Adaptive Multiresolution Mesh Refinement for the Solution of Evolution PDEs

S. Jain*, P. Tsiotras† and H.-M. Zhou‡
Georgia Institute of Technology, Atlanta, GA 30332-0150, USA

Abstract

In this paper we propose a novel multiresolution-based adaptive mesh refinement method for solving Initial-Boundary Value Problems (IBVP) for evolution partial differential equations. The proposed algorithm dynamically adapts the grid to any existing or emerging irregularities in the solution, thus refining the grid only at places where the solution exhibits sharp features. The main advantage of the proposed grid adaptation method is that it results in a grid with a fewer number of nodes when compared to adaptive grids generated by other wavelet- or multiresolution-based mesh refinement techniques. Several examples show the robustness, stability and savings, in terms of the CPU time, of our algorithm.

1 Introduction

It is well known that the solution of evolution partial differential equations is often not smooth even if the initial conditions are smooth. To capture discontinuities in the solution with high accuracy one needs to use a fine resolution grid. The use of a uniformly fine grid requires a large amount of computational resources in terms of both CPU time and memory. Hence, in order to solve evolution equations in a computationally efficient manner, the grid should adapt dynamically to reflect local changes in the solution.

Several adaptive gridding techniques for solving partial differential equations have been proposed in the literature. A nice survey on some of the earlier works can be found in [5, 43]. Adaptive methods currently popular for solving PDEs are: (i) moving mesh methods [1, 2, 4, 6, 7, 30, 13, 17, 35, 36] where one explicitly derives an equation governing the grid, which moves a grid of a fixed number of finite difference cells or finite elements so as to follow and resolve local irregularities in the solution, (ii) the so called "Adaptive Mesh Refinement" method [8, 9, 10, 11] in which the mesh is refined locally based on the difference between the solutions computed on the coarser and the finer grids, and (iii) wavelet-based or multiresolution-based methods [3, 12, 20, 21, 24, 25, 27, 44, 45, 46] which take advantage of the fact that functions with localized regions of sharp transition can be very well compressed. Our proposed method falls under this last category. Traditional wavelet-based adaptive methods can be further classified either as wavelet-Galerkin methods [25] or wavelet-collocation methods [12, 24, 27, 46]. The major difference between these two is that wavelet-Galerkin algorithms solve the problem in the wavelet coefficient space and, in general, can be considered as gridless methods. Wavelet-collocation methods, on the other

*Ph.D. candidate, School of Aerospace Engineering, Email: sachin_jain@ae.gatech.edu.
†Professor, School of Aerospace Engineering, Email: tsiotras@gatech.edu.
‡Assistant Professor, School of Mathematics, Email: hmzhou@math.gatech.edu.

hand, solve the problem in the physical space, using a dynamically adaptive grid. When one uses wavelets together with Galerkin methods, multiplication in physical space becomes convolution in wavelet space, which is very costly as it is very hard to compute this convolution efficiently. Whereas in the wavelet-collocation approach operations like multiplication and differentiation are fast. Furthermore, nonlinearities can be handled with ease. Wavelet-Galerkin methods also suffer from the fact that they become increasingly complex as one goes to higher dimensions.

What makes wavelets attractive for solving PDEs are their multiresolution properties [22, 33, 34]. Recently, Alves et al. [3] proposed an adaptive multiresolution mesh refinement scheme, similar to the multiresolution mesh refinement approach proposed by Harten[1] [20, 21] and the interpolating wavelet approach proposed by Bertoluzza [12] and Holmstrom [24]. The authors in [20, 21, 24] used the interpolating subdivision scheme of [15, 16, 20] to construct the interpolating polynomials, whereas Alves et al. [3] used the high-resolution schemes SMART [18] and MINMOD [19] for constructing the interpolating polynomials. These approaches share similar underlying ideas. Namely, the first step is to interpolate the function values at all the points belonging to a particular resolution level from the corresponding points at the coarser level, and find the interpolative error at each point of that particular resolution level. Once this step has been performed for all the resolution levels, all the points that have an interpolative error greater than a prescribed threshold are added to the grid, along with their neighboring points at the same level and the neighboring points at the next finer level. One also needs to add to the grid the points that have been used to predict the function values at all the previously added points in order to calculate the interpolative error at the previously added points during the next mesh adaptation. Henceforth, we call this approach the traditional mesh refinement approach.

In this paper, we propose a new multiresolution-based mesh refinement technique for solving initial-boundary value problems (IBVP) for evolution partial differential equations. The key feature of our algorithm is that it is a "top-down"approach, in which we use the most updated information to make predictions. Moreover, our interpolations are not restricted to use only the retained points at the coarser level, but also the retained points at the same level and even the finer level, which allows for more accurate interpolations leading to less points in the final grid. In the proposed algorithm we continuously keep on updating the grid as we go from the coarsest level to finer levels. If the interpolative error at a point that belongs to a particular level is greater than the prescribed threshold, we add that point to the grid. At the same time we add the neighboring points at the same level and the neighboring points of the next level to the grid. We predict the function value at a particular point only from the points that are already present in the grid. Therefore, in the proposed algorithm we make use of the fact that the neighboring points added to the grid can be also used to predict the remaining points at the same level and the levels below it.

The paper is organized as follows. We first formulate the problem we are interested in, and we then present the algorithm for mesh refinement. Next, we compare the proposed mesh refinement approach with the traditional approach, and we show that the proposed algorithm results, in general, in a fewer number of grid points compared to the traditional approach. We then present an algorithm for solving the IBVP for evolution equations on an adaptive nonuniform grid, generated using the proposed mesh refinement technique. This analysis is followed by several numerical examples that show a major speed up in terms of computational time compared to the uniform mesh case.

[1]The main difference is that Harten used the grid averages instead of the grid points for the multiresolution representation of the numerical solution.

2 Problem Statement

Many problems in engineering and physics can be written in the form of an initial-boundary value problem (IBVP) for an evolution equation:

$$(\text{IBVP}): \quad \begin{cases} u_t + f(u_{xx}, u_x, u, x) = 0 & \text{in } \mathcal{D} \times (0, \infty), \\ u = g & \text{on } \overline{\mathcal{D}} \times \{t = 0\}, \end{cases} \tag{1}$$

where $\overline{\mathcal{D}} = \mathcal{D} \cup \partial\mathcal{D}$, with $\mathcal{D} \subset \mathbb{R}$ bounded. The function $f : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathcal{D} \to \mathbb{R}$, and the initial function $g : \overline{\mathcal{D}} \to \mathbb{R}$ are given. The unknown is the function $u : \overline{\mathcal{D}} \times [0, \infty) \to \mathbb{R}$. The algorithm proposed in this paper works for many other boundary conditions but we only use *periodic*, *Dirichlet*, and *Neumann* boundary conditions for simplicity in the analysis below. Without loss of generality, we will further assume that $\mathcal{D} = (0, 1)$.

In (IBVP) the initial function g can be irregular. Even if g is smooth discontinuities such as shocks in hyperbolic conservation laws and kinks in Hamilton-Jacobi equations can develop in the solution u at some later time. Therefore, we would like to adapt the grid dynamically to any existing or emerging irregularities in the solution instead of using a fine mesh over the whole spatial and temporal domain. In the next section we propose a novel grid refinement technique for solving (IBVP) in a computationally efficient way.

3 Adaptive Gridding

Since $\overline{\mathcal{D}} = [0, 1]$ we consider dyadic grids of the form

$$\mathcal{V}_j = \{x_{j,k} \in [0, 1] : x_{j,k} = k/2^j,\ 0 \le k \le 2^j\}, \quad J_{\min} \le j \le J_{\max}, \tag{2}$$

where j denotes the resolution level, k the spatial location, and $J_{\min},\ J_{\max} \in \mathbb{Z}_0^+$. We denote by $\mathcal{W}_j$ the set of grid points belonging to $\mathcal{V}_{j+1} \setminus \mathcal{V}_j$. Therefore,

$$\mathcal{W}_j = \{y_{j,k} \in [0, 1] : y_{j,k} = (2k + 1)/2^{j+1},\ 0 \le k \le 2^j - 1\}, \quad J_{\min} \le j \le J_{\max} - 1. \tag{3}$$

Hence, $x_{j+1,k} \in \mathcal{V}_{j+1}$ is given by

$$x_{j+1,k} = \begin{cases} x_{j,k/2}, & \text{if } k \text{ is even}, \\ y_{j,(k-1)/2}, & \text{otherwise}. \end{cases} \tag{4}$$

An example of a dyadic grid with $J_{\min} = 0$ and $J_{\max} = 5$ is shown in Figure 1.

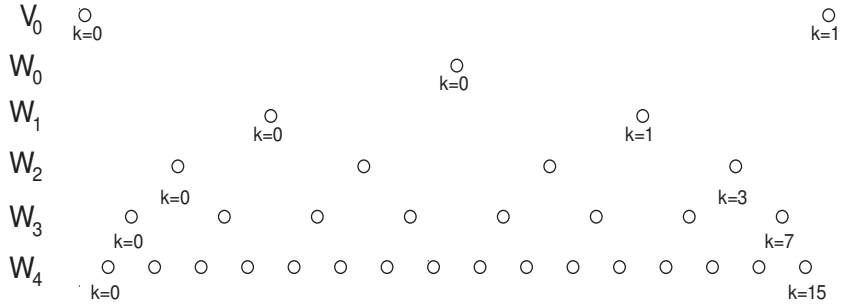


Figure 1: Example of a dyadic grid.

With a slight abuse of notation we write

$$\mathcal{V}_{j+1} = \mathcal{V}_j \oplus \mathcal{W}_j, \tag{5}$$

although $\mathcal{W}_j$ is not an orthogonal compliment of $\mathcal{V}_j$ in $\mathcal{V}_{j+1}$. The subspaces $\mathcal{V}_j$ are nested

$$\mathcal{V}_{J_{\min}} \subset \mathcal{V}_{J_{\min}+1} \cdots \subset \mathcal{V}_{J_{\max}}, \tag{6}$$

with $\lim_{J_{\max}\to\infty} \mathcal{V}_{J_{\max}} = \overline{\mathcal{D}}$. On the contrary, the sequence of subspaces $\mathcal{W}_j$ satisfy

$$\mathcal{W}_j \cap \mathcal{W}_\ell = \varnothing, \quad j \neq \ell. \tag{7}$$

3.1 Grid Adaptation

Assume we are given a nonuniform grid of the form

$$\begin{aligned} Grid = \quad & \{x_{j_i,k_i} : x_{j_i,k_i} \in [0,1],\ 0 \le k_i \le 2^{j_i},\ J_{\min} \le j_i \le J_{\max},\ \text{for } i = 0 \dots N, \\ & \text{and } x_{j_i,k_i} < x_{j_{i+1},k_{i+1}},\ \text{for } i = 0 \dots N-1\}. \end{aligned} \tag{8}$$

For simplicity of notations, we denote $u(x,t)$ evaluated at $x = x_{j,k}$ and $t = n\Delta t$ by $u^n_{j,k}$, where $0 \le k \le 2^j,\ J_{\min} \le j \le J_{\max},\ n \in \mathbb{Z}_0^+$, and Δt is the time step based on the Courant-Friedrichs-Levy condition [42] for hyperbolic equations and the von Neumann condition [42] for all other evolution equations.

