2005

An Evaluation Of A Multigrid Algorithm Error Spike Correction Method

Erika L. McMichael

An Evaluation Of A Multigrid Algorithm Error Spike Correction Method

by

Erika L. McMichael

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechanical Engineering

Approved By:

Dr. Amitabha Ghosh
Department of Mechanical Engineering

Dr. Ali Ogut
Department of Mechanical Engineering

Dr. P. Venkataraman
Department of Mechanical Engineering

Dr. Edward C. Hensel
Mechanical Engineering Department Head

Department of Mechanical Engineering
Rochester Institute of Technology
May 2005

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: An Evaluation Of A Multigrid Algorithm Error Spike Correction Method

I, Erika L. McMichael, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Erika L. McMichael

Erika L. McMichael

Date

Acknowledgments

I wish to thank my advisor, Dr. Ghosh, for his guidance and support in completing this project. I would also like to thank Dr. Ogut and Dr. Venkataraman for their assistance in reviewing this report.

Abstract

This study is an evaluation of a method of improving the multigrid process by correcting error spikes which are generated when moving from a coarser to finer level. The correction method was tested on nine one-dimensional problems governed by second order differential equations. Tests were performed with an accomodative, full approximation scheme, full multi-grid algorithm.

Results indicate that appropriate implementation of the correction can increase solution accuracy. Accuracy was increased in 75% of cases in which a single correction was applied to a point *in the central portion* of the grid. Single corrections performed on points *with error greater than the average error* were effective 86% of the time. Further study is required to determine a method of identifying this scenario.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multigrid is an iterative method which utilizes grids of multiple sizes to reduce error on multiple wavelengths. Many classical iterative methods (Gauss Seidel, etc.) efficiently reduce the error of high frequency components, "smoothing" the error. Multigrid uses a relaxation scheme to smooth the error on several grid levels of varying coarseness. In this way, multigrid can greatly reduce the time required to obtain a solution. Brandt introduced multigrid methods in his 1977 paper [1]. Since that time several books have been written on the topic, including [6], [5], [3], and [4].

In this study the multigrid process was examined graphically through plots of the residuals and error after each iteration in the multigrid cycle. From the examination of these plots it became apparent that spikes in the residual plots corresponded to spikes in the error plots. It was also noted that error spikes were prevalent when moving to finer grid levels. A correction method designed to mitigate these error spikes is described in this thesis. Testing showed that the removal of a single error spike could increase the accuracy of the solution while only minimally increasing the amount of computational work.

To evaluate the effectiveness of the correction method, a multigrid program was written and nine one-dimensional problems were tested. The multigrid program utilized an accomodative, full approximation scheme, full multi-grid algorithm (FAS FMG). Brandt provides a detailed description of an accomodative FAS FMG algorithm in [2]. One-dimensional problems were used for simplicity. For consistency, the problems evaluated were all second order differential equations. The differential equation solutions include five polynomials, three trigonometric functions, and one exponential function. The effects of several independent variables (such as grid level, location on grid, etc.) on solution accuracy and number of iterations required for convergence were recorded and are presented here.

A brief introduction to computational fluid dynamics (CFD) is given in Chapter 2 of this thesis. The basic multigrid process is outlined in Chapter 3, and the multigrid program used in this study is described in Chapter 4. Chapter 5 describes the proposed correction method and discusses the computational work required for its

implementation. The problems used in the study are presented in Chapter 6. Testing methods are discussed in Chapter 7, and results are reported in Chapter 8.

Chapter 2

Computational Fluid Dynamics

Many fluid flow problems cannot be solved analytically. These problems can be solved numerically using computation fluid dynamics.Computational fluid dynamics (CFD) is the term given to a variety of numerical mathematical techniques used to solve the equations that govern fluid flows and aerodynamics. CFD was developed in the 1970s after the introduction of high speed computers.

Because CFD solvers use numerical methods to obtain a solution, the region being analyzed must be divided into a grid with discrete grid points. A numerical solution is obtained at each of the grid points. To use CFD to solve a problem, the governing equations must be discretized so that the values of the variables are considered at the grid point only. If there are n grid points on the grid, n algebraic equations are developed by discretizing the governing differential equations. Several discretization schemes exist. In this thesis, differential equations were discretized using central differencing derived from the Taylor series expansion (5.1),(5.2). Other types of discretization include forward and backward differencing as well as forward, backward, or central differencing using a larger number of grid points (for example three and four point formulae).

When a problem is discretized it changes from a differential equation to a set of algebraic equations. If the problem is nonlinear it must be linearized before it can be solved using the methods described in this thesis. For simplicity, only linear problems are discussed in this thesis. If the set of equations is linear it can be represented in matrix form as:

$$Au = f \tag{2.1}$$

where, A is the known coefficient matrix, u is the vector of n dependent variables, and f is the vector of known values. Equation (2.1) can be solved using direct or iterative methods. Direct methods use a finite number of operations to obtain a solution and iterative methods use a variable number of iterations depending on the accuracy desired. The number of iterations is controlled by the stopping criteria, usually an average of the residuals (the difference between f and Au at each point). Multigrid methods and the Gauss Seidel iteration are iterative methods. The Gauss

Seidel iteration is given in equation (4.1).

Boundary conditions are specified conditions on the edges of the domain which aid in the solution of partial differential equations. Common boundary condition types are Dirichlet, Neumann, and Cauchy boundary conditions. Dirichlet boundary conditions specify the values of the function on the boundary. Neumann boundary conditions specify the derivative of the function on the boundary. Cauchy boundary conditions specify a weighted average of Dirichlet and Neumann boundary conditions. When Neumann boundary conditions are specified the boundary conditions are part of the computational domain (the computational domain is the area of the grid where the values of the dependent variable are unknown). Boundary grid points with Dirichlet boundary conditions are not included in the computational domain.

The following example illustrates the process of discretizing the partial differential equation (2.3) and expressing it in the matrix form $Au = f$ (2.6). The example applies the steady state heat conduction equation (Laplace equation) (2.2) to a plate with an unknown temperature distribution, known edge temperatures, and no heat generation. In (2.2) u is the temperature (the dependent variable) and the dimensions of the plate are $L \times L$. The temperature is constant along each edge of the plate. The temperatures on the edges of the plate are the boundary conditions and are designated B_1, B_2, B_3, B_4. For example, $u = B_1$ at $x = 0$ and $u = B_2$ at $y = L$. The example has Dirichlet boundary conditions (known boundary values). The 2-dimensional plate being analyzed is shown divided into a grid in Figure 2.1. The distance between grid points in the x and y directions are denoted by Δx and Δy respectively. In this case $\Delta x = \Delta y = \frac{L}{3}$. There are four points in the computational domain ($n = 4$) which are labeled u_1, u_2, u_3, and u_4. The grid coordinates are denoted by (i, j).

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{2.2}$$

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = 0 \tag{2.3}$$

$$u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i,j+1} + u_{i,j-1} = 0 \tag{2.4}$$

$$\begin{aligned}
&\text{Grid point } (\tfrac{L}{3}, \tfrac{2L}{3}): && u_3 + B_1 - 4u_1 + B_2 + u_2 = 0 \\
&\text{Grid point } (\tfrac{L}{3}, \tfrac{L}{3}): && u_4 + B_1 - 4u_2 + u_1 + B_4 = 0 \\
&\text{Grid point } (\tfrac{2L}{3}, \tfrac{2L}{3}): && B_3 + u_1 - 4u_3 + B_2 + u_4 = 0 \\
&\text{Grid point } (\tfrac{2L}{3}, \tfrac{L}{3}): && B_3 + u_2 - 4u_4 + u_3 + B_4 = 0
\end{aligned} \tag{2.5}$$

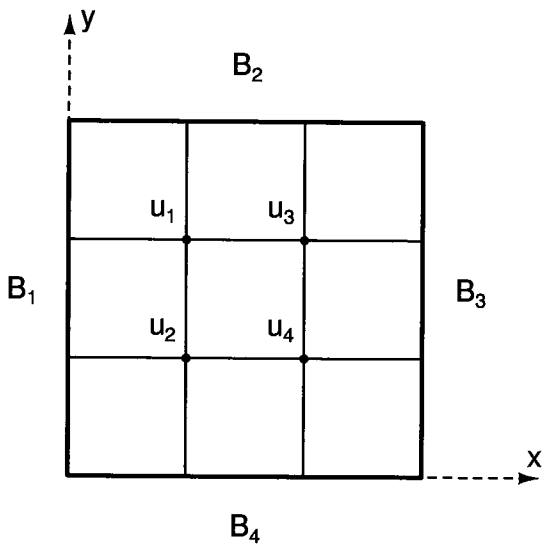


Figure 2.1: Example grid.

$$\begin{bmatrix} -4 & 1 & 1 & 0 \\ 1 & -4 & 0 & 1 \\ 1 & 0 & -4 & 1 \\ 0 & 1 & 1 & -4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} -B_1 - B_2 \\ -B_1 - B_4 \\ -B_3 - B_2 \\ -B_3 - B_4 \end{bmatrix} \tag{2.6}$$

Chapter 3

Multigrid

Multigrid methods are iterative methods used to solve partial differential equations. Mulitgrid methods use multiple grids of varying coarseness to speed up convergence. The two main components of multigrid methods are error smoothing and coarse grid correction.

Each grid used in solving the problem is identified by a grid level (for example the coarsest grid is referred to as level 1). On each grid level the error is "smoothed" with the one or more iterations of a relaxation scheme (e.g. Gauss-Seidel). The relaxation methods are efficient at reducing the error (4.9) of high frequency components. Figure 3.1 shows the effects of one and four Gauss-Seidel iterations on a one dimensional problem. The high frequency errors are quickly resolved giving the plot a smooth appearance.

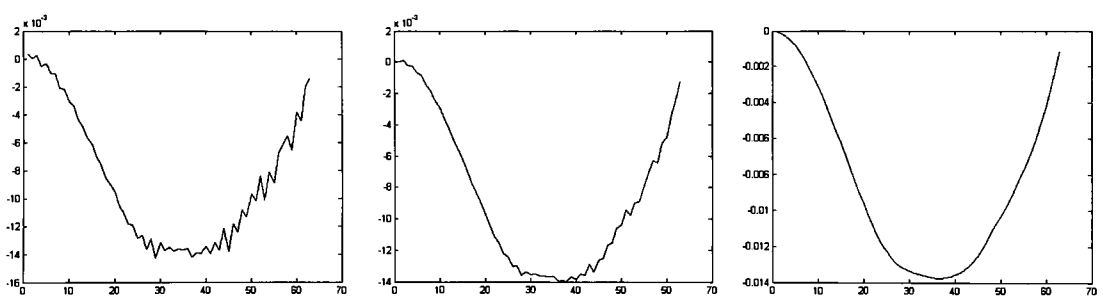


Figure 3.1: Error vs. Grid Point: After initial guess, After 1 GS iteration, After 4 GS iterations.

