A COMPARISON OF TWO MULTI-GRID METHODS FOR SOLVING THE POISSON PROBLEM

by

Richard K. Taft

Department of Atmospheric Science
Colorado State University
Fort Collins, Colorado

Paper No. 415

A COMPARISON OF TWO MULTI-GRID METHODS FOR SOLVING THE POISSON PROBLEM

by

Richard K. Taft

This report was prepared with support provided by the
Naval Environmental Prediction Research Facility, Monterey, CA,
under program element 61153N 'Multigrid Elliptic Solver Development'
ONR Grant N00014-87-K-0535

Principal Investigators: Wayne H. Schubert and Gerald D. Taylor

Department of Atmospheric Science
Colorado State University
Fort Collins, CO 80523

June 1987

Atmospheric Science Paper No. 415

ABSTRACT

This paper presents two simple multi-grid algorithms for solving the Poisson problem: a standard second-order method and a less-common fourth-order method. The two algorithms are compared theoretically and numerically, and conclusions are drawn concerning speed, accuracy, resolution, and computer storage. For the same set of grids, the fourth-order method gives an approximation of greater accuracy, but the second-order method is quicker. On the other hand, to achieve a given accuracy, the fourth-order method requires less computer time and storage, but the approximation obtained has less resolution than that obtained from the second-order method. The numerical calculations were performed on a Zenith PC and checked on a Cyber mainframe computer. These results suggest that the fourth-order method may begin to suffer from round-off errors and cancellation effects on coarser grids sooner than the second-order method. This prediction is discussed; and PC users, in particular, are warned to check and question numerical results before accepting them.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Many physical phenomena can be modelled as elliptic boundary-value problems. One of the simplest and most common such problem is the Poisson problem given by

$$\nabla^2 u = f \tag{1}$$

with suitable boundary conditions. Consequently, many numerical methods have been developed to solve elliptic boundary-value problems such as the Poisson problem.

Classically, the Poisson problem was solved by discretizing equation (1) and then solving the resulting linear system of equations, either directly using Gaussian elimination or iteratively using a relaxation scheme such as Gauss-Seidel, SOR or ADI. Following this, so called "fast Poisson solvers" based on cyclic reduction were developed in the 1970's (e.g. Swarztrauber [1]). More recently, a second class of "fast solvers" have been developed which give the solution with "optimal efficiency", i.e., the computational work involved is proportional to the number of unknowns.

Achi Brandt first introduced multi-grid methods in the early 1970's as tools for developing such "fast solvers" for elliptic boundary-value problems [2]. There are now a wide class of multi-grid methods that can solve the Poisson problem with "optimal efficiency." Furthermore, these multi-grid methods have an advantage over many of the other "fast solvers" in that multi-grid methods are easily generalized and adapted to much more complex problems. In fact, multi-grid methods have now become an important and very efficient class of techniques for solving a wide variety of problems. There are many good reviews that further describe the history, general concepts, and various applications of multi-grid methods (e.g. [3] – [7]).

The fundamental idea of multi-grid methods is to approximate the given continuous problem, not only on a single fine grid of desired resolution, but also on a sequence of increasingly coarser grids, and then to obtain a solution on the finest grid with optimal efficiency by cycling between the discrete problems on these various grids. On a given grid, a simple relaxation scheme is used to reduce efficiently the components of error which oscillate on the scale of that grid. This relaxation, however, does not effectively reduce smooth error components (those that are non-oscillatory on the scale of the grid). The key to multi-grid methods is the fact that error components which are smooth on one grid become more oscillatory on a coarser grid. Thus, the coarser grids can be viewed as correction grids which accelerate the convergence of the relaxation scheme on the finest grid by efficiently reducing the smooth error components. The basis for the good efficiency of multi-grid methods is now clear: much of the work is performed on the coarser grids which have relatively few points, and thus, the computational cost involved is greatly reduced.

Until recently, most multi-grid algorithms actually applied to an elliptic boundary-value problem such as the Poisson problem were second-order methods in that their underlying discretization schemes were second-order approximations. Higher-order multi-grid algorithms using discretization schemes of fourth- and sixth-order were much less common and were generally considered to be too expensive due to their increased complexity. Recently, however, these higher-order algorithms have gained new interest and support as methods for obtaining high-accuracy approximations [8].

The body of this paper will present two simple multi-grid algorithms for solving the Poisson problem: a standard second-order method and a less-common fourth-order method. After notation and basic concepts are introduced, the two algorithms will be described in detail and then compared theoretically and numerically. Finally, conclusions concerning speed, accuracy, resolution, and computer storage will be made.

CHAPTER II

NOTATION AND BASIC CONCEPTS

The model problem for this paper is the Poisson problem with Dirichlet boundary conditions:

$$\begin{aligned} Lu = \nabla^2 u = f \qquad & \text{in } D \\ u = g \qquad & \text{on } S, \end{aligned} \tag{2}$$

where L is the Laplace operator $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$, D is the unit square $[0,1] \times [0,1]$ in R^2, f is a continuous function defined in D, g is a continuous function defined on S (the boundary of D), and u is the continuous solution which is to be approximated.

For each $h = 1/N$, where N is a positive integer, a uniform grid D^h with mesh spacing h is defined on the unit square as the following set of discrete points of D:

$$D^h = \{(x_i, y_j) = (ih, jh) \ : \ i, j = 0, 1, 2, \ldots, N\}.$$

Such a grid will be called an N-grid, but note that D^h actually contains $(N+1) \times (N+1)$ points. Lexicographic ordering refers to taking the interior grid points (x_i, y_j) in the order $(i,j) = (1,1), (2,1), \ldots, (N-1,1), (1,2), \ldots, (N-1,N-1)$. The intersection of S with D^h is denoted by S^h.

The grid functions f^h and g^h designate the pointwise restrictions of the continuous functions f and g onto D^h and S^h, respectively. In particular, $f^h(x_i, y_j) = f(x_i, y_j) = f_{ij}$ and $g^h(x_i, y_j) = g(x_i, y_j) = g_{ij}$. The grid function u^h represents the discrete approximation to the true solution u; and hence, $u_{ij} = u^h(x_i, y_j)$ approximates the true value of u at (x_i, y_j).

Approximating (2) on a grid D^h using finite differences gives the discrete Poisson problem:

$$\begin{aligned} L^h u^h = F^h \qquad & \text{in } D^h \\ u^h = g^h \qquad & \text{on } S^h, \end{aligned} \tag{3}$$

where L^h is a finite difference operator and F^h is a grid function related to f^h. The residual for problem (3) is the grid function $r^h = F^h - L^h\tilde{u}^h$, where the grid function $\tilde{u}^h$ is the current approximation to the solution of (3). The notation $r_{ij} = r^h(x_i, y_j)$ is also used. The residual norm is a root-mean-square quantity defined by

$$RN = \sqrt{h^2 \sum_{i,j=1}^{N-1} (r_{ij})^2} \,. \tag{4}$$

The basic concept of multi-grid methods can now be demonstrated by the following simple two-level multi-grid cycle:

i. Input initial guess for $\tilde{u}^h$ on fine grid with resolution h.

ii. Perform relaxation on $L^h\tilde{u}^h = F^h$ to improve $\tilde{u}^h$.

iii. Transfer residual $r^h = F^h - L^h\tilde{u}^h$ to a coarser grid with resolution $2h$ using a fine-to-coarse grid transfer operator I_h^{2h}.

iv. Solve the coarse grid correction equation $L^{2h}\tilde{v}^{2h} = I_h^{2h}\left(F^h - L^h\tilde{u}^h\right)$.

v. Transfer the correction $\tilde{v}^{2h}$ back to the fine grid using a coarse-to-fine grid transfer operator I_{2h}^h, and replace $\tilde{u}^h$ by $\tilde{u}^h + I_{2h}^h\tilde{v}^{2h}$.

vi. Perform relaxation on $L^h\tilde{u}^h = F^h$ to improve $\tilde{u}^h$ further.

This cycle has two main parts: a smoothing part (steps ii and vi) to reduce the high-frequency error components of $\tilde{u}^h$ by relaxation and a coarse grid correction part (steps iii–v) to reduce the smooth error components of $\tilde{u}^h$.

The above two-level cycle can be repeated to improve the approximation $\tilde{u}^h$ even further. In addition, by using a set of more than two grids, a simple multi-level cycle is easily obtained from this two-level cycle by recursion; i.e., by solving the equation in step (iv) by another application of a two-level cycle, etc. These simple cycles are examples of multi-grid control algorithms. In particular, the multi-level cycle just described forms the basis for the fixed V-cycle control algorithm.

CHAPTER III

DESCRIPTION OF MULTI-GRID ALGORITHMS

A specific multi-grid algorithm has several important components: a discretization method, a set of grids, grid transfer operators, a relaxation scheme, and a control algorithm. A variety of possible choices exists for each of these components, and hence, a large number of different multi-grid algorithms are possible.

The two algorithms of this paper differ primarily in the discretization method used. The other components are essentially identical for the two algorithms and are typically the simplest choices possible. In particular, both algorithms share the following basic components: a set of uniform, overlapping grids $\left\{D^M, D^{2M}, \ldots, D^{1/2}\right\}$ where M is an element of $\{1/64,\ 1/32,\ 1/16\}$; simple injection for the fine-to-coarse grid transfer of residuals; bilinear interpolation for the coarse-to-fine grid transfer of corrections; lexicographic Gauss-Seidel relaxation; and a fixed V-cycle control algorithm. These components combine to form the basic structure for these two algorithms, as shown in Figure 1.

