that are involved in the use of this type of storage schemes.

The data we keep for each node are stored per level of refinement in a row sequential order. In particular, rows in subgrids are taken together and a "sparse" vector in our storage scheme contains all data belonging to all nodes of a row at a certain level. The data we store for each level of refinement are as follows:

- the number of rows;
- for each row,
  - a row index corresponding with its y-coordinate,
  - a pointer to the memory location of the data belonging to its first node;
- for each node,
  - a column index corresponding with its x-coordinate,
  - two pointers to memory locations where data belonging to the nodes directly above and below in the same grid is stored,
  - an integer indicating the position of a node in the domain, i.e., whether it lies on the physical boundary, on an internal boundary, or in the interior of the grid,
  - the solution and its time derivative at the beginning and end of a time step.

The row and column indices are, among others, used to find coinciding nodes on different refinement levels. This is needed for interpolation and injection. The row and column indices correspond to the x- and y-coordinates in the following way. Let i be the column index and j the row index of a node at refinement level l. Then,

$$x = x_0 + i\Delta x\, 2^{-(l-1)}, \qquad y = y_0 + j\Delta y\, 2^{-(l-1)}, \tag{4.1}$$

where Δx, Δy are the coarsest level mesh widths and (x_0, y_0) is the coordinate of the left lower corner of the physical, rectangular domain. It follows that a node on level l with column index and row index j coincides with a node on level $l+1$ with column index $2i$ and row index $2j$.

We will now outline how the actual storage is organized. The arrays used are divided in a number of blocks of the same size. The number of blocks equals the maximum of the number of refinement levels that is expected to be required during actual runtime. Each of the blocks is assigned a refinement level and all data for this refinement level are stored in this block. Hence the size of the blocks should be large enough to store all generated data of any refinement level that can occur during runtime. Herewith we implicitly determine the maximum number of levels and the maximum number of rows and nodes per level. The drawback of using memory blocks of the same size is that memory is wasted, as it is likely that the actual amount of memory needed differs per level. A clear advantage is computational simplicity, because there is no need for garbage collection. Because the size of the blocks is fixed, their initial subscript values are known which makes the data retrievement simple and fast. Nevertheless, the implementation of our algorithm is such that a garbage collector can be easily implemented. In this connection it is noted that with minor modifications our storage scheme can be extended to 3D. Needless to say that in 3D the memory waste could be prohibitive so that garbage collection becomes a necessity. All experiments reported in this paper have been carried out without garbage collection, indicating that in 2D this is not much of a problem.

Next we give an estimate of the amount of memory needed in bytes. Work arrays needed by the regridding algorithm and the time integrator are not included in this estimate. The work

arrays for the regridding algorithm are integer arrays and their total length is negligible compared to the estimate given below. Further, for the time integrator we use one and the same set of work arrays for all levels. The total length of these array is of course determined by the actual integrator in use (see Section 6).

The array in which we store the solution and its time derivative at the begin and end of an integration step is a real floating point array. The number of bytes per node this array uses is 32 times the number of PDEs (NPDE), assuming precision arithmetic involving 64 bits. The four arrays holding the column indices, the pointers to nodes above and below, and the position indicators, are 4-bytes integer arrays and together they take 16 bytes per node. The two arrays containing the row indices and the memory addresses of the data belonging to the first node of all rows are also integer arrays. These row indices and starting addresses require together 8 bytes of memory per row. Because the minimum number of nodes on a row is 7, it follows that these two arrays take at most 8/7 bytes per node. We thus arrive at a final estimate in bytes given by

$$\text{maxlev} * \text{nptspl} * (32 * \text{NPDE} + 16 + 8/7), \tag{4.2}$$

where maxlev is the maximum number of levels and nptspl is the maximum number of nodes per level.

5. Interface conditions

When a new fine grid is created, initial and boundary values have to be defined. Concerning the initial values, three cases are distinguished. If the time level at which initial values must be specified coincides with the physical initial time, then of course the prescribed initial function is used at any occurring node. If the time level does not coincide with the physical initial time, then two possible cases remain. A node coincides with a node at the same refinement level from a previous time step. In this case we adopt the available solution at this node as initial value. In the other case we must interpolate. The initial value is then obtained from interpolation on the nearest coarse grid. Obviously, the interpolation error should not diminish the accuracy. We use fourth-order Lagrangian interpolation and note that this way second-order accuracy is maintained in calculating first- and second-order spatial derivatives with the second-order difference scheme. Note that we use an explicit integration scheme which starts with a spatial differential operator calculation at the initial time level (stage (6.2b)). It is then prohibited to use straightforward linear interpolation, since this would yield zero second derivative values at the newly created nodes. When using implicit methods, this difficulty can be avoided by selecting the method such that no evaluation at the initial time is used.

Concerning boundary values, again three possible cases are distinguished. A boundary node of the new fine grid is located on the physical boundary. In this case the physical boundary condition is imposed. If a new boundary node belongs to the interior of the physical domain, still two different cases are possible. First, the node coincides with a node of the nearest coarse grid. Then we define Dirichlet boundary values by interpolating at the beginning and end of the new time interval to be covered. This time interval always coincides with a previous coarse grid time step. Fourth-order Hermite interpolation is applied, using the available solution and first time derivative from the coarse grid. Second, the node is new and hence does not coincide with a node of the nearest coarse grid. In this case a solution value is already available at the initial point of

the new time interval by the above Lagrangian interpolation procedure. To be able to apply again Hermite interpolation as outlined above, the solution and time derivative at the end point of the time interval are also generated by Lagrangian interpolation on the coarse grid.

Because we use second order in space and time in the discretization of the PDE, the fourth-order interpolation procedures should be sufficiently accurate, provided of course the local refinement has been carried through far enough at the time of the interpolations. Note that the prescription of interior Dirichlet boundary values is natural, since we solve initial boundary value problems involving second-order spatial differential operators.

6. Runge–Kutta–Chebyshev method

We will now describe the temporal integrator used for the experiments reported in this paper. For this purpose we introduce the system of ordinary differential equations (ODEs)

$$dU(t)/dt = F(t, U(t)), \tag{6.1}$$

assuming that this system originates from spatial discretization of the PDE problem (1.1) on the coarse base grid or on a refined local grid (method of lines). It is also assumed that boundary conditions, physical and artificial, have been incorporated into the continuous time, semi-discrete form (6.1). At this stage of development there is no need being more specific about (6.1).

The integration method is the explicit Runge–Kutta–Chebyshev (RKC) method of van der Houwen and Sommeijer [13]. This method has been designed for the numerical integration of large stiff systems of ODEs which originate from spatial discretization of multi-space-dimensional parabolic PDEs such as (1.1). We will present only a brief outline here. More details can be found in [13,20]. The latter paper is devoted to a convergence analysis of the RKC method.