For grid adaptation we first need a procedure for computing the function value at any point $x_{\text{interp}} \in (0,1)$ from the function values in $U = \{u^n_{j,k} : x_{j,k} \in Grid\}$, using an interpolating polynomial of degree p. To this end, the first step is to find the $p+1$ nearest points $\{x_{j_\ell,k_\ell}\}_{\ell=i}^{i+p} \in Grid$ to x_{interp}, where $0 \le i \le N-p$. By $p+1$ nearest points here we mean one neighboring point on the left of x_{interp}, one neighboring point on the right of x_{interp} and the remaining $p-1$ points are the points nearest to x_{interp} in the set *Grid*. In case two points are at the same distance, that is, if a point on the left and a point on the right are equidistant to x_{interp}, then we choose a point so as to equalize the number of points on both sides. For example, consider a grid at level $j = 3$ as shown in Figure 2. The set *Grid* consists of the solid circles. Let $p = 3$ and $x_{\text{interp}} = x_{3,4}$, shown by an empty square in Figure 2. The whole process involves three steps. Step a: We include in the set X_{near} one neighboring point on the left ($x_{3,2}$), shown by a left triangle and one neighboring point on the right ($x_{3,5}$), shown by a right triangle in Figure 2. Step b: We add to the set X_{near} the point $x_{3,6}$ since the distance from x_{interp} to $x_{3,0}$ is greater than the distance of x_{interp} to $x_{3,6}$. Step c: To choose the last point, we see that both points $x_{3,0}$ and $x_{3,8}$ are equidistant to x_{interp}. In this case, we choose $x_{3,0}$ in order to equalize the number of points on both sides. Hence, our final set X_{near} consists of points $x_{3,0},\ x_{3,2},\ x_{3,5}$, and $x_{3,6}$ as shown in Figure 2.

Suppose p is even and suppose we have already chosen $p-2$ points based on the previous methodology, such that $(p-2)/2$ points are on the left of x_{interp} and the remaining $(p-2)/2$ points are on the right of x_{interp}. In case both points on the left and the right are equidistant to x_{interp} we choose either of these points as the last point. Note that this situation will not arise if p is odd.

Once we have found the $p+1$ nearest points, we construct an interpolating polynomial $L_p(x)$ of order p passing through these $p+1$ points. One may use Neville's algorithm to construct the respective interpolating polynomials on the fly. The interpolating polynomials can also be constructed using high-resolution schemes such as by using the idea of ENO scheme [23, 40]. The only difference in that case would be that we would take one neighboring point on the left and one

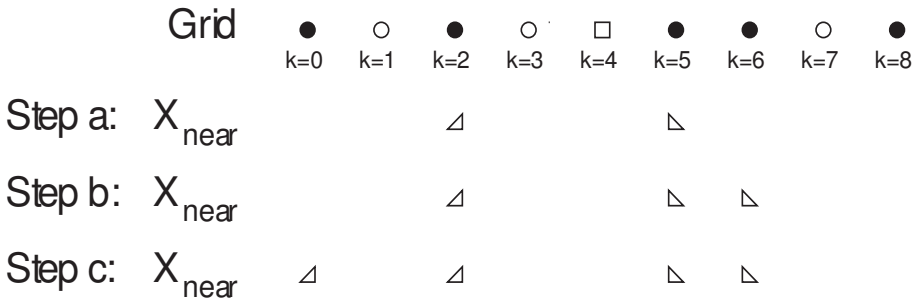


Figure 2: Demonstration of the procedure for finding the nearest points.

neighboring point on the right of x_{interp}, and choose the remaining $p-1$ points that give the least oscillatory polynomial.

The algorithm described above for calculating the function value at x_{interp} from the function values in U, using an interpolating polynomial of degree p, is summarized in the following psuedocode:

INTERPNEAREST$(x_{\text{interp}}, Grid, U, p)$

1 $X_{\text{near}} = \{x_{j_\ell,k_\ell}\}_{\ell=i}^{i+p}$; the $p+1$ nearest points to x_{interp} in the set $Grid$;
2 Construct $L_p(x)$ using the points in the set X_{near}
and the corresponding function values in U;
3 Calculate $L_p(x_{\text{interp}})$.

The "top-down" approach of our algorithm allows us to add and remove points using the most updated information. Now suppose $u(x, n\Delta t)$ is specified on a grid $Grid_{\text{old}}$, with corresponding solution values

$$U_{\text{old}} = \{u_{j,k}^n : x_{j,k} \in Grid_{\text{old}}\}, \tag{9}$$

where $Grid_{\text{old}}$ can be either regular or irregular[2]. Our aim is to find a new grid $Grid_{\text{new}}$, by adding or removing points from $Grid_{\text{old}}$, reflecting local changes in the solution. To this end, we initialize an intermediate grid $Grid_{\text{int}} = \mathcal{V}_{J_{\min}}$. We then set $j = J_{\min}$ and find the points belonging to the intersection of $\mathcal{W}_j$ and $Grid_{\text{old}}$. To this end, let this intersection be

$$Y = \{y_{j,k_i} : y_{j,k_i} \in \mathcal{W}_j \cap Grid_{\text{old}}, \text{ for } i = 1, \ldots, M\}. \tag{10}$$

Now we find the interpolated function value $\hat{u}_{j,k_1}$ at point $y_{j,k_1} \in Y$ from the points in $Grid_{\text{int}}$ and their function values in the set $U_{\text{int}} = \{u_{j,k}^n : u_{j,k}^n \in U_{\text{old}} \text{ and } x_{j,k} \in Grid_{\text{int}}\}$ using the procedure INTERPNEAREST mentioned above. Next we find the interpolative error coefficient d_{j,k_1} at the point y_{j,k_1}.

Definition 1. *The interpolative error coefficient $d_{j,k}$ at any point $y_{j,k} \in \mathcal{W}_j$, where $J_{\min} \le j \le J_{\max} - 1$ and $0 \le k \le 2^j - 1$, is defined as*

$$d_{j,k} = |u(y_{j,k}, n\Delta t) - \hat{u}(y_{j,k})|. \tag{11}$$

The interpolative error coefficient is a measure of the local smoothness of the function. If the value of $d_{j,k}$ is below the prescribed threshold ϵ, then $y_{j,k}$ is rejected since the function value at this point can be reconstructed from the points in grid $Grid_{\text{int}}$ with an error no larger than ϵ. A function with isolated singularities but is otherwise regular, will have most of the interpolative error

[2]Typically, $Grid_{\text{old}}$ at time $t = 0$ is regular with $Grid_{\text{old}} = \mathcal{V}_{J_{\max}}$.

coefficients close to zero. This allows a compact approximation of the function without significant loss of accuracy.

If the value of d_{j,k_1} is above the prescribed threshold ϵ, we add y_{j,k_1} to the intermediate grid $Grid_{\text{int}}$, along with N_1 points on the left and N_1 points on the right neighboring to the point y_{j,k_1} in $\mathcal{W}_j$. This step accounts for the possible displacement of any sharp features of the solution during the next time integration step. The value of N_1 dictates the frequency of mesh adaptation and is provided by the user. At the same time, we add to $Grid_{\text{int}}$ $2N_2$ neighboring points at the next level $\{y_{j+1,2k_1+\ell}\}_{\ell=-N_2+1}^{N_2}$. This step accounts for the possibility of the solution becoming steeper in this region and the value of N_2 is chosen based on the problem. We also add the function values at all the newly added points to U_{int}. If the function value at any of the newly added points is not known, we interpolate the function value at that point from the points in $Grid_{\text{old}}$ and their function values in U_{old} using the procedure INTERPNEAREST. Next we move to point $y_{j,k_2} \in Y$ and repeat the whole process, and similarly for all y_{j,k_i}, $i = 3, \ldots, M$. Once we have checked all the points in Y, we set $j = j + 1$, and repeat the above mentioned steps until $j = J_{\max} - 1$. This way we obtain the new nonuniform grid $Grid_{\text{new}}$.

In this work we consider two choices for the threshold ϵ. We can set either $\epsilon = \hat{\epsilon}$ or $\epsilon = \hat{\epsilon}/2^{j/2}$, where $\hat{\epsilon} > 0$ is chosen based on the desired accuracy in the solution. The choice $\epsilon = \hat{\epsilon}$ implies that ϵ is the same for all $\mathcal{W}_j$, whereas the choice $\epsilon = \hat{\epsilon}/2^{j/2}$ implies that ϵ changes with each level $\mathcal{W}_j$. In this case ϵ becomes increasingly smaller as one goes from $j = J_{\min}$ to $j = J_{\max} - 1$. This is to account for the reduction in the interpolation error with the decrease in the distance between the interpolating points. We use $\epsilon = \hat{\epsilon}/2^{j/2}$ for the case when the initial condition is not smooth and $\epsilon = \hat{\epsilon}$ for the case when the initial condition is smooth.

A pseudocode for the above mentioned grid adaptation procedure is as follows:

GRIDSELECTION($Grid_{\text{old}}, U_{\text{old}}, J_{\min}, J_{\max}, \mathcal{V}_{J_{\max}}, p, \epsilon, N_1, N_2$)

```
Grid_int = V_{J_min};
U_int = {u^n_{j,k} : u^n_{j,k} ∈ U_old and x_{j,k} ∈ Grid_int};
for j = J_min to J_max − 1
    Y = {y_{j,k} : y_{j,k} ∈ W_j ∩ Grid_old};
    for every y_{j,k} ∈ Y
        û(y_{j,k}) = INTERPNEAREST(y_{j,k}, Grid_int, U_int, p);
        d_{j,k} = |u(y_{j,k}, nΔt) − û(y_{j,k})|;
        if d_{j,k} > ε
            Add points {y_{j,k+ℓ}}_{ℓ=−N_1}^{N_1} to Grid_int;
            Add points {y_{j+1,2k+ℓ}}_{ℓ=−N_2+1}^{N_2} to Grid_int;
            Add the function values at all the newly added points to U_int;
    Empty Y;
Grid_new = Grid_int; U_new = U_int.
```

Next, we explain the proposed grid adaptation algorithm with the help of a simple toy example.

Example 1

Consider a dyadic grid $\mathcal{V}_4$ and the function

$$g(x) = \begin{cases} 1, & x = x_{4,k}, \\ 0, & \text{otherwise}, \end{cases} \tag{12}$$

with $k = 6$, so that g denotes an impulse located at $x = x_{4,6} = 0.375$. Let $J_{\min} = 0$, $J_{\max} = 4$, $p = 1$, $\epsilon = 0.1$, $N_1 = N_2 = 1$, and consider $Grid_{\text{old}} = \mathcal{V}_{J_{\max}}$. For this example the proposed grid adaptation algorithm is illustrated in Figure 3.

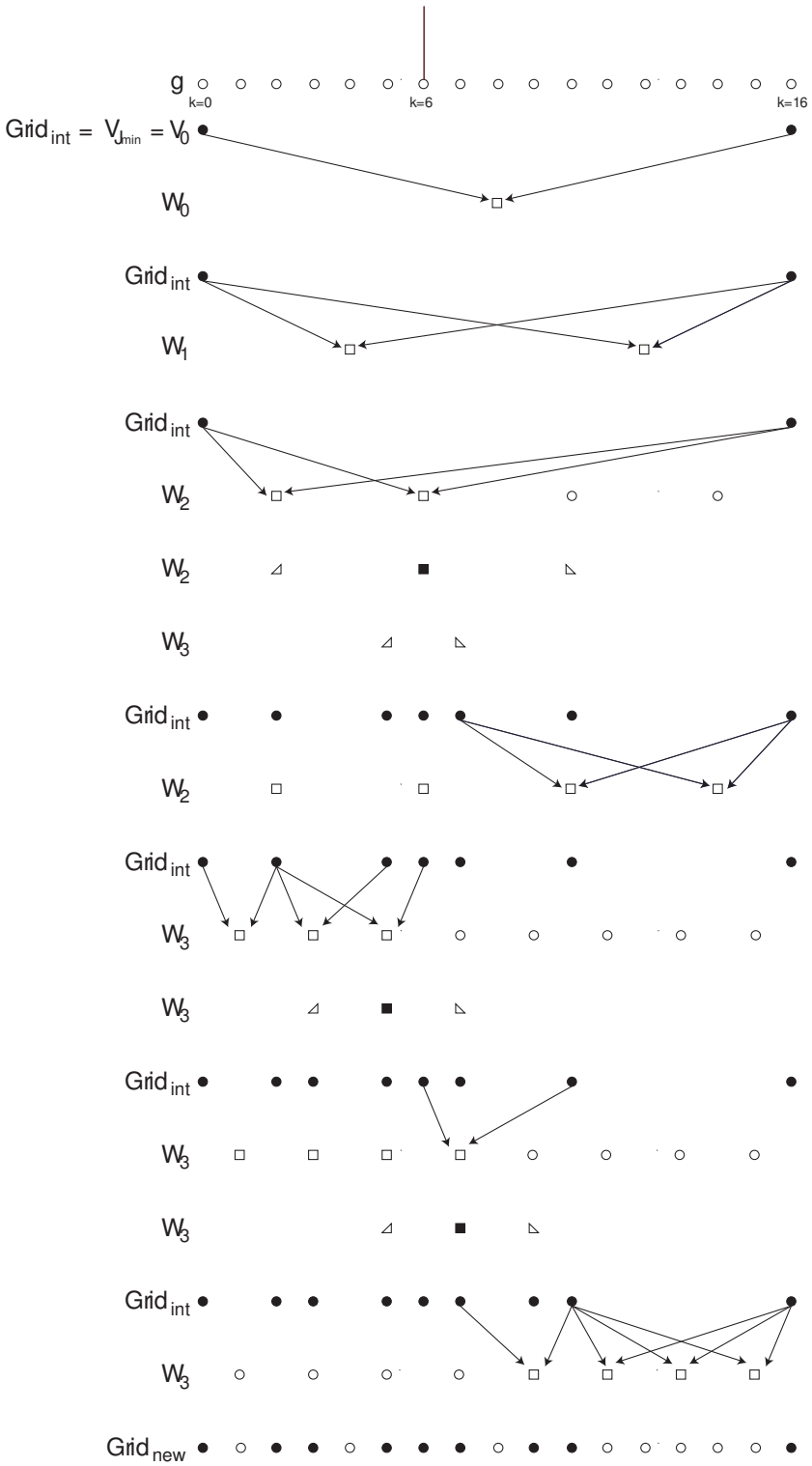


Figure 3: Demonstration of GRIDSELECTION procedure using Example 1.