3a. the second-order algorithm (Algorithm 1)

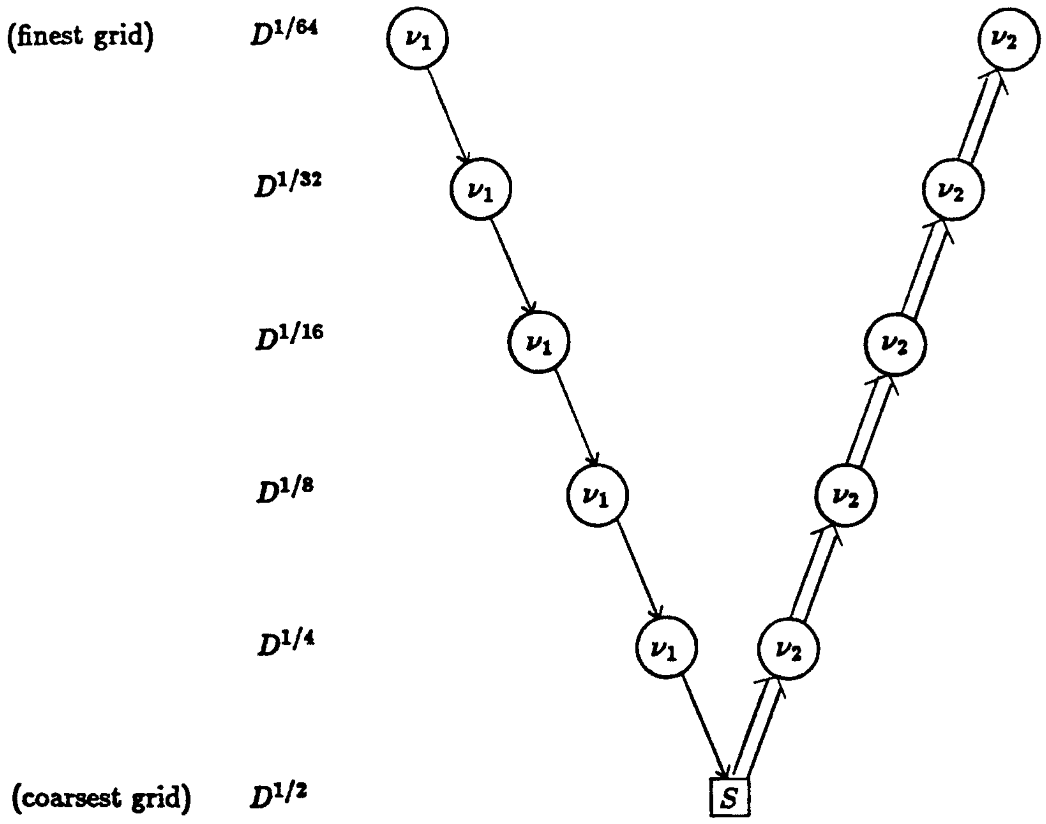
The discretization scheme of Algorithm 1 utilizes the standard second-order centered finite difference operator $(L^h)_2$ defined by the five-point difference stencil

$$\left(L^h\right)_2 = \left(1/h^2\right) \begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}_h .$$

For a given grid D^h, the discrete problem (3) now assumes the form

$$\begin{aligned} (1/h^2) \begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}_h u^h = f^h \qquad & \text{in } D^h \\ u^h = g^h \qquad & \text{on } S^h. \end{aligned}$$

Figure 1: The fixed V-cycle structure of the multi-grid algorithms of this paper (for the case where the finest grid is a 64-grid).



Key:		
	ν_1	perform ν_1 relaxation sweeps
	ν_2	perform ν_2 relaxation sweeps
	$\longrightarrow$	transfer residual by injection
	$\boxed{S}$	solve the discrete problem on the coarsest grid
	$\Longrightarrow$	transfer correction by bilinear interpolation

Reference: (adapted from [6])

The error in this approximation to the Poisson problem is of the order h^2 (i.e., the truncation error is $O(h^2)$), and hence, this algorithm is classified as second-order [9].

From (5), an equation of the following form is obtained for each interior grid point (x_i, y_j) in D^h:

$$(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - 4u_{ij})/h^2 = f_{ij}.$$

The equation for lexicographic Gauss-Seidel relaxation in Algorithm 1 therefore becomes

$$\tilde{u}_{ij}{}^{\text{new}} = \left(\tilde{u}_{i-1,j}{}^{\text{new}} + \tilde{u}_{i,j-1}{}^{\text{new}} + \tilde{u}_{i+1,j} + \tilde{u}_{i,j+1} - h^2 f_{ij}\right)/4\,. \tag{6}$$

Note that $\tilde{u}_{i-1,j}{}^{\text{new}}$ and $\tilde{u}_{i,j-1}{}^{\text{new}}$ are already computed before reaching the point (x_i, y_j). The residual for problem (5) can be expressed by the equation

$$r_{ij} = f_{ij} - (\tilde{u}_{i-1,j} + \tilde{u}_{i,j-1} + \tilde{u}_{i+1,j} + \tilde{u}_{i,j+1} - 4\tilde{u}_{ij})/h^2\,. \tag{7}$$

3b. the fourth-order algorithm (Algorithm 2)

The discretization scheme of Algorithm 2 utilizes a more complex fourth-order finite difference operator $(L^h)_4$ defined by the nine-point difference stencil

$$\left(L^h\right)_4 = \left(1/6h^2\right)\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}_h .$$

This operator is sometimes referred to as a Mehrstellen Verfahren difference operator [10]. For a given grid D^h, the discrete problem (3) is obtained by using this operator with a set of weights applied to f^h and assumes the form

$$\begin{aligned} (1/6h^2)\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}_h u^h &= (1/12)\begin{bmatrix} & 1 & \\ 1 & 8 & 1 \\ & 1 & \end{bmatrix}_h f^h \qquad \text{in } D^h \\ u^h &= g^h \qquad\qquad\qquad\qquad \text{on } S^h. \end{aligned} \tag{8}$$

This finite difference equation is easily derived using a Taylor series approach. The error in this approximation to the Poisson problem is of the order h^4 (i.e., the truncation error is $O(h^4)$), and hence, this algorithm is classified as fourth-order [8 and 11].

From (8), an equation of the following form is obtained for each interior grid point (x_i, y_j) in D^h:

$$[u_{i-1,j-1} + u_{i+1,j-1} + u_{i+1,j+1} + u_{i-1,j+1} - 20u_{ij}$$
$$+4(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1})]/6h^2 = F_{ij}$$

where $F_{ij} = (f_{i-1,j} + f_{i,j-1} + f_{i+1,j} + f_{i,j+1} + 8f_{ij})/12$. The equation for lexicographic Gauss-Seidel relaxation in Algorithm 2 therefore becomes

$$\tilde{u}_{ij}{}^{\text{new}} = \Big[\tilde{u}_{i-1,j-1}{}^{\text{new}} + u_{i+1,j-1}{}^{\text{new}} + \tilde{u}_{i+1,j+1} + \tilde{u}_{i-1,j+1}$$
$$+ 4\left(\tilde{u}_{i-1,j}{}^{\text{new}} + \tilde{u}_{i,j-1}{}^{\text{new}} + \tilde{u}_{i+1,j} + \tilde{u}_{i,j+1}\right) - 6h^2F_{ij}\Big]/20. \tag{9}$$

Note that all of the "new" quantities on the right-hand side of (9) have already been computed before reaching the point (x_i, y_j). The residual for problem (8) can be expressed by the equation

$$r_{ij} = F_{ij} - [\tilde{u}_{i-1,j-1} + \tilde{u}_{i+1,j-1} + \tilde{u}_{i+1,j+1} + \tilde{u}_{i-1,j+1} - 20\tilde{u}_{ij}$$
$$+ 4(\tilde{u}_{i-1,j} + \tilde{u}_{i,j-1} + \tilde{u}_{i+1,j} + \tilde{u}_{i,j+1})]/6h^2\,. \tag{10}$$

3c. pseudo-code for Algorithms 1 and 2

The use of Algorithms 1 and 2 for solving the discrete problem

$$L^H u^H = F^H \qquad \text{in } D^H$$
$$u^H = g^H \qquad \text{on } S^H\,,$$

where H is an element of {1/64, 1/32, 1/16}, can now be summarized by the following pseudo-code:

```
Initialize :  $\tilde{u}^H = 0$
              $F^H = f^H$                                                  for Algorithm 1
              $F^H = (1/12)\begin{bmatrix} & 1 & \\ 1 & 8 & 1 \\ & 1 & \end{bmatrix}_H f^H$   for Algorithm 2
              $h = H$
Input convergence tolerance TOL
Calculate initial residual $r^H$ using (7) for Algorithm 1 and (10) for
Algorithm 2
```

```
Calculate initial residual norm $RN$ using (4)
Perform the following until $RN < TOL$ {this gives the stopping criterion}
 Repeat until $h = 1/2$
  Perform $\nu_1$ relaxation sweeps on $L^h\tilde{u}^h = F^h$ using (6) for Algorithm 1
  and (9) for Algorithm 2
  Store $\tilde{u}^h$ and $F^h$
  Calculate residual $r^h$ using (7) for Algorithm 1 and (10) for Algorithm 2
  Set: $\tilde{u}^{2h} = 0$ {initialize coarse grid correction}
       $F^{2h} = I_h^{2h} r^h$ {transfer residual by injection}
       $h = 2h$
 Solve $L^{1/2}\tilde{u}^{1/2} = F^{1/2}$ {i.e., use $\nu_3$ relaxation sweeps}
 Repeat until $h = H$
  Set $\tilde{u}^{h/2} = \tilde{u}^{h/2} + I_h^{h/2}\tilde{u}^h$ {transfer correction using bilinear interpolation}
  Set $h = h/2$
  Perform $\nu_2$ relaxation sweeps on $L^h\tilde{u}^h = F^h$ using (6) for Algorithm 1
  and (9) for Algorithm 2
 Calculate residual $r^H$ using (7) for Algorithm 1 and (10) for Algorithm 2
 Calculate residual norm $RN$ using (4)
Set $u^H = \tilde{u}^H$    {final solution}.
```

The actual FORTRAN code used for Algorithms 1 and 2 is given in Appendices A and B. The values of ν_1, ν_2, and ν_3 used in this paper were 2, 1, and 3, respectively.