Let U_n denote an approximation to $U(t)$ at time $t = t_n$. Let Δt denote the stepsize in time. The next approximation U_{n+1} at time $t_{n+1} = t_n + \Delta t$ is then given by the s-stage integration method:

$$Y_0 = U_n, \tag{6.2a}$$

$$Y_1 = Y_0 + p_1 \Delta t\, F_0, \tag{6.2b}$$

$$Y_j = m_j Y_{j-1} + n_j Y_{j-2} + (1 - m_j - n_j) Y_0 + p_j \Delta t\, F_{j-1} + q_j \Delta t\, F_0, \quad 2 \leqslant j \leqslant s, \tag{6.2c}$$

$$U_{n+1} = Y_s, \tag{6.2d}$$

where $F_j = F(t_n + c_j \Delta t, Y_j)$ and the value Y_j represents an intermediate, auxiliary approximation to $U(t)$ at the time point $t = t_n + c_j \Delta t$. Thus, the RKC method is to be interpreted as a one-step, s-stage Runge–Kutta method.

The available method parameters are used for obtaining a very large real interval of stability. This is achieved by identifying the recursive formula (6.2c) with a stable three-term Chebyshev recursion, thus explaining the specific form of the integration method. The length of the real stability interval that is obtained this way is proportional to s^2, where the proportionality constant depends on the order of consistency and on a damping property imposed on the common stability function. This stability function is a shifted Chebyshev polynomial.

The notion of stability meant here is the common linear stability based upon the linear system

$$dU(t)/dt = F(t, U(t)) := MU(t) + f(t), \quad M \text{ symmetric.} \tag{6.3}$$

Specifically, we have stability, in the step-by-step sense, if Δt and s are such that the inequality

$$\Delta t\, \sigma(M) \leqslant \beta(s), \tag{6.4}$$

is satisfied, where $\sigma(M)$ represents the spectral radius of M and $\beta(s)$ is the real stability boundary of the method. We have worked with a second-order method [20, formulas (2.19)–(2.21)] of which the real stability boundary is given by

$$\beta(s) \approx \tfrac{2}{3}s^2. \tag{6.5}$$

If the system (6.1) is nonlinear, then criterion (6.4) is imposed in the common heuristic way. Specifically, M is then understood to represent the Jacobian matrix of the vector function F taken at an appropriate point. Experience has revealed that the linear theory is most reliable if F stems from a nonlinear parabolic problem and the Jacobian matrix is symmetric. The method is recommended only if the Jacobian is symmetric or "nearly symmetric". This excludes, e.g., convection-diffusion problems with dominating convection.

The RKC method is applied with variable stepsize governed by a local error indicator (cf. Section 7) and with a variable number of stages s, such that always the linear stability inequality (6.4) is satisfied with s minimal. The variable s-strategy leans upon two properties. First, the error indicator is independent of s. Second, thanks to an internal stability property associated with the three-step Chebyshev recursion, there is no practical limit on s. This in fact implies that the method can be applied as if it is unconditionally stable, simply by adjusting s at each integration step to satisfy (6.4) for given stepsize and given spectral radius. The computation of (a safe upper bound) of the spectral radius normally renders no problem for operators like (1.1).

When compared with the implicit approach, explicitness has an inherent advantage for static regridding, since the costs of the numerical algebra involved in the application of implicit methods is much larger than in standard (single grid) method of lines applications. Recall that at any time step a regridding may take place at different levels of refinement, thus introducing one or more new Jacobians of different order at any time step when using an implicit method. This degrades the efficiency of stiff ODE solvers, since these often benefit from integrating with old Jacobians over many time steps. In spite of this, there are of course many problems and situations where for stability reasons alone implicit time stepping becomes a necessity. Therefore, as a continuation of the research reported here, in the near future we will investigate the application of implicit Runge–Kutta methods for static regridding. An important aspect hereby is the choice of appropriate implicit equation solvers which can efficiently deal with various types of 2D systems (see, e.g., Hindmarsh and Nørsett [12]).

Concerning stability, the explicit RKC method should be positioned between classical explicit methods yielding a severe time step restriction and unconditionally stable implicit ones. For problems with symmetric Jacobians, the RKC method is still attractive in cases of substantial stiffness due to the quadratic dependence of the real stability boundary on s. Furthermore, the method is simple to implement and the explicitness offers natural prospects for vector-based implementations. Also the memory demand is low. We have used a variable stepsize FORTRAN code due to Sommeijer [19] that needs only six arrays of storage.

It is noted that our version of this code slightly differs from Sommeijer's original one. This concerns merely the local error indicator. As we will outline in the next section, the third-derivative estimator of the original code, based on the genuine local truncation error, does not function well in our adaptive grid application and has therefore been replaced by a more simple error

indicator. However, the actual strategy associated with varying the stepsize, threshold factors and the like, has not been changed. It would lead us too far here to discuss this strategy in detail and it suffices to remark that, apart from the error indicator itself, the variable stepsize strategy is conceptually similar to strategies in existing ODE codes.

7. Error indicators

In static-regridding methods three differential kinds of local errors show up, viz., spatial discretization errors, time integration errors and interpolation errors. The asymptotic behaviour of the local space and time errors for decreasing spatial and temporal grid sizes is well understood for single grid applications. This is also true for the propagation of resulting global errors, at least for interesting model situations (see e.g. [17,20]). Needless to say that the static regridding complicates the error analysis considerably. Such an analysis should provide insight into how these three different local errors interfere with each other and propagate or accumulate under regridding. Subsequently, this insight then should assist us in the choice of mathematically correct, practical local error estimators. Without good estimators it is likely that one wastes computational effort due to bad balancing of space and time errors. This in fact is also true for standard, single-grid method of lines applications (see Berzins [5] who studies the balancing of space and time errors in applications using the BDF method). However, the question of balancing space and time errors is most interesting for a static-regridding method, because for such a method the regridding apparatus is available and it is natural to let the local refinement in space be governed by the genuine space truncation error.

In a sequel to this paper we will present a convergence analysis of the static-regridding method. This analysis is supposed to cover both explicit and implicit Runge–Kutta methods for the time integration and shall be aimed at practically balancing genuine local space and time errors. Here we confine ourselves to illustrating the convergence of the method numerically (next section) and to using simple heuristic, local error indicators. These error indicators are cheap and function quite satisfactorily from the point of view of ease of use, viz., they automatically invoke refinement in regions with high gradients. On the other hand, they do not guarantee a good balance between spatial and temporal local errors.