Once the error on the fine grid is smoothed it is transferred to a coarser grid. The error is smoothed on the coarse grids and the coarse grid approximations are used to correct the finest grid approximation. Applying the relaxation scheme to grids of varying coarseness results in efficient error reduction at multiple frequencies.

3.1 Basic Multigrid Process

This section describes the multigrid process in very general terms. A detailed description of the algorithm used in this study is given in the next chapter. The discretized equation to be solved is (2.1). On the k^{th} grid level the discretized equation is represented as:

$$A^k u^k = f^k \tag{3.1}$$

Restriction and prolongation operators are used to move between grid levels. Restriction operators are used to transfer to coarser levels and either directly inject or average fine grid values to obtain a coarse grid value. Prolongation operators use interpolation to transfer to finer grids. The restriction and prolongation operators are denoted by R and P respectively.

$$Ru^{k+1} = u^k \tag{3.2}$$

$$Pu^k = u^{k+1} \tag{3.3}$$

The first step in the multigrid process is presmoothing, the application of $v1$ iterations of an appropriate relaxation scheme. This is denoted by:

$$S(u^k, A^k, f^k, v_1) \tag{3.4}$$

The process then moves to a lower (coarser) grid level. On the lower grid level, f^k is modified and a relaxation scheme is implemented to approximate u^k (3.1). This coarse grid approximation is used to correct the finer grid approximation. The value of f^k and the meaning of u^k depend on the multigrid scheme (see Coarse Grid Processing, below). The final step of the process is postsmoothing, relaxation on the fine grid level. These steps can be performed multiple times on any number of grid levels.

3.2 Multigrid Algorithms

Starting Point

Full Multi-Grid (FMG) algorithms begin on the coarsest level and move up. The coarsest level solution may be obtained by relaxation or a direct method.[2]

Cycling algorithms begin with an approximation on the finest grid. The approximation can be trivial, such as $u = 0$, or the solution of a previous similar problem.[2]

Program Progression

An algorithm is described as accomodative or fixed. Accomodative algorithms use internal checks to determine when a switch will be made to a finer or coarser level. The checks are usually based on relative magnitudes of residuals.

Fixed algorithms have no internal checks and operate based on a predetermined flow. Fixed algorithms may execute more quickly than accomodative algorithms because they do not require the calculation of residuals after each iteration.

Coarse Grid Processing

A multigrid algorithm can use the Correction Scheme or the Full Approximation Scheme described in this section. In the Correction Scheme the long wavelength error of the fine grid approximation (u^{k+1}) is approximated on the coarse grid by u^k. On the coarse grid, the right hand side of equation (3.1) is given by (3.6), the restriction of the fine grid residual error (r^{k+1} (3.5)). To correct the fine grid approximation (u^{k+1}), u^k is prolongated and added to u^{k+1} (3.7).[2]

$$r^{k+1} = f^{k+1} - A^{k+1}u^{k+1} \tag{3.5}$$

$$f^k = Rr^{k+1} \tag{3.6}$$

$$u^{k+1}_{new} = u^{k+1}_{old} + Pu^k \tag{3.7}$$

In the Full Approximation Scheme (FAS) the coarse grid approximation (u^k) approximates the solution to the differential equation on level k. When transferring to a coarser level k, an initial value of u^k is the restriction of u^{k+1} (3.8). This initial value is used in the calculation of f^k (3.9). The fine grid is corrected with (3.10).[2]

$$u^k = Ru^{k+1} \tag{3.8}$$

$$f^k = A^k u^k + Rr^{k+1} \tag{3.9}$$

$$u^{k+1}_{new} = u^{k+1}_{old} + P(u^k - Ru^{k+1}_{old}) \tag{3.10}$$

The algorithm used is fully described by one item from each of the above three sections. The program used in this study is an accomodative, full approximation scheme, full multi-grid (FAS FMG) algorithm.

Chapter 4

Multigrid Program

A multigrid program written for this study is described in this section. This program provided a means of testing the correction method. The program uses an accomodative full approximation scheme full multi-grid (FAS FMG) algorithm. It is capable of solving one dimensional problems with Dirichlet boundary conditions. The program source code is in Appendix A.

4.1 Basic Elements

Smoothing Method

The Gauss-Seidel iteration is used for error smoothing on all grid levels. For the case of M equations the $n+1^{th}$ round of Gauss-Seidel iterations can be written as:

$$u_i^{n+1} = \frac{1}{a_{ii}} \left[f_i - \sum_{j=1}^{i-1} a_{ij} u_j^{n+1} - \sum_{j=i+1}^{M} a_{ij} u_j^n \right] \qquad \text{for} \quad i = 1 \quad \text{to} \quad M \tag{4.1}$$

Gauss-Seidel iterations are performed by the subroutine `gs.for`, see Appendix A.3.

Prolongation/Restriction Operators

The restriction and prolongation operators used in grid transfer are defined by the following equations:

$$Ru_j = \frac{1}{4} u_{2j-1} + \frac{1}{2} u_{2j} + \frac{1}{4} u_{2j+1} \qquad \text{Restriction} \tag{4.2}$$

$$Pu_{2j} = u_j \qquad Pu_{2j+1} = \frac{1}{2}(u_j + u_{j+1}) \qquad \text{Prolongation} \tag{4.3}$$

These are taken from [6]. Prolongation and restriction matrices are created in the subroutine `prores.for`, see Appendix A.4.

Coefficient Matrix

The finest level coefficient matrix A from (2.1) is provided by the user. To perform smoothing operations on all grid levels, an A matrix must be obtained for each level. Here the Galerkin coarse grid approximation is used:

$$A^k = RA^{k+1}P \tag{4.4}$$

4.2 User Inputs

Seven items must be supplied by the user for the program to operate:

1. n_m The number of grid points in the computational domain on the finest level
2. The exact value of the center point of the computational domain
3. *domain* The length of the grid $(x_{max} - x_{min})$
4. Dirichlet boundary conditions
5. ε_m Finest level convergence criteria
6. A The coefficient matrix in (2.1)
7. f The vector of known values in (2.1)

4.3 Structure

The number of grid levels m used in computation is calculated from n, the size of the computational domain (number of grid points) on the finest level. For the program to function n must be able to satisfy (4.5) where m is an integer. The size of the computational domain on levels 1-6 is shown in Table 4.1. Level 1 is the coarsest level.

$$n = 2^m - 1 \tag{4.5}$$

Level	1	2	3	4	5	6
n	1	3	7	15	31	63

Table 4.1: Size of computational domain per grid level.

The program begins computation on the coarsest level 1 and works up to the finest level m. Level 1 consists of only three grid points. The exact values of these three points are supplied. The two end points are the Dirichlet boundary conditions. The center point is provided by the user. It is assumed that this point can be obtained through iterative or direct methods. Because the values of u on level 1 (u^1) are known this level is not revisited.

The prolongation of u^1 gives an initial estimate for u^2. One Gauss-Seidel (GS) iteration is then performed on u^2. Convergence on the current level is then checked as described in section 4.4. If convergence has been obtained on level 2 the program moves up to level 3 (3.10). If not, another GS iteration is performed on u^2.

After a transfer is made to any level, one Gauss-Seidel iteration is performed and both convergence and convergence rate are checked as described in section 4.4. If the convergence on the current level has been obtained the program moves to a finer level (3.10). Otherwise if the relaxation convergence rate is acceptable, another GS iteration is performed on the current level. If the relaxation convergence rate is slow the program moves to a coarser level using (3.8) - (3.9). This procedure is repeated until the convergence criteria are met on the finest grid level.

4.4 Convergence Testing

After each GS iteration, the norm of the residuals e_k (4.6) is computed and convergence and convergence rate are tested. The convergence criteria for the current operation level is denoted by ε_k. Convergence has been obtained on the current level if $e_k \leq \varepsilon_k$. Finest level convergence ε_m is provided by the user. Initial values of ε_k for all other levels are obtained from (4.7) unless this calculated ε_k is smaller than ε_m, in which case ε_k is set equal to ε_m.

$$e_k = \frac{[\sum_{i=1}^{n_k} x_i{}^2]^{1/2}}{n_k} \tag{4.6}$$

$$\varepsilon_k = \left[\frac{domain}{n_k + 1}\right]^2 \tag{4.7}$$

Equation 4.7 sets ε_k equal to the distance between grid points squared. Each time a transfer is made to a coarser level ε_k is updated:

$$\varepsilon_k = 0.2 e_{k+1} \tag{4.8}$$

To correct the fine grid solution, the coarse grid residuals should be smaller than the fine grid residuals. [2]

If $e_k > \varepsilon_k$, convergence rate is tested. The convergence rate is satisfactory if $e_k \leq \eta \bar{e}_k$, where $\eta = 0.5$ and $\bar{e}_k$ holds previous values of e_k. If the convergence rate is satisfactory $\bar{e}_k$ is set equal to e_k ($\bar{e}_k = e_k$) before another GS iteration is performed. When a transfer is made to a coarser level, $\bar{e}_k$ is set equal to the norm of the residuals immediately after the transfer is made (before any GS iterations are performed). The value of $\bar{e}_k$ is not changed immediately after a fine grid transfer because testing showed that this decreased program efficiency.

If $e_k > \eta \bar{e}_k$ the convergence rate is slow. This indicates that the error is smooth and should be approximated on a coarser grid.

4.5 Program Outputs

After each Gauss Seidel iteration, the error and residuals of each grid point are written to a file. The error referred to in this thesis is obtained by subtracting the current values from the exact solution (4.9). Residuals are calculated from (4.10).

$$error = u_{exact} - u \tag{4.9}$$

$$residuals = Au - f \tag{4.10}$$

4.6 Program Efficiency

This program can solve equations much more efficiently than a solver which uses standard Gauss Seidel iterative methods. Table 4.2 shows the number of iterations required by the program outlined above and a Gauss Seidel program to reach the indicated convergence. The number of iterations required for the multigrid program are weighted to account for iterations on different grid levels. The number indicated is the equivalent number of finest level iterations. A single iteration on level k requires the same amount of work as $2^{-(m-k)}$ iterations on level m (the finest level, here level 6). The Gauss Seidel program is executed on grid level 6 and begins with an initial guess of $u = 0$. Convergence is reached when the residual norm (4.6) is less than or equal to the convergence criteria.