CHAPTER IV

THEORETICAL COMPARISONS

From the pseudo-code at the end of the last section, it is clear that Algorithms 1 and 2 have identical structures and differ only in the formulas used at various steps. This observation simply reflects the fact that the two algorithms are actually the same multi-grid method applied to two different finite difference equations, namely (5) and (8), respectively.

4a. complexity versus truncation error

The finite difference equation (8) is clearly more complex than the finite difference equation (5). Consequently, the formulas used in Algorithm 2 are more complex than the corresponding formulas in Algorithm 1; e.g., compare the relaxation formula (9) for Algorithm 2 with the corresponding formula (6) for Algorithm 1. Therefore, an apparent disadvantage of Algorithm 2 is its increased complexity. Until recently, this was a common criticism of such higher-order methods.

Algorithm 2, however, also has a possible advantage. Since the finite difference equation (8) has a smaller truncation error than the finite difference equation (5), it follows that Algorithm 2 has the potential to produce a more accurate solution to the Poisson problem than Algorithm 1 under the same conditions. In addition, since the truncation error depends on the mesh spacing of the finest grid, Algorithm 2 has the potential of producing on a coarser grid as accurate a solution as Algorithm 1 produces on a finer grid. It is this balance between the complexity disadvantage and the truncation error advantage of Algorithm 2 that will be examined numerically in the next section.

4b. work and computer storage requirements

From the pseudo-code for the two algorithms, it is clear that $\tilde{u}^h$ and F^h must be stored at each grid before transferring to the next coarser grid. Letting S represent the storage requirement for the two grid functions $\tilde{u}^h$ and F^h on the finest grid and noting that each coarser grid has approximately one-fourth as many points as the next finer grid, the storage requirement for the two algorithms can be approximated by

$$\left(1 + \frac{1}{4} + \frac{1}{16} + \ldots + \frac{1}{4^{m-1}}\right) S < \left(\frac{4}{3}\right) S$$

where m is the number of grid levels used. Thus, the storage requirements for the two algorithms are not much greater than those for solving the problem just on the finest grid.

By a similar argument, it follows that the work required per V-cycle (ignoring the work involved in residual transfers and interpolation—which is typically small) is less than 4/3 times the work required for $(\nu_1 + \nu_2)$ relaxation sweeps on the finest grid alone. Thus, if the finest grid has $0(N^2)$ points, the work per V-cycle is also $0(N^2)$ operations.

In coding the two algorithms, a large one-dimensional array was used to store the information of each grid. The table below indicates the required size of this storage array as a function of the finest grid used:

finest grid	size of storage array
$D^{1/64}$	11436
$D^{1/32}$	2986
$D^{1/16}$	808

Clearly, in terms of computer storage it is best to use the coarsest resolution possible to achieve a desired accuracy.

4c. analysis of relaxation and convergence

The relaxation process and speed of convergence of a multi-grid method can be analyzed using a technique called local mode analysis. Although this technique contains certain underlying assumptions, such as periodic boundary conditions, Achi Brandt claims that it always gives reliable estimates of the overall convergence rate of any multi-grid method [3]. This technique, in its simplest form, uses Fourier analysis to examine the effects of the relaxation scheme on the high frequency components of the error. In this simple form, the grid transfer processes are not considered.

Let $v^h = u^h - \tilde{u}^h$ and $v^{h,\text{new}} = u^h - \tilde{u}^{h,\text{new}}$ be the error [between the exact solution to the finite difference equation (u^h) and the current approximation to this solution] before and after a single relaxation sweep, respectively. Using the discrete Fourier modes $E_\theta\left(x_j, y_k\right) = \exp\left[i\left(j\theta_1 + k\theta_2\right)\right]$, where $\theta = (\theta_1, \theta_2)$ is the discrete vector wavenumber and the components θ_1 and θ_2 are integral multiples of $2\pi h$ between $-\pi$ and π, one can represent the errors v^h and $v^{h,\text{new}}$ as sums of components of the form $A_\theta E_\theta$ and $A_\theta^{\text{new}} E_\theta$, respectively. One relaxation sweep reduces the amplitude of the error component E_θ by the convergence factor $\mu(\theta) = \left|A_\theta^{\text{new}}/A_\theta\right|$. The smoothing factor $\bar{\mu}$ is defined as the maximum convergence factor $\mu(\theta)$ for the high wavenumbers, i.e.,

$$\bar{\mu} = \max_{\frac{\pi}{2} < |\theta| < \pi} \mu(\theta) \,,$$

where $|\theta| = \max\left(|\theta_1|, |\theta_2|\right)$. The smoothing factor is a measure of the rate at which the high frequency components of error are reduced by the relaxation process. In particular, $-(\log \bar{\mu})^{-1}$ relaxation sweeps are needed to reduce all the high frequency components by an order of magnitude.

It is easily shown (see [8]) that the convergence factors for Algorithms 1 and 2 are given by

$$\mu(\theta) = \left|\frac{e^{i\theta_1} + e^{i\theta_2}}{4 - e^{-i\theta_1} - e^{-i\theta_2}}\right|$$

and

$$\mu(\theta) = \left|\frac{4\left(e^{i\theta_1} + e^{i\theta_2}\right) + e^{i(\theta_1+\theta_2)} + e^{-i(\theta_1-\theta_2)}}{20 - 4\left(e^{-i\theta_1} + e^{-i\theta_2}\right) - e^{-i(\theta_1+\theta_2)} - e^{i(\theta_1-\theta_2)}}\right| \,,$$

respectively. The smoothing factors are therefore calculated to be

$$\begin{aligned} \bar{\mu} &= 0.500 \qquad \text{(Algorithm 1)} \\ \bar{\mu} &= 0.464 \qquad \text{(Algorithm 2)} \end{aligned}$$

It follows that 3.32 relaxation sweeps are required by Algorithm 1 to reduce all the high frequency components of the error by an order of magnitude, while Algorithm 2 only requires 3.00 relaxation sweeps. Thus, the relaxation scheme of Algorithm 2 is slightly more efficient as a smoothing process than that of Algorithm 1, and hence, the rate of convergence for Algorithm 2 should be slightly faster than that for Algorithm 1.

CHAPTER V

NUMERICAL COMPARISONS

The two algorithms were coded in Microsoft FORTRAN (version 3.31)—see Appendices for program listings—and run on a Zenith ZF-158-42 PC (an IBM compatible) under MS-DOS (version 3.1). The PC was run at a 8 Mhz clock speed and was equipped with an 8088 processor, an 8087-2 math coprocessor, and 640 KB of dynamic RAM.

5a. test problems

The model test problem is the following Poisson problem with Dirichlet boundary conditions:

$$\begin{aligned} \nabla^2 u(x,y) &= -4\pi^2 (a^2+b^2) \sin(2\pi a x + 2\pi b y) && \text{in } D = [0,1]\times[0,1] \\ u(x,y) &= \sin(2\pi a x + 2\pi b y) && \text{on } S = \partial D. \end{aligned}$$

In particular, the following three specific cases were tested:

Test Problem	Wavenumber a	Wavenumber b	Analytical Solution
1	1.0	2.0	$u(x,y) = \sin(2\pi x + 4\pi y)$
2	3.5	3.0	$u(x,y) = \sin(7\pi x + 6\pi y)$
3	0.5	3.5	$u(x,y) = \sin(\pi x + 7\pi y)$

Note the varying degree of oscillation in the solutions to these three different test problems.

