JOURNAL OF COMPUTATIONAL PHYSICS 94, 201–224 (1991)

A Two-Dimensional Adaptive Mesh Generation Method

IRFAN ALTAS AND JOHN W. STEPHENSON

Department of Mathematics, University of Saskatchewan, Saskatoon, Saskatchewan, Canada S7N OWO

Received March 6, 1989; revised September 12, 1989

An automatic two-dimensional adaptive mesh generation method is persented. The method is designed so that a small portion of the mesh can be modified without disturbing a large number of adjacent mesh points. The method can be used with or without boundary-fitted coordinate generation procedures. On the generated mesh a differential equation can be discretized by using classical difference formulas designed for uniform meshes as well as the difference formulas developed in this work. Both cases are illustrated by applying the method to the Hiemenz flow for which the exact solution of the Navier–Stokes equation is known [1] and to a two-dimensional viscous internal flow model problem. © 1991 Academic Press, Inc.

1. INTRODUCTION

Adaptive mesh generation is a promising technique in the numerical solution of differential equations. An adaptive mesh generation procedure adjusts the location of mesh points, or adds and subtracts mesh points using feedback from a previous numerical solution of a problem. In the past two decades, as stated in the review paper [2], researchers have developed many sophisticated adaptive mesh methods for the solution of ordinary differential equations. Significant interest has appeared during the last decade in generating and applying adaptive mesh to the numerical solution of partial differential equations [2–5]. Significant progress has been made in finite element adaptive mesh methods [6–8]. Progress has been slower in finite difference application because of the difficulties associated with discretization on nonuniform mesh. In this paper, we present an adaptive mesh generation procedure which adds additional mesh points locally. The resulting mesh in nonuniform and we derive discretization formulas for this nonuniform mesh.

In order to find suitable positions for mesh points, most adaptive methods construct a positive weight function by using some general features of the solution which may be obtained after a small amount of calculation. The solution domain is then divided into subregions such that the positive weight function has roughly equal value over each subregion. Adaptive algorithms mostly differ from each other in the choice of the weight function as stated in review paper [2].

In this paper, we consider a weight function which measures how well the solution function can be represented by a certain degree of polynomial over a subregion. We do this by applying a numerical quadrature rule which uses an approximate solution obtained in the subregion. This approach does not need a very accurate solution in contrast to some adaptive algortihms in the literature which form the weight function by using the derivatives of the solution, for example, [9, 10]. It also provides us with an elegant stopping criteria for the adaptive mesh generation algorithm.

The main obstacle in the extension of one-dimensional adaptive methods to higher dimensional cases is that difference formulas on irregular meshes may not exist on an arbitrary set of mesh points. Most of the existing adaptive methods overcome this problem in the following way. They transform the irregular physical domain to a square computational domain. Then they use a uniform mesh to generate the difference scheme for the transformed equation. However, a major deficiency of some of the adaptive algorithms using this approach, as stated in [11], is a lack of control of mesh skewness in the physical plane. This may cause larger truncation errors of difference expressions and curvilinear coordinates might overlap in the physical plane. Another disadvantage of this approach, as stated in [10], is that it is not possible to modify a small portion of the mesh during the calculation without disturbing a large number of adjacent mesh points.

In [10], P. Luchini has suggested an adaptive method in the physical domain to modify a small portion of the mesh during the calculation. This method maintains each point at the center of a symmetrical cross formed with four other mesh points, with the exception of points lying in the neighborhood of the boundary. This method applies only to those elliptic partial differential equations in which the second-order derivatives appear in the form of the Laplacian operator. If there are boundaries passing between mesh points, Luchini suggested using interpolation or, to transform the nonrectangular region to a rectangular region with a conformal mapping, to avoid introducing the mixed derivatives. In the first case, using interpolation causes poor representation of boundary values and in most cases the boundary values are dominant in the solution of the differential equation. In the second case, the application of conformal mapping is not always possible for arbitrary boundaries.

Our adaptive procedure has no difficulty handling mixed derivatives because of our six-point discretization formula described in Section 2. Consequently, we do not restrict our adaptive algorithm to certain types of boundary fitted coordinate generation procedures. A mesh similar to that obtained by Luchini [10] can be obtained using our adaptive mesh generation procedure when restrictions are applied to the mesh. These restrictions will be discussed in Section 3.

We introduce the adaptive mesh generation procedure in Section 2, and discuss the existence of difference formulas in the Appendix. In Section 3, we give additional restrictions to the mesh generation procedure which guarantees that simple classical finite difference formulas can be used. In Section 4, we demonstrate some applications of the method.

2. Adaptive Method

The adaptive mesh generation procedure which we have developed, is based upon using quadrature errors to estimate E, the variation of the solution function. In our applications, we choose the trapezoidal rule to decide whether the solution function can be represented by a low degree polynomial over a subregion. However, the trapezoidal rule may be replaced by other quadrature rules or may be combined with other formulas if we wish to represent the solution by a higher order polynomial.

We will explain the procedure for a square domain, because for irregular domains, we propose to use a numerical boundary fitted coordinate generation procedure which first transforms the given domain into a square domain. Then, the adaptive procedure given here can be employed.

The adaptive mesh generation procedure can be briefly described as follows. Let us assume that a solution function $u(x, y)$ of the differential equation is available in the solution domain. The construction of such a solution will be explained later. We start by dividing the original square into subsquares using a uniform mesh. We can commence with four subsquares. Then we calculate E for every subsquare in the solution domain. A subsquare with a value of E which is larger than some tolerance ε, is subdivided again into four subsquares. The idea of subdividing a rectangular cell into subrectangles based on the value of an error measure is also used in finite element procedures [7]. See also review papers [6, 12]. However, the generated adaptive mesh in this work will be combined with a finite difference

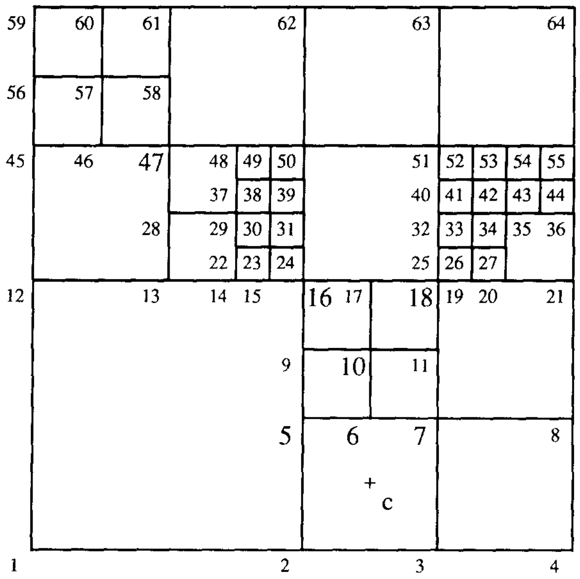

FIG. 1. Different mesh types.

procedure. This means that some special discretizations on nonuniform mesh will have to be presented in the sequel.

In order to describe the quantity E mentioned above, consider a subsquare S, defined by the vertices (x_i, y_j), (x_{i+1}, y_j), (x_i, y_{j+1}), and (x_{i+1}, y_{j+1}). For example, in Fig. 1 the subsquare defined by the vertices (5, 6, 10, 9). The trapezoidal quadrature rule, T_1, applied to $u(x, y)$ on S leads to the following error term

$$\iint_S u(x, y)\, ds - T_1$$

$$= E_{T_1} = -\frac{1}{12}(x_{i+1} - x_i)^2 (y_{i+1} - y_i)^2 [u_{xx}(\xi_1, \eta_1) + u_{yy}(\xi_2, \eta_2)], \quad (2.1)$$

where $T_1 = (x_{i+1} - x_i)(y_{j+1} - y_j)[u(x_i, y_j) + u(x_{i+1}, y_j) + u(x_i, y_{j+1}) + u(x_{i+1}, y_{j+1})]/4$ and (ξ_1, η_1), $(\xi_2, \eta_2) \in S$. The left-hand side of Eq. (2.1) measures how well the solution function $u(x, y)$ can be approximated by a linear function over S. We do not know the exact value of $\iint_S u\, ds$, however, we can obtain a better approximation of $\iint_S u\, ds$.