Problem	Gauss Seidel Iterations	Multigrid Iterations	Convergence Criteria
1	2522	16.06	1.0E-06
2	1519	12.38	1.0E-06
3	5925	16.94	1.0E-07
4	1756	12.81	1.0E-06
5	2456	20.44	5.0E-08
6	4351	15.69	5.0E-07
7	2748	15.56	7.0E-06
8	2666	14.94	5.0E-07
9	1509	13.00	1.0E-06

Table 4.2: Number of iterations required for convergence.

Chapter 5

Correction Method

5.1 Description of Correction

The proposed correction improves efficiency by adjusting the values of grid points where large residual spikes occur. The adjustments are made immediately after prolongation of the approximate solution when moving to a finer level. After the new fine level solution is obtained the residuals are calculated. The value of the point at the location of the absolute maximum residual is changed. The new value is obtained from the discretized equation (5.1), where u_i is the point that is corrected.:

$$u_i = \frac{1}{a_{ii}} \left[f_i - \sum_{j=1}^{i-1} a_{ij} u_j - \sum_{j=i+1}^{M} a_{ij} u_j \right] \qquad \text{for} \quad i = 1 \quad \text{to} \quad M \tag{5.1}$$

Only the spike which corresponds to the maximum residual is corrected. This can effectively increase accuracy and requires a minimal amount of additional work. This type of correction works well when moving to a fine level because the error is typically not smooth after prolongation.

This method is effective in reducing error because residual spikes correspond to error spikes. Figure 5.1 shows the plots of corresponding residual and error curves (Problem 5, Iteration 14). The largest residual is at point five, Figure 5.2 shows the error plot after this point has been replaced.

The new value of point 5 is more accurate than the original. After one GS iteration the effect of the correction is still apparent (Figure 5.3).

When implemented at appropriate times this correction method can reduce the error of the solution and reduce the number of iterations required to achieve residual convergence. The code used for the correction is in Appendix A.2. This code was inserted into the multigrid program code.

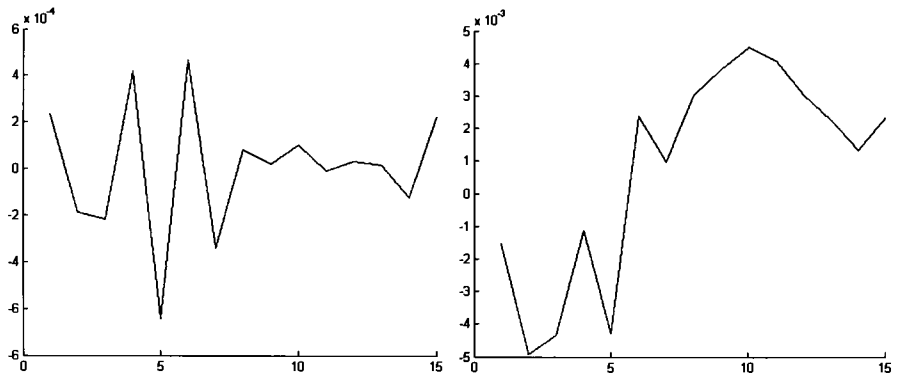


Figure 5.1: Problem 5, Iteration 14. Residual vs. Grid Point (left) and Error vs. Grid Point (right).

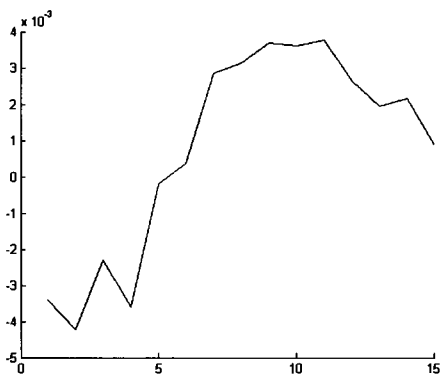


Figure 5.2: Problem 5, Iteration 14. Error vs. Grid Point after replacement of point 5.

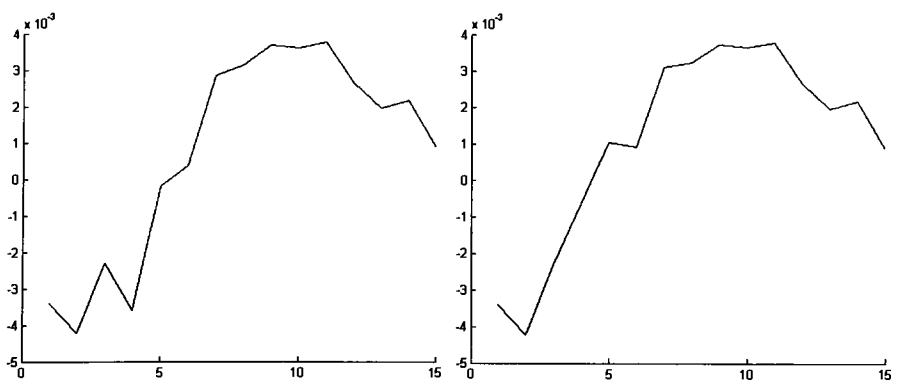


Figure 5.3: Error vs. Grid Point one GS iteration later without (left) and with (right) correction.

5.2 Correction Processing Time

The amount of time required to perform a single correction was compared to the amount of time required to perform a single Gauss Seidel iteration. If residuals are calculated for all grid points, one correction takes approximately the same amount of time to implement as one Gauss Seidel iteration on the same level. Table 5.1 shows the amount of work required for a correction as a fraction of a Gauss Seidel iteration.

One correction requires the calculation of residuals, a comparison of the residuals (to determine the maximum residual), and the calculation of the new value of the corrected point. Two programs were written to compare the processing time of a Gauss Seidel iteration to a correction, see Appendix B. Each program was run four times, each time with a different number of grid point values. The level number which these corresponded to is shown in Table 5.1.

The running time required for each program was recorded and is presented in Table 5.1. In the table, 'GS Time' is the running time of the Gauss Seidel program. The correction program determines the max residual (includes the calculation of residuals and a comparison of the residuals). This running time is 'Max Residual Time' in Table 5.1. 'Residual/GS' is the ratio of 'Max Residual Time' to 'GS Time'. 'New Value Calculation' accounts for the calculation of the new value of the corrected point. The amount of work required for the calculation of the new value is equal to a fraction of a Gauss Seidel iteration, $1/n_k$ (n_k is the number of grid points on level k). 'Total Correction/GS' is the sum of Residual/GS and New Value Calculation, it is the amount of work required for a correction expressed as a fraction of a Gauss Seidel iteration.

Level	3	4	5	6
GS Time (s)	10.77	18.99	35.03	67.21
Max Residual Time (s)	8.54	16.01	32.02	60.76
Residual/GS	0.79	0.84	0.91	0.90
Calculation of New Value	0.143	0.067	0.032	0.016
Total Correction/GS	0.935	0.910	0.946	0.920

Table 5.1: Work required for a correction compared to a Gauss Seidel iteration.

Table 5.1 indicates that a single correction requires approximately the same amount of work as one Gauss Seidel iteration on the same level. This means that a correction performed on level 3 will increase the total amount of work required to solve a problem by only 0.125 equivalent level 6 GS iterations. Between 12 and 21 equivalent level 6 GS iterations were required for the solution of each problem (Table 4.2). Thus, a level 3 correction would increase computational work by 1% or less.

Chapter 6

Problems

The correction was tested on the nine problems shown in Table 6.1 below. All are second order differential equations with Dirichlet boundary conditions.

Problem	Equation	Classification
1	$-2u'' + 3u' + 6u = 6x^5 + 27x^4 + 14x^3 - 15x^2 - 66x + 35$	hyperbolic
2	$u'' - 4u' + 2u = 2x^5 - 16x^4 - 22x^3 + 86x^2 - 36x - 2$	hyperbolic
3	$u'' + 2u' + 3u = 2\sin x + 2\cos x$	elliptic
4	$u'' - 3u + u = -8\sin 3x - 9\cos 3x$	hyperbolic
5	$u'' + 5u' - u = -3x^2 + 23x + 45$	hyperbolic
6	$2u'' - 2u' + 3u = 3x^3 - 3x^2 + 2x + 38$	elliptic
7	$5u'' + 4u' + 5u = -12x^2\sin x + 30\cos x - 60x\sin x + 24x\cos x$	elliptic
8	$u'' + 5u' + 2u = 24e^x - 8e^x$	hyperbolic
9	$u'' + 2u' - 3u = -12x^4 + 50x^3 + 15x^2 - 64x + 23$	hyperbolic

Problem	Solution	Domain	Lower BC	Upper BC
1	$u = x^5 + 2x^4 + 5x^3 - 2x^2 + x + 4$	$0 - 1.0$	4.0	11.0
2	$u = x^5 + 2x^4 - 5x^3 + x^2 + x$	$0 - 1.0$	0.0	0.0
3	$u = \sin x$	$0 - \pi/2$	0.0	1.0
4	$u = \sin 3x$	$0 - \pi/2$	0.0	1.0
5	$u = 3x^2 + 7x - 4$	$0 - 1.0$	-4.0	6.0
6	$u = x^3 + x^2 - 2x + 10$	$0 - 1.0$	10.0	10.0
7	$u = 3x^2\cos x$	$0 - \pi/2$	0.0	0.0
8	$u = 3e^x + 4e^{-x}$	$0 - 1.0$	7.0	9.6
9	$u = 4x^4 - 6x^3 - x^2 + 8x - 3$	$0 - 1.0$	-3.0	2.0

Table 6.1: Problems

The problems were discretized using three point central differencing (6.1), (6.2). The domain of each problem was split into 64 elements of equal size h, yielding a computational domain of $n = 63$ grid points. Six grid levels ($m = 6$) were used in multigrid

computations (refer to (4.5)).

$$\left.\frac{du}{dx}\right|_i = \frac{1}{2h}(u_{i+1} - u_{i-1}) \tag{6.1}$$

$$\left.\frac{d^2u}{dx^2}\right|_i = \frac{1}{h^2}(u_{i-1} - 2u_i + u_{i+1}) \tag{6.2}$$

Chapter 7

Methods

To test the effectiveness of the correction, each problem was run several times. These runs are referred to as trials. The effects of the correction on solution accuracy and the number of iterations required for convergence were measured. The correction was implemented up to three times per trial.