5b. measurement of numerical performance

Execution time (real-time) and final solution error are used to measure the numerical performance of the two algorithms in solving these test problems. The execution time is a meaningful and reproducible quantity, in this case, since the computer used is a single-user machine. This real-time execution time is obtained by interfacing to the MS-DOS subroutine TIME before and after the multi-grid algorithm executes. The final solution error is given by the grid function $E^h = U^h - \tilde{u}^h$, where U^h is the actual solution projected on D^h and $\tilde{u}^h$ is the final approximation to U^h. The final solution error is measured using a discrete analog of the continuous Euclidean norm of E^h,

$$\left\|E^h\right\|_{2,h} = \left[\sum_{(x,y)\epsilon D^h} \left(E^h(x,y)\right)^2 h^2\right]^{1/2} .$$

5c. truncation error and convergence tolerance

Since the actual solutions to the test problems are known, the actual truncation errors involved can be computed. If the grid function R^h is defined as $R^h = F^h - L^h U^h$, where U^h is the actual solution projected on D^h, then the truncation error (TE) becomes

$$TE = \left[\sum_{(x,y)\epsilon D^h} \left(R^h(x,y)\right)^2 h^2\right]^{1/2} .$$

Thus, the truncation error is simply the residual norm (equation 4) calculated using the actual solution; i.e., the actual solution to the partial differential equation solves the finite difference equation with an error of TE. To solve the finite difference equation to a greater accuracy than TE is therefore not meaningful, and hence, TE was used as the convergence tolerance (see the pseudo-code of the two algorithms).

The truncation errors for the three test problems and for a variety of grids are given in Table 1. Note that for each problem, the truncation error of Algorithm 2 is always smaller than that of Algorithm 1. In fact, for test problems 1 and 2 the truncation error for Algorithm 2 on a 16-grid is actually smaller than that for Algorithm 1 on a 64-grid. Also note that the truncation errors for both algorithms decrease as the grid mesh spacing decreases.

5d. numerical results

The results of applying Algorithms 1 and 2 to test problems 1, 2, and 3 are summarized in Table 2.

Table 1: truncation error analysis for the three test problems (using Zenith PC)

Test Problem	Grid	Mesh Spacing	Truncation Error Algorithm 1	Truncation Error Algorithm 2
1	64	.01563	0.37472	0.0028675
1	32	.03125	1.4690	0.0053262
1	16	.06250	5.5938	0.080740
2	64	.01563	5.0810	0.015203
2	32	.03125	19.781	0.20172
2	16	.06250	73.245	2.1938
3	64	.01563	3.3000	0.019108
3	32	.03125	12.833	0.29093
3	16	.06250	47.312	4.2243

Table 2: numerical results of applying Algorithms 1 and 2 to the three test problems (using Zenith PC)

Test Problem	Alg.	Finest Grid	Required V-Cycles	Final Solution Error (x1000)	Execution Time (sec.)
1	1	64	3	1.805	40
	1	32	2	7.827	7
	1	16	2	29.01	4
	2	64	5	0.002121	92
	2	32	4	0.02703	19
	2	16	3	0.4310	4
2	1	64	2	6.103	27
	1	32	2	23.32	7
	1	16	1	280.4	1
	2	64	4	0.01608	73
	2	32	3	0.2577	14
	2	16	2	3.476	3
3	1	64	3	7.335	40
	1	32	2	32.08	7
	1	16	1	170.1	1
	2	64	4	0.04494	73
	2	32	3	0.6444	14
	2	16	2	11.05	3

5e. check of numerical results

The numerical results obtained from computer calculations (particularly those obtained on a PC) should always be checked and questioned since computers can only perform finite precision arithmetic. Round-off errors and cancellation effects can reduce the number of significant digits in a computed result, even to the point that no significant digits remain.

There is an easy check that can be performed on the truncation error results in Table 1. Since the truncation error is $0(h^2)$ for Algorithm 1 and $0(h^4)$ for Algorithm 2, the truncation error for a given problem should decrease by approximately a factor of four for Algorithm 1 and sixteen for Algorithm 2 as the mesh spacing is halfed (i.e., resolution doubled). The results in Table 1 follow these patterns, except the result for test problem 1 using Algorithm 2 on a 64 grid. This particular result is certainly questionable; and it is possible, therefore, that all the results are questionable.

Probably the best method for checking the reliability of a numerical result obtained using single precision arithmetic is to redo the calculation either in double precision on the same computer or in single precision on a different computer which has a greater number of significant digits. The Zenith PC used to obtain the results of Tables 1 and 2 is a 16-bit machine which uses two bytes to store a real number in single precision. This storage corresponds to approximately seven significant digits. To check the results of Tables 1 and 2, the calculations were redone in single precision on a Cyber 720 mainframe computer, a 64-bit machine which stores a real number with approximately twice the significant digits as the Zenith PC. These new results (see Tables 3 and 4) are less prone to the problems associated with finite precision arithmetic and are, therefore, considered more reliable than those in Tables 1 and 2.

Note that the truncation error results of Table 3 now completely follow the expected pattern mentioned earlier in this section. The one result of Table 1 that was questionable since it did not follow the expected pattern was, in fact, a meaningless result. (The more reliable result of Table 3 is an order of magnitude different.) Also note that three other results of Table 1 were significantly affected by round-off errors and cancellation effects, even though these three results "passed" the expected pattern test.

Table 3: truncation error analysis for the three test problems (using Cyber mainframe computer)

Test		Mesh	Truncation Error		
Problem	Grid	Spacing	Algorithm 1	Algorithm 2	
1	64	.01563	0.37470	0.00033647	*
1	32	.03125	1.4690	0.0052814	*
1	16	.06250	5.5938	0.080735	
2	64	.01563	5.0810	0.013865	*
2	32	.03125	19.781	0.20173	
2	16	.06250	73.245	2.1938	
3	64	.01563	3.3000	0.018774	*
3	32	.03125	12.833	0.29093	
3	16	.06250	47.312	4.2243	

Note: an asterisk indicates that the corresponding value from Table 1 does not agree (to at least two digits) with this value.

Table 4: numerical results of applying Algorithms 1 and 2 to the three test problems (using Cyber mainframe computer)

Test		Finest	Required	Final Solution	Extrapolated Execution
Problem	Alg.	Grid	V-Cycles	Error (x1000)	Time (sec.)
1	1	64	3	1.805	40
	1	32	2	7.827	7
	1	16	2	29.01	4
	2	64	6	0.001612 *	110
	2	32	4	0.02703	19
	2	16	3	0.4310	4
2	1	64	2	6.103	27
	1	32	2	23.32	7
	1	16	1	280.4	1
	2	64	4	0.01605	73
	2	32	3	0.2577	14
	2	16	2	3.476	3
3	1	64	3	7.335	40
	1	32	2	32.08	7
	1	16	1	170.1	1
	2	64	4	0.04485	73
	2	32	3	0.6444	14
	2	16	2	11.05	3

Note: an asterisk indicates that the corresponding value from Table 2 is significantly different from this result.

The more reliable truncation error results of Table 3 were used as the convergence tolerances when obtaining the results in Table 4 on the Cyber mainframe computer. Note that only one result in Table 4 was significantly different from the corresponding result in Table 2. This difference is probably due more to the use of a smaller convergence tolerance in obtaining the result in Table 4 than to problems with finite precision arithmetic. The execution time listed in Table 4 is not the execution time for the Cyber mainframe computer, but is an extrapolated value for the execution time on the Zenith PC. This extrapolated value was obtain using the number of V-cycles required on the Cyber mainframe computer and the results in Table 2.

In conclusion, it appears that the truncation error analysis for Algorithm 2 begins to have problems with round-off errors and cancellation effects when used on a 64-grid. If progressively finer grids are used, the truncation error analysis for both algorithms would eventually develop problems due to this finite precision arithmetic. It is reasonable that such problems develop sooner (i.e., on a coarser grid) for Algorithm 2 than for Algorithm 1 since formula (10) for the residual in Algorithm 2 involves approximately twice the number of subtractions as does formula (7) for the residual in Algorithm 1. These observations suggest that Algorithm 2 itself will suffer such round-off and cancellation problems on coarser grids sooner than Algorithm 1, but further testing is needed to verify this prediction. In any case, the above comments should be a warning, particularly to PC users, that numerical results must be checked and questioned.

CHAPTER VI

CONCLUSIONS

The results obtained (see Tables 2 and 4) are very similar for each of the three test problems. In particular, for all three test problems, the following observations can be made:

i. For a given resolution, Algorithm 1 is faster than Algorithm 2.

ii. For a given resolution, Algorithm 2 is more accurate than Algorithm 1.

iii. Algorithm 2 at a given resolution is faster and more accurate than Algorithm 1 at twice that resolution.

Note that observations (i) and (ii) are expected results, since Algorithm 2 is more complex and has a better truncation error than Algorithm 1. Observation (iii), on the other hand, is a less expected and more interesting result.

From these observations, it is clear that conclusions concerning which algorithm is best for solving the Poisson problem are case dependent. Consider the case where one is forced by resolution requirements to use a particular finest grid. Then Algorithm 1 is expected to be "best" in terms of speed, while Algorithm 2 is expected to be "best" in terms of accuracy. For example, if one were solving a problem, the solution of which was expected to have local behavior requiring approximately a 64-grid to resolve, then one would be forced to use at least a 64-grid as the finest grid. In such a case, Algorithm 2 should be used if accuracy is more important than speed, while Algorithm 1 should be used if speed (and hence, computer expense) is of greater concern than very high accuracy.