Let us first define ests, the spatial error indicator first introduced in Section 3. In this paper ests is based on the "curvature" expression,

$$(\Delta x)^2 \mid u_{xx} \mid + (\Delta y)^2 \mid u_{yy} \mid, \tag{7.1}$$

which is computed with the three-point finite-difference scheme. To our experience, this "curvature" expression functions well in measuring the degree of spatial difficulty of the problem. We have also used it successfully in 1D moving grid computations [21]. Note that the genuine space truncation error is also of second order in the mesh widths, since we use the second-order finite-difference scheme for spatial discretization. We thus have proportionality between the tolerance tols and the spatial truncation error.

We next define estt, the temporal local error indicator that determines the stepsize in time. According to the outline presented in Section 2, estt is computed after every time step at any grid level. Naturally, the choice of estt should be determined by the integration method (cf. the

discussion at the end of Section 6). To our experience, the original third derivative estimator of the RKC method functions quite satisfactorily in standard, fixed grid applications. However, we have encountered difficulties with this estimator in our adaptive grid application. These difficulties are inherent to static regridding and similar to those reported by Petzold [15,16].

The following observations are in order. The regridding implies that components of a solution vector that acts as the initial vector for a following time step may be of three different types. Components may result directly from a preceding integration step on the same grid level, components may result from injection from a higher preceding level, and components may be obtained from interpolation at the next coarser level. While it is supposed that the accuracy is not adversely affected by the interpolation and, trivially so, by the injection, the effect of the regridding thus is that small "discontinuities" are introduced into the initial vector, which in turn introduce small stiff transient solution components in time. These small transients are damped by the stability of the numerical method. They are, however, seen by common local error estimators like the one implemented in the original RKC code. This estimator computes an approximation to the third solution derivative from solution data that has become available within the current time step. It suffices to recall that this computation is to be interpreted as explicit numerical differencing which holds true for any common local error estimator. The estimators then have a tendency to choose stepsizes which are smaller than what is really required to maintain the accuracy, since on nonsmooth data explicit numerical differencing tends to overestimate higher derivatives. When considered on its own, this is not so much of a problem. However, it may also result in unnecessary step rejections which of course should be avoided. For a static-regridding method using different levels of refinement, the sensitivity of the local error estimator for the observed nonsmoothness is even worse, because this non-smoothness increases for decreasing spatial mesh widths. This implies that irrespective of the strategy used, there will always be a tendency to use too small stepsizes on fine grids, which by themselves are already more expensive to process than coarser grids.

For implicit solvers an often used remedy to the problem of nonsmooth error estimates is "implicit filtering" by which the original, explicit local error estimate is "smoothed" in an implicit way [16]. This filtering step kills all high frequent components in the estimated error and leaves, up to $O(\Delta t)$, the low frequent components of the error unchanged. Because we work here with the explicit RKC scheme, "implicit filtering" cannot be used. We do not advocate explicit smoothers, since these only work well if the errors to be smoothed do have a regular structure. This is obviously not the case with static regridding, as already mentioned above. To circumvent the difficulty, we therefore base our variable stepsize on the simple, heuristic local error indicator

$$\text{estt} \approx (\Delta t)^2 \,|\, u(t+\Delta t) - u(t) \,| = (\Delta t)^3 \,|\, \mathrm{d}u(t)/\mathrm{d}t \,| + O\big((\Delta t)^4\big), \tag{7.2}$$

which is of the same order in stepsize as the local truncation error estimated in the original code. The rationale behind this first-derivative indicator is that this way we ignore the small, high frequent error components which we wish to ignore for stepsize prediction purposes. To our experience, the indicator is succesful in this respect. Usually, the imposed tolerance tolt should be chosen smaller than when using the original estimator. This is to be expected, since for solutions with steep temporal gradients, the third-derivative shall be larger than the first one. It may also be advisable for keeping ahead of instabilities. In a time stepping process, emerging instabilities result in high frequent error components which are detected by the local error

estimator. The estimator reacts by reducing the stepsize and consequently restores stability. As indicated, the error indicator (7.2) detects high frequent error components later than the original estimator. Needless to say that the combination (7.1)–(7.2) is inappropriate for subtle error balancing purposes. However, when using an explicit method like RKC, they form a good compromise because they are cheap and prevent the stepsize selection from being hindered by the inherent non-smoothness of the numerical solution.

As already mentioned at the end of the previous section, we have used the existing variable stepsize code of Sommeijer and have only replaced the existing estimator by (7.2). The stepsize selection strategy within the code has not been altered. When comparing the estimates with tolt, the maximum norm is used. Finally we mention that the stepsize prediction is carried out levelwise. More precisely, when entering a new level, the last predicted stepsize taken on that level is used. The stepsizes are always fitted to hit the end point of integration. For example, if at level 2 the predicted stepsize is smaller than that currently in use at the coarsest level 1, but greater than half this stepsize, then the level-2 stepsize is taken to be half of the level-1 stepsize. During the first base time step the initial stepsize is the same for any level that may be introduced.

To illustrate the foregoing we conclude this section with a pseudo FORTRAN description of the entire LUMR algorithm. In this description T(level) denotes the end time of the "level" level. For level = 1 the end time is simply the final physical time. For level > 1 this end time changes dynamically with the introduction and removal of the refined grids and is always smaller than or equal to the forward time value of the current coarse grid stepsize:

```
      Program LUMR
      level = 1
      T(level) = physical end time
      t(level) = physical initial time
      Δt(level) = initial stepsize
      call PDEsol
      end LUMR

      subroutine PDEsol
10    if t(level) < T(level) then
20      - advance on "level" level from t(level) to t(level) + Δt(level)
        - compute time error indicator estt and predict new Δt(level)
        if estt > tolt then
          - decrease Δt(level)
          - go to 20
        end if
        - compute space error indicator ests and estsm for all nodes at "level" level
        if estsm > tols then
          compute tolspc
        end if
        - flag all untolerable nodes
        if nodes are flagged then
          - generate new "level" level + 1
```

- $t(\text{level}+1) = t(\text{level})$
- $\Delta t(\text{level}+1) = \Delta t(\text{level})$ or previously predicted $\Delta t(\text{level}+1)$
- $T(\text{level}+1) = t(\text{level}) + \Delta t(\text{level})$
- level = level + 1
- go to 10

else

- $t(\text{level}) = t(\text{level}) + \Delta t(\text{level})$
- $\Delta t(\text{level}) =$ new $\Delta t(\text{level})$
- go to 10

end if

end if

if level > 1 **then**

- update the solution on "level − 1" with "level" values in coinciding nodes
- level = level − 1
- $t(\text{level}) = t(\text{level}) + \Delta t(\text{level})$
- $\Delta t(\text{level}) =$ new $\Delta t(\text{level})$
- go to 10

end if

end PDEsol

8. Numerical examples

We present two examples of parabolic problems (1.1) that were solved with our static-regridding method using the explicit RKC scheme for time integration. The entire code is written in standard FORTRAN and the numerical experiments have been carried out on a SUN/SPARC station 1.