As stated in Chapter 5, corrections were made immediately after prolongation when moving to a finer grid level. In addition corrections were only made after a Gauss Seidel iteration had been performed on each level. The correction is not effective when applied to the initial iterations in a problem because the error at each grid point is high. It is most effective when there is a large amount of localized error.

A trial was performed for each correction possible based on the above criteria (indicated in yellow in Figure 7.1). Some additional test runs with multiple corrections per run were also performed.

The accomodative algorithm did not control the flow of the correction trials. Instead the structure of the multigrid runs without corrections (labeled trial 0) was preserved in each of the correction trials. The structure of the runs can have a significant impact on the solution, and any variance can make it difficult to evaluate the effects of the correction. While the flow, or pattern, of the runs was preserved, the number of iterations on each level was varied in accordance with changes in the residual norm.

A generic structure is shown in Figure 7.1. It represents the basic flow that the accomodative algorithm used in this study produced. The structure of each of the trials performed in this study was a variation of that shown in Figure 7.1, with a varying number of iterations performed on each of the points in the figure. Points at which a correction could be applied are colored yellow. Plots of iteration vs. grid level for each multigrid run without corrections (trial 0 for each problem) are presented in Appendix C. From the plot of Problem 1, Trial 0 it can be seen that the trial began on level 1, then moved up with one Gauss Seidel iteration performed on levels 2,3, and 4 and two GS iterations performed on level 5 and level 6, and so on.

The effects of the correction were measured in terms of the number of iterations

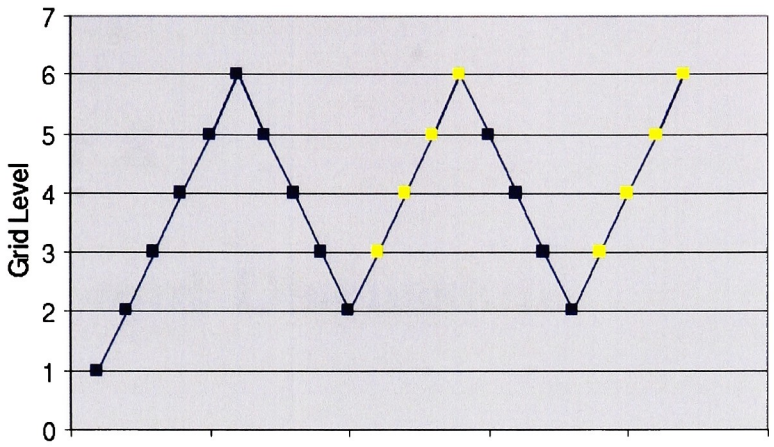


Figure 7.1: Generic multigrid structure.

required and the accuracy of the solution. In Chapter 8 percent change in error and percent change in iterations are referred to. These are the change in number of iterations and the change in error with respect to the trial without corrections (trial 0).

$$\%\Delta Error = \frac{|error| - |error_{trial0}|}{|error_{trial0}|} * 100 \tag{7.1}$$

$$\%\Delta Iterations = \frac{iterations - iterations_{trial0}}{iterations_{trial0}} * 100 \tag{7.2}$$

Increased solution accuracy is indicated by a negative change in error. Similarly, a negative change in iterations indicates a decrease in the number of iterations required for convergence.

Chapter 8

Results and Discussion

Several variables influenced the effectiveness of the correction. These included the nature of the corrected error spike, the grid level that the correction was implemented on, the time it was implemented, and the grid point that was corrected. Error was decreased in 68% of trials with one correction. The majority of the results summarized below are of trials with one correction only. The results of multiple correction trials are discussed only in section 8.7. Specific results of each trial are presented in Appendix E.

8.1 Error Spike Direction

The direction of the error spike that was corrected had a significant effect on whether the correction increased or decreased the error of the final solution. If the error of the corrected point was greater than the error of the two points adjacent to it, correcting it generally decreased the error of the solution (Figure 8.1). In this document, this is referred to as a "correct direction" spike because in the graphical view, the error spike appears to be "pointing" in the desirable direction. In some cases the error of the corrected point was negative and the error of the two surrounding points was positive, or vise versa. This situation is labeled "partially correct" and is shown in Figure 8.2. Corrections on partially correct spikes were not as effective as those made on correct spikes.

In the case of an "incorrect" direction spike, the error of the corrected point is smaller than the error of the adjacent points. Correcting an incorrect direction spike does not always yield positive results because the correction increases the error of the corrected point. Figures 8.1 and 8.2 show examples of error spikes in the "correct", "partially correct", and "incorrect" directions, as defined in this paper. Overall 58% of all spikes were in the correct direction, 28% in the incorrect direction, and 14% of spikes were partially in the correct direction.

Tables 8.1 and 8.2 show the effect of the direction of the corrected spike on the final

error for trials with a single correction. Corrections applied to spikes in the correct direction produced a decrease in solution error 86% of the time. In the trials in which corrections of correct direction spikes increased the solution error, it was increased an average of only 0.8%. Corrections made on spikes in the incorrect and partially correct directions caused an increase in solution error the majority of the time.

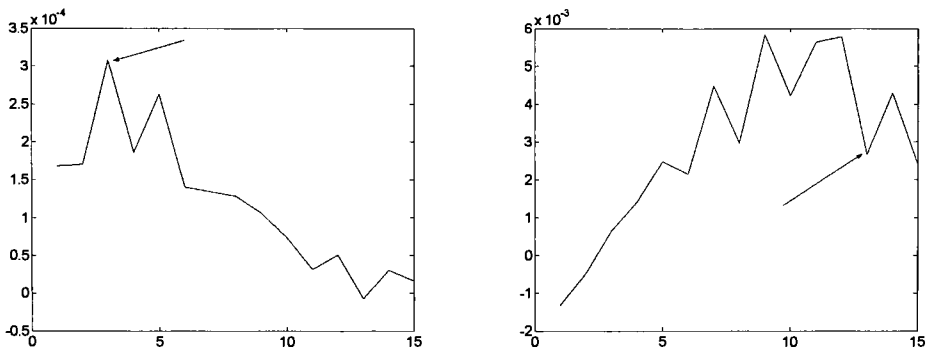


Figure 8.1: Error vs. Grid Point, "Correct" spike (left) and "Incorrect" spike (right).

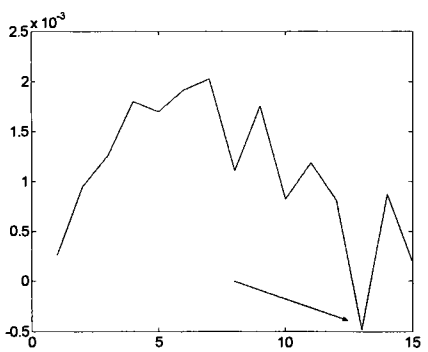


Figure 8.2: Error vs. Grid Point, "Partially Correct" spike.

It is difficult to determine if an error spike is in the correct, partially correct, or incorrect direction when the exact solution is not known. The difficulty lies in determining whether the error of the points adjacent to the spike have a positive or negative margin. The direction of the spike (pointing up or down in the plot) can be determined from the residuals. If the diagonal of the A matrix is a negative value, the residual spike and error spike will point in the same direction (residual and error calculated from (4.10) and (4.9)). If the diagonal of the A matrix is a positive value, the error spike will point in the opposite direction of the residual spike (Figure 8.3).

	Error	
Direction	Decreased	Increased
Correct	86%	14%
Incorrect	40%	60%
Partially Correct	44%	56%
Overall	68%	32%

Table 8.1: Percentage of trials error was decreased or increased per error spike direction.

	Error	
Direction	Decreased	Increased
Correct	5.1	0.8
Incorrect	2.4	1.9
Partially Correct	16.7	17.1
Total	6.4	7.3

Table 8.2: Average percentage change in error per error spike direction.

Unfortunately, a method of determining whether the surrounding error has a negative or positive margin has not been devised. Without this information, knowledge of the direction of the spike is not useful.

8.2 Grid Level

Corrections on coarser grid levels proved to be more effective than corrections on finer grid levels. Coarse level corrections correct error on a larger scale than finer level corrections and therefore have a greater influence on the accuracy of the final solution. This is illustrated in Table 8.3 and Figure 8.6. Table 8.3 shows the average of the absolute value of the percentage change in final error for trials with a single correction on the specified grid level.

Figure 8.6 shows the change in error produced by each trial with a single correction and indicates which corrections yielded a decreased number of total iterations. Each data point is the result of one trial. The level that each correction was implemented on is indicated by the color of the data point.

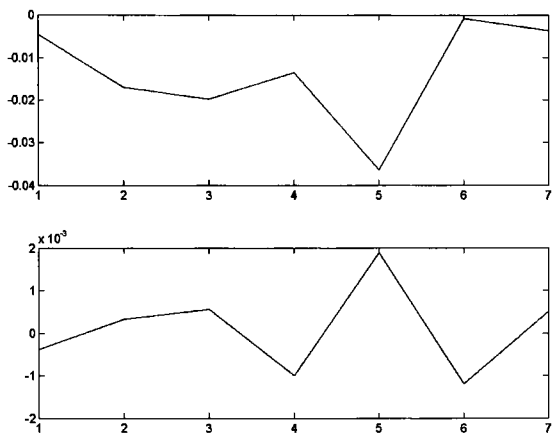


Figure 8.3: Error plot (top) and residual plot (bottom) for Problem 1, Trial 1 (level 3).

Level	Average | % Δ Error |
3	15.9
4	7.5
5	1.4
6	2.8

Table 8.3: Average change in error per grid level.

8.3 Point in Multigrid Cycle

The point in the multigrid cycle that a correction is made can also effect the accuracy of the solution. Corrections made earlier in the cycle have a greater impact on solution accuracy than those made later in the cycle. For example, if a problem requires 36 iterations for convergence, a correction made on the 15th iteration will generally have a greater effect on accuracy than a correction made on the 34th iteration. Corrections made near the end of the cycle produced smaller errors when error increased.

Table 8.4 and Figure 8.5 illustrate this. In the table and figure the location of the corrected iteration in the multigrid cycle is given as a percentage of the total number of iterations. Figure 8.5 shows the results of all of the single correction trials (in the same way that Figure 8.6 does). One data point shows the change in error produced by each trial.

In Table 8.4 the location of the correction in the cycle is reported in three segments. The first segment (20-60%) represents the first set of corrections made on any grid level as illustrated in Figure 8.4. The final set of corrections is made in the last segment (75-100%) (Fig. 8.4). If three sets of corrections were possible, the second set was made in the middle segment (60-75%) (Fig. 8.4). Problem 5 was the only problem in which corrections were made in the 60-75% range of the cycle (Fig. 8.5).