If the square S was divided into four equal subsquares, and we applied the trapezoidal rule to each subsquare and added the results we would obtain

$$\iint_S u\, ds - T_2 = E_{T_2} \quad (2.2)$$

where

$$T_2 = (x_{i+1} - x_i)(y_{j+1} - y_j)[u(x_i, y_j) + u(x_{i+1}, y_j) + u(x_i, y_{j+1})$$
$$+ u(x_{i+1}, y_{j+1}) + 2(u(x_{i+1/2}, y_j) + u(x_{i+1}, y_{j+1/2})$$
$$+ u(x_{i+1/2}, y_{j+1}) + u(x_i, y_{j+1/2})) + 4u(x_{i+1/2}, y_{j+1/2})]/16$$

and $(x_{i+1/2}, y_{j+1/2})$ is the center of S.

By subtracting Eq. (2.1) from Eq. (2.2) we obtain

$$E = |T_1 - T_2| = (x_{i+1} - x_i)(y_{j+1} - y_j)\, |[3(u(x_i, y_j) + u(x_{i+1}, y_j)$$
$$+ u(x_i, y_{j+1}) + u(x_{i+1}, y_{j+1})) - 2(u(x_{i+1/2}, y_j)$$
$$+ u(x_{i+1}, y_{j+1/2}) + u(x_{i+1/2}, y_{j+1})$$
$$+ u(x_i, y_{j+1/2})) - 4u(x_{i+1/2}, y_{j+1/2})]|/16. \quad (2.3)$$

The quantity E in Eq. (2.3) measures the variation of the solution function and is in a form that can be used in a numerical algorithm. By using Taylor expansions of the functions about the center point $(x_{i+1/2}, y_{j+1/2})$ of S one can show that theoretically the quantity given by the right-hand side of Eq. (2.3) is equal to $(x_{i+1} - x_i)(y_{j+1} - y_j)\, |2(x_{i+1} - x_i)^2 u_{xx}(x_{i+1/2}, y_{j+1/2}) + 2(y_{j+1} - y_j)^2 u_{yy}(x_{i+1/2}, y_{j+1/2}) + R|/16$, where R is the remainder term in Taylor expansions.

In order to evaluate E given in Eq. (2.3) we need to obtain values of the solution function u at the center and midpoints of the edges of S. However, u is only known at the vertices of S. We note that S is a subsquare of a larger square LS, defined by the vertices (5, 7, 18, 16) in Fig. 1. We construct an interpolation polynomial

$$P(x, y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 + a_6 x^2 y + a_7 xy^2 \tag{2.4}$$

with

$$\begin{aligned}
a_0 &= u_{10} \\
a_1 &= (4u_{11} - 4u_{10} - u_5 - u_{18} - u_7 - u_{16} + 2u_{17} + 2u_6)/4h \\
a_2 &= (u_{17} - u_6)/2h \\
a_3 &= (u_5 + u_{18} + u_{16} - 2u_{17} - 2u_6 + u_7)/4h^2 \\
a_4 &= (u_{18} + u_5 - u_{16} - u_7)/4h^2 \\
a_5 &= (u_{17} + u_6 - 2u_{10})/2h^2 \\
a_6 &= (-u_5 - u_7 + u_{18} + u_{16} + 2u_6 - 2u_{17})/4h^3 \\
a_7 &= (u_7 + u_{18} - u_{17} - u_6 + 2u_{10} - 2u_{11})/2h^3,
\end{aligned} \tag{2.5}$$

where u_i is the value of the function u at the mesh point i and $h = x_1 - x_0$. Note that we have only used eight of the nine available values of the solution by excluding the mesh point 9. For another combination of eight mesh points we obtain a different interpolation polynomial. In this sense, the interpolation polynomial (2.4) is not unique.

The interpolation polynomial $P(x, y)$ can be used for any required value of the solution function u in LS. This procedure can always be applied in our adaptive mesh, since every subsquare can be embedded in a larger square. These interpolations on subregions are expensive and require some reasonable amount of bookkeeping. Since the values of u in Eq. (2.3) are needed for the purpose of mesh generation only, therefore, some reasonable approximation will do the same job. Let us assume that the subsquare under investigation is (5, 7, 18, 16) in Fig. 1 and the values of u are required at the mesh points 6, 11, 17, 9, and 10. We estimate the value of u_{11} by $u_{11} = wu_{18} + (1-w)u_7$, where $0 < w < 1$. The value of w can be chosen depending on the slope of u at the mesh point 11. In our applications, after several experiments, we selected $w = \frac{4}{7}$ if $u_y = (u_{18} - u_7)/(y_{18} - y_7)$ is positive, otherwise $w = \frac{3}{7}$. The values of u_{17}, u_9, and u_6 are obtained similarly. For the midpoint 10 we simply average the values of u at the corner points 5, 7, 18, 16. Experimentation has shown us that this procedure gives an adaptive mesh almost identical to the mesh produced using the interpolation formula (2.4).

The following algorithm uses Eq. (2.3) to generate an adaptive mesh.

ALGORITHM 2.1. Given a discrete solution function $u(x, y)$ on a uniform mesh in $R = [0, 1] \times [0, 1]$, construct a set of adaptive mesh points $\{(x_i, y_j), i = 0, ..., N$

and $j = 0, \ldots, N\}$ which partitions R into square subregions on which E, evaluated by Eq. (2.3), is less than some tolerance ε.

1. Start by using the subregions generated by a uniform mesh.
2. Evaluate E using Eq. (2.3) on each subregion.
3. Subdivide the regions with the quantity E larger than a given tolerance ε into four equal subregions.
4. On the new mesh points, either obtain a new approximate solution to the problem or use interpolated values of the previously obtained solution.
5. Continue steps 2 to 4 until the largest value of E is less than ε.
6. Solve the problem on the final mesh.

In practical applications of Algorithm 2.1 we have employed both of the suggested ways in the fourth step. We obtained a more evenly graduated mesh by solving the problem after completion of step 3 than by using the interpolated values of the solution on the new mesh points. We have also observed that in these intermediate steps, it is not necessary to obtain an accurate solution. An approximate solution obtained from a few iterations of SOR will do. Experiment has shown us that the mesh obtained using this crude approximation is almost the same as that obtained using a fully converged solution. Hence, for these intermediate steps we use relatively large error tolerances in the iterative solution of the system. The value of ε can be specified after the first completion of the step 2 by using some combination of the largest and smallest values of the quantity E. Hence, this tolerance is problem dependent and can be automatically determined.

The mesh generated by Algorithm 2.1 is nonuniform. We cannot directly apply the classical difference schemes like central differences. Hence, in order to generate difference formulas about every interior mesh point, which we call a central mesh point, we need to select a certain number of neighbouring points. We call this set of mesh points a computational cell. In the selection of computational cells we apply certain criteria which leads us to develop some useful difference formulas. First, a difference formula should exist on the chosen mesh points. Second, the mesh points in a computational cell should be as close as possible to the central mesh point in order to reduce the magnitude of the truncation error of the difference formula. We define a mesh ratio on a computational cell as the maximum ratio of the distances of two mesh points from the center point of the cell. In other words, mesh points should be chosen so as to keep the mesh ratio small. Finally, the mesh points of a computational cell should be chosen from as many different directions as possible.

In order to form a difference approximation for each interior mesh point, we must first choose an appropriate computational cell. The choice of mesh points in the computational cell depends upon the classification of the interior mesh point. An interior mesh point can belong to either three squares or four squares. A mesh point belonging to three squares, is the vertex of two squares and lies on one of the edges of the third square. We can separate those types of mesh points in two different categories. The first case is when the two squares which have the mesh point as a vertex are equal. The second type is when the two squares are not equal. The mesh points which are the vertices of four squares can be separated into five categories. The first case is when the four squares are equal. Second, three of the squares are equal. The third case is when two of the squares are equal to each other and the other two squares are also equal to each other. The fourth case is when two of the squares are equal to each other and the other two are unequal. The fifth case is when the four squares are unequal. Hence, we can identify all interior mesh points in seven different categories. Those seven different types of mesh points are given in Fig. 1 as the mesh points 5, 6, 7, 10, 16, 18, and 47. We only have to generate difference formulas for these seven different types of mesh points.