Problem 8.1. This test problem is hypothetical and due to Adjerid and Flaherty [1]. The equation is the linear parabolic model equation

$$u_t = u_{xx} + u_{yy} + f(x, y, t), \qquad 0 < x, y < 1, \quad t > 0, \tag{8.1}$$

and the initial function at $t = 0$, the Dirichlet boundary conditions for $t > 0$, and the source term f are selected so that the exact solution is

$$u(x, y, t) = \exp\Big(-80\big((x - r(t))^2 + (y - s(t))^2\big)\Big), \tag{8.2}$$

where

$$r(t) = 0.25[2 + \sin(\pi t)], \qquad s(t) = 0.25[2 + \cos(\pi t)]. \tag{8.3}$$

This solution is a cone that is initially centered at $(\frac{1}{2}, \frac{3}{4})$ and that symmetrically rotates around the center $(\frac{1}{2}, \frac{1}{2})$ of the domain in a clockwise direction. The speed of rotation is constant and one rotation has a period of 2. This problem is not a very difficult one in the sense that the spatial gradients of the solution are not extremely large, that is, the cone is not that steep. However, the problem is suitable to subdue an LUMR regridding algorithm like ours to a convergence test.

Table 1
Results of the convergence test on Problem 8.1

Levels	1	1–2	1–2–3	1–2–3–4	1–2–3–4–5
tols	4.0	1.0	0.25	0.125	0.03125
Δt	0.01	0.005	0.0025	0.00125	0.000625
Time point	Global errors measured in the maximum norm over the coarsest 10-by-10 grid. The numbers in brackets are the corresponding error ratios				
0.10	0.19120	0.03886 (4.9)	0.01154 (3.7)	0.00340 (3.4)	0.00112 (3.0)
0.15	0.24592	0.04904 (5.0)	0.01386 (3.5)	0.00420 (3.3)	0.00132 (3.2)
0.20	0.18575	0.03877 (4.8)	0.01150 (3.4)	0.00359 (3.2)	0.00119 (3.0)
0.25	0.21759	0.04511 (4.8)	0.01273 (3.5)	0.00383 (3.3)	0.00121 (3.2)

The solution is computed five times over the time interval [0, 0.25] with an increasing number of levels and decreasing time step. In the first computation only one level is used, in the second two, and so on. The addition of a new level is governed by selecting tols appropriately. Of course, the movement of the refined grids is governed by the regridding algorithm itself. For each computation a constant time step is taken which is the same for all levels (the time step control was switched off). This time step is halved in the next computation when a new level is added. Thus, in view of our regridding strategy based upon using the second tolerance parameter tolspc, we expect the error to decrease with a factor 4, due to the second-order spatial differencing and the second-order consistency of the RKC method. Note that the number of explicit Runge–Kutta stage varies with Δt and the spatial mesh width according to formulas (6.4)–(6.5). We refer to [20] for a convergence test using the fixed grid approach (one single level) where the second order nicely shows up.

Results of the convergence test have been collected in Table 1 and Fig. 4. The figure shows two grid structures used in the level-4 run. Observe that (away from the physical boundary) the grids accurately reflect the symmetry of the rotating cone. This gives confidence that the principles underlying the grid strategy of Section 3 work out very satisfactorily. The size of the

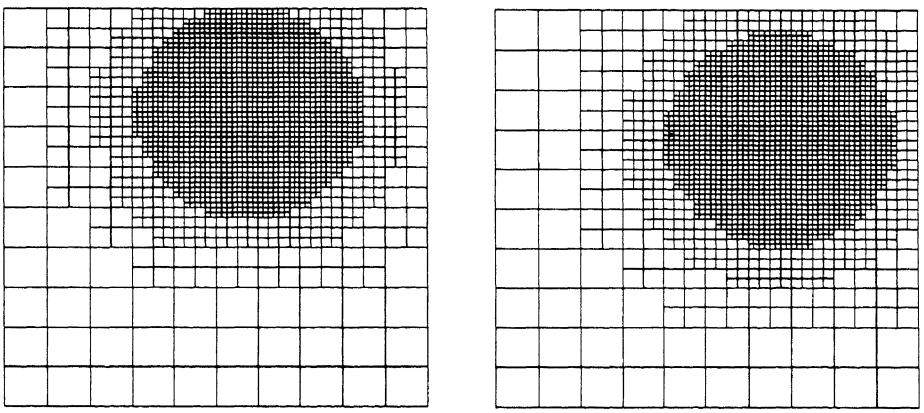


Fig. 4. Grid structures used in the level-4 run with Problem 8.1.

region of refinement is still considerable, even at the fourth level. This is due to the fact that the cone is not that steep, as we mentioned earlier.

Table 1 contains global errors at specified times given in the maximum norm over the base grid (level 1). The base grid is uniform with a grid distance of 0.1. We observe that the accuracy nicely decreases with the number of levels. Inspection of all output data also revealed that the maximum error was indeed found on the finest grid, as we anticipated in the development of the regridding strategy in Section 3. However, quite interestingly, we also observe a slight reduction in order for increasing number of levels. For instance, at $t = 0.25$ the successive ratios of the errors are approximately 4.8, 3.5, 3.3, 3.2 and thus tend to deviate from 4, although the deviation itself settles down. We believe this order reduction to be inherent in the LUMR approach and to originate from the small "discontinuities" in the grid functions which are caused by the interpolation and injection (cf. the discussion of Section 7). A particular role hereby is played by the internal Dirichlet boundaries. The following experiment serves to illustrate this observation.