Now consider the case where the choice of which finest grid to use is much less restricted and the primary concern is to obtain a solution of some desired accuracy. In this case, one would hope to choose a finest grid and an algorithm which would give a solution of the desired accuracy in the least amount of time. It is in this context that observation (iii) becomes relevant. For example, suppose one were solving a problem, the solution of which was expected to be fairly smooth locally so that even a 16-grid could resolve it, and suppose that, while using Algorithm 1, one was forced to use a 64-grid as the finest grid in order to achieve a desired accuracy. In this case, it may be "better" to use Algorithm 2 and only a 32-grid as the finest grid. It is expected that this combination would produce a more accurate solution in less time; however, this combination has a possible disadvantage in that one obtains the solution at only about one-quarter as many discrete points of the domain. If this disadvantage is a problem in a particular context, it may be possible to use a fourth-order interpolation method, such as cubic spline interpolation, to fill in the solution between the grid points.

In conclusion, both algorithms can be practical methods for solving the Poisson problem and neither method is in general better than the other. Although Algorithm 2 is generally thought to be more accurate, but slower, than Algorithm 1, it appears that in at least certain cases Algorithm 2 can be used in such a way that it is both more accurate and faster than Algorithm 1. Thus, even though standard second-order methods such as Algorithm 1 are more typically used, higher-order methods such as Algorithm 2 deserve to be considered in many cases.

Note that the above conclusions do not take into consideration the potential problems which may occur due to finite precision arithmetic. The truncation error results suggest that Algorithm 2 will begin to suffer from round-off errors and cancellation effects on coarser grids sooner than Algorithm 1. This prediction, if true, is a potential disadvantage of Algorithm 2. For example, if one were forced to work on a particular grid on which Algorithm 2 suffered from these round-off and cancellation problems, but Algorithm 1 did not, then Algorithm 1 would be the method of choice, regardless of speed and accuracy considerations. PC users, in particular, may be forced to consider this potential disadvantage of Algorithm 2.

REFERENCES

[1] P.N. SWARZTRAUBER, "The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle," *SIAM Review*, **19** (1977), pp. 490–501.

[2] A. BRANDT, "Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems," Proceedings Third International Conference on Numerical Methods in Fluid Mechanics (Paris, 1972), *Lecture Notes in Physics*, Springer-Verlag, Berlin, **18** (1973), pp. 82–89.

[3] A. BRANDT, "Multi-level adaptive solutions to boundary-value problems," *Math. Comp.*, **31** (1977), pp. 333–390.

[4] A. BRANDT, "Guide to multigrid development," Multigrid Methods (W. Hackbusch, U. Trottenberg, eds.), *Lecture Notes in Mathematics*, Springer-Verlag, Berlin, **960** (1982), pp. 220–312.

[5] K. STUBEN and U. TROTTENBERG, "Multigrid methods: fundamental algorithms, model problem analysis and applications," Multigrid Methods (W. Hackbusch, U. Trottenberg, eds.), *Lecture Notes in Mathematics*, Springer-Verlag, Berlin, **960** (1982), pp. 1–176.

[6] S.R. FULTON, P.E. CIESIELSKI, and W.H. SCHUBERT, "Multigrid methods for elliptic problems: a review," *Mon. Wea. Rev.*, **114** (1986), pp. 943–959.

[7] A. BRANDT, "Multi-level approaches to large scale problems," to appear: Proceedings International Congress of Mathematicians, 1986, Berkley, CA.

[8] S. SCHAFFER, "Higher order multi-grid methods," *Math. Comp.*, **43** (1984), pp. 89–115.

[9] A.R. MITCHELL and D.F. GRIFFITHS, *The Finite Difference Method in Partial Differential Equations*, John Wiley & Sons, New York, 1980, p. 104.

[10] L. COLLATZ, *The Numerical Treatment of Differential Equations*, Springer-Verlag, Berlin, 1960, pp. 384–387.

[11] E.N. HOUSTIS and T.S. PAPATHEODOROU, "High-order fast elliptic equation solver," *ACM Trans. Math. Software*, **5** (Dec. 1979), pp. 431–441.