In order to classify the mesh points according to the above discussion, we first number every subsquare and every mesh point in the initial mesh. We store the numbers of the subsquares belonging to a mesh point in a vector. Another vector holds the mesh numbers of the vertices of a subsquare. In a third vector we store the mesh type according to the above classification. When a subsquare is refined, the information in the vectors is updated.

In practise, without any restriction in mesh generation, some of the desired properties of a computational cell might not be met. For example, we can experience some difficulties in the convergence of iterative methods used to obtain a solution because of high mesh ratios. In order to avoid such problems, we require a control mechanism in step 3 of Algorithm 2.1. A successful control was developed using a measure called edge ratios, the maximum ratios of the edges of neighbouring squares which contain a given mesh point. The control imposed by the edge ratio aborts a subdivision determined by the quantity E if the edge ratio in the sugdivision will exceed a given number. All the desired properties of a computational cell were obtained in a relatively simple way by this device.

In our computations, we restricted the edge ratio to be 2. However, we have not observed a significant difference in the adaptive mesh generated with the edge ratio 2 and the edge ratio 4. Furthermore, the mesh types described above are reduced from 7 to 4 with edge ratio 2, whereas it is only reduced to 6 with the edge ratio 4. The mesh types denoted by the numbers 5, 16, and 18 in Fig. 1, are eliminated if the edge ratio is 2. The detailed explanations and comparisions of those two cases are given in [13]. Consequently, the data management is simplified. Another advantage with the edge ratio 2 is that it is possible to obtain the classical five-point symmetrical computational cell which will be explained later in Section 3.

For the remaining four mesh types, we choose the computational cells as follows. For the central mesh point 6 in Fig. 1, we choose the mesh points $\{6, 7, 11, 10, 5, 2\}$ as the computational cell. Note that the mesh point 11 can be replaced by a nearer mesh point to the center point in this computational cell whenever there is a further subdivision of the subsquare defined by mesh points (11, 18, 17, 10). For the mesh type denoted by 10 in Fig. 1, the selected computational cell is $\{10, 11, 18, 17, 9, 6\}$. The computational cell for the mesh point 7 is $\{7, 8, 11, 10, 6, 3\}$. This mesh type can also be in the form given by the mesh number 34 in Fig. 1. In this case the chosen computational cell is $\{34, 35, 43, 42, 33, 27\}$. The last mesh type is denoted by either the mesh number 29 or 47 in Fig. 1. The computational cells for those mesh points are $\{29, 30, 38, 37, 28, 22\}$ and $\{47, 48, 58, 46, 28, 37\}$, respectively. We also have rotations of the given mesh types above by 90°, 180°, and 270°. Rotations do not affect the existence of difference formulas. So, we do not discuss those cases here. We give the generated difference formulas on those computational cells in the Appendix.

3. Symmetric Computational Cells

When the differential equation does not contain a mixed derivative, we derive another advantage by restricting the edge ratio to 2. In this case, we do not have to generate special difference formulas. Classical finite difference formulas, for example central difference, can be used with these symmetric five point computational cells. We illustrate this by applying our adaptive mesh generation technique to some elliptic problems in Section 4. For the mesh points 6, 10, 7, 34, 29, and 47 discussed in Section 3, we can easily choose symmetric computational cells except for the mesh point 6 as we can see from Fig. 1. The symmetric computational cells for the mesh points 10, 7, 34, 29, and 47 are $\{10, 11, 17, 9, 6\}$, $\{7, 8, 18, 5, 3\}$, $\{34, 35, 42, 33, 27\}$, $\{29, 31, 48, 28, 14\}$, and $\{47, 48, 58, 46, 28\}$, respectively.

From Fig. 1, we can see that we must add an extra mesh point to the adaptive mesh in order to define a symmetric computational cell for the mesh point 6. In Fig. 1 we indicate this mesh point by + and call it c. It is the intersection point of the diagonals of the square with the vertices (2, 3, 7, 5). In this case, the symmetric computational cells for the mesh point 6 is $\{6, 7, 10, 5, c\}$. This additional mesh point creates one more mesh type. If we consider the point c as the origin and the length of one of the edges of the square (2, 3, 7, 5) as $2h$, then the derivatives at c can be approximated in three ways:

(i) If the second derivatives in the differential equation occur only in the form of the Laplace operator, then we can approximate derivatives with the following difference schemes.

$$\begin{aligned} u_{xx}+u_{yy} &= (u_7+u_5+u_2+u_3-4u_c)/(2h^2)+O(h^2) \\ u_y &= (u_7+u_5-u_2-u_3)/(4h)+O(h^2) \\ u_x &= (u_7-u_5-u_2+u_3)/(4h)+O(h^2). \end{aligned} \tag{3.1}$$

(ii) We can approximate the value of the solution function at the mesh point c by interpolation. We use the mesh points $\{7, 10, 16, 5, 2, 6\}$ to obtain a second-degree interpolation polynomial. This approach removes the restriction in case (i) that the second derivatives appear only aggregated in the Laplacian form. However, we do not recommend this interpolation, although it is one of most common techniques in the literature, since it lowers the accuracy of the overall difference scheme, as we will demonstrate in Setion 4.

(iii) Suppose that the differential equation under consideration is invariant to rotation. This is the case in many of the fluid dynamics problems if there is no external force acting. Then the only difference between this mesh type and the mesh type of the mesh point 10 is in the mesh width. The mesh width of this mesh type would be $\sqrt{2}h$ if the mesh width of the point 10 is h. However, if the differential equation is variant to rotation, then it can be transformed for this mesh type by rotating the coordinate variables by 45°.

4. Applications

In this section, our purpose is to demonstrate the application of the adaptive mesh generation procedure which is proposed here and to show that the adaptive procedure could be used with the classical difference scheme as well as the difference formulas proposed in this work. We were not trying to obtain extremely accurate solutions to the chosen problems.

We applied the adaptive mesh generation technique to three different problems. Problem 4.1 is a linear problem taken from [14]. Problems 4.2 and 4.3 represent the extension of the method to nonlinear problems. For the problems given here we obtained crude solutions either on a 16×16 or a 8×8 uniform mesh and used these in the adaptive mesh generation. We obtained the uniform mesh solutions by using central difference approximations of the derivatives. We used successive over relaxation to solve the systems which result from discretization of the differential equations. We used 10^{-6} to be the convergence criteria for the iterations. We performed all the computations in double precision on a VAX 8650.

Problem 4.1.

$$u_{xx} + u_{yy} - 100u = 0.5(p^2 - 100)\cosh(py)/\cosh p \tag{4.1}$$

with the exact solution

$$u(x, y) = 0.5(\cosh(10x)/\cosh 10 + \cosh(py)/\cosh p) \tag{4.2}$$

and boundary conditions obtained from the exact solution.

This problem is taken from [14]. Contours of the exact solution are given in Fig. 4. For $p \neq 10$, boundary layers occur near both $x = 1$ and $y = 1$. The boundary layer along $x = 1$ is thicker than the boundary layer along $y = 1$. The variable p adjusts the strength of the boundary layer along $y = 1$. When the value of p increases, the boundary layer becomes thinner. We solved this problem for $p = 80$ which is the largest possible value without having overflow in our machine. The generated adaptive mesh and contours of the numerical solution are given in Fig. 5

TABLE I

The Maximum Absolute Errors Obtained from Problem 4.1

Mesh No.	160	423	818
Max. Abs. Error	0.232	0.128	0.037

and Fig. 6, respectively. The numerical results are reported in Table I. The results are obtained using our six-point formula.

The efficiency of the adaptive mesh procedure can be appreciated when we compare the absolute maximum error obtained with a uniform mesh. The error is 0.039 for this problem when we use a uniform mesh with 4225 mesh points. This result corresponds to the adaptive solution obtained with 818 mesh points which is $\frac{1}{5}$ of the uniform mesh number.