We have repeated the convergence test while omitting injection of fine grid results into coarse grid nodes. Hence, the only change in the algorithm is that coarse grid results are never updated so as to reduce the nonsmoothness in the grid functions. Note that this way for each level solutions are obtained only by integration or by interpolation. Further, the number of nodes where interpolation takes place will be relatively small since this is needed only at locations where the current level not yet exists. In this non-update convergence test we observed an overall improvement in accuracy. For instance, for increasing number of levels, the errors at $t = 0.25$ with their successive ratios now are, respectively, 0.21759, 0.04425, 0.01045, 0.00240, 0.00082 and 4.9, 4.2, 4.4, 2.9. Apparently, till level 4 the order has improved, but it drops again at level 5. Inspection of the output data revealed that in the level-5 run the maximal error was not always committed at this finest level-5 grid. This means that in the level-5 run the heuristic spatial error indicator (7.1) probably failed in adequately identifying the highest error region and this observation should explain the drop in order. If this, more or less technical, problem can be overcome, the "non-update version" of the LUMR algorithm might be a remedy to the reduction in order and a good alternative for the more standard "update version". On the other hand, omitting updating may be somewhat dangerous because the coarse grid solution can become so dispersed that the algorithm decides that mesh refinement is no longer necessary. Of course, this can also happen when updating is used since we must rely here on error indicators or estimators. More precisely, since error indicators or estimators always underly some form of asymptotics, approximations serving as input for them should be of a certain minimal degree of accuracy in order to let them detect inaccuracies safely. For an LUMR method this implies that some care must be exercised in selecting the base grid parameters not too large since this might work out in the wrong way.

We conclude this numerical example with a measurement of overhead, in terms of total execution time and total number of PDE evaluations counted per node. The estimated part of the total execution time that is spent outside the integration routine is defined as overhead. Table 2 contains execution times and number of PDE evaluations for four runs over the time interval [0, 0.25], using, respectively, 1, 2, 3 and 4 refinement levels. The data are obtained with the "update" version using now also variable stepsizes in time. The computations were organized such that at each run the finest grid has $\Delta x = \Delta y = 0.0125$. Hence, the level-1 run is performed on an 80-by-80 level-1 mesh with $\Delta x = \Delta y = 0.0125$, the level-2 run on a 40-by-40 level-1 mesh accompanied with a level-2 mesh with $\Delta x = \Delta y = 0.0125$, and so on. This way the maximal error

Table 2
Results of time measurements for Problem 8.1

Levels	PDE evaluations	Execution times (sec)	Overhead
1	9749646	865	
1-2	3604712	365	12%
1-2-3	3427610	381	20%
1-2-3-4	4147352	512	28%

committed during each of the four runs is more or less equal, so that the measurement of overhead is not interfered by large differences in accuracy (in this reasoning we neglect the small effects of the above observed order-reduction phenomenon). The finest grid with $\Delta x = \Delta y = 0.0125$ is invoked by the spatial error indicator value tols = 0.125. The time step tolerance parameter tolt = 2.0E−7 and the initial time step equals 0.005. The parameter tolt was tuned so as to obtain nearly the same accuracy as in the level-4 run of the convergence experiment.

The entries "overhead" in Table 2 are percentages defined by

$$\text{overhead} = \frac{(\text{time(level)} - (\text{eval(level)} * 865)/9749646)}{\text{time(level)}} * 100\%, \tag{8.4}$$

where eval(level) is the total number of PDE evaluations and time(level) is the execution time. Thus in our measurement of overhead the work load for the single 80-by-80 mesh is used as a reference point. When inspecting Table 2 one should realize that the overhead factors are valid only for this particular experiment. Overhead is not only solution dependent (size of the refined grids), but, since we measure CPU times, also depends on the differential equation (expensive expressions in the operator) and on the degree of optimality in the FORTRAN code for the data structure and the like. As yet, we have paid little attention to the matter of optimal coding. In our opinion, the overhead factors found in this experiment are quite acceptable, although we should note that the use of garbage collection will increase the overhead (cf. Section 4). Finally we wish to note that for the present experiment the decrease in execution time shown in Table 2 is minor due to the fact that the refined grids are still of considerable size.

Problem 8.2. Our second example problem has also been borrowed from [1] and stems from combustion theory. The problem is a model for a so-called single one-step reaction of a mixture of two chemicals. In the problem, the dependent variable u represents the temperature of the mixture. The equation reads

$$u_t = d(u_{xx} + u_{yy}) + D(1 + \alpha - u)\, e^{-\delta/u}, \qquad 0 < x,\ y < 1, \quad t > 0, \tag{8.5}$$

and is subjected to the following initial and boundary conditions,

$$\begin{aligned} &u(x, y, 0) = 1, \quad 0 \leqslant x,\ y \leqslant 1, \\ &u_x(0, y, t) = 0, \quad u(1, y, t) = 1, \qquad 0 \leqslant y \leqslant 1, \quad t > 0, \\ &u_y(x, 0, t) = 0, \quad u(x, 1, t) = 1, \qquad 0 \leqslant x \leqslant 1, \quad t > 0. \end{aligned} \tag{8.6}$$