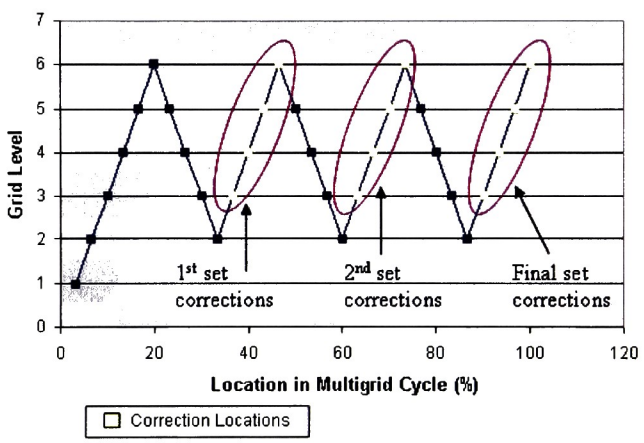


Figure 8.4: Generic multigrid structure, grid level vs. point in cycle.

	Location of Correction In Multigrid Cycle					
	20-60%		60-75%		75-100%	
Level	Dec. Error	Inc. Error	Dec. Error	Inc. Error	Dec. Error	Inc. Error
3	-15.4	43.5	-0.3	—	-11.4	11.9
4	-13.9	8.6	-0.3	—	-3.0	9.5
5	-1.2	1.1	-5.6	—	-1.3	0.6
6	-5.8	0.8	-19.7	—	-0.1	0.3
All	-9.08	13.51	-6.45	—	-3.95	5.57

Table 8.4: Average percentage change in error per location in multigrid cycle.

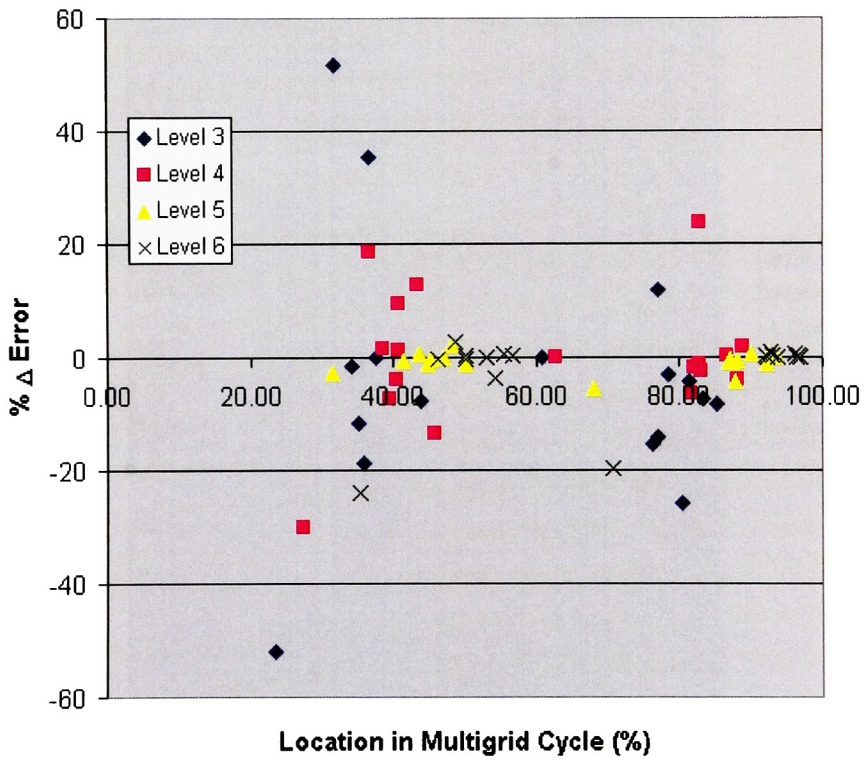


Figure 8.5: Change in error vs. point in multigrid cycle.

8.4 Grid Location

Corrections on points located in the center of the grid were more effective than corrections on points near the grid boundaries. In the following discussion grid point locations are expressed as a percentage of the computational domain of the problem. It should be noted that Dirichlet boundary conditions were used in all of the trials.

Corrections on points located in the center of the grid had a greater impact on solution accuracy than those performed within a distance of 10% (of the grid length) of the boundaries. Figure 8.6 shows the percentage change in solution error vs. the location of corrected point as a percentage of the length of the grid. The percentage change in error for all single correction trials is plotted in Figure 8.6. Each of the plotted points is the result of one trial. On finer grids errors near the edge of the grid are reduced by the boundary conditions, and corrections near the grid edges are not as effective. Corrections implemented in the center of the grid also decreased error more frequently than corrections near the boundary conditions (Table 8.5).

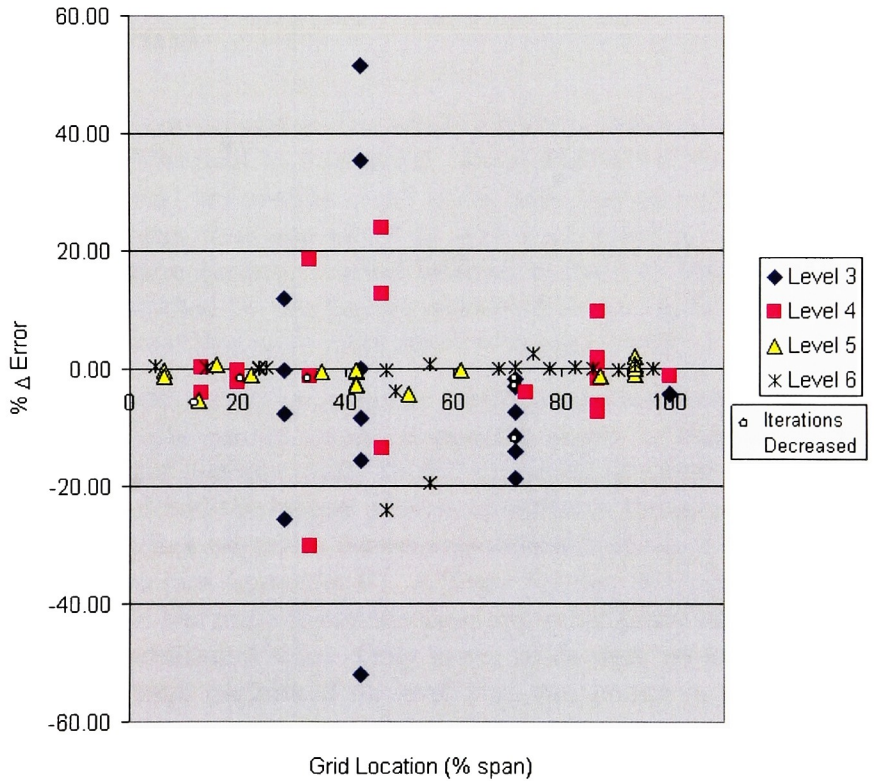


Figure 8.6: Change in final error vs. correction location and level.

	Error of Final Solution	
Distance from Boundary (% grid length)	Decreased	Increased
0-10%	8	5
10-25%	12	9
25-50%	30	10

Table 8.5: Number of trials in which final error was increased or decreased vs. correction location.

8.5 Residual Magnitude

There was no correlation between the size of the residual spike and the percentage change in final error. Figure 8.7 shows the change in the error of the solution as a function of the residual/norm (norm refers to the residual norm) of the corrected point.

8.6 Problems

The effect of single corrections on the error of each problem is shown in Figure 8.8. This plot is nearly identical to Figure 8.6. As in Figure 8.6, each data point is the result of one trial and the results of all trials with one correction are plotted. In Figure 8.8 the problem that was solved in each trial is indicated. The effect of the corrections on solution accuracy varied between problems. Most of the differences can largely be attributed to the factors discussed above (spike direction, location, and grid level). Corrections were most effective in problem 5.

The number of single correction trials in which corrections were made on correct, incorrect, and partially correct direction spikes is shown in Figure 8.9. In problem 6, 100% of the spikes corresponding to the maximum residual were in the correct direction. Problem 8 had the largest number of spikes in the incorrect direction. The number of trials performed varies between problems because of the variation in the structure of the runs (see Appendix C). A larger number of trials were performed on problem 5 because it has more iterations than any other problem and moves between the grid levels an additional time. Only seven trials were performed on problem 2 because iterations were performed on level 2 at two points in the cycle instead of three.

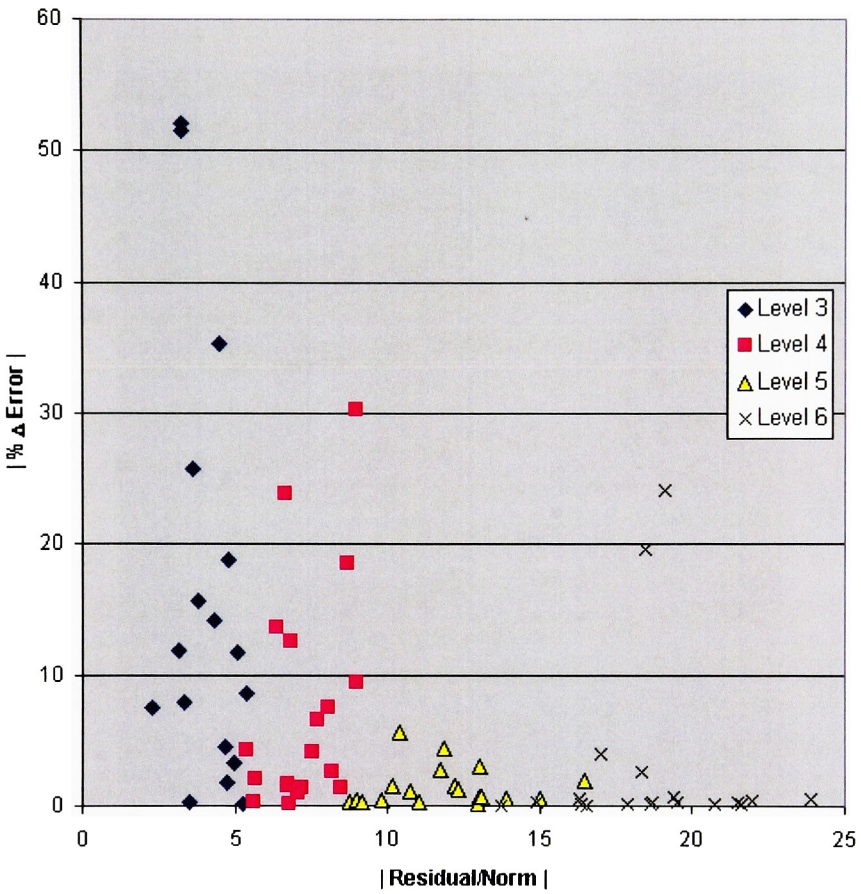


Figure 8.7: Change in final error vs. residual/residual norm.