Problem 4.2. As a second example, we applied the adaptive mesh generation technique to the Hiemenz flow for which the exact solution of the Navier–Stokes equation is known [1]. The fundamental equations for two-dimensional incompressible flow of a Newtonian fluid with no body forces are two momentum equations

$$\partial \underline{u}/\partial \underline{t} + \underline{u}\, \partial \underline{u}/\partial \underline{x} + \underline{v}\, \partial \underline{u}/\partial \underline{y} = -1/\underline{\pi}\, \partial \underline{p}/\partial \underline{x} + \underline{\mu}(\partial^2 \underline{u}/\partial \underline{x}^2 + \partial^2 \underline{u}/\partial \underline{y}^2) \tag{4.3}$$

and

$$\partial \underline{v}/\partial \underline{t} + \underline{u}\, \partial \underline{v}/\partial \underline{x} + \underline{v}\, \partial \underline{v}/\partial \underline{y} = -1/\underline{\pi}\, \partial \underline{p}/\partial \underline{y} + \underline{\mu}(\partial^2 \underline{v}/\partial \underline{x}^2 + \partial^2 \underline{v}/\partial \underline{y}^2) \tag{4.4}$$

and the continuity equation,

$$\partial \underline{u}/\partial \underline{x} + \partial \underline{v}/\partial \underline{y} = 0. \tag{4.5}$$

The underbars indicate dimensional quantities and $\underline{u}$ and $\underline{v}$ represent velocity components, while $\underline{p}$, $\underline{\pi}$ and $\underline{\mu}$ are the pressure, mass density and kinematic viscosity respectively.

We can obtain a numerical solution from Eq. (4.3)–(4.5). However, we prefer to use the stream function–vorticity formulation given in [15]. Then, the relevant equations, in terms of scalars, are invariant to rotations. This property can be used for the symmetric computational cell designed for the mesh point c in Fig. 1 and explained in Section 3.

The stream function–vorticity formulation of the Navier–Stokes equations, in a generic cartesian coordinate system, can be written as

$$\Delta_2 \psi = w \tag{4.6}$$

$$\Delta_2 w = R(\partial\psi/\partial y\, \partial w/\partial x - \partial\psi/\partial x\, \partial w/\partial y), \tag{4.7}$$

where $R = u_0 L/\mu$ is the Reynolds number, ψ is the stream function, w is the vorticity, u_0 is the characteristic velocity, and L is the characteristic length.

We now obtain the exact solution of Hiemenz flow. For viscous flow, we make [illegible]

function. By this assumption, the equation of continuity, Eq. (4.5), is satisfied identically. The corresponding solution of the Navier–Stokes equation in stream function–vorticity form is

$$\psi = xF(y), \qquad w = x\, d^2F/dy^2. \tag{4.8}$$

By substituting Eq. (4.8) into both Eq. (4.6) and Eq. (4.7), we see that the first one is satisfied. The latter one, after few manipulations, takes the form

$$d^3F/dy^3 - R[-F(d^2F/dy^2) + (dF/dy)^2 - 1] = 0 \tag{4.9}$$

with the boundary conditions $F(0) = dF/dy|_0 = 0$, $dF/dy = 1$ as $y \to \infty$. Similar solutions can be obtained by eliminating R. We write $\eta = \sqrt{R}y$ and $F = \Phi(\eta)/\sqrt{R}$ and substitute these values into Eq. (4.9) to obtain

$$d^3\Phi/d\eta^3 + \Phi(d^2\Phi/d\eta^2) - (d\Phi/d\eta)^2 - 1 = 0 \tag{4.10}$$

with the boundary conditions $\Phi(0) = d\Phi/d\eta|_0 = 0$, $d\Phi/d\eta = 1$ as $\eta \to \infty$.

In order to discretize Eq. (4.6) and Eq. (4.7), we choose the difference scheme given in [10] and treat boundary conditions in a different way. The difference scheme applied to this problem is not an efficient difference scheme. We have tried to obtain the solution with 64×64 uniform mesh points for $R = 100$, but the iteration procedure did not converge in a reasonable time. In this difference scheme, the upwinded form of the equations is used, and each convective term is modified with a suitable correction factor in order to restore the approximation of central differences. The central difference approximation is used for Eq. (4.6). The difference approximation of Eq. (4.7) is

$$w_1 + w_2 + w_3 + w_4 - 4w_0 = R[c_1(\psi_2 - \psi_4)(w_0 - w_3) - c_2(\psi_1 - \psi_3)(w_2 - w_0)]/4, \tag{4.11}$$

where the subscripts show the mesh numbers given in Fig. 3. We assume $\psi_2 - \psi_4 > 0$, $\psi_1 - \psi_3 > 0$ and

$$c_1 = (w_1 - w_3)/(w_0 - w_3), \; c_2 = (w_2 - w_4)/(w_2 - w_0). \tag{4.12}$$

If $\psi_2 - \psi_4$ and/or $\psi_1 - \psi_3$ are negative, the upwinded differences must be taken in the opposite direction in both Eq. (4.11) and Eq. (4.12).

In the iteration process, w_0 is recalculated at each step as

$$\begin{aligned} w_0 = \{ & w_1 + w_2 + w_3 + w_4 - aw_0 + R[c_1(\psi_2 - \psi_4)w_3 \\ & + c_2(\psi_1 - \psi_3)w_2]/4\}/\{4 - a + R[c_1(\psi_2 - \psi_4) + c_2(\psi_1 - \psi_3)]/4\}, \end{aligned} \tag{4.13}$$

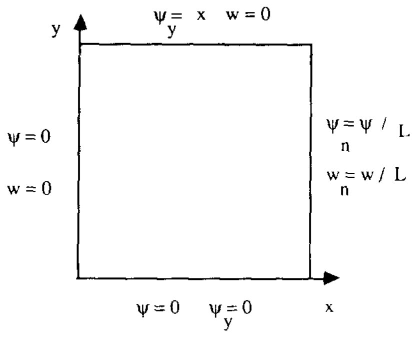

FIG. 2. Boundary conditions for Hiemenz flow.

while the old value of w_0 is used in c_1 and c_2 in Eq. (4.12). The parameter, a, can be used to overrelax or underrelax the iteration process. If the denominator of c_1 and/or c_2 is near to zero in machine precision, then we take c_1 and/or c_2 to be 1. The reason is that the ratio must be close to 1 when the difference approximations approach the exact value of the derivative.

In order to obtain the numerical solution, we use a square domain with a vertex at the stagnation point and two sides of the square lying along the coordinate axes. The boundary conditions of the problem, as in [10], are shown in Fig. 2.

Note that the y axis is an axis of symmetry for the solution. The boundary condition $\psi_y = 0$ on the lower side of the square can be expressed as a condition for w by using the Thom formula [15] which will be given below. On the upper side of the square, the flow field is given by the asymptotic ittotational form of the stream coming from infinity. This approximation is valid if the size of the square is large enough as stated in [10]. However, this condition is not very critical since its error is of exponentially small order. On the right side, the boundary conditions are given in an exact form by using the exact form of the solution (4.8), where L is the length of a side of the square.

We now explain the approximation of the derivatives on the boundaries for the right boundary of the square shown in Fig. 2. The approximation for the other boundaries is similar.

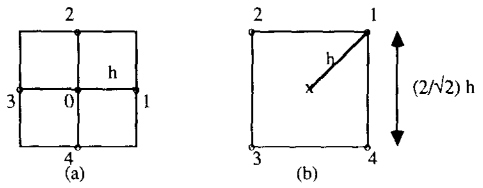

FIG. 3. Symmetric computational cells.

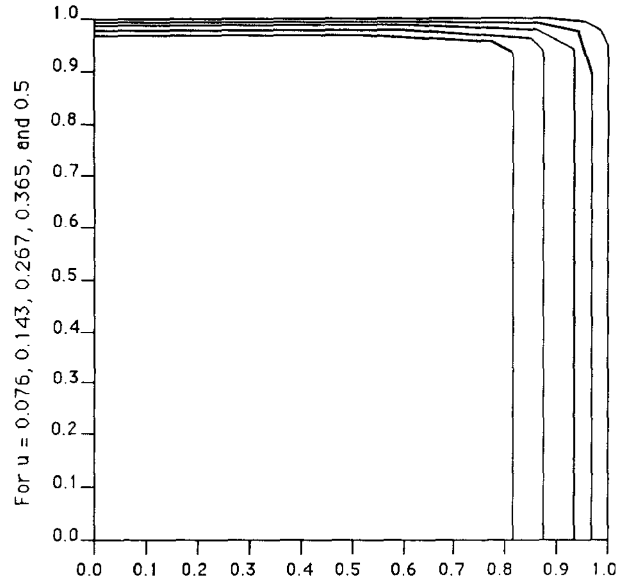

FIG. 4. Contours of the exact solution of Problem 4.1.