In Figure 3, the solid circles show the points belonging to the intermediate grid $Grid_{\text{int}}$ and those belonging to $Grid_{\text{new}}$. The empty squares show the points that are being tested or have been tested for inclusion in $Grid_{\text{int}}$. If the interpolative error coefficient at a point is greater than the prescribed threshold then we show that point by a solid square. The left and the right neighbors are shown by left and right triangles, respectively. For reference, all points at that particular level are shown by empty circles.

First, we initialize $Grid_{\text{int}}$ with $\mathcal{V}_{J_{\min}}$. Next, we check if the function value at the point $y_{0,0} \in \mathcal{W}_0$ can be interpolated from the nearest $p+1=2$ points in $Grid_{\text{int}}$, which in this case are the points $x_{0,0}$ and $x_{0,1}$. Since for this example $g(y_{0,0})$ can be interpolated from the points in $Grid_{\text{int}}$, we do not include $y_{0,0}$ in $Grid_{\text{int}}$. Next, we consider the level $\mathcal{W}_1$ and check the point $y_{1,0}$. Since $g(y_{1,0})$ can again be interpolated from the function values at points $x_{0,0}, x_{0,1} \in Grid_{\text{int}}$, we do not include $y_{1,0}$ in $Grid_{\text{int}}$, and move on to the next point $y_{1,1}$. For the same reason as before, we do not include this point and the point $y_{2,0}$ belonging to the next level $\mathcal{W}_2$. Moving further to $y_{2,1}$, we find that $g(y_{2,1})$ cannot be interpolated from the neighboring two points $x_{0,0}, x_{0,1} \in Grid_{\text{int}}$. Hence, we include $y_{2,1}$ in $Grid_{\text{int}}$ along with points $y_{2,0}, y_{2,2} \in \mathcal{W}_2$ and $y_{3,2}, y_{3,3} \in \mathcal{W}_3$. Next, we check point $y_{2,2}$. The nearest points to $y_{2,2}$ in $Grid_{\text{int}}$ are $y_{3,3}$ and $x_{0,1}$. Since $g(y_{2,2})$ can be interpolated from $y_{3,3}$ and $x_{0,1}$, we do not include $y_{2,2}$ in the grid. For the same reason, we do not include $y_{2,3}$. Now we move to the next level $\mathcal{W}_3$ and check points $y_{3,0}$ and $y_{3,1}$. Since both these points can be interpolated from the existing points in $Grid_{\text{int}}$ we do not include these points in the grid. Subsequently we check $y_{3,2}$. Since $g(y_{3,2})$ cannot be interpolated from the nearest two points $y_{2,0}, y_{2,1} \in Grid_{\text{int}}$ we include $y_{3,2}$ along with points $y_{3,1}$ and $y_{3,3}$ (which in any case is already present in $Grid_{\text{int}}$) in $Grid_{\text{int}}$. Moving on to the next point $y_{3,3}$, we see again that $g(y_{3,3})$ cannot be interpolated from the nearest two points $y_{2,1}, y_{2,2} \in Grid_{\text{int}}$. Hence, we include $y_{3,3}$ along with $y_{3,2}$ (which in any case is already present in $Grid_{\text{int}}$) and $y_{3,4}$ in $Grid_{\text{int}}$. The next point in $\mathcal{W}_3$ is $y_{3,4}$. Since $g(y_{3,4})$ can be interpolated from the two nearest points $y_{3,3}, y_{2,2} \in Grid_{\text{int}}$, we move on to the next point $y_{3,5}$. The nearest two points to $y_{3,5}$ in $Grid_{\text{int}}$ are $y_{2,2}$, $x_{0,1}$, and since $g(y_{3,5})$ can be interpolated from these two points, we do not include $y_{3,5}$ in $Grid_{\text{int}}$. For the same reason we do not add points $y_{3,6}$ and $y_{3,7}$. The final adaptive grid $Grid_{\text{new}}$ is shown by the solid circles in Figure 3.

The adaptive grid generated using the previous algorithm depends on how we select points along the grid, that is, whether we move from left to right or from right to left across each level. It also depends on the location of the singularity. If the singularity is located in the middle, then it does not matter whether we move from left to right or from right to left. The result will be the same nonuniform grid. If on the other hand, the singularity is not in the middle, then the grid depends on the way in which we traverse across each level. To see this fact, we again consider Example 1, but this time with $k=1$. Hence, the impulse is located at $x = x_{4,1}$. If we go from left to right then the adaptive grid consists of the points $x_{4,0}$, $x_{4,1}$, $x_{4,3}$, $x_{4,5}$, $x_{4,16}$. If we go from right to left then the grid consists of the points $x_{4,0}, x_{4,1}, x_{4,3}, x_{4,16}$. Now let $k=15$, which implies that the impulse is located at $x = x_{4,15}$. If we go from left to right then the grid consists of the points $x_{4,0}$, $x_{4,13}$, $x_{4,15}$, $x_{4,16}$, and if we go from right to left then the grid consists of the points $x_{4,0}, x_{4,11}, x_{4,13}, x_{4,15}, x_{4,16}$. Note that in the proposed algorithm it is not mandatory to traverse across a level only from the leftmost point or from the rightmost point. We can instead start from any point at that level, each time resulting in a different grid. This suggests that by using a suitable probability distribution function to choose the order in which the points at each particular level are selected, one may be able to further optimize the grid. We will not elaborate on this observation in this paper.

3.2 The Traditional Grid Adaptation Approach

In this section, we briefly summarize the main idea underlying the traditional grid adaptation approach. For the details on the traditional grid adaptation approach the reader is referred to [3, 20, 21, 24].

Consider a set of dyadic grids $\mathcal{V}_j$ and $\mathcal{W}_j$ as described in equations (2) and (3) before. We explain pictorially, with the help of Example 1, how the traditional approach works (see Figure 4). The symbols used in Figure 4 are the same as those used in Figure 3 except for a new symbol (a triangle facing down), which is used to show the parents of points in $\mathcal{W}_j$, that is, the points in $\mathcal{V}_j$ that are used for interpolating the function values at points in $\mathcal{W}_j$. In the traditional approach, we first interpolate the function values at the points belonging to $\mathcal{W}_j$ from the corresponding points in $\mathcal{V}_j$ for $j = 0, \ldots, 3$, respectively, as shown in Figure 4. Then we find the points which have interpolative error coefficients greater than the prescribed threshold ϵ. We see that points $y_{2,1},\ y_{3,2},\ y_{3,3}$ have interpolative error coefficients greater than the threshold (shown by solid squares) and add these points in the grid. Next, we add the points $y_{2,0}, y_{2,2}$ neighboring to $y_{2,1}$ in $\mathcal{W}_2$ (shown by level A) and the points $y_{3,2},\ y_{3,3}$ neighboring to $y_{2,1}$ in $\mathcal{W}_3$ (shown by level B). Similarly, we add points $y_{3,1},\ y_{3,3}$ neighboring to $y_{3,2}$ in $\mathcal{W}_3$ (shown by level C) and the points $y_{3,2},\ y_{3,4}$ neighboring to $y_{3,3}$ in $\mathcal{W}_3$ (shown by level D). Finally, we include the parents of all the points added previously to the adaptive grid (shown by levels $A_p,\ B_p,\ C_p$ and D_p) resulting in the nonuniform grid $Grid_{\text{new}}$ shown in Figure 4.

3.3 Comparison between the Traditional and the Proposed Approach

The proposed grid adaptation algorithm results, in general, in a fewer number of grid points when compared to the traditional approach. First, we explain why this is so and then we give several examples to demonstrate this fact.

In the traditional approach, one interpolates $\{g(y_{j,k})\}_{k=0}^{2^j}$ only from the function values at the points belonging to $\mathcal{V}_j$ for $j = J_{\min} \ldots J_{\max} - 1$, and only then, one adds to the adaptive grid, the points $y_{j,k}$ along with the points $\{y_{j,k+\ell}\}_{\ell=-N_1}^{N_1}$ and $\{y_{j+1,2k+\ell}\}_{\ell=-N_2+1}^{N_2}$ for all the pairs (j,k), such that $d_{j,k} > \epsilon$. In the proposed method, we continuously keep on updating the adaptive grid instead. If the interpolative error coefficient at $y_{j,k}$, where $0 \le k \le 2^j - 1$ and $J_{\min} \le j \le J_{\max} - 1$, is greater than the prescribed threshold, we add $y_{j,k}$ to the adaptive grid, and at the same time we add to the adaptive grid the neighboring points at the same level $\{y_{j,k+\ell}\}_{\ell=-N_1}^{N_1}$, as well as the neighboring points at the next level $\{y_{j+1,2k+\ell}\}_{\ell=-N_2+1}^{N_2}$. We use these newly added points also for interpolating the remaining points at level $\mathcal{W}_j$ and the levels below it. In other words, in the proposed approach, $\{g(y_{j,k})\}_{k=0}^{2^j}$ are interpolated from the function values at the points in $\mathcal{V}_j \oplus \mathcal{W}_j \oplus \mathcal{W}_{j+1}$ for $J_{\min} \le j \le J_{\max} - 2$ and in $\mathcal{V}_j \oplus \mathcal{W}_j$ for $j = J_{\max} - 1$. Hence, by making use of the extra information from levels $\mathcal{W}_j$ and $\mathcal{W}_{j+1}$, which in any case will be added to the adaptive grid, we are able to reduce the number of grid points in the final grid. Moreover, in the traditional approach, when a point $y_{j,k}$, where $0 \le k \le 2^j - 1$ and $J_{\min} \le j \le J_{\max} - 1$, is added to the grid then we also include its parents, which were used to predict the function value at that point. The parents are not needed for approximating g to the prescribed accuracy but are included just for calculating the interpolative error coefficient at the point $y_{j,k}$ during the next mesh adaptation. In the proposed algorithm, on the other hand, whenever a point is being checked for inclusion in the adaptive grid, we predict the function value at that point only from the points which already exist in the adaptive grid. Hence, if that point is inserted in the grid we do not need to add any extra points (i.e., its parents). This also saves us from keeping a separate track of the parents to distinguish them from the rest of the points as in the traditional approach.