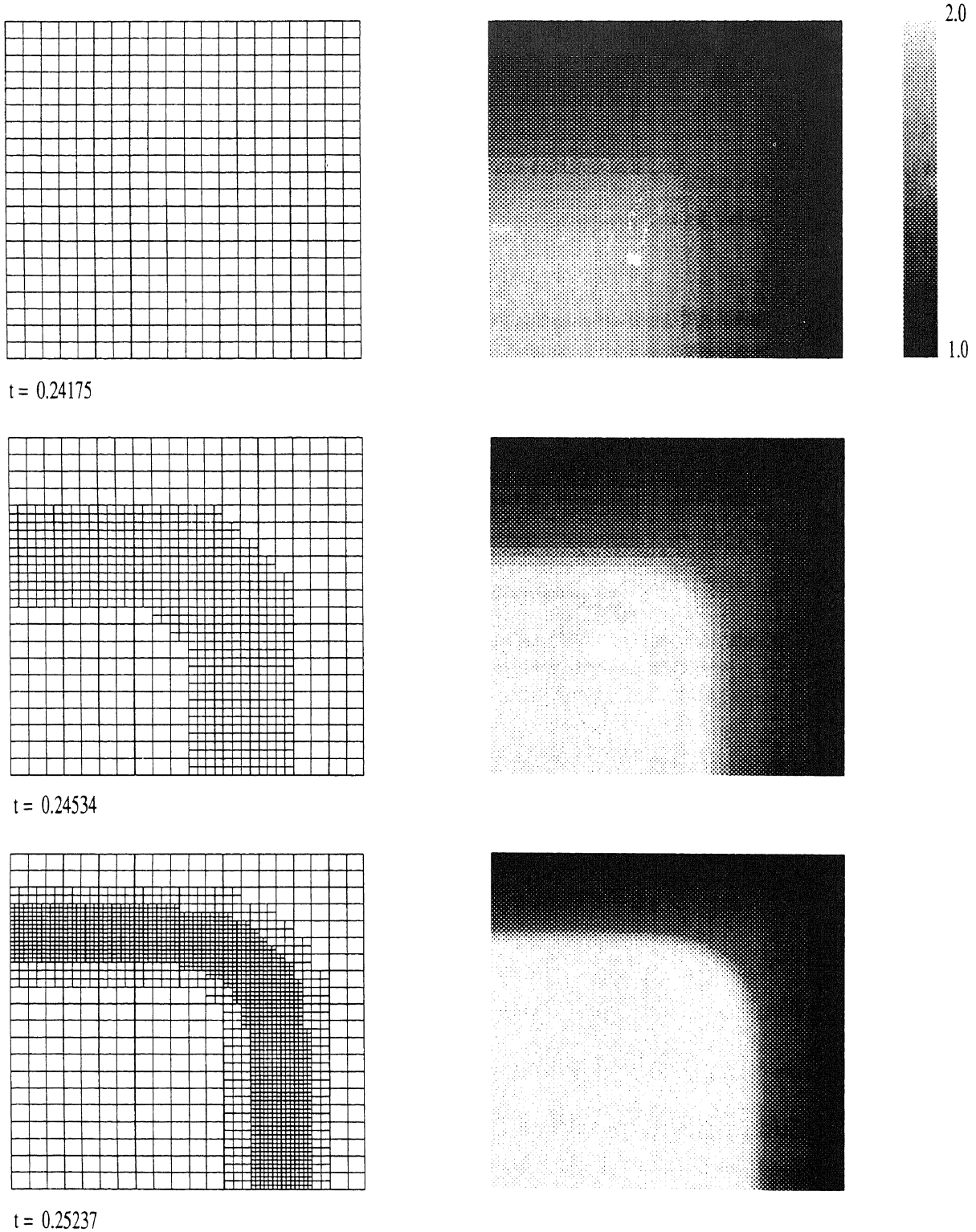
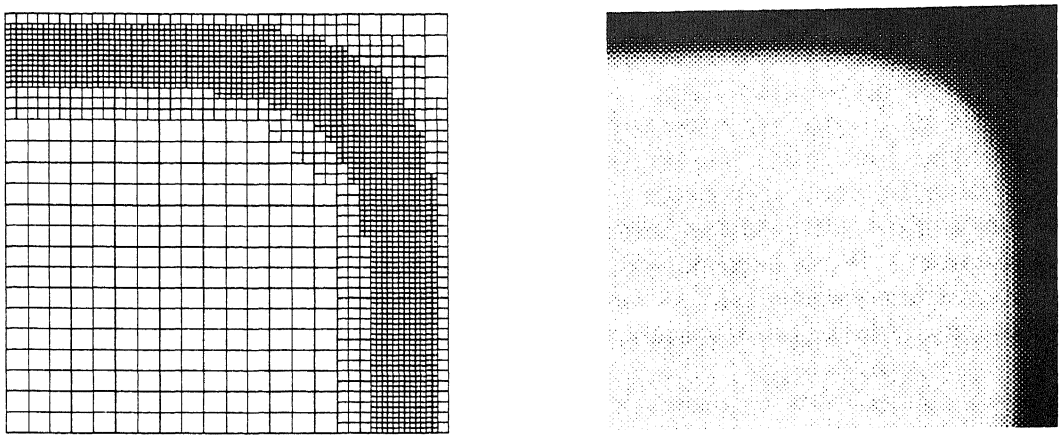
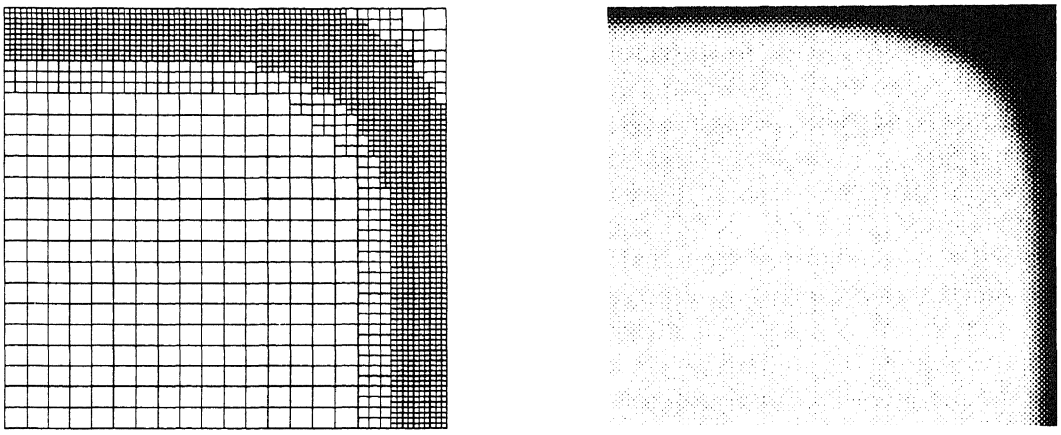


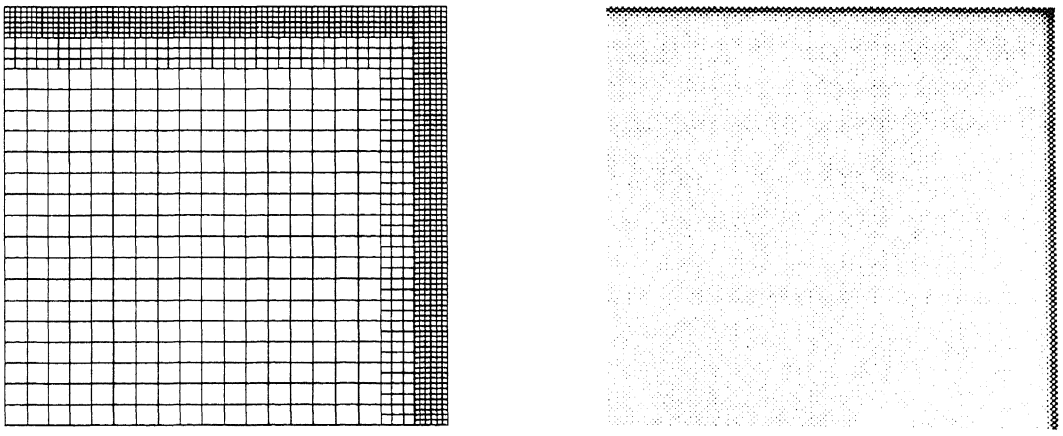
Fig. 5. Grids and solutions of Problem 8.2 at some specified output points.



t = 0.26796



t = 0.27653



t = 0.35000

Fig. 5. (continued).