The derivatives can be approximated by a first-order backward difference formula. For the mesh types given in Figs. 3(a) and (b), they are respectively

$$\begin{aligned}\psi_x|_1 &= (\psi_1 - \psi_0)/h + O(h)\\ \psi_x|_1 &= (\psi_1 - \psi_2)/[(2/\sqrt{2})h] + O(h).\end{aligned}\tag{4.14}$$

If we wish to approximate the derivatives on the boundary with a second-order formula, then we use a three-point formula for the symmetric computational cells in Fig. 3(a) to obtain

$$\psi_x|_1 = (\psi_3 + 3\psi_1 - 4\psi_0)/h + O(h^2).\tag{4.15}$$

For the computational cell in Fig. 3(b), we start by once again examining its structure. If we look at Fig. 1, we must have a mesh point on the midpoint of either the side $\{1, 2\}$, $\{2, 3\}$, or $\{3, 4\}$ of the square given in Fig. 3(b). Let us call this mesh point 5:

(i) If the mesh point 5 is on the side $\{1, 2\}$ of the square, then by using System (5.1) given in the Appendix we can obtain the approximation of the derivatives at the point 4 in the form

$$\psi_x|_4 = -\sqrt{2}(4\psi_5 - 2\psi_2 + \psi_3 - 2\psi_1 - \psi_4)/(2h) + O(h^2)\tag{4.16}$$

and at the mesh point 1, we can use the three-point formula (4.15) and replace h by $(1/\sqrt{2})h$.

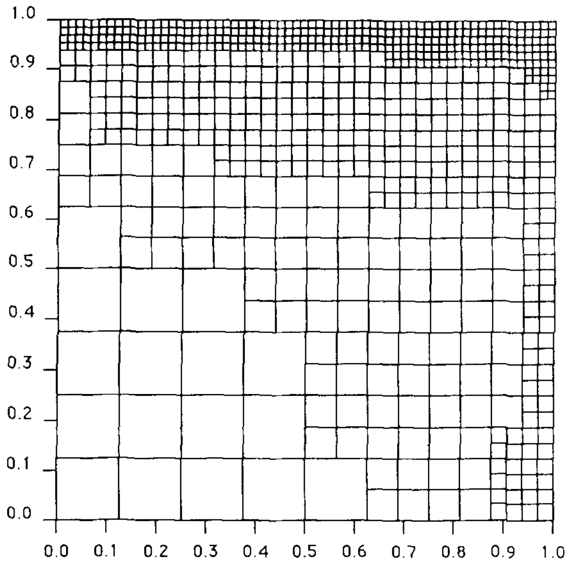

FIG. 5. Adaptive mesh for Problem 4.1.

(ii) Similarly, if the mesh point 5 is on the side $\{3, 4\}$, then at the point 1 we obtain

$$\psi_x|_1 = -\sqrt{2}(4\psi_5 + \psi_2 - 2\psi_3 - \psi_1 - 2\psi_4)/(2h) + O(h^2) \tag{4.17}$$

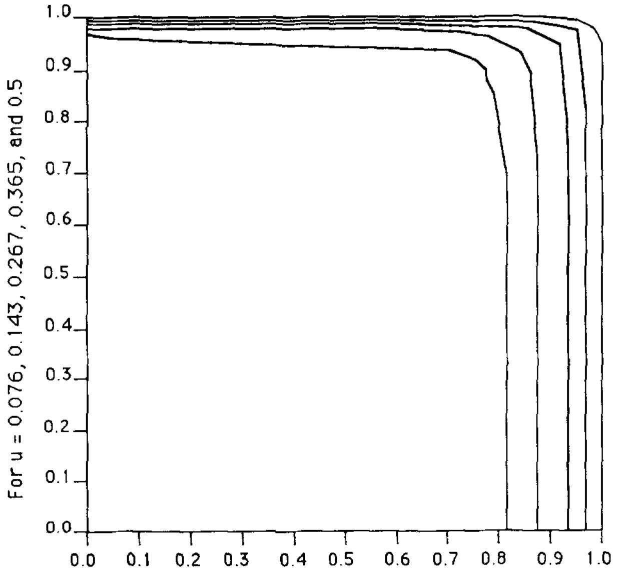

FIG. 6. Contours of the numerical solution of Problem 4.1.

and at the mesh point 4, we can use the three-point formula (4.15) again by replacing h with $(1/\sqrt{2})h$.

(iii) If the point 5 is on the side $\{2, 3\}$, then we have

$$\begin{aligned} \psi_x|_4 &= -\sqrt{2}(-4\psi_5 + \psi_2 + 2\psi_3 - \psi_1 + 4\psi_0 - 2\psi_4)/(2h) + O(h^2) \\ \psi_x|_1 &= -\sqrt{2}(-4\psi_5 + 2\psi_2 + \psi_3 + 4\psi_0 - \psi_4 - 2\psi_1)/(2h) + O(h^2). \end{aligned} \tag{4.18}$$

In order to transform the boundary condition $\psi_y = 0$ at the wall to a condition for the vorticity function w, we use either the two- or three-point, no-slip condition Thom formula [15]. For the mesh types 1–4 in Fig. 3(a), the formulas are standard ones. That is, the two- and three-point no-slip conditions are given, assuming the mesh point 4 is on the wall, by

$$w_4 = 2(\psi_0 - \psi_4)/h^2 + O(h)$$

and

$$w_4 = 3(\psi_0 - \psi_4)/h^2 - \tfrac{1}{2}w_0 + O(h^2), \tag{4.19}$$

respectively. For the computational cell in Fig. 3(b), the two- and three-point no-slip conditions become, respectively,

$$w_4 = 2(\psi_1 - \psi_4)/[(2/\sqrt{2})h]^2 = (\psi_1 - \psi_4)/h^2 + O(h)$$

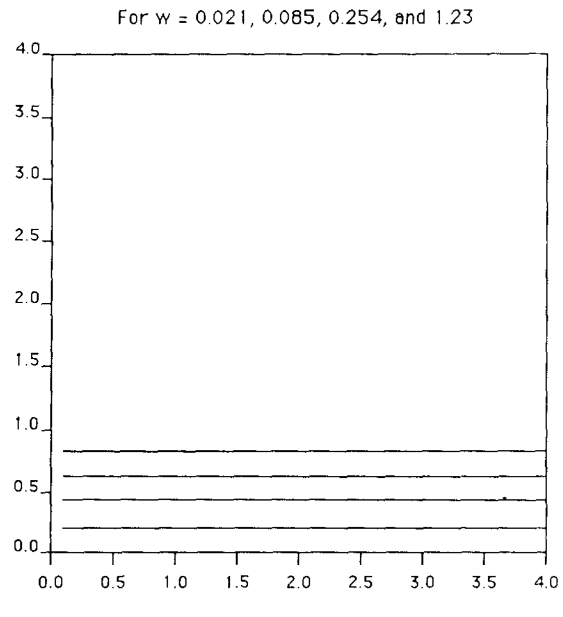

FIG. 7. Contours of the vorticity obtained from Eq. (4.10), $R = 9$.