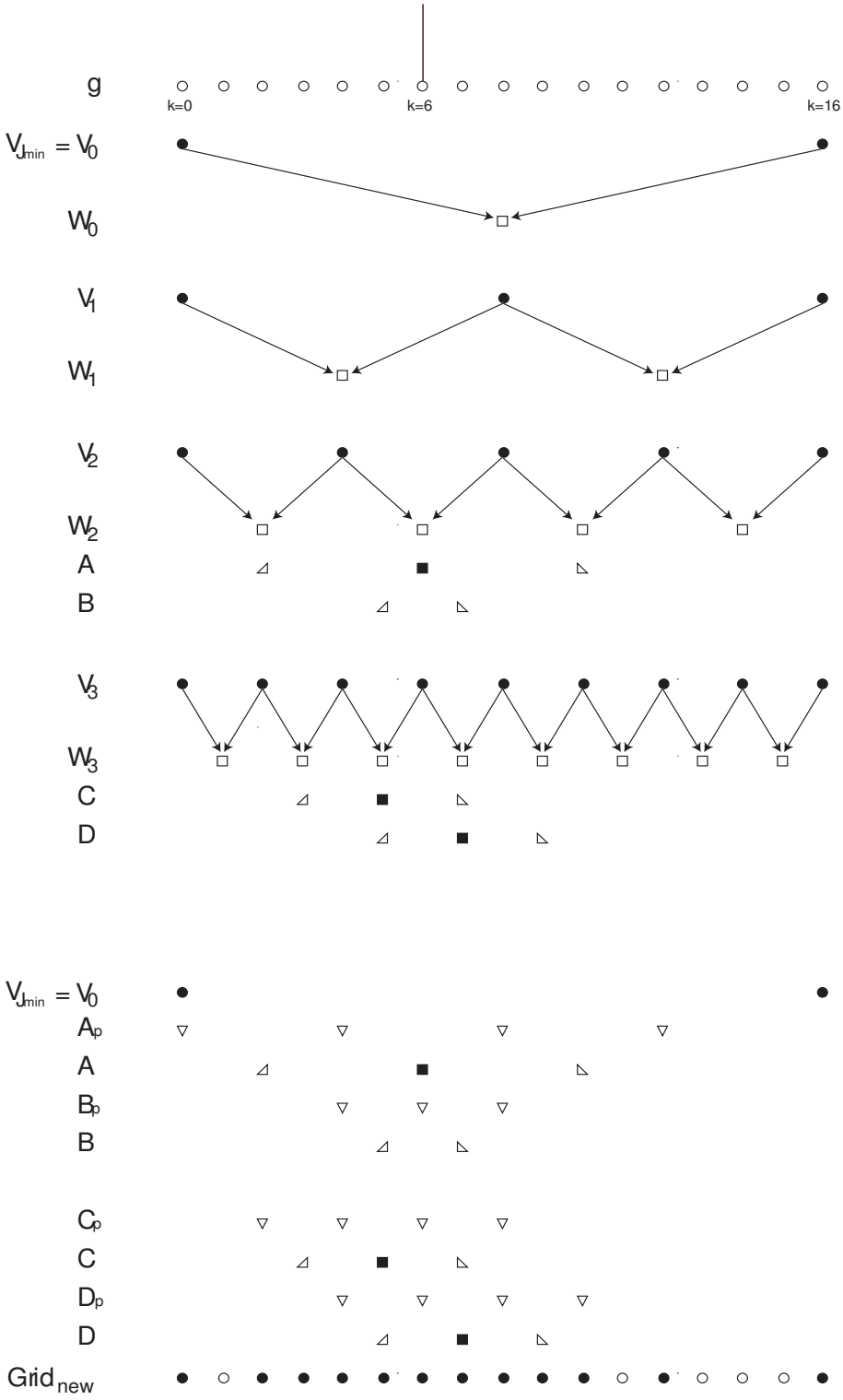


Figure 4: Demonstration of traditional grid adaptation approach using Example 1.

Example 2
Consider a dyadic grid $\mathcal{V}_4$ and the function

$$g(x) = \begin{cases} 1, & x = x_{4,k}, \\ 0, & \text{otherwise}, \end{cases} \tag{13}$$

with an impulse located at $x = k/2^4$, where $0 \leq k \leq 16$. Let $J_{\min} = 0$, $J_{\max} = 4$, $p = 1$, $\epsilon = 0.1$, and $N_1 = N_2 = 1$. Table 1 shows the number of grid points (N_{g}) in the adaptive grid using both the traditional and the proposed methods for $k = 0, \ldots, 16$. We found that when the impulse is located at either the left boundary ($k = 0$) or the right boundary ($k = 16$) or in the middle of the domain ($k = 8$) both the traditional approach and the proposed approach result in the same grid. For all other cases, the grids generated by both the algorithms are different and we see that the proposed algorithm results in a fewer number of grid points. For this example the proposed algorithm outperforms the traditional approach by up to 33%.

k	N_{g} using Traditional Method	N_{g} using Proposed Method	Ratio
0	9	9	1
1	6	5	0.83
2	9	7	0.78
3	8	6	0.75
4	12	11	0.92
5	9	6	0.67
6	12	9	0.75
7	9	6	0.67
8	13	13	1
9	9	6	0.67
10	12	9	0.75
11	9	6	0.67
12	12	11	0.92
13	7	5	0.71
14	9	7	0.78
15	6	4	0.67
16	9	9	1

Table 1: Example 2. Traditional method vs proposed method.

Example 3
Now consider another example with $g(x)$ a step function, given by

$$g(x) = \begin{cases} 1, & \frac{1}{3} \leq x \leq \frac{2}{3}, \\ 0, & \text{otherwise}. \end{cases} \tag{14}$$

For this example, we consider a grid with $J_{\min} = 2$ and $J_{\max} = 10$. The results for different thresholds using interpolating polynomials of degree $p = 1$ and $p = 3$ are summarized in Table 2. We consider $N_1 = N_2 = 1$. In this example we found that if the points are predicted using linear interpolation both the algorithms result in the same grids for different thresholds, whereas if the

p	ϵ	N_g using Traditional Method	N_g using Proposed Method	Ratio
1	0.3	51	51	1
1	0.1	51	51	1
3	0.05	89	76	0.85
3	0.01	89	76	0.85
3	0.001	89	76	0.85
3	0.0001	89	76	0.85

Table 2: Example 3: Traditional method vs proposed method.

points are predicted using cubic interpolation, the algorithms result in different grids, and the proposed algorithm always results in approximately 15% fewer number of points.

Example 4
Next, we consider the function

$$g(x) = \sin(2\pi x) + e^{-\beta(x-0.5)^2}, \tag{15}$$

which is smooth everywhere except for the peak at $x = 0.5$. Here β is a parameter that controls the width of the Gaussian. We let $\beta = 20000$. For this example we let $J_{\min} = 2$, $J_{\max} = 10$, $p = 3$, and $N_1 = N_2 = 1$. The results for different thresholds are summarized in Table 3. We observe once

ϵ	N_g using Traditional Method	N_g using Proposed Method	Ratio
10^{-2}	69	49	0.71
10^{-3}	89	79	0.89
10^{-4}	125	116	0.93
10^{-5}	181	150	0.83
10^{-6}	277	185	0.67
10^{-7}	429	304	0.71
10^{-8}	545	530	0.97
10^{-9}	1005	957	0.95
10^{-10}	1025	1021	0.99

Table 3: Example 4. Traditional method vs proposed method.

again that the proposed algorithm outperforms the traditional approach by up to 33%.

We are now ready to present the algorithm for solving the (IBVP) on an adaptive, nonuniform grid.

4 Numerical Solution of the IBVP for Evolution Equations

The numerical scheme for discretizing (IBVP) depends on $f(u_{xx}, u_x, u, x)$. The proposed grid adaptation algorithm will work for many numerically stable discretization schemes for (IBVP). We use different schemes for different numerical examples discussed in this paper depending on the problem. Hence, in the next section we only describe the techniques we use for calculating the

spatial derivatives u_x and u_{xx} on the nonuniform grid and we state the numerical schemes in the examples themselves.

4.1 Calculation of the Spatial Derivatives

For calculating the derivative u_x on the adaptive nonuniform grid $Grid_{\text{new}}$ we extend the essentially nonoscillatory (ENO) [23, 40, 41] and weighted ENO (WENO) schemes [28, 29, 32, 37] to nonuniform grids. To this end, let the nonuniform grid be given as in (8). Now define

$$D^+u^n_{j_i,k_i} = \frac{u^n_{j_{i+1},k_{i+1}} - u^n_{j_i,k_i}}{x_{j_{i+1},k_{i+1}} - x_{j_i,k_i}}, \qquad D^-u^n_{j_i,k_i} = \frac{u^n_{j_i,k_i} - u^n_{j_{i-1},k_{i-1}}}{x_{j_i,k_i} - x_{j_{i-1},k_{i-1}}}. \tag{16}$$

A third-order ENO approximation to $(u^{\pm}_x)^n_{j_i,k_i} = u^{\pm}_x(x_{j_i,k_i}, n\Delta t)$ is given by one of the following expressions

$$((u^{\pm}_x)^n_{j_i,k_i})_1 = \frac{v_1}{3} - \frac{7v_2}{6} + \frac{11v_3}{6} \tag{17}$$

or

$$((u^{\pm}_x)^n_{j_i,k_i})_2 = -\frac{v_2}{6} + \frac{5v_3}{6} + \frac{v_4}{3} \tag{18}$$

or

$$((u^{\pm}_x)^n_{j_i,k_i})_3 = \frac{v_3}{3} + \frac{5v_4}{6} - \frac{v_5}{6}, \tag{19}$$

where for calculating $(u^-_x)^n_{j_i,k_i}$, we use $v_1 = D^-u^n_{j_{i-2},k_{i-2}}$, $v_2 = D^-u^n_{j_{i-1},k_{i-1}}$, $v_3 = D^-u^n_{j_i,k_i}$, $v_4 = D^-u^n_{j_{i+1},k_{i+1}}$, $v_5 = D^-u^n_{j_{i+2},k_{i+2}}$, and for calculating $(u^+_x)^n_{j_i,k_i}$, we use $v_1 = D^+u^n_{j_{i+2},k_{i+2}}$, $v_2 = D^+u^n_{j_{i+1},k_{i+1}}$, $v_3 = D^+u^n_{j_i,k_i}$, $v_4 = D^+u^n_{j_{i-1},k_{i-1}}$, $v_5 = D^+u^n_{j_{i-2},k_{i-2}}$. The basic idea behind a third-order ENO scheme is to choose either $((u^{\pm}_x)^n_{j_i,k_i})_1$ or $((u^{\pm}_x)^n_{j_i,k_i})_2$ or $((u^{\pm}_x)^n_{j_i,k_i})_3$ for approximating $(u^{\pm}_x)^n_{j_i,k_i}$ by choosing the smoothest possible polynomial interpolation of u.

It is reminded that a WENO approximation of $(u^{\pm}_x)^n_{j_i,k_i}$ is a convex combination of the approximations in equations (17), (18) and (19), that is,

$$(u^{\pm}_x)^n_{j_i,k_i} = \sum_{\ell=1}^{3} \omega_\ell ((u^{\pm}_x)^n_{j_i,k_i})_\ell, \tag{20}$$

where $0 \le \omega_\ell \le 1$ for $\ell = 1, 2, 3$ and $\omega_1 + \omega_2 + \omega_3 = 1$. The weights for a fifth-order accuracy are given by [28, 37]

$$\omega_\ell = \frac{\alpha_\ell}{\alpha_1 + \alpha_2 + \alpha_3}, \qquad \ell = 1, 2, 3, \tag{21}$$

where

$$\alpha_\ell = \frac{\bar{\alpha}_\ell}{(S_\ell + \delta)^2}, \qquad \ell = 1, 2, 3, \tag{22}$$

$$S_1 = \frac{13}{12}(v_1 - 2v_2 + v_3)^2 + \frac{1}{4}(v_1 - 4v_2 + 3v_3)^2, \tag{23}$$

$$S_2 = \frac{13}{12}(v_2 - 2v_3 + v_4)^2 + \frac{1}{4}(v_2 - v_4)^2, \tag{24}$$

$$S_3 = \frac{13}{12}(v_3 - 2v_4 + v_5)^2 + \frac{1}{4}(3v_3 - 4v_4 + v_5)^2, \tag{25}$$

and

$$\bar{\alpha}_1 = 0.1, \quad \bar{\alpha}_2 = 0.6, \quad \bar{\alpha}_3 = 0.3. \tag{26}$$

In (22) δ is used to prevent the denominator from becoming zero. In our computations, we have used $\delta = 10^{-6}$.