The parameter α is the heat release, $D = Re^{\delta}/\alpha\delta$ the Damkohler number, δ the activation energy, and R is the reaction rate. For small times the temperature gradually increases in a circular area around the origin. Then, provided the reaction rate is large enough, at a finite time "ignition" occurs causing the temperature to suddenly jump from near unity to $1 + \alpha$, while simultaneously a reaction front is formed which circularly propagates towards the outer Dirichlet boundary. When the front reaches the boundary the problem runs into steady state. Following [1] we select the parameter values $\alpha = 1$, $\delta = 20$, $R = 5$, but choose a different value for the diffusion parameter. While in [1] the diffusion coefficient $d = 1.0$, we here put $d = 0.1$. A smaller diffusion coefficient has the effect that the wave front becomes steeper, particularly so upon approaching steady state [14]. With this choice of parameters the "ignition" takes place at about $t = 0.24$ and the solution is in "steady state" at about $t = 0.35$.

In spite of the fact that a fine grid is necessary for combustion problems of this type, the explicit RKC scheme is a natural candidate for the numerical integration. Two arguments support this observation. First, we must follow a travelling front on static, i.e., nonmoving space grids. This naturally limits the temporal stepsize of any integration scheme, be it explicit or implicit. Second, during the time evolution this special combustion problem is "locally unstable". Inspection of the reaction term reveals that for $1 \leqslant u \leqslant 2$ its derivative varies approximately between $+1000$ (for $u \approx 1.6$) and -5500 (for $u \approx 2.0$). Consequently, irrespective the integrator used, quite small integration steps are required to maintain sufficient accuracy in regions of "local instability" and before steady state will be reached the advantage of unconditional stability as provided by implicit methods shall not be fully exploited. To our experience, for this problem the explicit, stabilized RKC method is a good alternative.

Figure 5 shows generated grids and solutions at six specified time points. These are obtained using the full variable stepsize in time option using a uniform base grid with $\Delta x = \Delta y = 0.05$, tols = 0.6, tolt = 1.0E-7 and initial stepsize of 0.005. With this choice of parameters the method uses at most three levels, but integrates till $t = 0.24175$ only on the coarse base grid. A notable point is that the grids accurately reflect the symmetrical shape of the combustion wave front, thus again showing that our regridding strategy works very well. The steepening up of the wave front for evolving time is also clearly visible from the width of the finest level-3 grid. The solution at the end time $t = 0.35$ is, approximately, in steady state and shows a thin layer at the outer Dirichlet boundaries. At this time point the level-3 grid near the layer is only six cells wide, which is the minimum number that is possible near a boundary.

Inherent in static-regridding methods is that they have some difficulty in efficiently approaching steady state. When using a single grid without refinement, and assuming stability, a good

Table 3
Results of time measurements for Problem 8.2. The number of rejected steps is negligible in all three cases. The overhead is defined in the same way as for Problem 8.1, now relative to experiment (iii)

	Accepted time steps			PDE evaluations	Execution times (sec)	Overhead
	Level 1	Level 2	Level 3			
(i)	186	193	178	1530249	264	37%
(ii)	199	168	154	1470343	252	37%
(iii)	195	0	0	6206706	671	

variable stepsize solver will steadily increase the stepsize upon approaching steady state. On fine level grids this steady increase of stepsize will be less for a static-regridding method like ours, at least when using the "update version", due to the fact that the injection of solutions from fine grids into coarse ones has the effect that the numerical solution at the coarse grids is "slightly perturbed". The "slight perturbations" hinder the approximation from steadily becoming stationary like in single grid computations. This negative effect will be most pronounced at higher levels of refinement. Fortunately, the drawback can be largerly overcome by using the "non-update version" where injection is omitted.

By way of illustration we have included Table 3. This table contains information concerning the temporal integration for three runs:

(i) the run corresponding with Fig. 5, where "updating" was used,
(ii) a similar run, but now without "updating", and
(iii) a run using a single 80-by-80 grid without refinement, with the same parameter values for the variable stepsize control.

We note that in case (ii) the generated grids are nearly the same as in case (i) and that the plot accuracy in the three runs is the same. In particular, the finest mesh width in space for all three runs is $\Delta x = \Delta y = 0.0125$, so that, since the accuracy is very much the same, we can compare the workload. We see that for (i) and (ii) the workload is nearly the same; (ii) requires more steps at level 1, but less at levels 2 and 3 as is to be expected. The number of time steps in (iii) is nearly the same as at level 1 in (i) and (ii). The reduction in execution time, when comparing (i) and (ii) with (iii) is approximately a factor of 2.7. This includes CPU time for overhead which here appears to be larger than for Problem 8.1.

9. Final remarks and future plans

The LUMR algorithm presented in this paper is based on a mix of various techniques and builds on previous work started by Berger and Oliger [4] and continued by Gropp [9–11], Arney and Flaherty [2] and others. While Berger and Oliger [4] consider hyperbolic problems, we have focussed on parabolic problems and have applied a special, explicit time integrator using variable time steps. The integrator is special in that it possesses an extended real stability interval. For parabolic problems of type (1.1), the integrator is therefore an attractive alternative in situations where the good stability properties of the more common implicit solvers can not be fully exploited due to inherent stepsize restrictions. Our example Problem 8.2 illustrates such a situation.

Further, while Berger and Oliger [4] and Arney and Flaherty [2] use noncellular refinement and truly rectangular subgrids which may overlap and rotate to align with an evolving dynamic structure, we have chosen to use cellular refinement and to avoid overlapping subgrids. On the other hand, our subgrids are not necessarily truly rectangular and may of course be disjunct with neighbouring subgrids at the same level of refinement. This approach allows a simpler data structure and, most importantly, makes it possible in a relatively simple and transparent way to exclusively interpolate missing initial and boundary conditions at internal grid interfaces in low error regions. It is emphasized that our method, unlike the method of Berger and Oliger [4] due to overlapping subgrids, does not allow steep solutions to intersect grid interfaces. For this

purpose, a reliable refinement strategy is of crucial importance. A good refinement strategy should refine in such a way that the accuracy obtained at the current highest level grid is comparable to the accuracy obtained on this grid if it would be used without any adaptation. This way the interpolation errors will never become visible. Our refinement strategy, as discussed in Section 3, attempts to achieve this through the use of the refinement condition (3.6). This condition in fact looks ahead such that all cells at the current level are refined, except those for which at the anticipated highest level the solution is already sufficiently accurate. The rationale behind this strategy is that when at a certain level a cell is not refined, we never return to this cell within the current base time step as we work with nested subgrids.

In connection with the use of nested subgrids, we recall that on each time step we restart from the base grid, but also that we keep the finest grid solution in storage for possible use in the next time step. Actually, for evolving time we integrate on different grid levels with the understanding that the integration domains are nested per level and change in time. The nesting requires the interpolation of Dirichlet boundary values and the change in time requires interpolation of initial values, but only for those nodal points not already used at the previous time step. Most of the time we thus restart from the finest grid solution that already exists at a given nodal point.