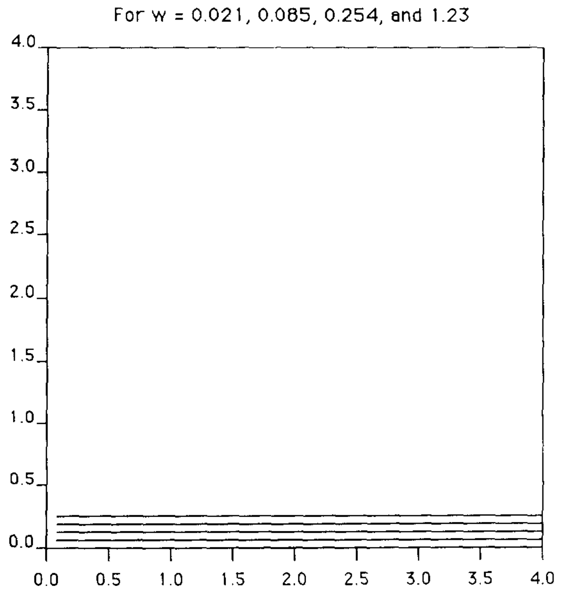

FIG. 8. Contours of the vorticity obtained from Eq. (4.10), $R = 100$.

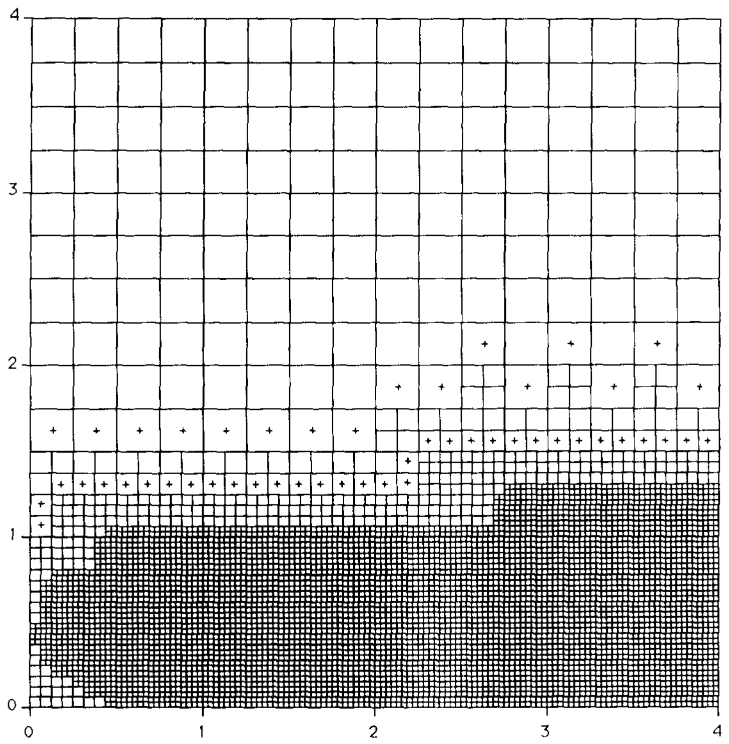

FIG. 9. Adaptive mesh for Hiemenz flow, $R = 9$.

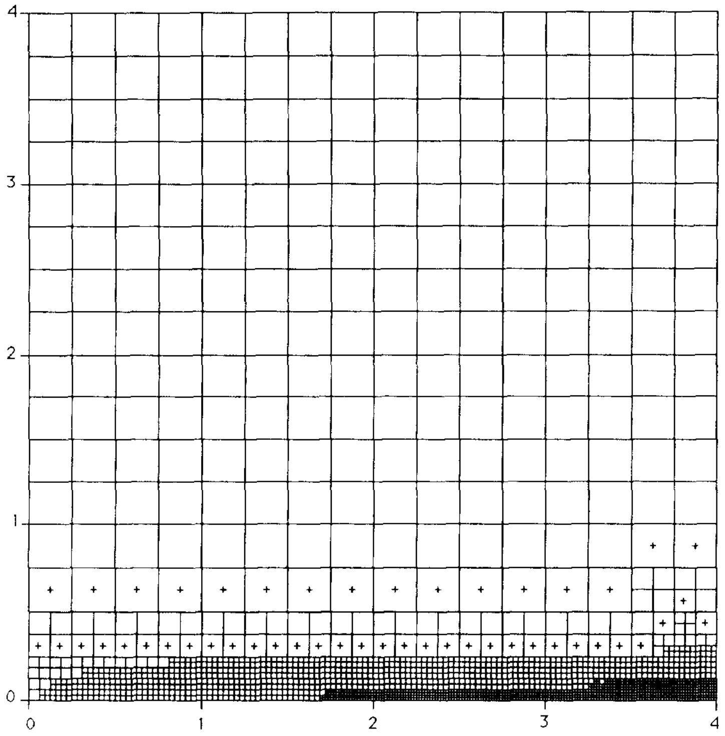

FIG. 10. Adaptive mesh for Hiemenz flow, $R = 100$.

and

$$w = 3(\psi - \psi)/[(2/\sqrt{2})h]^2 - \tfrac{1}{2}w + O(h^2) \qquad (4.20)$$

We generate the adaptive mesh according to the vorticity function w. In this problem a boundary layer appears near the wall for the vorticity function w. The value of w decreases to 0 as the distance from the wall increases. The thickness of the boundary layer depends on the Reynolds number R. The thickness of the boundary layer region shrinks with the ratio $1/\sqrt{R}$.

We solve this problem for two different values of the Reynolds numbers, namely, $R = 9$ and $R = 100$ in a square whose side length, L, is 4. For $R = 9$ and $R = 100$, the boundary layers occur approximately in the regions $0 < y < 1$ and $0 < y < 0.3$, respectively, as shown in Figs. 7 and 8 which are obtained from the solution of the ordinary differential equation (4.10). The adaptive mesh generated for $R = 9$ and $R = 100$ and contours of w obtained from the solution of Eq. (4.7) are given in Figs. 9, 10, 11, and 12, respectively.

In [10], it has been stated that the second-order approximation of the derivatives on the boundary were also tested. There was really no significant improvement over the results obtained by first-order approximations. We observe

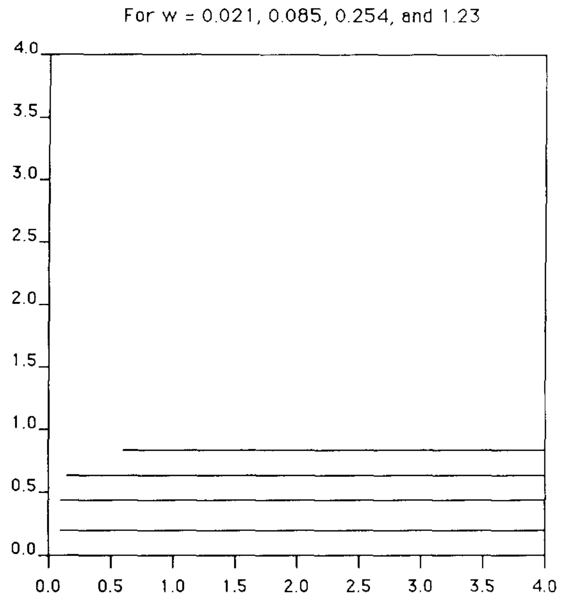

FIG. 11. Contours of the vorticity obtained from Eq. (4.7), $R = 9$.

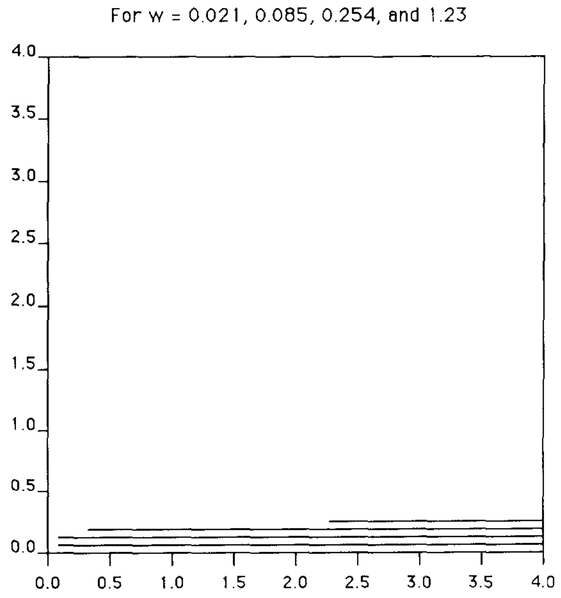

FIG. 12. Contours of the vorticity obtained from Eq. (4.7), $R = 100$.