For calculating u_{xx}, we extend the centered second difference scheme for the uniform mesh [42] to the nonuniform mesh (8) as follows

$$(u_{xx})^n_{j_i,k_i} = \frac{2\left(\frac{u^n_{j_{i+1},k_{i+1}} - u^n_{j_i,k_i}}{x^n_{j_{i+1},k_{i+1}} - x^n_{j_i,k_i}} - \frac{u^n_{j_i,k_i} - u^n_{j_{i-1},k_{i-1}}}{x^n_{j_i,k_i} - x^n_{j_{i-1},k_{i-1}}}\right)}{x^n_{j_{i+1},k_{i+1}} - x^n_{j_{i-1},k_{i-1}}}. \tag{27}$$

Now we are ready to give the algorithm for solving IBVP for evolution equations (1).

4.2 Solution of the IBVP for Evolution PDEs

Based on the problem and the desired accuracy we choose the minimum resolution level $J_{\min}$, the maximum resolution level $J_{\max}$, the threshold ϵ, the order of the interpolating polynomial p and the parameters N_1, N_2 required for the GRIDSELECTION procedure. The final time t_f is assumed to be given.

To solve (IBVP) on an adaptive grid, we first initialize $Grid_{\text{old}} = \mathcal{V}_{J_{\max}}$ and $U_{\text{old}} = \{g(x_{J_{\max},k})\}_{k=0}^{2^{J_{\max}}}$. Next, we find the new grid $Grid_{\text{new}}$ and the function values at all the points in $Grid_{\text{new}}$, $U_{\text{new}} = \{u^n_{j,k} : x_{j,k} \in Grid_{\text{new}}\}$, using the GRIDSELECTION procedure. The new grid $Grid_{\text{new}}$ is the grid on which we will propagate the solution from time $t = 0$ to time $t = \Delta t_{\text{adapt}}$, where Δt_{adapt} is the time after which the grid should be adapted again. It is calculated based on the approximate time the solution will take to move N_1 grid points. Therefore, the value of N_1 will dictate the frequency of mesh adaptation. The larger the N_1, the smaller the frequency of mesh adaptation will be, at the expense of a larger number of grid points in the adaptive grid. Hence, there is a trade-off between the frequency of mesh adaptation and the number of grid points. Next, we find the solution at time $t + \Delta t$ at all the points belonging to $Grid_{\text{new}}$, with values $U_{\text{new}} = \{u^{n+1}_{j,k} : x_{j,k} \in Grid_{\text{new}}\}$, using any numerically stable scheme. This way we keep on propagating the solution on $Grid_{\text{new}}$ until $t < \Delta t_{\text{adapt}}$. We then reassign the sets $Grid_{\text{old}}$, U_{old} to $Grid_{\text{new}}$, U_{new} respectively, and find the new time at which the next mesh adaptation should take place $t_{\text{adapt}} = t + \Delta t_{\text{adapt}}$. We keep on repeating the above mentioned steps until the final time t_f is reached. A pseudocode of the above algorithm is as follows:

IBVP_EE($g, t_f, J_{\min}, J_{\max}, \mathcal{V}_{J_{\max}}, p, \epsilon, N_1, N_2$)

$t = 0,\ n = 0;$
$Grid_{\text{old}} = \mathcal{V}_{J_{\max}};\ U_{\text{old}} = \{g(x_{J_{\max},k})\}_{k=0}^{2^{J_{\max}}};$
$t_{\text{adapt}} = \Delta t_{\text{adapt}};$
while $(t \le t_f)$ **do**
  $[Grid_{\text{new}}, U_{\text{new}}] = \text{GRIDSELECTION}(Grid_{\text{old}}, U_{\text{old}}, J_{\min}, J_{\max}, \mathcal{V}_{J_{\max}}, p, \epsilon, N_1, N_2);$
  while $t < t_{\text{adapt}}$ **do**
    find $U_{\text{new}} = \{u_{j,k}^{n+1} : x_{j,k} \in Grid_{\text{new}}\}$ using any numerically stable scheme;
    $t = t + \Delta t;\ n = n + 1;$
  Empty $Grid_{\text{old}}$, U_{old};
  $Grid_{\text{old}} = Grid_{\text{new}};\ U_{\text{old}} = U_{\text{new}};$
  Empty $Grid_{\text{new}}$, U_{new};
  $t_{\text{adapt}} = t + \Delta t_{\text{adapt}};$
$Grid_{\text{new}} = Grid_{\text{old}};\ U_{\text{new}} = U_{\text{old}}.$

5 Numerical Examples

In this section we present several examples to demonstrate the stability and robustness of our algorithm. These examples also validate the extension of ENO, WENO, and second difference schemes to the nonuniform grid and illustrate the algorithm's ability to automatically capture and follow any existing or self-sharpenning features of the solution that develop in time.

Example 5
Consider the linear advection equation

$$u_t + u_x = 0, \tag{28}$$

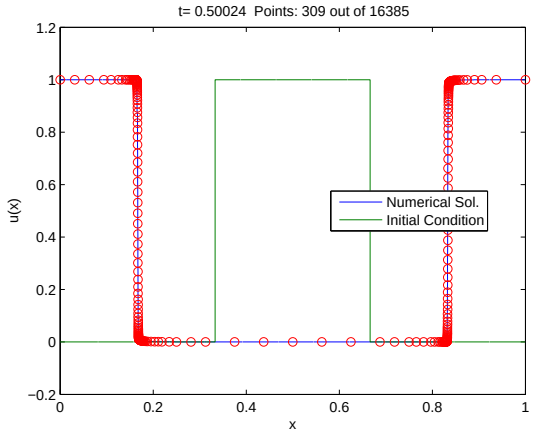
with the periodic boundary condition

$$u(0,t) = u(1,t), \tag{29}$$

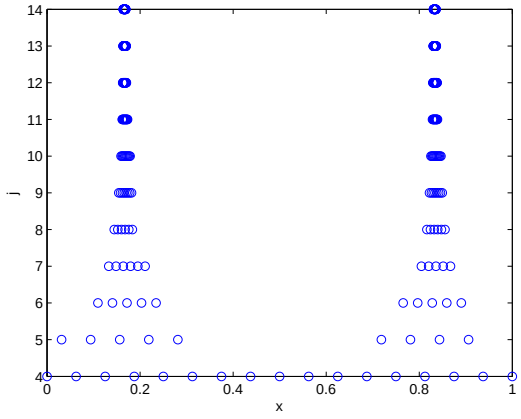
and initial condition

$$g(x) = \begin{cases} 1, & 1/3 \le x \le 2/3, \\ 0, & \text{otherwise.} \end{cases} \tag{30}$$

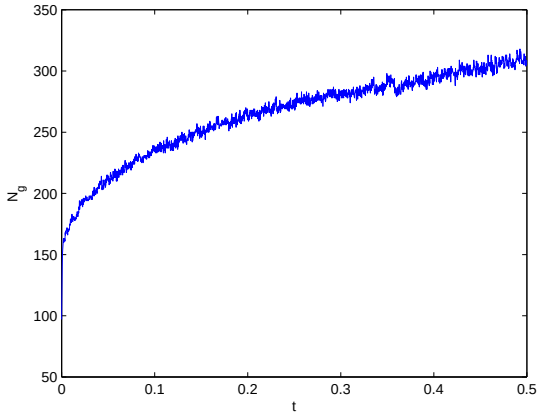
The final time is $t_f = 0.5\,\text{s}$. We approximate u_x by u_x^- using a third-order ENO scheme and use a third-order TVD RK scheme [40] for temporal integration. The numerical solution at the final time using a grid with $J_{\min} = 4$ and $J_{\max} = 14$ is shown in Figure 5(a). The other parameters used in the GRIDSELECTION procedure are $p = 1$, $\epsilon = 0.001/2^{j/2}$, $N_1 = 2$, and $N_2 = 1$. The grid point distribution in terms of position and resolution level at $t = 0.50024\,\text{s}$ is shown in Figure 5(b). The regions where the solution is smooth are well represented by the coarse resolution levels; the higher resolution levels are needed only near the edges of the step function, thus illustrating the efficient data compression of the adaptive strategy proposed in this paper. Since the numerical scheme is dissipative, the sharp edges of the step function tend to smear out continuously. Hence, the grid adaptation algorithm is continuously adding points near the edges to maintain the desired accuracy. This can also be observed by looking at the time evolution of the number of grid points in Figure 5(c). We see that the number of grid points is constantly increasing, which again shows the efficiency of the proposed grid adaptation algorithm. A comparison of CPU times for the uniform



(a) Solution $u(x, 0.50024)$.



(b) Grid point distribution at $t = 0.50024$ s.



(c) Time evolution of the number of grid points.

Figure 5: Example 5. Parameters used in the simulation are $J_{\min} = 4$, $J_{\max} = 14$, $p = 1, \epsilon = 0.001/2^{j/2}$, $N_1 = 2$, and $N_2 = 1$.

and adaptive grids, along with the average number of points used for $J_{\max} = 10, 11, 12, 13, 14$ are summarized in Table 4.

In the next two examples we consider the evolution equation (1) with

$$f(u_{xx}, u_x, u, x) = f(u_x, u, x). \tag{31}$$

We use the Lax-Friedrich's (LF) scheme [14, 37, 38, 41] for discretizing $f(u_x, u, x)$ as follows

$$f(u_x, u, x) = \hat{f}^{\text{LF}}(u_x^-, u_x^+, u, x) = f\left(\frac{u_x^+ + u_x^-}{2}, u, x\right) - \frac{1}{2}\alpha^x(u_x^+ - u_x^-), \tag{32}$$

where,

$$\alpha^x = \max_{u_x \in I^x} |f_1(u_x, u, x)|, \tag{33}$$

where f_1 is the partial derivative of f with respect to u_x, $I^x = [u_x^{\min}, u_x^{\max}]$, and the minimum and the maximum values of u_x are identified by considering all the values of u_x^- and u_x^+ on the

	Uniform Mesh		Adaptive Mesh		
$J_{\max}$	$N_{\rm g}$ in $\mathcal{V}_{J_{\max}}$	$t_{\rm cpu}$ (s)	$N_{\rm g_{ave}}$ in $Grid_{\rm new}$	$t_{\rm cpu}$ (s)	Speed Up
10	$2^{10}+1=1025$	0.2715	135	0.0907	3.0
11	$2^{11}+1=2049$	1.4821	159	0.1914	7.7
12	$2^{12}+1=4097$	5.6090	188	0.4788	11.7
13	$2^{13}+1=8193$	22.2160	222	1.3575	16.4
14	$2^{14}+1=16385$	85.6133	264	4.1690	20.5

Table 4: Example 5. Computational times for uniform vs. adaptive mesh.

nonuniform grid.