APPENDIX A

FORTRAN CODE FOR ALGORITHM 1

FORTRAN Code for Setting Up Multi-Grid Solver:

```
$STORAGE:2
$NOFLOATCALLS
$LARGE
C
C     ************************************************************
C
      INTERFACE TO SUBROUTINE DATE (N,STR)
C
      CHARACTER*10  STR [NEAR,REFERENCE]
      INTEGER*2     N [VALUE]
C
      END
C
C     *********************************************************
C
      PROGRAM MULTIG
C
C       AUTHOR:  Richard Taft
C         DATE:  March 18, 1987
C     LANGUAGE:  Microsoft FORTRAN (version 3.31)
C                  (Libraries used:  8087.LIB and FORTRAN.LIB)
C
C      PURPOSE:  This routine sets up and uses subroutine MGSOLV to
C                  approximate the solution to the 2-D Poisson problem
C
C               u   + u   = -4*pi*pi*(A*A + B*B)sin(2*pi*A*x + 2*pi*B*y)
C                xx    yy
C                                                           in [0,1]x[0,1]
C
C                  u(x,y) = sin(2*pi*A*x + 2*pi*B*y)   on the boundary.
C
C                  using either a standard second-order or a less common
C                  fourth-order fixed V-cycle multi-grid algorithm.  The
C                  analytical solution is given by
C
C                      u(x,y) = sin(2*pi*A*x + 2*pi*B*y).
C
      INTEGER        FGRID,CGRID,MAXGRD,GRIDPT,I,J,V1,VC,V2,ALG,VMAX
      PARAMETER      (MAXGRD=65)
      REAL           A,B,TWOPI,WX,WY,H,TOL,RNORM,ERROR
      REAL           UCALC(MAXGRD,MAXGRD),F(MAXGRD,MAXGRD)
      REAL           UEXACT(MAXGRD,MAXGRD)
      CHARACTER      ANSWER,STORE
      CHARACTER*10   TODAY
      CHARACTER*12   OUTFIL
      CHARACTER*14   NAME,PURPOS,PLACE,PROG
      CHARACTER*45   STENCIL
      DATA           VMAX,V1,VC,V2,FGRID,CGRID,ALG/9,2,3,1,64,2,1/
      DATA           A,B,TOL/1.0,2.0,3.7472E-01/
      DATA           NAME,PURPOS/'RICHARD TAFT','MASTER''S WORK'/
      DATA           PLACE,PROG/'CSU MATH DEPT','MULTIG PROGRAM'/
C
C                     *** SET-UP PROBLEM SPECS ***
C
   20 WRITE(*,1000)'MULTIGRID SOLVER'
      WRITE(*,1050)'PROBLEM SPECS:'
      WRITE(*,1100)A,B
C
   40 WRITE(*,1200)'ENTER:  P  to modify problem specs',
     +                     'M  to see method specs',
     +                     'Q  to quit this session'
      READ(*,'(1A)')ANSWER
C
      IF (ANSWER.EQ.'P') THEN
        WRITE(*,'(//5X,A\)')'Input new wavenumber A ...(XX.X)...>'
        READ(*,'(F4.1)')A
        WRITE(*,'(5X,A\)')'Input new wavenumber B ...(YY.Y)...>'
        READ(*,'(F4.1)')B
        GOTO 20
      ELSEIF (ANSWER.EQ.'M') THEN
        GOTO 60
      ELSEIF (ANSWER.EQ.'Q') THEN
        STOP
      ELSE
        GOTO 40
      ENDIF
C
C                      *** SET-UP METHOD SPECS ***
C
   60 H = 1.0/REAL(FGRID)
   80 WRITE(*,1000)'MULTIGRID SOLVER'
      WRITE(*,1050)'METHOD SPECS:'
      IF (ALG.EQ.1) THEN
        STENCIL = 'Standard Second-order FDE (5-Point Stencil)'
      ELSE
        STENCIL = 'A Fourth-order FDE (9-Point Stencil)'
      ENDIF
      WRITE(*,1300)STENCIL,V1,VC,V2,FGRID,CGRID,H,TOL,VMAX
C
  100 WRITE(*,1400)'ENTER:  M  to modify method specs',
     +                     'P  to review problem specs',
     +                     'S  to solve problem',
     +                     'Q  to quit this session'
      READ(*,'(1A)')ANSWER

C
      IF (ANSWER.EQ.'M') THEN
  120   WRITE(*,1400)'ENTER:  D  to modify the discretization scheme',
     +                       'V  to modify the V-cycle sweeps or VMAX',
     +                          'G  to modify the grid intervals',
     +                          'T  to modify the convergence tolerance',
     +                          'M  to return to Method Specs'
        READ(*,'(1A)')ANSWER
        IF (ANSWER.EQ.'D') THEN
          ALG = 2/ALG
          GOTO 80
        ELSEIF (ANSWER.EQ.'V') THEN
          WRITE(*,'(//5X,A\)')'Input new value for V1 ...(1-9)...>'
          READ(*,'(I1)')V1
          WRITE(*,'(5X,A\)')'Input new value for VC ...(1-9)...>'
          READ(*,'(I1)')VC
          WRITE(*,'(5X,A\)')'Input new value for V2 ...(1-9)...>'
          READ(*,'(I1)')V2
          WRITE(*,'(5X,A\)')'Input new value for VMAX ...(1-9)..>'
          READ(*,'(I1)')VMAX
          GOTO 80
        ELSEIF (ANSWER.EQ.'G') THEN
          WRITE(*,'(//5X,A,A\)')'Input intervals for finest grid',
     +                             ' ...(4,8,16,32,64)...>'
          READ(*,'(I2)')FGRID
          WRITE(*,'(5X,A,A\)')'Input intervals for coarsest grid',
     +                           ' ...2,4,8,16,32 and < fine grid)...>'
          READ(*,'(I2)')CGRID
          GOTO 60
        ELSEIF (ANSWER.EQ.'T') THEN
          WRITE(*,'(//5X,A,A\)')'Input new value for convergence',
     +                             ' tolerance ...(X.XXXXE-YY)...>'
          READ(*,'(E10.4)')TOL
          GOTO 80
        ELSEIF (ANSWER.EQ.'M') THEN
          GOTO 80
        ELSE
          GOTO 120
        ENDIF
      ELSEIF (ANSWER.EQ.'P') THEN
        GOTO 20
      ELSEIF (ANSWER.EQ.'S') THEN
        GOTO 140
      ELSEIF (ANSWER.EQ.'Q') THEN
        STOP
      ELSE
        GOTO 100
      ENDIF
C
C               *** REPORT PROBLEM AND METHOD SPECS ***
C
  140 OPEN(1,FILE='PRN')
      CALL DATE (10,TODAY)
      WRITE(1,1450)NAME,PURPOS,PLACE,TODAY,PROG
      WRITE(1,1000)'MULTIGRID SOLVER'
      WRITE(1,1150)'PROBLEM SPECS:'
      WRITE(1,1100)A,B
      WRITE(1,1250)'METHOD SPECS:'
      WRITE(1,1300)STENCIL,V1,VC,V2,FGRID,CGRID,H,TOL,VMAX
C
```

```
C          *** DISCRETIZE ANALYTICAL SOLUTION AND RIGHT-HAND SIDE OF ***
C          *** FDE AND SET-UP INITIAL GUESS FOR CALCULATED SOLUTION  ***
C
      TWOPI = 2.0*ACOS(-1.0)
      WX = TWOPI*A
      WY = TWOPI*B
      GRIDPT = FGRID + 1
C
      CALL U2GRID(UEXACT,WX,WY,H,GRIDPT)
      CALL F2GRID(F,WX,WY,H,GRIDPT,ALG)
      CALL C2GRID(0.0,UCALC,GRIDPT)
      CALL U2BDRY(UCALC,WX,WY,H,GRIDPT)
C
C                    *** SOLVE AND REPORT RESULTS ***
C
      CALL MGSOLV(UCALC,F,FGRID,CGRID,V1,VC,V2,VMAX,TOL,RNORM,ALG)
      ERROR = DIFNRM(UEXACT,UCALC,H,GRIDPT)
      WRITE(1,1500)'RESULTS:','Final Residual Norm',RNORM,
     +             'Final Solution Error',ERROR
C
C                       *** STORE SOLUTION ***
C
      WRITE(*,'(//5X,A\)')'Store solution? ...(Y or N)...>'
      READ(*,'(1A)')STORE
C
      IF (STORE.EQ.'Y') THEN