TABLE II

The Numerical Results for Hiemenz Flow with $R = 9$. First-Order Approximation of Boundary Conditions

		$x=1, y=0.5$		$x=3, y=0.5$		$x=3.75, y=0.5$	
Mesh no.	Error type	ψ	w	ψ	w	ψ	w
146	Absolute	0.046	0.11	0.045	0.10	0.044	0.10
	Relative	0.13	0.26	0.044	0.085	0.035	0.068
716	Absolute	0.010	0.025	0.010	0.024	0.0098	0.029
	Relative	0.034	0.050	0.010	0.015	0.0086	0.012
1469	Absolute	0.0028	0.0066	0.0026	0.0061	0.0026	0.0061
	Relative	0.0094	0.012	0.0029	0.0039	0.0023	0.0031

in our numerical experiments that the second-order approximation of the derivatives on the boundary improves the results. We give the numerical results obtained from the first- and second-order approximations of derivatives on the boundaries in Tables II and III, respectively.

Interpolation is one of most common techniques used to handle interface mesh points. The broad discussion of this subject can be found in [16]. For this problem, we also decided to use interpolation for the mesh type denoted by c in Fig. 1. For this type of mesh, we employ a second-degree polynomial interpolation which was explained in Section 3 as (ii). On the other mesh types we still discretize the differential equation and use first-order approximations of the derivatives on the boundaries. Table IV shows the numerical results obtained from such a difference scheme. If we examine the table, we see how the results are spoiled by interpolation, especially for larger mesh numbers.

The numerical results obtained for $R = 100$ are given in Tables V and VI.

TABLE III

The Numerical Results for Hiemenz Flow with $R = 9$. Second-Order Approximation of Boundary Conditions

		$x=1, y=0.5$		$x=3, y=0.5$		$x=3.75, y=0.5$	
Mesh no.	Error type	ψ	w	ψ	w	ψ	w
146	Absolute	0.025	0.079	0.023	0.039	0.023	0.040
	Relative	0.042	0.086	0.024	0.026	0.019	0.021
716	Absolute	0.0045	0.0077	0.0039	0.006	0.0037	0.0053
	Relative	0.015	0.014	0.0044	0.0038	0.0033	0.0027
1469	Absolute	0.0013	0.0020	0.0011	0.0013	0.0011	0.0016
	Relative	0.0045	0.0039	0.0016	0.0010	0.0010	0.0008

TABLE IV

The Numerical Results for Hiemenz Flow with $R=9$. Second-Degree Interpolation Approximation of the Mesh Types denoted by c in Fig. 1

		$x=1, y=0.5$		$x=3, y=0.5$		$x=3.75, y=0.5$	
Mesh no.	Error type	ψ	w	ψ	w	ψ	w
146	Absolute	0.032	0.10	0.067	0.065	0.064	0.093
	Relative	0.12	0.24	0.099	0.046	0.074	0.056
716	Absolute	0.089	0.065	0.096	0.052	0.093	0.066
	Relative	0.43	0.14	0.16	0.036	0.12	0.038
1469	Absolute	0.10	0.056	0.10	0.059	0.11	0.054
	Relative	0.57	0.11	0.19	0.042	0.16	0.030

Problem 4.3.

$$u_{xx}+u_{yy}+y(1-y)(1-2x)\,Ruu_x+x(1-x)(1-2y)\,Ruu_y$$
$$-[x(1-x)+y(1-y)]\,R(u^2-1)=0. \tag{4.21}$$

Equation (4.21) is the representative of a two-dimensional viscous internal flow model problem and is taken from [17]. The solution domain is the unit square $0 \leqslant x \leqslant 1$, $0 \leqslant y \leqslant 1$. The boundary conditions used are

$$u(0, y)=u(1, y)=u(x, 0)=u(x, 1)=0. \tag{4.22}$$

The exact (steady state) solution of Eq. (4.21) together with Eq. (4.22) is

$$u(x, y)=\tanh(x(1-x)\,y(1-y)\,R/2). \tag{4.23}$$

TABLE V

The Numerical Results for Hiemenz Flow with $R=100$. First-Order Approximation of Boundary Conditions

		$x=1, y=0.125$		$x=3, y=0.125$		$x=3.75, y=0.125$	
Mesh no.	Error type	ψ	w	ψ	w	ψ	w
382	Absolute	0.037	1.34	0.030	1.34	0.036	1.30
	Relative	0.35	0.98	0.11	0.32	0.095	0.26
800	Absolute	0.010	0.25	0.010	0.25	0.010	0.22
	Relative	0.13	0.10	0.045	0.034	0.035	0.023
1673	Absolute	0.0054	0.077	0.0048	0.056	0.0012	0.085
	Relative	0.077	0.029	0.022	0.0070	0.0052	0.0081

TABLE VI

The Numerical Results for Hiemenz Flow with $R = 100$. Second-Order Approximation of Boundary Conditions

		$x=1,\ y=0.125$		$x=3,\ y=0.125$		$x=3.75,\ y=0.125$	
Mesh no.	Error type	ψ	w	ψ	w	ψ	w
382	Absolute	0.025	0.60	0.023	0.51	0.018	0.19
	Relative	0.27	0.28	0.087	0.077	0.057	0.020
800	Absolute	0.0064	0.044	0.0063	0.030	0.0059	0.011
	Relative	0.087	0.016	0.028	0.0037	0.021	0.0011
1673	Absolute	0.0043	0.016	0.0032	0.0052	0.0013	0.0094
	Relative	0.061	0.0060	0.015	0.0007	0.0054	0.0009

In this problem, very large gradients are developed on all four boundaries as $R \to \infty$ and the resulting flow characteristic is a very thin boundary layer at all the surfaces. The generated mesh for $R = 300$ is given in Fig. 13. Linearization is done by using old values at each iteration step. The numerical results, which are obtained by using both symmetrical and six-point computational cells, are reported in Table VII. The uniform mesh needs 4225 mesh points in order to obtain 0.051 accuracy in terms of absolute maximum error for this problem.

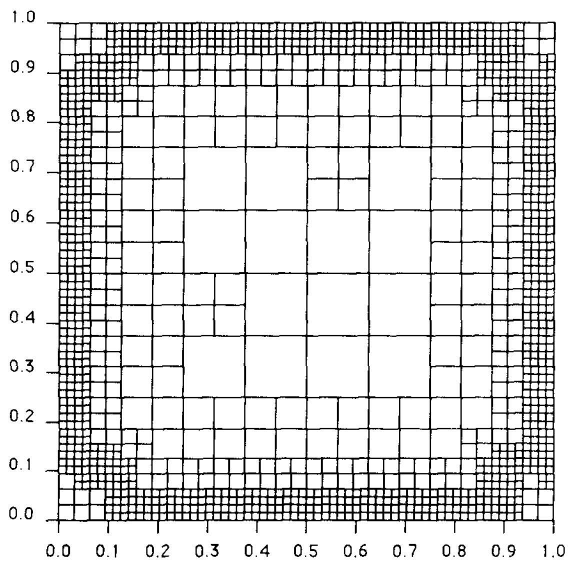

FIG. 13. Adaptive mesh for internal viscous flow, $R = 300$.

TABLE VII

The Maximum Absolute and Mean Square Root Errors for Problem 4.3, $R = 300$

Mesh no.	Error type	6-point formula	Central difference
437	Mean	0.13	0.10
	Max. abs.	0.34	0.27
1397	Mean	0.026	0.022
	Max. abs.	0.061	0.052

5. Conclusions

We have developed an adaptive mesh generation method using quadrature rules. We modify mesh locally by adding new mesh points. In our approach, the addition of new mesh points into the mesh is straightforward. For the deletion of mesh points, we reverse the algorithm. In order to delete a mesh point, we check the quantity E, given by Eq. (2.3), over four subsquares which are obtained from the subdivision of the same subsquare. If the quantity E is less than a specified tolerance for all four subsquares (and the edge ratio restriction is satisfied), then the mesh points of that subblock are deteled. However, this approach needs extra bookkeeping and therefore may not be an efficient way. The work in this direction is still under investigation.