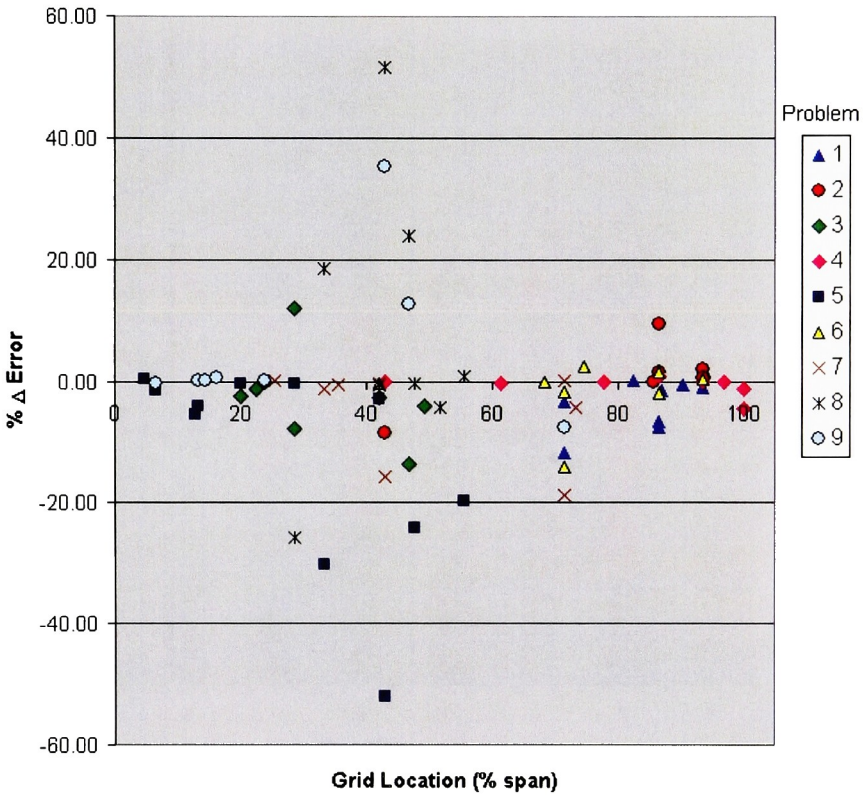


Figure 8.8: Change in final error vs. correction location and problem.

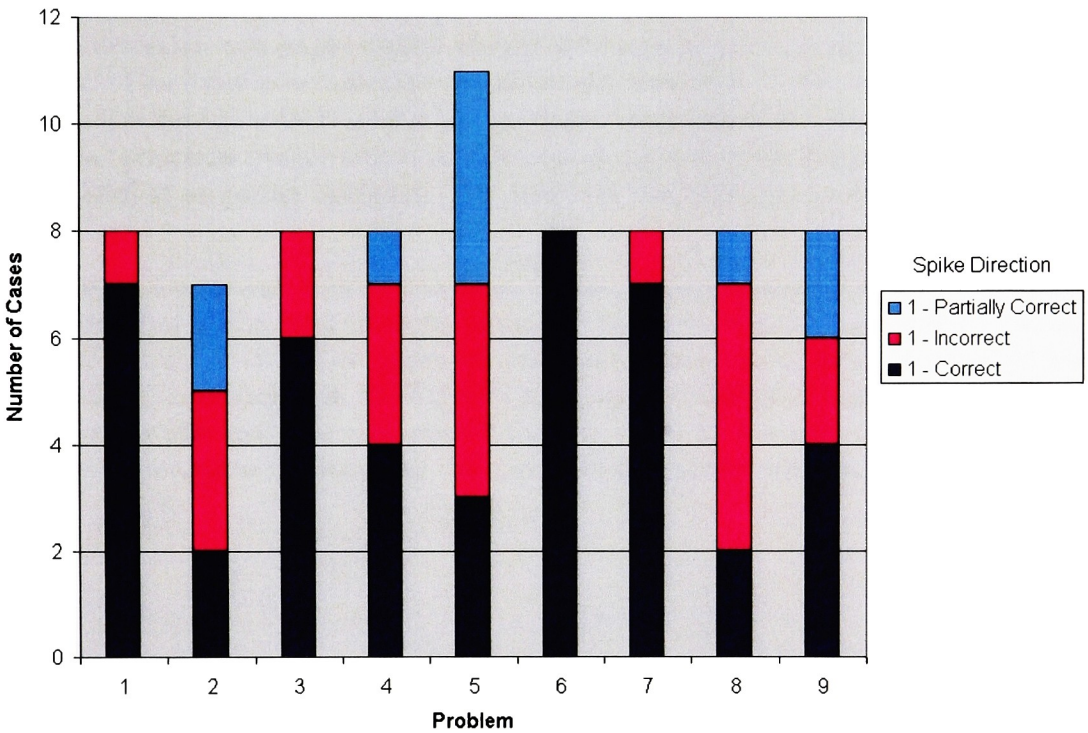


Figure 8.9: Direction of error spikes per problem.

8.7 Multiple Corrections

The results presented in the previous sections are the results of trials with a single correction. Twenty-six trials with multiple corrections were performed, 21 with two corrections, and 5 with three corrections. Appendix D shows the results of these trials. The percent change in error (7.1) and the percent change in number of iterations (7.2) are reported. In some cases the number of iterations did not change and therefore the percent change in number of iterations does not show up in the plot.

The graphs show the effects of the multiple correction trials and the corresponding single correction trials (performed at the same points in the cycle). The iteration that a correction was implemented after is indicated in parentheses next to the trial number. The iterations indicated in the single correction trials are not always the same as the iterations indicated in the multiple correction trial. This is because some multiple correction trials resulted in a decreased number of iterations and a move to a higher level at an earlier iteration. The details of the trials can be found in Appendix E.

In 12 trials the change in error produced by the multiple correction trial was within 6% of the sum of the error changes produced by the corresponding single correction trials. Six trials produced an improvement in solution accuracy over the combination of the single corrections. In 23 of the 26 trials the the change in number of iterations produced by the multiple correction trial was equal to the sum of the changes in number of iterations produced by the corresponding single correction trials.

Chapter 9

Conclusions

This study is an initial evaluation performed to assess the viability of the correction method. The results indicate that it has the capacity to improve solution accuracy and potentially increase solver efficiency for a problem solved with multigrid methods. The correction can also cause a decrease in solution accuracy if it applied at an improper time.

The most important factor in determining whether a correction will produce positive results is the direction of the corrected error spike. Almost all of the corrections performed on "correct direction" error spikes increased solution accuracy. Unfortunately it is difficult to determine whether or not the spike is in the "correct" direction. This would be possible if a method was devised to determine whether the error had a positive or negative margin.

There was also a correlation between the location of the correction on the grid and correction effectiveness. Error was decreased in 75% of single correction trials in which the corrected point was in the central portion of the grid.

In most cases implementation of the correction did not affect the number of iterations required for convergence. In these cases the amount of computational work was increased by an amount approximately equal to that required for one Gauss Seidel iteration. It is up to the end user to decide whether the accuracy improvements produced in these cases are worth this extra computation time.

Maximum efficiency is achieved when the correction is implemented on coarser grid levels. Corrections made on coarser grid levels have a greater effect on solution accuracy and require the computation of fewer residuals. The efficiency of a fine level correction could be improved by calculating the residuals of points located near the center of the grid only. Multiple corrections can be as efficient as single corrections, and can be used to further increase accuracy.

Due to the limited number of problems tested in this study, it is difficult to determine if specific types of problems are better suited to this correction method. Future studies

should examine a more diverse set of problems, including two and three dimensional problems.

Bibliography

[1] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, (31):333–390, 1977.

[2] Achi Brandt and Nathan Dinar. Multigrid solutions to elliptic flow problems. In Seymore V. Parter, editor, *Numerical Methods for Partial Differential Equations*, pages 53–147. Academic Press, 1979.

[3] W.L. Briggs. *A Multigrid Tutorial.* SIAM Books, Philadelphia, 1987.

[4] Wolfgang Hackbusch. *Multi-Grid Methods and Applications.* Springer Verlag, Berlin, 1985.

[5] U. Trottenburg, C.W. Oosterlee, and A. Schuller. *Multigrid.* Academic Press, 2001.

[6] Pieter Wesseling. *An Introduction To Multigrid Methods.* John Wiley & Sons, 1992.