The advantage of this approach, which is typical for LUMR methods, is that one can integrate on uniform subgrids and avoid the use of truly nonuniform grids such as obtained in a pointwise refinement procedure (see, e.g., Adjerid and Flaherty [1]). Admittedly, the approach may be a bit wasteful in situations where the sharp transitions move very slowly, e.g., when approaching steady state. On the other hand, the computational effort for the coarser grids normally shall not be large and finite-difference or finite-element expressions on uniform grids are more accurate and cheaper to process than on nonuniform grids. Also note that an inherent drawback of pointwise refinement and a truly nonuniform grid is a more complex and expensive data structure.

There are two major reasons why the development of LUMR methods is of interest. The first reason is obvious and of a purely practical nature: by refining the spatial grid locally in regions of high spatial activity, it is attempted to obtain accurate numerical solutions at significantly lower costs than required in the standard approach without refinement. Our second numerical example illustrates this nicely in respect with the execution time (see Table 3). Of course, since any LUMR method necessarily involves considerable overhead arising from the data structure, the regridding, the repeated integrations, etc., these methods are of interest only when the spatial solution variations are sufficiently large, like in our Problem 8.2. We also conclude that in both our numerical examples the actual regridding strategy based on the refinement condition (3.6) has functioned very well. This follows directly from inspection of the plotted grids. On the other hand, in this paper the actual input for the regridding still stems from the heuristic indicator (7.1). Our choices of tols illustrate very clearly that this "curvature indicator" bears no good resemblance with the true spatial errors. No doubt it will be very worthwile to replace (7.1) by an accurate estimator of the genuine local space error, assuming this is feasible. This of course is also true for the time error indicator (7.2).

These observations lead us to the second reason why LUMR methods are of interest. This second reason is of a more fundamental nature: these methods offer the natural environment for balancing genuine local space and time errors and to let the local refinement be governed by the genuine local space error. Balancing local space and time errors has so far got only very little attention in the literature, but is of obvious importance when one aims at efficiency and

robustness in solving time-dependent PDEs (see Berzins [5] and Schönauer et al. [18]). We will therefore continue our work on adaptive grid methods with an investigation to convergence properties of the present LUMR method. In this investigation we plan to analyse both implicit and explicit methods and an important goal will be balancing genuine local time and space errors. In this connection it should be noted that L-stable implicit Runge–Kutta methods are of interest because of their inherent smoothing properties. This will no doubt help to reduce any difficulty originating from the unavoidable "nonsmoothness" in the numerical solution.

Acknowledgement

We would like to thank Joke Blom and Paul Zegeling for reading the manuscript and for suggesting many corrections and clarifying remarks.

References

[1] S. Adjerid and J.E. Flaherty, A local refinement finite element method for two-dimensional parabolic systems, Rept. 86-7, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY (1986).
[2] D.C. Arney and J.E. Flaherty, An adaptive local mesh refinement method for time-dependent partial differential equations, *Appl. Numer. Math.* 5 (1989) 257–274.
[3] M.J. Berger, Data structures for adaptive grid generation, *SIAM J. Sci. Statist. Comput.* 7 (1986) 904–916.
[4] M.J. Berger and J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, *J. Comput. Phys.* 53 (1984) 484–512.
[5] M. Berzins, Global error estimation in the method of lines for parabolic equations, *SIAM J. Sci. Statist. Comput.* 9 (1988) 687–703.
[6] I.S. Duff, A.M. Erisman and J.K. Reid, *Direct Methods for Sparse Matrices* (Clarendon Press, Oxford, 1986).
[7] R.E. Ewing, Adaptive grid refinements for transient flow problems, in: J.E. Flaherty, P.J. Paslow, M.S. Shephard and J.D. Vasilakis, eds., *Adaptive Methods for Partial Differential Equations* (SIAM, Philadelphia, PA, 1989) 194–205.
[8] J.E. Flaherty, P.J. Paslow, M.S. Shephard and J.D. Vasilakis, eds., *Adaptive Methods for Partial Differential Equations* (SIAM, Philadelphia PA, 1989).
[9] W.D. Gropp, A test of moving mesh refinement for 2D-scalar hyperbolic problems, *SIAM J. Sci. Statist. Comput.* 1 (1980) 191–197.
[10] W.D. Gropp, Local uniform mesh refinement on vector and parallel processors, in: P. Deuflhard and B. Engquist, eds., *Large Scale Scientific Computing*, Birkhäuser Series Progress in Scientific Computing 7 (Birkhäuser, Basel, 1987) 349–367.
[11] W.D. Gropp, Local uniform mesh refinement with moving grids, *SIAM J. Sci. Statist. Comput.* 8 (1987) 292–304.
[12] A.C. Hindmarsh and S.P. Nørsett, KRYSI, An ODE solver combining a semi-implicit Runge–Kutta method and a preconditioned Krylov method, Rept. UCID-21422, Lawrence Livermore National Laboratory (1988).
[13] P.J. van der Houwen and B.P. Sommeijer, On the internal stability of explicit m-stage Runge–Kutta methods for large m-values, *Z. Angew. Math. Mech.* 60 (1980) 479–485.
[14] K. Miller, Private Communication (1988).
[15] L.R. Petzold, Observations on an adaptive moving grid method for one-dimensional systems of partial differential equations, *Appl. Numer. Math.* 3 (1987) 347–360.
[16] L.R. Petzold, Adaptive moving grid strategies for one-dimensional systems of partial differential equations, Preprint UCRL-96190, Lawrence Livermore National Laboratory (1987).
[17] J.M. Sanz-Serna, J.G. Verwer and W.H. Hundsdorfer, Convergence and order reduction of Runge–Kutta schemes applied to evolutionary problems in partial differential equations, *Numer. Math.* 50 (1987) 405–418.

[18] W. Schönauer, E. Schnepf and K. Raith, Numerical engineering: Experiences in designing PDE software with self adaptive variable stepsize/order difference methods, *Computing Suppl.* 5 (1984) 227–242.
[19] B.P. Sommeijer, Private Communication (1989).
[20] J.G. Verwer, W.H. Hundsdorfer and B.P. Sommeijer, Convergence properties of the Runge–Kutta–Chebyshev method, *Numer. Math.* 57 (1990) 157–178.
[21] J.G. Verwer, J.G. Blom and J.M. Sanz-Serna, An adaptive moving-grid method for one-dimensional systems of partial differential equations, *J. Comput. Phys.* 82 (1989) 454–486.