Example 6
We again consider the linear advection equation

$$u_t - u_x = 0, \tag{34}$$

with periodic boundary condition

$$u(0,t) = u(1,t), \tag{35}$$

and initial condition

$$g(x) = \sin(2\pi x) + e^{-\beta(x-1/2)^2}. \tag{36}$$

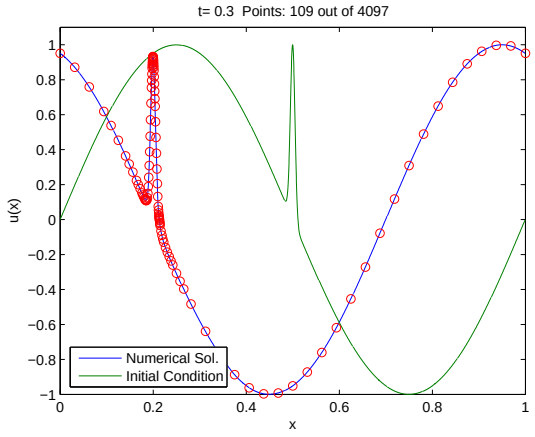
As mentioned earlier the initial function is mostly smooth with a peak located at $x = 0.5$. Here we again consider $\beta = 20000$. The final time is $t_f = 0.3\,$s. Since we are solving a linear advection equation, we may approximate u_x by u_x^+ as we did in the previous example where we approximated u_x by u_x^-. We use an LF scheme to check the accuracy of the scheme on the resulting nonuniform grid since the exact solution of this problem is known. We approximate the derivatives u_x^+, u_x^- in the LF discretization using a WENO scheme and use a third-order TVD RK scheme for temporal integration. The numerical results using a grid with $J_{\min} = 5$ and $J_{\max} = 14$ are shown in Figure 6. The other parameters used in the GRIDSELECTION procedure are $p = 3$, $\epsilon = 0.02/2^{j/2}$, and $N_1 = N_2 = 2$. As can be seen from Figure 6, the proposed grid adaptation algorithm performed well with the LF scheme combined with a third-order TVD RK scheme. The grid point distribution (Figure 6(b)) again validates the efficiency of the proposed algorithm.

A comparison of CPU times for the uniform and adaptive grids, along with the average number of points used for $J_{\max} = 10, 11, 12, 13, 14$ are summarized in Table 5. We see a major speed up in the computational time compared to the uniform mesh, and the speed-up factors increase significantly with mesh refinement at an approximate rate of two. There is a trade-off between

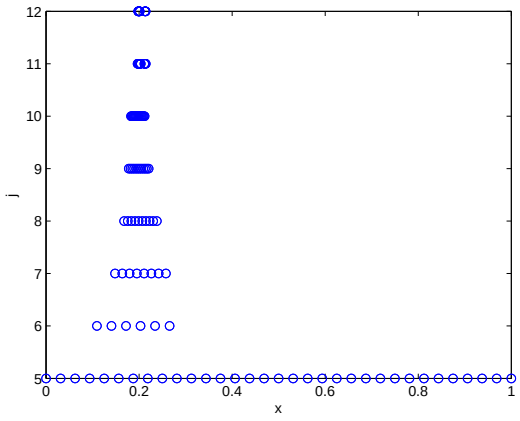
	Uniform Mesh		Adaptive Mesh		
$J_{\max}$	$N_{\rm g}$ in $\mathcal{V}_{J_{\max}}$	$t_{\rm cpu}$ (s)	$N_{\rm g_{ave}}$ in $Grid_{\rm new}$	$t_{\rm cpu}$ (s)	Speed Up
9	$2^{9}+1=513$	8.0605	70	1.2635	6.4
10	$2^{10}+1=1025$	43.3547	87	3.9434	11.0
11	$2^{11}+1=2049$	190.4138	103	8.0187	23.7
12	$2^{12}+1=4097$	747.6864	105	14.6014	51.2

Table 5: Example 6. Computational times for uniform vs. adaptive mesh.

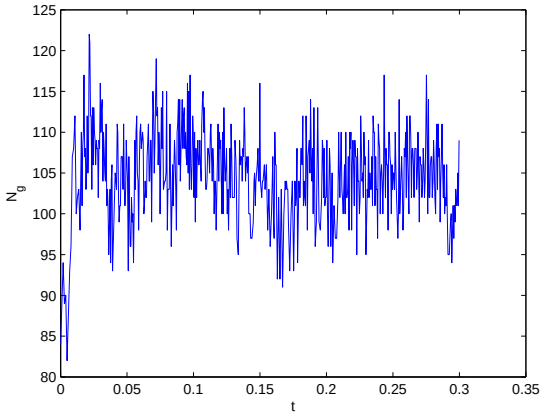
accuracy and speed up of computations. We solved this problem again for $J_{\max} = 12$ using the



(a) Solution $u(x, 0.3)$.



(b) Grid point distribution at time $t = 0.3\,\mathrm{s}$.



(c) Time evolution of the number of grid points.

Figure 6: Example 6. Parameters used in the simulation are $J_{\min} = 5$, $J_{\max} = 12$, $p = 3, \epsilon = 0.02/2^{j/2}$, and $N_1 = N_2 = 2$.

same parameters as before, but this time with $\epsilon = 0.01/2^{j/2}$ and $\epsilon = 0.005/2^{j/2}$. The numerical solution at the final time for both thresholds are shown in Figure 7. From Figure 7 we see that as we decrease the threshold, the dissipation in the height of the peak is decreasing at the cost of a greater CPU time, 22.1503 s for $\epsilon = 0.01/2^{j/2}$ and 29.0653 s for $\epsilon = 0.005/2^{j/2}$. However, it should be noted that the CPU time for both the cases is still significantly less compared to the CPU time for the case of a uniform mesh.

Example 7

In this example we solve the inviscid Burger's equation

$$u_t + uu_x = 0. \tag{37}$$

Burger's equation is similar to the linear advection equations (28) and (34), except from the fact that the convective term is nonlinear, thus allowing discontinuous solutions to develop in time even if the initial condition is smooth. We use the same smooth initial condition and the Dirichlet

boundary condition as in [3], that is,

$$g(x) = \sin(2\pi x) + \frac{1}{2}\sin(\pi x), \tag{38}$$

$$u(0,t) = u(1,t) = 0, \tag{39}$$

to check the ability of the proposed algorithm to capture the shock. The solution is a wave that develops a very steep gradient and subsequently moves towards $x = 1$. Because of the zero boundary values, the wave amplitude diminishes with increasing time.

We use a WENO scheme for calculating the derivatives u_x^-, u_x^+ in the LF discretization and we use a third-order TVD RK method for temporal integration. The numerical solutions at different times $t = 0\,\mathrm{s}$, $t = 0.158\,\mathrm{s}$, $t = 0.5\,\mathrm{s}$ and $t = 1\,\mathrm{s}$ using a grid with $J_{\min} = 4$ and $J_{\max} = 12$ are shown in Figure 8(a). The other parameters used in the GRIDSELECTION procedure are $p = 3$, $\epsilon = 0.0005$, $N_1 = 2$, and $N_2 = 1$. Figure 8 also shows the grid point distribution in the adaptive mesh at times $t = 0\,\mathrm{s}$, $t = 0.14\,\mathrm{s}$, $t = 0.158\,\mathrm{s}$ and $t = 1\,\mathrm{s}$. We see that as the shock continues to develop the algorithm adds points at the finer levels of resolution in the region where the shock is developing, and removes points from the regions where the solution is getting smoother and smoother. Similar conclusions can be drawn by looking at the time evolution of the number of grid points (Figure 8(f)). We see in the Figure 8(f) that the number of grid points is increasing as the shock continues to develop, and once the discontinuity appears in the solution the number of grid points starts to decrease since the solution is getting smoother in most of the remaining region. Once the solution is smooth everywhere except for the region of the shock the number of grid points is pretty much steady oscillating about a mean value of 68. This clearly shows that the proposed strategy uses only the grid points that are actually necessary to attain a given precision, and the algorithm is able to add and remove points when and where is needed.

A comparison of CPU times for the uniform and adaptive grids, along with the average number of points used for different $J_{\max} = 8, 9, 10, 11, 12$ are summarized in Table 6. We observe a major speed up in the computational time compared to the uniform mesh, and the speed-up factors are again increasing at an approximate rate of two with mesh refinement showing the efficiency and good performance of our algorithm.

	Uniform Mesh		Adaptive Mesh		
$J_{\max}$	N_g in $\mathcal{V}_{J_{\max}}$	t_{cpu} (s)	$N_{g_{\text{ave}}}$ in $Grid_{\text{new}}$	t_{cpu} (s)	Speed Up
8	$2^8 + 1 = 257$	2.5474	49	0.5319	4.8
9	$2^9 + 1 = 513$	18.1617	55	2.1699	8.4
10	$2^{10} + 1 = 1025$	137.6709	59	9.1435	15.1
11	$2^{11} + 1 = 2049$	1142.5316	65	43.8749	26.0
12	$2^{12} + 1 = 4097$	9352.2561	69	188.9315	49.5

Table 6: Example 7. Computational times for uniform vs. adaptive mesh.

Example 8

Consider the scalar reaction-diffusion problem from combustion theory [1, 31]

$$u_t - u_{xx} - \frac{Re^{\delta}}{a\delta}(1 + a - u)e^{-\delta/u} = 0, \tag{40}$$

$$u_x(0,t) = 0, \quad u(1,t) = 1, \tag{41}$$

$$u(x,0) = 1. \tag{42}$$

The solution u represents the temperature of a reactant in a chemical system, a is the heat release, δ is the activation energy, and R is the reaction rate. For small times the temperature gradually increases from unity with a "hot spot" forming at $x = 0$. At a finite time, ignition occurs, and the temperature at $x = 0$ jumps rapidly from near unity to near $1 + a$. A flame front then forms and propagates towards $x = 1$ with speed proportional to $e^{a\delta/2(1+a)}$. In real problems, a is about unity and δ is large, thus the flame front moves exponentially fast after ignition. We use the same problem parameters as in [1] namely, $a = 1$ and $R = 5$. We use $\delta = 30$ as in [26, 39] since the flame layer in this case is much thinner, and higher mesh adaptation is required.

We use (27) to discretize u_{xx} and use a second-order TVD RK scheme [40] for temporal integration. To illustrate how we apply Neumann boundary condition on a nonuniform grid we again consider the grid of the form (8). To apply the Neumann boundary condition $u_x(0,t) = 0$ we introduce a fictitious node $x_{j_{-1},k_{-1}} = -x_{j_1,k_1}$. We approximate the boundary condition by[3]

$$(u_x)^n_{j_0,0} = \frac{u^n_{j_1,k_1} - u^n_{j_{-1},k_{-1}}}{x_{j_1,k_1} - x_{j_{-1},k_{-1}}} = 0, \tag{43}$$

which implies

$$u^n_{j_{-1},k_{-1}} = u^n_{j_1,k_1}. \tag{44}$$

Hence, at the boundary $x = 0$, equation (27) reduces to

$$(u_{xx})^n_{j_0,0} = \frac{2\left(\frac{u^n_{j_1,k_1} - u^n_{j_0,0}}{x^n_{j_1,k_1} - x^n_{j_0,0}} - \frac{u^n_{j_0,0} - u^n_{j_{-1},k_{-1}}}{x^n_{j_0,0} - x^n_{j_{-1},k_{-1}}}\right)}{x^n_{j_1,k_1} - x^n_{j_{-1},k_{-1}}} = \frac{2(u^n_{j_1,k_1} - u^n_{j_0,0})}{(x^n_{j_1,k_1} - x^n_{j_0,0})^2}. \tag{45}$$

The numerical solutions at times $t = 0\,\mathrm{s}$, $t = 0.24\,\mathrm{s}$, $t = 0.2403\,\mathrm{s}$, $t = 0.24035\,\mathrm{s}$, $t = 0.241\,\mathrm{s}$ and $t = 0.244\,\mathrm{s}$ using a grid with $J_{\min} = 4$ and $J_{\max} = 10$ are shown in Figure 9(a). The other parameters used in the GRIDSELECTION procedure are $p = 3$, $\epsilon = 0.001$, $N_1 = 2$, and $N_2 = 1$. One of the main challenges of this problem is the fact that one needs to use a very small time step to capture the transition layer during the time of ignition. This is achieved automatically by the proposed algorithm since the proposed algorithm is adaptive both in time and space. As the mesh gets refined, Δt in the procedure IBVP_EE also decreases. In Figure 9(f) we show the time evolution of grid points starting at $t = 0.23\,\mathrm{s}$ since the number of points for $t \le 0.23\,\mathrm{s}$ remains constant. We see from Figure 9(f) that for time $t < 0.239\,\mathrm{s}$ the proposed algorithm found the solution at grid $\mathcal{V}_4$ and efficiently added points at finer levels starting at $t = 0.239\,\mathrm{s}$. As the points from finer grid levels are being added, the algorithm automatically decreases the time step and is able to accurately capture the transition layer during the time of ignition.