Appendix A

Multigrid Program - Fortran 90 Code

A.1 Multigrid - main.for

```
program main

integer ::ai,aifine,aicoarse,halfai,aj
integer ::aicoarsea,alocation
integer ::ajcoarse1,ajfine1,ajfine2
integer ::v, size,digits
integer ::coarse, rn,divs
integer ::k,l,m,div2,aim,index,lenname
integer ::iteration !iteration number
character(len=80) ::name
character(len=80) ::name2
character(len=80) ::name3
character(len=80) ::n4
character(len=80) ::name5
character(len=80) ::name6
character(len=80) ::name8
character(len=10) ::chariteration
character(len=100) ::errorname
character(len=80) ::indexfile
real ::error,convrate,scriptem,avgerror
real ::proresidual,resresidual
real ::n,p
real ::normf,sumgs,weight
real ::sumfdiff
real ::bc1,bc2,bcplace,level1
real ::scripteinit
real ::xrange
real,dimension(:,:),allocatable ::asizes
real,dimension(:,:),allocatable ::a
real,dimension(:,:),allocatable ::x
real,dimension(:,:),allocatable ::f
real,dimension(:,:),allocatable ::ffine
real,dimension(:,:),allocatable ::fres
real,dimension(:,:),allocatable ::master
real,dimension(:,:),allocatable ::pro
real,dimension(:,:),allocatable ::res
real,dimension(:,:),allocatable ::e
real,dimension(:,:),allocatable ::olde
real,dimension(:,:),allocatable ::ebar
real,dimension(:,:),allocatable ::scripte
real,dimension(:,:),allocatable ::exactx
real,dimension(:,:),allocatable ::residual
```

```
real,dimension(:,:),allocatable ::residual2
real,dimension(:,:),allocatable ::currentf
real,dimension(:,:),allocatable ::abserror
integer,dimension(:,:),allocatable ::smooth


!get user input

print*,'What is the index file path?'
read*,indexfile
!connect to index file
open (16,file= indexfile,status="old")
!read from file
read (16,*)ai,name,name2,name5,level1,xrange,bc1,bc2
close(unit=16)
ai=ai+2
print*,'ai=',ai !number of grid points
print*,'name=',name
print*,'name2=',name2
print*,'name5=',name5
print*,'level1=',level1
print*,'xrange=',xrange
print*,'bc1=',bc1
print*,'bc2=',bc2


print*,'Where do you want the output to be written?'
read*,name3
print*,'Where do you want the error text files?'
read*,n4
n=0.5
print*,'What is the stopping criteria?'
read*,scriptem


div2=0
aim=ai-1
do
if (mod((aim),2)/=0) exit
aim=aim/2
div2=div2+1
end do
m=div2  !number of grid levels

!creates asizes vector
!column vector to hold sizes of A beginning with level 1
size=ai-2
allocate(asizes(m,1))
asizes(m,1)=size
do i= m-1,1,-1
size=(size+1)/2-1
asizes(i,1)=size
end do
aj=sum(asizes)
print*,asizes

allocate(a(ai-2,aj))
allocate(f(ai-2,m))
allocate(exactx(ai-2,m))


!connect to coefficient matrix file
open (2,file= name,status="old")
!read from file
```

```
read (2,'(<ai-2>(f15.7))')((a(i,j),j=1,ai-2),i=1,ai-2)
close(unit=2)

!connect to solution vector file
open (3,file= name2,status="old")
!read from file
read (3,'(f15.7)')(f(i,m),i=1,ai-2)
close(unit=3)

!connect to exact solution file
open (9,file= name5,status="old")
!read from file
read (9,'(f15.7)')(exactx(i,m),i=1,ai-2)
close(unit=9)


!initial values
allocate(x(ai,m))
   !boundary conditions
x(1,1)=bc1
x(3,1)=bc2

   !level 1 solution
x(2,1)=level1
   !epsilon for each level
allocate(scripte(m,1))
scripte(m,1)=scriptem
divs=4
do i=2,m-1
scripte(i,1)=(xrange/divs)**2
if (scripte(i,1)<scriptem) then
do j=i,m-1
scripte(i,1)=scriptem
end do
exit
end if
divs=divs*2
end do
scripte(1,1)=100.0

l=m
k=1
aicoarse=ai
alocation=asizes(m,1)
ajcoarse1=1
allocate(master(ai,m))
allocate(pro(1,1))  !random size
allocate(res(1,1))  !random size
allocate(residual(1,1))  !random size
allocate(residual2(1,1))  !random size
allocate(currentf(1,1))  !random size
allocate(abserror(1,1))  !random size
allocate(e(m,1))
allocate(olde(m,1))
allocate(ebar(m,1))
allocate(smooth(m,1))
allocate(ffine(ai-2,1))
smooth=0

e=0
coarse=0
olde=1
iteration=0
ebar=100000
```

```
rn=0
digits=1
p=2


open (4,file= name3,status="replace")
write (4,*)'convergence criteria=',scripte(m,1)
write (4,*)'initial epsilon=',scripte(2:m-1,1)
close (unit=4)


lenname=len(trim(name3))-4
print*,'lenname=',lenname
name8=name3(:lenname)//'2.txt'
open (8,file=name8,status="replace")
write(8,*)'Iteration  Level  Error'
write(8,*)' '
close(unit=8)


    !initial f and a matrices
ffine(2:aicoarse-3,1)=f(2:aicoarse-3,m)
ffine(1,1)=f(1,m)+bc1*a(2,ajcoarse1)
ffine(aicoarse-2,1)=f(aicoarse-2,m)+bc2*a(aicoarse-3,alocation)

do j=m-1,2,-1
k=j
aifine=aicoarse
aicoarse=(aifine+1)/2
aicoarsea=aicoarse-2
deallocate(res)
deallocate(pro)
allocate(res(aicoarse,aifine))
allocate(pro(aifine,aicoarse))
call prores(aifine,aicoarse,pro,res)
f(1:aicoarsea,j)=matmul(res(2:aicoarse-1,2:aifine-1),
     &  ffine(1:aifine-2,1))
ffine(1:aicoarsea,1)=f(1:aicoarsea,j)



alocation=sum(asizes(k:m,1))
ajcoarse1=1+alocation-(aicoarse-2)
ajfine1=alocation-(aicoarse-2)-(aifine-2)+1
ajfine2=alocation-(aicoarse-2)
a(1:aicoarse-2,ajcoarse1:alocation)=
     &  matmul(matmul(res(2:aicoarse-1,2:aifine-1),
     &     a(1:aifine-2,ajfine1:ajfine2)),pro(2:aifine-1,2:aicoarse-1))
!acoarse=matmul(matmul(res,afine),pro)

f(1,j)=ffine(1,1)-bc1*a(2,ajcoarse1)
f(aicoarsea,j)=ffine(aicoarsea,1)-bc2*a(aicoarsea-1,alocation)

exactx(1:aicoarsea,j)=matmul(res(2:aicoarse-1,2:aifine-1),
     & exactx(1:aifine-2,j+1))



end do



k=1
```

```
aifine=aicoarse
aicoarse=(aifine+1)/2
aicoarsea=aicoarse-2


print*,'aifine=',aifine
print*,'aicoarse=',aicoarse


open (4,file= name3,position="append")
write (4,*)'grid level=',k
close (unit=4)

open (4,file= name3,position="append")
write(4,*)'iteration=',iteration
write(4,*)' '
close(unit=4)

open (5,file=name8,position="append")
write(5,'(i9)',advance="no")iteration
write(5,'(i5)')k
write(5,*)' '
close (unit=5)
go to 2



! -Presmoothing using gauss seidel
 1 open (4,file= name3,position="append")
write (4,*)'grid level=',k
close (unit=4)

call gs2(a,alocation,x,f,aicoarse,error,k,ai,aj,
     &   m,name3,n4,iteration,exactx,aicoarsea,digits,avgerror)
open (4,file= name3,position="append")
if (rn==1) then
write(4,*)'prev residual norm=',resresidual
end if
if (rn==2) then
write(4,*)'pro prev residual norm=',proresidual
end if
write (4,*)'residual norm=',error
convrate=error/olde(k,1)
write (4,*)'convergence rate=',convrate
write (4,*)' '
close (unit=4)

 4 open (5,file=name8,position="append")
write(5,'(i9)',advance="no")iteration
write(5,'(i5)',advance="no")k
write(5,'(es11.2)')error
write(5,*)' '
close (unit=5)
e(k,1)=error
olde(k,1)=error

smooth(k,1)= smooth(k,1) + 1
 2 rn=0



if (e(k,1)<=scripte(k,1)) then
! print*,'I'
```

```
!I
if (k==1) then
!B
!exit
go to 3
else   !switch to finer level


  rn=2
deallocate(pro)
deallocate(res)
allocate(pro(aifine,aicoarse))
allocate(res(aicoarse,aifine))

      call prores(aifine,aicoarse,pro,res)

      x(1:aifine,k+1)= master(1:aifine,k+1) + matmul
     & (pro(1:aifine,1:aicoarse),(x(1:aicoarse,k)-matmul
     & (res(1:aicoarse,1:aifine),master(1:aifine,k+1))))
      !xfine= xfineold + matmul(pro,(xcoarse-matmul(res,xfineold)))



if (k==l-1) then !if at second highest level
aicoarse=aifine
else
aicoarse=aifine
aifine=(aicoarse*2)-1
end if

k=k+1
alocation=sum(asizes(k:m,1))
aicoarsea=aicoarse-2

!calculate residuals of current solution
deallocate(residual)
deallocate(residual2)
deallocate(currentf)
allocate(residual(aicoarsea,1))
allocate(residual2(aicoarsea,1))
allocate(currentf(aicoarsea,1))

currentf(1:aicoarsea,1)=
     & matmul(a(1:aicoarsea,1+alocation-aicoarsea:alocation),
     &  x(2:aicoarse-1,k))

do i=1,aicoarsea
residual(i,1)=currentf(i,1)-f(i,k)
end do

write(chariteration,"(I<digits>)")iteration
errorname=trim(n4)//"rn"//trim(chariteration)//".txt"

open (12,file= errorname,status="replace")
do i=1,aicoarsea
write(12,*)residual(i,1)
end do
close(unit=12)

!compute norm of residuals
do i=1,aicoarsea
residual2(i,1)=(residual(i,1))**2
end do
```

```
proresidual=((sum(residual2))**.5)/aicoarsea


*******************************************************************************
*******************************************************************************
                      ! If Using Correction, Insert Correction Code Here
*******************************************************************************
*******************************************************************************


!Calculate Absolute Error of current solution
deallocate(abserror)
allocate(abserror(aicoarsea,1))

do i=1,aicoarsea
abserror(i,1)=exactx(i,k)-x(i+1,k)
end do

write(chariteration,"(I<digits>)")iteration
errorname=trim(n4)//"n"//trim(chariteration)//".txt"

open (15,file= errorname,status="replace")
do i=1,aicoarsea
write(15,*)abserror(i,1)
end do
close(unit=15)



!back to GS
!cycle
goto 1
end if

else if (e(k,1)<=(n*ebar(k,1)) .or. k==1) then
ebar(k,1)=e(k,1)
go to 1

end if



!if at crudest level
if (k==2) then
!save values of current level
master(1:aicoarse,k)=x(1:aicoarse,k)
goto 1
end if

!save values of current level
master(1:aicoarse,k)=x(1:aicoarse,k)

!switch to a coarser level
rn=1
coarse=coarse+1
k=k-1
alocation=sum(asizes(k:m,1))
if (k==(l-1)) then !if you have moved to the second highest level
aicoarse=(aifine+1)/2
else
aifine=aicoarse
aicoarse=(aifine+1)/2
```

```
end if

deallocate(pro)
deallocate(res)
allocate(pro(aifine,aicoarse))
allocate(res(aicoarse,aifine))
call prores(aifine,aicoarse,pro,res)
x(1:aicoarse,k)=
     &  matmul(res(1:aicoarse,1:aifine),x(1:aifine,k+1))
!xcoarse=matmul(res,xfine)



ajcoarse1=1+alocation-(aicoarse-2)
ajfine1=alocation-(aicoarse-2)-(aifine-2)+1
ajfine2=alocation-(aicoarse-2)
! a(1:aicoarse-2,ajcoarse1:alocation)=
     &!  matmul(matmul(res(2:aicoarse-1,2:aifine-1),
     & !    a(1:aifine-2,ajfine1:ajfine2)),pro(2:aifine-1,2:aicoarse-1))
!acoarse=matmul(matmul(res,afine),pro)



f(1:aicoarse-2,k)=matmul(a(1:aicoarse-2,ajcoarse1:alocation),
     &     x(2:aicoarse-1,k)) + matmul(res(2:aicoarse-1,2:aifine-1),
     &  (f(1:aifine-2,k+1)-matmul(a(1:aifine-2,ajfine1:ajfine2),
     &  x(2:aifine-1,k+1))))
!fcoarse=matmul(acoarse,xcoarse) + matmul(res,(ffine-matmul(afine,xfine)))


if (0.7*scriptem>.2*e(k+1,1)) then
scripte(k,1)=.7*scriptem
else
scripte(k,1)=.2*e(k+1,1)
end if
aicoarsea=aicoarse-2



!calculate residuals of current solution
deallocate(residual)
deallocate(residual2)
deallocate(currentf)
allocate(residual(aicoarsea,1))
allocate(residual2(aicoarsea,1))
allocate(currentf(aicoarsea,1))

currentf(1:aicoarsea,1)=
     & matmul(a(1:aicoarsea,1+alocation-aicoarsea:alocation),
     &  x(2:aicoarse-1,k))

do i=1,aicoarsea
residual(i,1)=currentf(i,1)-f(i,k)
end do

write(chariteration,"(I<digits>)")iteration
errorname=trim(n4)//"rn"//trim(chariteration)//".txt"

open (13,file= errorname,status="replace")
do i=1,aicoarsea
write(13,*)residual(i,1)
end do
close(unit=13)
```

```
!compute norm of residuals
do i=1,aicoarsea
residual2(i,1)=(residual(i,1))**2
end do

resresidual=((sum(residual2))**.5)/aicoarsea

!Calculate Absolute Error of current solution
deallocate(abserror)
allocate(abserror(aicoarsea,1))

do i=1,aicoarsea
abserror(i,1)=exactx(i,k)-x(i+1,k)
end do

write(chariteration,"(I<digits>)")iteration
errorname=trim(n4)//"n"//trim(chariteration)//".txt"

open (14,file= errorname,status="replace")
do i=1,aicoarsea
write(14,*)abserror(i,1)
end do
close(unit=14)

ebar(k,1)=resresidual
olde(k,1)=resresidual
goto 1



 3 weight=1
do i=m,1,-1
sumgs = weight*smooth(i,1)+sumgs
weight=.25*weight
end do



!print # iterations
open (4,file= name3,position="append")
write(4,*)'Number of coarse grid transfers=',coarse
write(4,*)' '
write(4,*)'Number of GS smoothing operations per level asc. order'
write(4,'(<1>(i4))')((smooth(i,j),j=1,1),i=1,m)
write(4,*)'Sum of GS iterations=',sumgs
close(unit=4)

end program main
```