        WRITE(*,'(/5X,A,A\)')'Input new file name for storage ...',
     +                       '(Y:XXXXXX.ZZZ)...>'
        READ(*,'(A)')OUTFIL
        OPEN(2,FILE=OUTFIL,STATUS='NEW')
        WRITE(2,1600)'SOLUTION:','I','J','ACTUAL U(I,J)',
     +               ' CALC U(I,J) ','I','J','ACTUAL U(I,J)',
     +               ' CALC U(I,J) '
        DO 500 J=1,FGRID-1,2
          DO 600 I=1,GRIDPT
            WRITE(2,1700)I-1,J-1,UEXACT(I,J),UCALC(I,J),
     +                   I-1,J,UEXACT(I,J+1),UCALC(I,J+1)
  600     CONTINUE
          WRITE(2,*)
  500   CONTINUE

C
        J=GRIDPT
        DO 700 I=1,GRIDPT
          WRITE(2,1800)I-1,J-1,UEXACT(I,J),UCALC(I,J)
  700   CONTINUE
        CLOSE(2)
      ENDIF
C
      CLOSE(1)
      GOTO 20
C
C                    *** FORMAT STATEMENTS ***
C
 1000 FORMAT(//30X,20('*')/30X,'* ',A,' *'/30X,20('*')//)
 1050 FORMAT(/5X,A/)
 1100 FORMAT(7X,'2-D Poisson Problem',7X,'u  (x,y) + u  (x,y) = ',
     +        'f(x,y)'/34X,'xx',9X,'yy'//7X,
     +        'Domain in RxR',13X,'[0,1]x[0,1]'/7X,'Boundary Conditions',
     +        7X,'Dirichlet'/7X,'Input Function',12X,
     +        'f(x,y) = C*sin(2*pi*A*x + 2*pi*B*y)'/36X,'where C = ',
     +        '-4*pi*pi*(A*A + B*B)' /7X,'Analytical Solution',
     +        7X,'u(x,y) = sin(2*pi*A*x + 2*pi*B*y)'/7X,
     +        'Wavenumbers',15X,'A = ',F4.1/33X,'B = ',F4.1/)
 1150 FORMAT(/5X,A/'+',4X,13('_')/)
 1200 FORMAT(/5X,A/2(13X,A/))
 1250 FORMAT(/5X,A/'+',4X,12('_')/)
 1300 FORMAT(7X,'Discretization Scheme',5X,A/7X,'Control Algorithm',
     +        9X,'Repeated Fixed V-Cycles'/
     +        36X,'V1 = ',I1,' [relax. sweep(s) per level down]'/
     +        36X,'VC = ',I1,' [relax. sweep(s) on coarsest grid]'/
     +        36X,'V2 = ',I1,' [relax. sweep(s) per level up]'/
     +        7X,'Grid Intervals',12X,'FGRID = ',I2,' (finest grid)'/
     +        33X,'CGRID = ',I2,' (coarsest grid)'/
     +        7X,'Mesh Spacing',14X,'H = ',F7.5,' (finest grid)'/
     +        7X,'Relaxation Scheme',9X,'Gauss-Seidel (lexicographic)'/
     +        7X,'Init. Solution Guess',6X,'U = 0.0   (for interior',
     +        ' grid pts.)'/7X,'Convergence Tolerance',5X,'TOL = ',1P,
     +        E10.4/7X,'Max Allowed V-Cycles',6X,'VMAX = ',I2/)
 1400 FORMAT(/5X,A/4(13X,A/))
 1450 FORMAT(//5(60X,A/))
 1500 FORMAT(/5X,A/'+',4X,7('_')//7X,A,7X,1P,E13.6/7X,A,6X,E13.6/'1')
 1600 FORMAT(/2X,A//2(2(2X,A),2(1X,A),1X)/
     +        2(2(1X,2('=')),2(1X,13('=')),1X)//)
 1700 FORMAT(2(2(1X,I2),1P,2(1X,E13.6)))
 1800 FORMAT(2(1X,I2),1P,2(1X,E13.6))
C
      END
C
C     ***************************************************************
C

      SUBROUTINE F2GRID(GRID,WX,WY,H,GRIDPT,ALG)
C
C     PURPOSE:  Puts the function FFUN on [0,1]x[0,1] into the
C                   GRIDPTxGRIDPT array GRID using a uniform mesh spacing
C                   of H and according to the right-hand side of the
C                   finite difference equation corresponding to algorithm
C                   ALG.
C
      INTEGER     MAXGRD,GRIDPT,I,J,ALG
      PARAMETER   (MAXGRD=65)
      REAL        H,X,Y,WX,WY,FFUN
      REAL        GRID(MAXGRD,MAXGRD),TEMP(MAXGRD,MAXGRD)
C
      FFUN(X,Y) = -(WX*WX + WY*WY)*SIN(WX*X + WY*Y)
C
C                    *** DISCRETIZE FFUN ***
C
      DO 20 J=1,GRIDPT
        Y = 0.0 + REAL(J-1)*H
        DO 10 I=1,GRIDPT
          X = 0.0 + REAL(I-1)*H
          GRID(I,J) = FFUN(X,Y)
          TEMP(I,J) = FFUN(X,Y)
   10   CONTINUE
   20 CONTINUE
C
C              *** FORM RIGHT SIDE OF APPROPRIATE FDE ***
C
      IF (ALG.EQ.1) RETURN
C
      DO 30 J=2,GRIDPT-1
      DO 30 I=2,GRIDPT-1
        GRID(I,J) = TEMP(I,J+1)+TEMP(I-1,J)+TEMP(I,J-1)+TEMP(I+1,J)
        GRID(I,J) = (GRID(I,J) + 8.0*TEMP(I,J))/12.0
   30 CONTINUE
C
      RETURN
      END
C
C     **************************************************************
C
      SUBROUTINE U2GRID(GRID,WX,WY,H,GRIDPT)
C
C     PURPOSE:  Puts the function UFUN on [0,1]x[0,1] into the
C               GRIDPTxGRIDPT array GRID using a uniform mesh spacing H.
C
      INTEGER    MAXGRD,GRIDPT,I,J
      PARAMETER  (MAXGRD=65)
      REAL       H,X,Y,WX,WY,UFUN,GRID(MAXGRD,MAXGRD)
C
      UFUN(X,Y) = SIN(WX*X + WY*Y)

C
      DO 20 J=1,GRIDPT
        Y = 0.0 + REAL(J-1)*H
        DO 10 I=1,GRIDPT
          X = 0.0 + REAL(I-1)*H
          GRID(I,J) = UFUN(X,Y)
   10   CONTINUE
   20 CONTINUE
C
      RETURN
      END
C
C     **************************************************************
C
      SUBROUTINE C2GRID(VALUE,GRID,GRIDPT)
C
C     PURPOSE:  Puts the constant VALUE into the GRIDPTxGRIDPT array
C                 GRID.
C
      INTEGER    MAXGRD,GRIDPT,I,J
      PARAMETER  (MAXGRD=65)
      REAL       VALUE,GRID(MAXGRD,MAXGRD)
C
      DO 10 J=1,GRIDPT
      DO 10 I=1,GRIDPT
        GRID(I,J) = VALUE
   10 CONTINUE
```

```
C
      RETURN
      END
C
C     ****************************************************************
C
      SUBROUTINE U2BDRY(GRID,WX,WY,H,GRIDPT)
C
C     PURPOSE:  Puts the function UFUN on the boundary of [0,1]x[0,1]
C                 into the boundary of the GRIDPTxGRIDPT array GRID
C                 using a uniform mesh spacing of H.
C
      INTEGER     MAXGRD,GRIDPT,I,J
      PARAMETER   (MAXGRD=65)
      REAL        H,X,Y,WX,WY,UFUN,GRID(MAXGRD,MAXGRD)
C
      UFUN(X,Y) = SIN(WX*X + WY*Y)
C
      DO 10 I=1,GRIDPT
        X = 0.0 + REAL(I-1)*H
        GRID(I,1) = UFUN(X,0.0)
        GRID(I,GRIDPT) = UFUN(X,1.0)
   10 CONTINUE
C

      DO 20 J=2,GRIDPT-1
        Y = 0.0 + REAL(J-1)*H
        GRID(1,J) = UFUN(0.0,Y)
        GRID(GRIDPT,J) = UFUN(1.0,Y)
   20 CONTINUE
C
      RETURN
      END
C
C     ****************************************************************
C
      REAL FUNCTION DIFNRM(GRID1,GRID2,H,GRIDPT)
C
C     PURPOSE:  Calculates a discrete analog of the continuous
C                 Euclidean norm of the difference of two GRIDPTxGRIDPT
C                 arrays (GRID1 AND GRID2) both having a uniform mesh
C                 spacing of H.
C
      INTEGER     MAXGRD,GRIDPT,I,J
      PARAMETER   (MAXGRD=65)
      REAL        H,DIFF,GRID1(MAXGRD,MAXGRD),GRID2(MAXGRD,MAXGRD)
C
      DIFNRM = 0.0
      DO 10 J=1,GRIDPT
      DO 10 I=1,GRIDPT
        DIFF = GRID2(I,J) - GRID1(I,J)
        DIFNRM = DIFNRM + DIFF*DIFF
   10 CONTINUE
      DIFNRM = SQRT(DIFNRM*H*H)
C
      RETURN
      END
```