A comparison of CPU times for the uniform and adaptive grids, along with the average number of points used for $J_{\max} = 8, 9, 10$ are summarized in Table 7. We see again that speed-up factors are increasing with mesh refinement.

6 Conclusions

In this paper, we have proposed a novel grid adaptation algorithm, which is shown to outperform the traditional grid adaptation approach. Specifically, we showed that the proposed approach results

[3]Note that $k_0 = 0$.

	Uniform Mesh		Adaptive Mesh		
$J_{\max}$	$N_{\rm g}$ in $\mathcal{V}_{J_{\max}}$	$t_{\rm cpu}$ (s)	$N_{\rm g_{ave}}$ in $Grid_{\rm new}$	$t_{\rm cpu}$ (s)	Speed Up
8	$2^8+1=257$	11.3496	39	0.9141	12.4
9	$2^9+1=513$	86.0208	51	3.9216	21.94
10	$2^{10}+1=1025$	646.5237	59	18.4145	35.1

Table 7: Example 8. Computational times for uniform vs. adaptive mesh.

in fewer grid points in the grid compared to the traditional approach. For the examples considered here, we observed savings of up to 33% in terms of the number of grid points. We have solved the initial-boundary value problem for evolution equations in a computationally efficient manner using the proposed grid adaptation algorithm. The proposed algorithm is adaptive both in space and time. Moreover, the frequency of mesh adaptation is dynamically adjusted in order to optimize the algorithm even further. Several examples have demonstrated the stability and robustness of the proposed algorithm. The grid was able to adapt dynamically to any existing or emerging irregularities in the solution, and the grid point distribution (in terms of position and resolution level) is evident of this fact. In all examples considered, the algorithm was able to automatically allocate more grid points to the region where the solution exhibits sharp features and fewer points to the region where the solution is smooth, thereby reducing the computational time as well as the memory usage dramatically, while maintaining an accuracy equivalent to one obtained using a fine uniform mesh.

References

[1] S. Adjerid and J.E. Flahery. A moving finite element method with error estimation and refinement for one-dimensional time dependent partial differential equations. *SIAM Journal on Numerical Analysis*, 23:778–795, 1986.

[2] S. Adjerid and J.E. Flahery. A moving mesh finite element method with local refinement for parabolic partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 56:3–26, 1986.

[3] M.A. Alves, P. Cruz, A. Mendes, F.D. Magalhães, F.T. Pinho, and P.J. Oliveira. Adaptive multiresolution approach for solution of hyperbolic PDEs. *Computer Methods in Applied Mechanics and Engineering*, 191:3909–3928, 2002.

[4] D.C. Arney and J.E. Flahery. A two-dimensional mesh moving technique for time dependent partial differential equations. *Journal of Computational Physics*, 67:124–144, 1986.

[5] I. Babuška, J. Chandra, and J.E. Flaherty, editors. *Adaptive Computational Methods for Partial Differential Equations*. SIAM, Philadelphia, 1983.

[6] M.J. Baines. *Moving Finite Elements*. Oxford University Press, New York, 1994.

[7] M.J. Baines and A.J. Wathen. Moving finite element methods for evolutionary problems. I. Theory. *Journal of Computational Physics*, 79:245–269, 1988.

[8] J. Bell, M.J. Berger, J. Saltzman, and M. Welcome. Three-dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM Journal on Scientific Computing*, 15:127–138, 1994.

[9] M.J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84, 1989.

[10] M.J. Berger and R.J. Leveque. Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems. *SIAM Journal on Numerical Analysis*, 35:2298–2316, 1998.

[11] M.J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, 1984.

[12] S. Bertoluzza. Adaptive wavelet collocation method for the solution of Burger's equation. *Transport Theory and Statistical Physics*, 25:339–352, 1996.

[13] N.N. Carlson and K. Miller. Design and application of a gradient weighted moving finite element code I: In one dimension. *SIAM Journal on Scientific Computing*, 19:728–765, 1998.

[14] M. Crandall and P.L. Lions. Two approximations of solutions of Hamilton-Jacobi equations. *Math. Comput.*, 43:1–19, 1984.

[15] G. Deslauriers and S. Dubuc. Symmetric iterative interpolation processes. *Constructive Approximation*, 5:49–68, 1989.

[16] D. L. Donoho. Interpolating wavelet transforms. Technical report, Department of Statistics, Stanford University, Stanford, CA, 1992.

[17] E.A. Dorfi and L.O'c. Drury. Simple adaptive grids for 1-d initial value problems. *Journal of Computational Physics*, 69:175–195, 1987.

[18] P.H. Gaskell and A.K.C. Lau. Curvature compensated convective transport: SMART, a new boundedness preserving transport algorithm. *International Journal for Numerical Methods in Fluids*, 8:617–641, 1988.

[19] A. Harten. High resolution schemes for hyperbolic conservation laws. *Journal of Computational Physics*, 49:357–393, 1983.

[20] A. Harten. Adaptive multiresolution schemes for shock computations. *Journal of Computational Physics*, 115:319–338, 1994.

[21] A. Harten. Multiresolution algorithms for the numerical solution of hyperbolic conservation laws. *Comm. Pure Appl. Math.*, 48:1305–1342, 1995.

[22] A. Harten. Multiresolution representation of data: A general framework. *SIAM Journal on Numerical Analysis*, 33(3):1205–1256, 1996.

[23] A. Harten, B. Enquist, S. Osher, and S. Chakravarthy. Uniformly high-order accurate essentially non-oscillatory schemes III. *Journal of Computational Physics*, 71:231–303, 1987.

[24] M. Holmstrom. Solving hyperbolic PDEs using interpolating wavelets. *SIAM Journal on Scientific Computing*, 21, 1999.

[25] M. Holmstrom and J. Walden. Adaptive wavelet methods for hyperbolic PDEs. *Journal of Scientific Computing*, 13(1):19–49, 1998.

[26] W. Huang, Y. Ren, and R.D. Russell. Moving mesh methods based on moving mesh partial differential equations. *Journal of Computational Physics*, 113:279–290, 1994.

[27] L. Jameson. A wavelet-optimized, very high order adaptive grid and order numerical method. *SIAM Journal on Scientific Computing*, 19:1980–2013, 1998.

[28] G.S. Jiang and D. Peng. Weighted ENO schemes for Hamilton-Jacobi equations. *SIAM Journal on Scientific Computing*, 21(6):2126–2143, 2000.

[29] G.S. Jiang and C.W. Shu. Efficient implementation of weighted ENO schemes. *Journal of Computational Physics*, 126(1):202–228, 1996.

[30] I.W. Johnson, A.J. Wathen, and M.J. Wathen. Moving finite element methods for evolutionary problems. II. Applications. *Journal of Computational Physics*, 79:270–297, 1988.

[31] A.K. Kapila. *Asymptotic treatment of chemically reacting systems.* Pitman Advanced Publishing Program, Boston, 1983.

[32] X.D. Liu, S. Osher, and T. Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115(1):200–212, 1994.

[33] S. G. Mallat. Multiresolution approximations and wavelet orthonormal bases of $L^2(\mathbb{R})$. In *Transactions of the American Mathematical Society*, volume 315, pages 69–87, 1989.

[34] S.G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intellegence*, 2(7):674–693, 1989.

[35] K. Miller. Moving finite elements II. *SIAM Journal on Numerical Analysis*, 18:1033–1057, 1981.

[36] K. Miller and R.N. Miller. Moving finite elements I. *SIAM Journal on Numerical Analysis*, 18:1019–1032, 1981.

[37] S. Osher and R.P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces.* Springer, New York, 2003.

[38] S. Osher and C.W. Shu. High order essentially non-oscillatory schemes for Hamilton-Jacobi equations. *SIAM Journal of Numerical Analysis*, 28:902–921, 1991.

[39] L.R. Petzold. Observations on an adaptive moving grid method for one-dimensional systems for partial differential equations. *Applied Numerical Mathematics*, 3:347–360, 1987.

[40] C.W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock capturing schemes. *Journal of Computational Physics*, 77:439–471, 1988.

[41] C.W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock capturing schemes II. *Journal of Computational Physics*, 83:32–78, 1989.

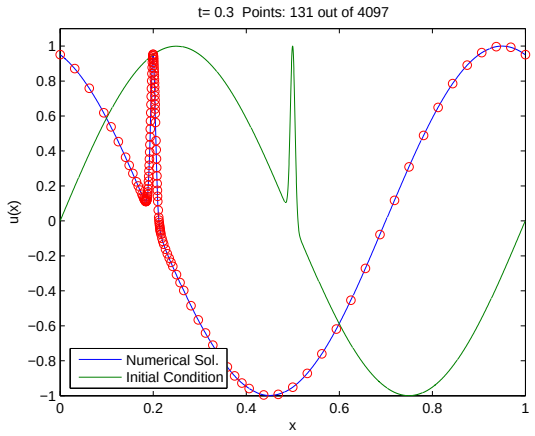
[42] J.W. Thomas. *Numerical Partial Differential Equations.* Springer, NY, 1995.

[43] J.F. Thompson. A survey of dynamically-adaptive grids in the numerical solution of partial differential equations. *Applied Numerical Mathematics*, 1:3–27, 1985.

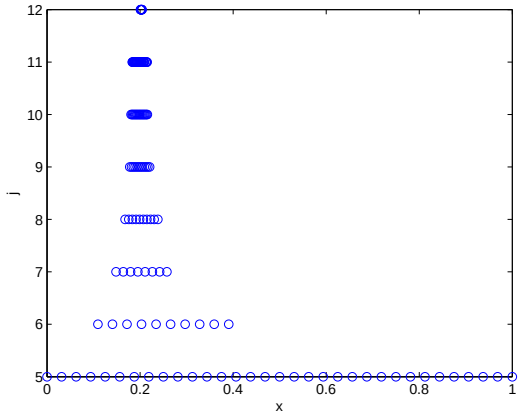
[44] O.V. Vasilyev. Solving multi-dimensional evolution problems with localized structures using second generation wavelets. *International Journal of Computational Fluid Dynamics*, 17(2):151–168, 2003.

[45] O.V. Vasilyev and C. Bowman. Second-generation wavelet collocatioin method for the solution of partial differential equations. *Journal of Computational Physics*, 165:660–693, 2000.

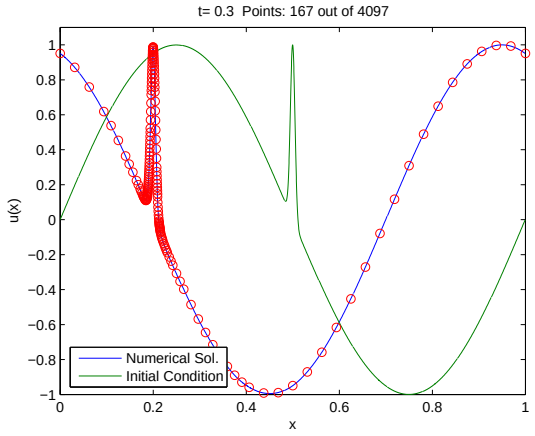
[46] J. Waldén. Filter bank methods for hyperbolic PDEs. *Journal of Numerical Analysis*, 36:1183–1233, 1999.



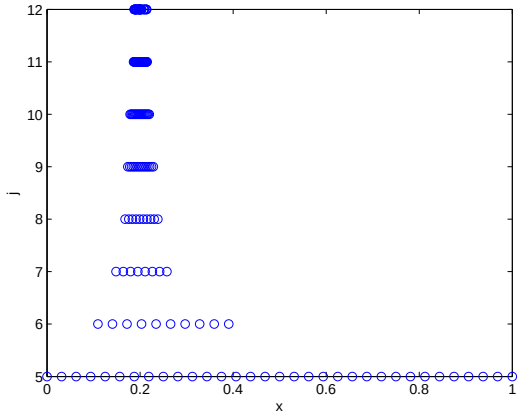
(a) Solution $u(x, 0.3)$ for $\epsilon = 0.01/2^{j/2}$.