The adaptive method given here can be used with or without boundary-fitted coordinate generation procedures. It does not require any a priori knowledge of the locations of large variations in the solution function. It automatically generates the complete mesh. Computational cells in our adaptive mesh can be chosen to apply our six-point finite difference formulas as well as to appy the classical finite difference formulas.

The adaptive method generates a well-suited mesh for problems whose solutions have large variations like large gradients, boundary layers, and sharp peaks. The method also handles problems having mild singularities in their solutions. Unfortunately, our adaptive method does not work well for highly oscillatory problems.

In order to get the same accuracy from the adaptive and the uniform mesh, the mesh number for the adaptive mesh is usually $\frac{1}{5}$ of the uniform mesh number for the examples given in Section 4. However, when we require more accurate solutions, this ratio would be smaller. The reason is that the adaptive mesh generation procedure puts more points in the regions where they are needed.

To answer the question whether a better mesh can be obtained by using a more accurate solution, we used the exact solution to generate the adaptive meshes for Problem 4.1. We have observed that this did not make a significant difference in the generation of the adaptive meshes. However, a cheaper and more efficient way to generate adaptive mesh would be to combine the method with a multigrid algorithm. Work in this direction is progressing. We also plan to generalize the adaptive mesh and formula generation to higher dimensional problems in our future work.

Appendix

using Taylor series expansion about the central mesh point. Let $u = u(x, y) \in C^3$ and $Z = \{(x_i, y_i) \mid i = 0, 1, ..., 5 \text{ and } (x_0, y_0) = (0, 0)\}$ represent a computational cell. The value of u at (x_i, y_i) is defined by $u_i = u(x_i, y_i)$. From the Taylor series expansion of u_i's about $(0, 0)$, the central mesh point, we have

$$u_i - u_0 = x_i u_x + y_i u_y + (x_i^2/2) u_{xx} + x_i y_i u_{xy} + (y_i^2/2) u_{yy}, \qquad i = 1, ..., 5. \quad (5.1)$$

Here, we have neglected the remainder terms in Eq. (3.1). We may interpret Eq. (5.1) as a set of linear equations in five unknowns u_x, u_y, u_{xx}, u_{xy}, u_{yy}. Now, if the above system (5.1) has a unique solution for the computational cell, then we have difference approximations for the derivatives. This is guaranteed by the following theorem and the proof of this theorem is given in [18].

THEOREM 5.1. *Let $z_i = (x_i, y_i)$, $i = 0, ..., 4$, be distinct points in the x–y plane with at most three of them on the same line in the x–y plane. Let w_j be the vectors $w_j = \{x_j, y_j, (x_j^2/2), x_j y_j, (y_j^2/2)\}$, $j = 1, ..., 4$. Assume that d_j, $j = 1, ..., 5$, are the determinant values obtained by deleting the jth column of the matrix $[w_1, w_2, w_3, w_4]^T$. Then, the above system has a unique solution if the sixth point, z_5, is not chosen on the conic $d_1 x - d_2 y + (d_3/2) x^2 - d_4 xy + (d_5/2) y^2 = 0$.*

One can show that the hypothesis of Theorem 5.1 is satisfied with the computational cells chosen in this paper. We illustrate the generated difference formula for the computational cell about the mesh point 6 in Fig. 1 by taking the coordinates of the mesh points 6, 7, 11, 10, 5, and 2 in terms of h as follows:

$$\begin{aligned} &z_0 = (0, 0), \quad &z_1 = (h, 0), \quad &z_2 = (ah, bh), \\ &z_3 = (0, h), \quad &z_4 = (-h, 0), \quad &z_5 = (-h, -2h). \end{aligned} \quad (5.2)$$

We let the coordinates of $z_2 = (ah, bh)$, where a and b are constants, since the position of the mesh point 11 might be changed depending on the distance to the center mesh point 6. Then, the solution of Eq. (5.1) gives

$$\begin{aligned} u_y = [&(4ba - 2b^2 + 2 - 2a^2) u_0 + (a + a^2) u_1 - 2u_2 + (-4ab + 2b^2) u_3 \\ &+ (a^2 - a - ab) u_4 + ab u_5] / [-2hb(3a - b + 1)] \end{aligned}$$

$$\begin{aligned}
u_{yy} &= [(2-2ba-2b-2a^2)u_0+(a+a^2)u_1-2u_2+(ab+2b)u_3 \\
&\quad +(a^2-a-ab)u_4+abu_5]/[h^2b(3a-b+1)] \\
u_{xy} &= [(6-2b^2-4b-6a^2)u_0+(3a+3a^2)u_1-6u_2+(4b+2b^2)u_3 \\
&\quad +(3a^2-3a-b^2+b)u_4+(b^2-b)u_5]/[-2h^2b(3a-b+1)] \\
u_{xx} &= (u_1+u_4-2u_0)/h^2 \\
u_x &= (u_1-u_4)/2h. \qquad (5.3)
\end{aligned}$$

From Eq. (5.3), we see that the formulas would not exist if either $b=0$ or $3a-b+1=0$. This corresponds to the lines $y=0$ and $y=3x+1$, respectively. Note that those two lines are the degenerated form of the conic defined in Theorem 5.1 by the mesh points z_0, z_1, z_3, z_4, and z_5. In our computation cell choice strategy, the mesh point z_2 cannot be on either line. Similarly, difference formulas for the other types of computational cells may be obtained by solving (5.1) in each case.

REFERENCES

1. H. SCHLICHTING, *Boundary Layer Theory* (McGraw-Hill, New York, 1968), p. 78.
2. J. F. THOMPSON, *Appl. Numer. Math.* **1**, 3 (1985).
3. R. D. RUSSELL AND J. CHRISTIANSEN, *SIAM J. Numer. Anal.* **15**, 59 (1978).
4. D. A. ANDERSON, in *Proceedings, International Conference on the Numerical Generation of Curvilinear Coordinate Systems and Their Use in the Numerical Solution of Partial Differential equations, Nashville, Tennessee, 1982*, edited by J. F. Thompson (North-Holland, Amsterdam, 1982), p. 317.
5. H. A. DWYER, *AIAA J.* **22**, 1705 (1984).
6. M. S. SHEPHARD, in *Proceedings, Third National Congress on Pressure Vessels and Piping, California, 1979*, edited by M. S. Shephard and R. H. Gallagher (ASME, New York, 1979), p. 1.
7. R. E. BANK, in *Proceedings, Workshop Held at The University of Maryland, College Park, Maryland, 1983*, edited by I. Babuska *et al.* (SIAM, Philadelphia, 1983), p. 74.
8. R. LOHNER, K. MORGAN, AND O. C. ZIENKIEWICZ, *Comput. Meth. Appl. Mech. Eng.* **51**, 441 (1985).
9. E. A. DORFI AND L. O'C. DRURY, *J. Comput. Phys.* **69**, 175 (1987).
10. P. LUCHINI, *J. Comput. Phys.* **68**, 283 (1987).
11. K. NAKASHI AND G. S. DEIWERT, *AIAA J.* **24**, 948 (1986).
12. R. B. SIMPSON, in *Proceedings, Ninth Manitoba Conference on Numerical Mathematics and Computing, Manitoba, Canada, 1979*, p. 49.
13. I. ALTAS, *Adaptive Mesh Generation*, Ph. D. thesis, 1988, Department of Mathematics, University of Saskatchewan, Saskatoon, Canada.
14. J. R. RICE, E. N. HOUSTIS, AND W. R. DYKSEN, *Math. Comput.* **36**, 475 (1981).
15. J. P. ROACHE, *Computational Fluid Dynamics* (Hermosa, Albuquerque, NM, 1972).
16. M. J. BERGER, *Math. Comput.* **45**, 301 (1985).
17. K. N. GHIA, U. GHIA, AND C. T. SHIN, in *Proceedings, Applied Mechanics, Bioengineering, and Fluids Engineering Conference, Houston, Texas, 1983*, edited by K. N. Ghia *et al.* (ASME, New York, 1983), p. 35.
18. I. ALTAS AND J. W. STEPHENSON, *Appl. Math. Lett.*, to appear.