APPENDIX B

FORTRAN CODE FOR ALGORITHM 2

FORTRAN Code for Multi-Grid Solver:

```
$STORAGE:2
$NOFLOATCALLS
$LARGE
C
C     ************************************************************
C
      INTERFACE TO SUBROUTINE TIME (N,STR)
C
      CHARACTER*10  STR [NEAR,REFERENCE]
      INTEGER*2     N [VALUE]
C
      END
C
C     **************************************************************
C
      SUBROUTINE MGSOLV(U,F,FGRID,CGRID,V1,VC,V2,VMAX,TOL,RNORM,ALG)
C
C       AUTHOR:  Richard Taft
C         DATE:  March 18, 1987
C     LANGUAGE:  Microsoft FORTRAN (version 3.31)
C                  (Libraries used:  8087.LIB and FORTRAN.LIB)
C
C      PURPOSE:  This routine uses a fixed V-cycle multi-grid algorithm
C                 to approximate the solution to the finite difference
C                 equation for the 2-D Poisson problem with Dirichlet
C                 boundary conditions obtained using either the
C                 standard second-order 5-point stencil (ALG=1) or a
C                 less common fourth-order 9-point stencil (ALG=2).
C                 The initial solution guess is inputted in array U and
C                 the final calculated solution is returned in array U.
C                 The right-hand side of the FDE is inputted in array F.
C                 FGRID and CGRID give the number of intervals of the
C                 finest and coarsest (uniform) grids, respectively.
C                 The fixed V-cycle structure is defined by V1, VC, and
C                 V2.  V-cycles are repeated until either the residual
C                 norm (RNORM) is less than the convergence tolerance
C                 (TOL) or the number of V-cycles performed equals the
C                 maximum allowed V-cycles (VMAX).  A trace of the
C                 residual norm versus the number of V-cycles performed
C                 and the execution time for the V-cycles performed are
C                 reported.
```

```
C
      INTEGER        MAXGRD,WSPACE,MAXLEV
      PARAMETER      (MAXGRD=65,WSPACE=11436,MAXLEV=6)
      INTEGER        FGRID,CGRID,GSIZE,GRDPTS(MAXLEV)
      INTEGER        WPOINT,UBEGIN(MAXLEV),FBEGIN(MAXLEV)
      INTEGER        LEVEL,FLEVEL,CLEVEL,STEP,ALG
      INTEGER        V1,VC,V2,SWEEP,VCOUNT,VMAX
      REAL           MESH,H(MAXLEV),TOL,RNORM
      REAL           U(MAXGRD,MAXGRD),F(MAXGRD,MAXGRD),W(WSPACE)
      CHARACTER*10   TSTART,TDONE
      PARAMETER      (FLEVEL=1,STEP=1)
C
C                      *** SET-UP STORAGE ARRAY W ***
C
      WPOINT = 1
      LEVEL = 0
      GSIZE = FGRID
      MESH = 1.0/REAL(FGRID)
C
    5 LEVEL = LEVEL + STEP
      H(LEVEL) = MESH
      GRDPTS(LEVEL) = GSIZE+1
      UBEGIN(LEVEL) = WPOINT
      WPOINT = WPOINT + GRDPTS(LEVEL)*GRDPTS(LEVEL)
      FBEGIN(LEVEL) = WPOINT
      WPOINT = WPOINT + GRDPTS(LEVEL)*GRDPTS(LEVEL)
      GSIZE = GSIZE/2
      MESH = 2.0 * MESH
      IF (GSIZE.GE.CGRID) GOTO 5
      CLEVEL = LEVEL
C
C                  *** PREPARE FOR FIRST V-CYCLE ***
C
      LEVEL = FLEVEL
      VCOUNT = 0
C
      CALL RESNRM(U,F,H(LEVEL),GRDPTS(LEVEL),RNORM,ALG)
      WRITE(1,1000)'SOLVING:','(Using Subroutine MGSOLV)',
     +             'NUMBER OF V-CYCLES COMPLETED','RESIDUAL NORM',
     +             '0',RNORM
C
C                      *** PERFORM A V-CYCLE ***
C
      CALL TIME (10,TSTART)
  100 DO 10 SWEEP=1,V1
        CALL GSRLAX(U,F,H(LEVEL),GRDPTS(LEVEL),ALG)
   10 CONTINUE
C
      CALL STOREW(U,F,GRDPTS(LEVEL),UBEGIN(LEVEL),FBEGIN(LEVEL),W)
      CALL F2CORS(U,F,H(LEVEL),GRDPTS(LEVEL),GRDPTS(LEVEL+1),ALG)
      LEVEL = LEVEL + STEP
      IF (LEVEL.LT.CLEVEL) GOTO 100

C
      DO 20 SWEEP=1,VC
        CALL GSRLAX(U,F,H(LEVEL),GRDPTS(LEVEL),ALG)
   20 CONTINUE
```

```
C
  200 CALL C2FINE(U,W,GRDPTS(LEVEL),GRDPTS(LEVEL-1),UBEGIN(LEVEL-1))
      LEVEL = LEVEL - STEP
      CALL RETRVW(U,F,GRDPTS(LEVEL),UBEGIN(LEVEL),FBEGIN(LEVEL),W)
C
      DO 30 SWEEP=1,V2
        CALL GSRLAX(U,F,H(LEVEL),GRDPTS(LEVEL),ALG)
   30 CONTINUE
C
      IF (LEVEL.GT.FLEVEL) GOTO 200
C
C                   *** PERFORM ANOTHER V-CYCLE IF NEEDED ***
C
      VCOUNT = VCOUNT + 1
      CALL RESNRM(U,F,H(LEVEL),GRDPTS(LEVEL),RNORM,ALG)
      WRITE(1,1100)VCOUNT,RNORM
C
      IF (RNORM.GT.TOL) THEN
        IF (VCOUNT.LT.VMAX) THEN
          GOTO 100
        ELSE
          WRITE(1,1150)'WARNING:  MAXIMUM ALLOWED V-CYCLES PERFORMED'
        ENDIF
      ENDIF
C
      CALL TIME (10,TDONE)
      WRITE(1,1200)'** Execution Time For Performing V-Cycles **',
     +              'Started:',TSTART,'Completed:',TDONE
C
C                       *** FORMAT STATEMENTS ***
C
 1000 FORMAT('1'/5X,A/'+',4X,7('_'),11X,A///10X,2(5X,A)/15X,28('='),
     +       5X,13('=')/29X,A,18X,1P,E13.6)
 1100 FORMAT(28X,I2,18X,1P,E13.6)
 1150 FORMAT(//15X,A/)
 1200 FORMAT(//15X,A/22X,A,3X,A/20X,A,3X,A/)
C
      RETURN
      END
C
C     ****************************************************************
C
      SUBROUTINE GSRLAX(U,F,H,GRDPTS,ALG)
C
C     PURPOSE:  This routine performs lexicographic Gauss-Seidel
C                 relaxation upon the finite difference equation
C                 associated with algorithm ALG.

C
      INTEGER     MAXGRD,GRDPTS,I,J,ALG
      PARAMETER   (MAXGRD=65)
      REAL        U(MAXGRD,MAXGRD),F(MAXGRD,MAXGRD)
      REAL        H,H2,H26,A
C
      IF (ALG.EQ.1) THEN
        H2 = H*H
        DO 10 J=2,GRDPTS-1
        DO 10 I=2,GRDPTS-1
          U(I,J) = (U(I-1,J)+U(I,J-1)+U(I+1,J)+U(I,J+1)-H2*F(I,J))/4.0
   10   CONTINUE
      ELSE
        H26 = 6.0*H*H
        DO 20 J=2,GRDPTS-1
        DO 20 I=2,GRDPTS-1
          A = 4.0*(U(I-1,J)+U(I,J-1)+U(I+1,J)+U(I,J+1)) - H26*F(I,J)
          U(I,J) = (A+U(I+1,J+1)+U(I-1,J+1)+U(I-1,J-1)+U(I+1,J-1))/20.0
   20   CONTINUE
      ENDIF
C
      RETURN
      END
C
C     ****************************************************************
C
      SUBROUTINE RESNRM(U,F,H,GRDPTS,RNORM,ALG)
C
C     PURPOSE:  This routine calculates the residual norm for the
C                finite difference equation associated with algorithm
C                ALG.
C
      INTEGER     MAXGRD,GRDPTS,I,J,ALG
      PARAMETER  (MAXGRD=65)
      REAL        H,H2,H26,LU,R,RNORM
      REAL        U(MAXGRD,MAXGRD),F(MAXGRD,MAXGRD)
C
      RNORM = 0.0
      H2 = H*H
C
      IF (ALG.EQ.1) THEN
        DO 10 J=2,GRDPTS-1
        DO 10 I=2,GRDPTS-1
          R = F(I,J)-(U(I-1,J)+U(I,J-1)+U(I+1,J)+U(I,J+1)-4.0*U(I,J))/H2
          RNORM = RNORM + R*R
   10   CONTINUE
      ELSE
        H26 = 6.0*H2
        DO 20 J=2,GRDPTS-1
        DO 20 I=2,GRDPTS-1
          LU = 4.0*(U(I-1,J)+U(I,J-1)+U(I+1,J)+U(I,J+1)) - 20.0*U(I,J)

          LU = LU + U(I+1,J+1)+U(I-1,J+1)+U(I-1,J-1)+U(I+1,J-1)
          R = F(I,J) - LU/(H26)
          RNORM = RNORM + R*R
   20   CONTINUE
      ENDIF
C
      RNORM = SQRT(RNORM*H2)
C
      RETURN
      END
C
C     ****************************************************************
C
      SUBROUTINE STOREW(U,F,GRDPTS,STARTU,STARTF,W)
```

```
C
C     PURPOSE:  Copies the 2-D arrays U and F into the 1-D array W.
C
      INTEGER     MAXGRD,STARTU,STARTF,DIFF,SUB,GRDPTS,I,J
      PARAMETER   (MAXGRD=65)
      REAL        U(MAXGRD,MAXGRD),F(MAXGRD,MAXGRD),W(*)
C
      DIFF = STARTF - STARTU
C
      DO 10 J=1,GRDPTS
      DO 10 I=1,GRDPTS
        SUB = STARTU + (I-1) + (J-1)*GRDPTS
        W(SUB) = U(I,J)
        W(SUB + DIFF) = F(I,J)
   10 CONTINUE
C
      RETURN
      END
C
C     ************************************************************
C
      SUBROUTINE F2CORS(U,F,HF,FGPTS,CGPTS,ALG)
C
C     PURPOSE:  Transfers the interior residual for the finite
C                 difference equation corresponding to algorithm ALG to
C                 the next coarser grid F by injection and initializes
C                 the next coarser grid U.
C
      INTEGER     MAXGRD,FGPTS,CGPTS,CI,CJ,FI,FJ,ALG
      PARAMETER   (MAXGRD=65)
      REAL        HF,HF2,HF26,LU,U(MAXGRD,MAXGRD),F(MAXGRD,MAXGRD)
C
C                 *** TRANSFER RESIDUAL TO COARSE GRID F ***
C
      HF2 = HF*HF
C

      IF (ALG.EQ.1) THEN
        DO 10 CJ=2,CGPTS-1
        DO 10 CI=2,CGPTS-1
          FJ = 2*CJ - 1
          FI = 2*CI - 1
          LU = U(FI-1,FJ)+U(FI,FJ-1)+U(FI+1,FJ)+U(FI,FJ+1)-4.0*U(FI,FJ)
          F(CI,CJ) = F(FI,FJ) - LU/HF2
   10   CONTINUE
      ELSE
        HF26 = 6.0*HF2
        DO 20 CJ=2,CGPTS-1
        DO 20 CI=2,CGPTS-1
          FJ = (2*CJ)-1
          FI = (2*CI)-1
          LU = 4.0*(U(FI-1,FJ) + U(FI,FJ-1) + U(FI+1,FJ) + U(FI,FJ+1))
          LU = LU+U(FI+1,FJ+1)+U(FI-1,FJ+1)+U(FI-1,FJ-1)+U(FI+1,FJ-1)
          F(CI,CJ) = F(FI,FJ) - (LU - 20.0*U(FI,FJ))/HF26
   20   CONTINUE
      ENDIF
C
```

```
C                    *** INITIALIZE COARSE GRID U ***
C
      DO 30 CJ=1,CGPTS
      DO 30 CI=1,CGPTS
        U(CI,CJ) = 0.0
   30 CONTINUE
C
      RETURN
      END
C
C     ************************************************************
C
      SUBROUTINE C2FINE(U,W,CGPTS,FGPTS,START)
C
C     PURPOSE:  Transfers the coarse grid correction (array U) to the
C                 next finer grid by bilinear interpolation and adds it
C                 to the previous solution on this finer grid (stored
C                 in array W)
C
      INTEGER     MAXGRD,CGPTS,FGPTS,START,SUB,CI,CJ,FI,FJ
      PARAMETER   (MAXGRD=65)
      REAL        AVG1,AVG2,U(MAXGRD,MAXGRD),W(*)
C
      W(START) = W(START) + U(1,1)
C
      DO 10 CI=2,CGPTS
        FI = 2*CI - 1
        SUB = START + FI - 1
        W(SUB) = W(SUB) + U(CI,1)
        W(SUB-1) = W(SUB-1) + 0.5*(U(CI-1,1) + U(CI,1))
   10 CONTINUE

C
      DO 20 CJ=2,CGPTS
        FJ = 2*CJ - 1
        AVG2 = 0.5*(U(1,CJ-1) + U(1,CJ))
        SUB = START + (FJ-1)*FGPTS
        W(SUB) = W(SUB) + U(1,CJ)
        W(SUB-FGPTS) = W(SUB-FGPTS) + AVG2
C
        DO 30 CI=2,CGPTS
          FI = 2*CI -1
          AVG1 = AVG2
          AVG2 = 0.5*(U(CI,CJ-1) + U(CI,CJ))
          SUB = START + (FI-1) + (FJ-1)*FGPTS
          W(SUB) = W(SUB) + U(CI,CJ)
          W(SUB-FGPTS) = W(SUB-FGPTS) + AVG2
          W(SUB-1) = W(SUB-1) + 0.5*(U(CI-1,CJ) + U(CI,CJ))
          W(SUB-FGPTS-1) = W(SUB-FGPTS-1) + 0.5*(AVG1 + AVG2)
   30   CONTINUE
C
   20 CONTINUE
C
      RETURN
      END
C
C     ************************************************************
```

```
C
      SUBROUTINE RETRVW(U,F,GRDPTS,STARTU,STARTF,W)
C
C     PURPOSE:  Retrieves info stored in the 1-D array W and copies
C                  it into the 2-D arrays U and F.
C
      INTEGER    MAXGRD,STARTU,STARTF,DIFF,SUB,GRDPTS,I,J
      PARAMETER  (MAXGRD=65)
      REAL       U(MAXGRD,MAXGRD),F(MAXGRD,MAXGRD),W(*)
C
      DIFF = STARTF - STARTU
C
      DO 10 J=1,GRDPTS
      DO 10 I=1,GRDPTS
        SUB = STARTU + (I-1) + (J-1)*GRDPTS
        U(I,J) = W(SUB)
        F(I,J) = W(SUB + DIFF)
   10 CONTINUE
C
      RETURN
      END
```