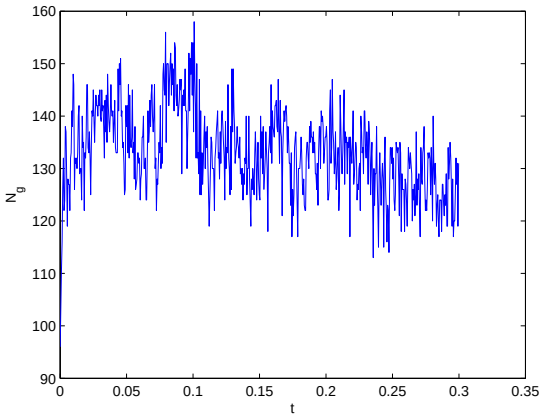
(b) Grid point distribution at time $t = 0.3\,\mathrm{s}$ for $\epsilon = 0.01/2^{j/2}$.



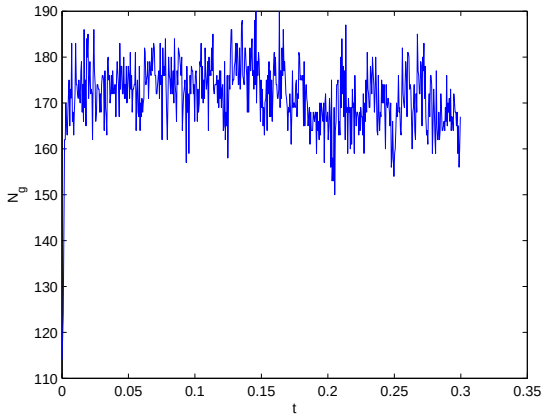
(c) Solution $u(x, 0.3)$ for $\epsilon = 0.005/2^{j/2}$.



(d) Grid point distribution at time $t = 0.3\,\mathrm{s}$ for $\epsilon = 0.005/2^{j/2}$.

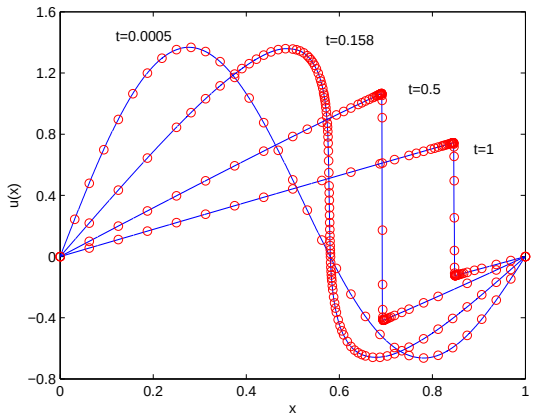


(e) Time evolution of the number of grid points for $\epsilon = 0.01/2^{j/2}$.

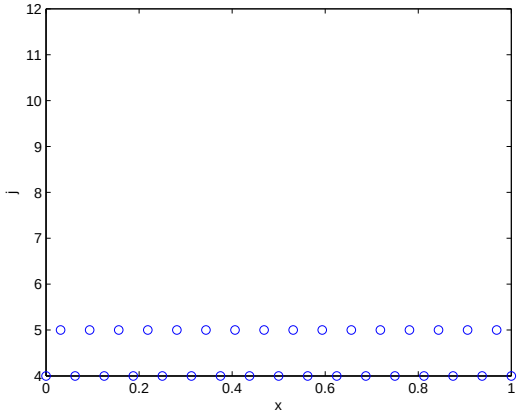


(f) Time evolution of the number of grid points for $\epsilon = 0.005/2^{j/2}$.

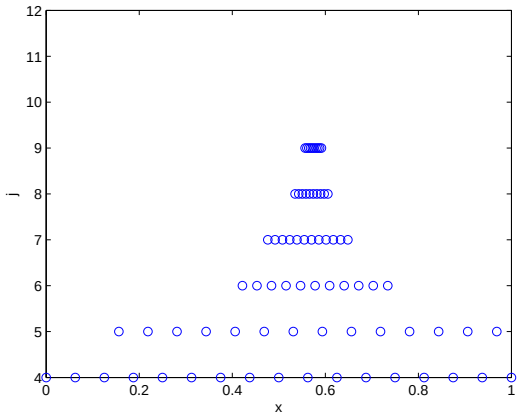
Figure 7: Example 6. Parameters used in the simulation are $J_{\min} = 5$, $J_{\max} = 12$, $p = 3$, and $N_1 = N_2 = 2$.



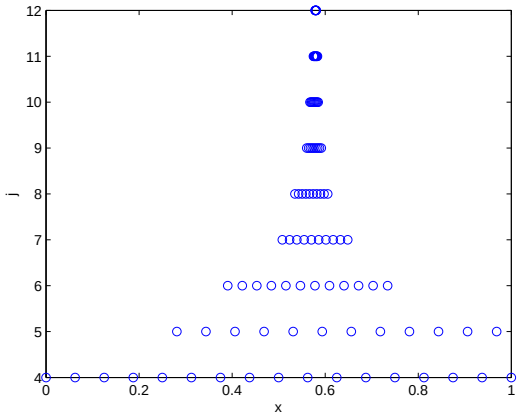
(a) Solution $u(x,t)$.



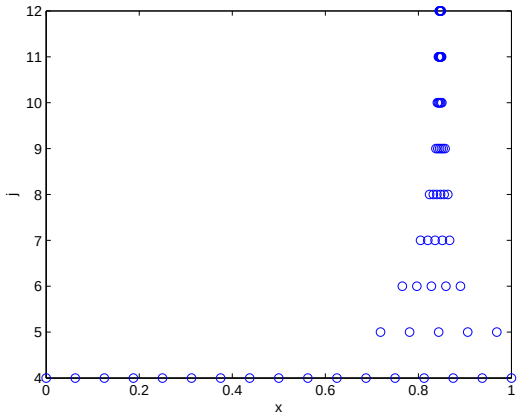
(b) Grid point distribution at $t = 0.0005\,\mathrm{s}$.



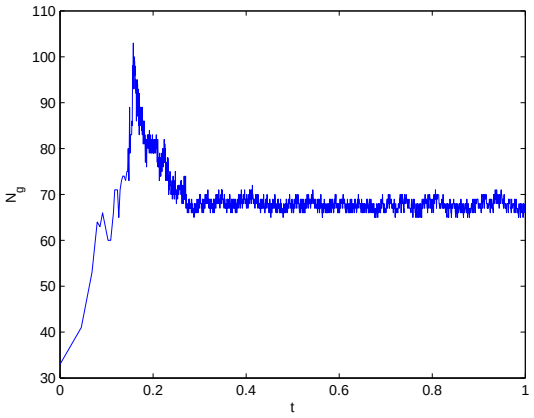
(c) Grid point distribution at $t = 0.14\,\mathrm{s}$.



(d) Grid point distribution at $t = 0.158\,\mathrm{s}$.

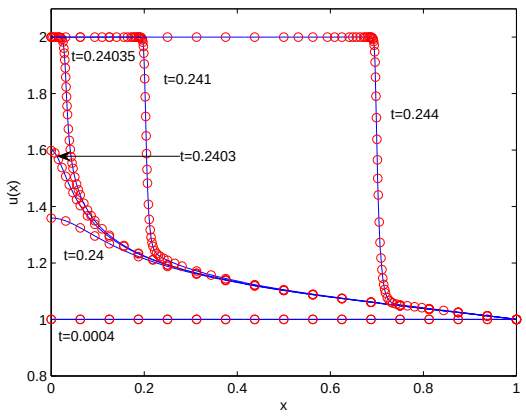


(e) Grid point distribution at $t = 1\,\mathrm{s}$.

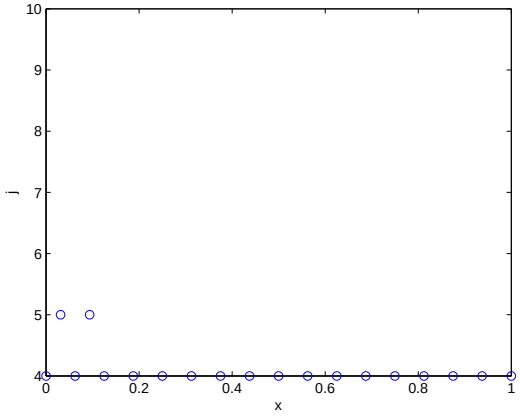


(f) Time evolution of the number of grid points.

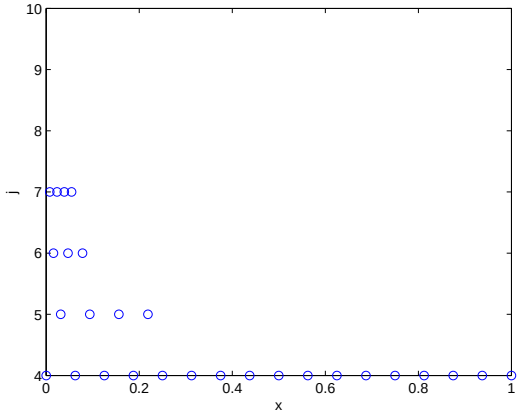
Figure 8: Example 7. Parameters used in the simulation are $J_{\min} = 4$, $J_{\max} = 12$, $p = 3, \epsilon = 0.0005$, $N_1 = 2$, and $N_2 = 1$.



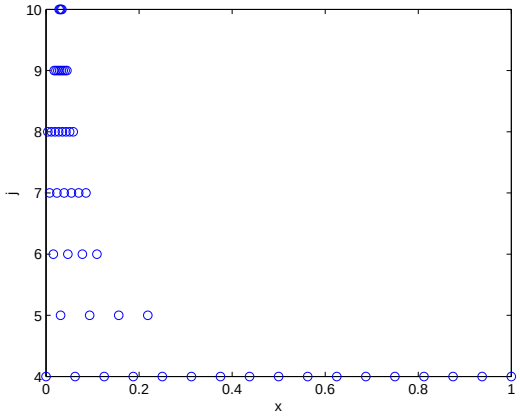
(a) Solution $u(x,t)$.



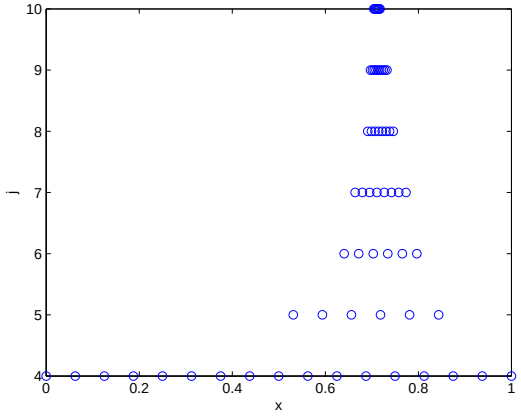
(b) Grid point distribution at $t = 0.24\,\mathrm{s}$.



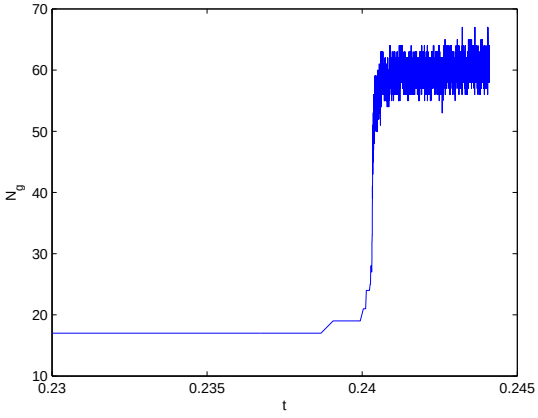
(c) Grid point distribution at $t = 0.2403\,\mathrm{s}$.



(d) Grid point distribution at $t = 0.24035\,\mathrm{s}$.



(e) Grid point distribution at $t = 0.244\,\mathrm{s}$.



(f) Time evolution of the number of grid points.

Figure 9: Example 8. Parameters used in the simulation are $J_{\min} = 4$, $J_{\max} = 10$, $p = 3, \epsilon = 0.001$, $N_1 = 2$, and $N_2 = 1$.