A.2 Correction

```
     !This code is inserted into main.for in the specified location

!Enter iteration of correction
if ((iteration==39)) then !.or. (iteration==11) ) then
     & ! .or. (iteration==26)) then
maxresm=maxloc(residual2(2:aicoarsea))
```

```
maxres=maxresm(1)+1
maxratio=abs(residual(maxres))/proresidual
if (maxratio>2)then
open (4,file= name3,position="append")
write(4,*)'maxres=',maxres
write(4,*)'maxratio=',maxratio
write(4,*)' '
write(4,*)' '
close(unit=4)

if (maxres==aicoarsea) then
x(maxres+1,k)=(f(maxres,k)-x(maxres,k)
     & *a(2,1+alocation-aicoarsea))
     & /a(2,2+ alocation-aicoarsea)
else
x(maxres+1,k)=(f(maxres,k)-x(maxres,k)
     & *a(2,1+alocation-aicoarsea)-x(maxres+2,k)
     & *a(2,3+alocation-aicoarsea))
     & /a(2,2+ alocation-aicoarsea)
end if

end if
end if
```

A.3 Gauss Seidel - gs.for

```
subroutine gs2(a,alocation,x,f,aicoarse,error,k,ai,aj,
     & m,name3,n4,iteration,exactx,aicoarsea,digits,avgerror)

!declare variables
integer ::aicoarse, imxres,factor
integer ::iter,c
integer,intent(out) ::digits
integer,intent(inout) ::iteration
integer,intent(in) ::alocation,ai,aj,m
integer,intent(in) ::k,aicoarsea
real ::num, mxres
real,intent(out) ::error,avgerror
character(len=80) ::name
character(len=80) ::name2
character(len=80) ::name3
character(len=80) ::n4
character(len=10) ::chariteration
character(len=100) ::errorname
real,dimension(ai-2,aj) ::a
real,dimension(ai,m),intent(inout)    ::x
real,dimension(aicoarsea,1)       ::xnew
real,dimension(ai-2,m)   ::f
real,dimension(aicoarsea,aicoarsea+1) ::b
real,dimension(aicoarsea,1) ::residuals
real,dimension(aicoarsea,1) ::residuals2
real,dimension(aicoarsea,1) ::currentsol
real,dimension(aicoarsea,1) ::abserror
real,dimension(ai-2,m),intent(in)   ::exactx



b=0
!define augmented matrix b
```

```
do i=1,aicoarsea
do j=1,aicoarsea
b(i,j)=a(i,j+alocation-aicoarsea)
end do
end do
do i=1,aicoarsea
b(i,aicoarsea+1)=f(i,k)
end do



!GS iterations
iteration=iteration+1

imxres=0
mxres=0
do i=1,aicoarsea
num=b(i,aicoarsea+1)
do j=1,aicoarsea
num= num-x(j+1,k)*b(i,j)
end do
xnew(i,1)=(num+x(i+1,k)*b(i,i))/b(i,i)
x(i+1,k)=xnew(i,1)
end do



currentsol(1:aicoarsea,1)=matmul(b(1:aicoarsea,1:aicoarsea),
     &   x(2:aicoarse-1,k))

do i=1,aicoarsea
residuals(i,1)=(currentsol(i,1)-f(i,k))
if(abs(residuals(i,1))>abs(mxres)) then
mxres = residuals(i,1)
imxres = i*2**(m-k)
end if
end do

open (4,file= name3,position="append")
write (4,*)'iteration=',iteration
write (4,*) 'max residual=', mxres,' at location=',imxres
close (unit=4)

!Exact solution

do i=1,aicoarsea
abserror(i,1)=exactx(i,k)-x(i+1,k)
end do
!average of abserror
if (k==m) then
avgerror=sum(abserror)/aicoarsea
end if


if (iteration < 10) then
digits=1
else if (iteration <100) then
digits=2
else if (iteration<1000) then
digits=3
else if (iteration<10000) then
digits=4
end if
```

```
!write error text files
write(chariteration,"(I<digits>)")iteration
errorname=trim(n4)//"r"//trim(chariteration)//".txt"

open (7,file= errorname,status="replace")
do i=1,aicoarsea
write(7,*)residuals(i,1)
end do
close(unit=7)

write(chariteration,"(I<digits>)")iteration
errorname=trim(n4)//trim(chariteration)//".txt"

open (8,file= errorname,status="replace")
do i=1,aicoarsea
write(8,*)abserror(i,1)
end do
close(unit=8)

!compute norm of residuals
do i=1,aicoarsea
residuals2(i,1)=(residuals(i,1))**2
end do

error=((sum(residuals2))**.5)/aicoarsea


end subroutine gs2
```

A.4 Prolongation and Restriction Operators - prores.for

```
subroutine prores(aifine,aicoarse,pro,res)

!variables
integer ::c
integer,intent(in) ::aifine
integer,intent(in) ::aicoarse
real,dimension(aifine,aicoarse),intent(out)::pro
real,dimension(aicoarse,aifine),intent(out) ::res


!define prolongation operator matrix
pro=0
pro(1,1)=1
c=2
!odd rows
do i=3,aifine,2
pro(i,c)=1
c=c+1
end do
c=1
!even rows
do i=2,aifine-1,2
pro(i,c)=.5
pro(i,c+1)=.5
c=c+1
end do

!define restriction operator matrix
```

```
res=.5*transpose(pro)
res(1,1)=1
res(1,2)=0
res(aicoarse,aifine)=1
res(aicoarse,aifine-1)=0

end subroutine prores
```

Appendix B

Efficiency Test Programs - Fortran 90 Code

B.1 Gauss Seidel Efficiency Test Program

```
program gstest

real ::x
real ::f
real ::a1
real ::a2
real ::a3
real ::a4
real ::a5
real ::a6
real ::a7
real ::x2
real ::x3
real ::x4
real ::x5
real ::x6
real ::x7



f=  5.463667
a1= 3.254534
a2= 5.979375
a3= -37.85091
a4= 4.235096
a5= -15.49078
a6= 7.094582
a7= 9.430092
a8= 4.687798
a9= 3.068973
a10=-6.975475
a11=3.254534
a12= 5.979375
a13= -37.85091
a14= 4.235096
a15= -15.49078
a16= 7.094582
a17= 9.432092
```

```
a18= 4.687798
a19= 6.045973
a20=-6.978475
a21=3.254534
a22= 5.979375
a23= -3.550912
a24= 4.735056
a25= -15.49078
a26= 7.094582
a27= 9.439092
a28= 4.687798
a29= 5.045973
a30=-6.978475
a31= 2.655665
a32= 7.484155
a33= -2.342189
a34= 4.235096
a35= -15.49078
a36= 8.094582
a37= 9.430092
a38= 4.647798
a39= 2.068973
a40=-6.975475
a41=3.554534
a42= 5.979375
a43= -37.87091
a44= 4.235096
a45= -15.49078
a46= 7.094582
a47= 9.432092
a48= 4.687798
a49= 6.045973
a50=-6.978475
a51=3.254534
a52= 5.979375
a53= -3.550912
a54= 4.735056
a55= -15.49078
a56= 7.098582
a57= 9.439092
a58= 5.687798
a59= 5.045973
a60=-6.978475
a61= 2.555665
a62= 3.484155
a63= -2.347189

x2= -9.086549
x3= 3.576876
x4= 6.875432
x5=-6.873684
x6= 14.586123
x7= 9.6612868
x8=-5.298981
x9= 4.459823
x10=8.945722
x11=-4.864655
x12= -5.086549
x13= 3.579876
x14= 6.875432
x15=-6.273684
x16= 13.58612
x17= 9.661986
x18=-5.098981
```

```
 x19= 4.459523
 x20=6.935722
 x21=-4.896655
 x22= -5.026549
 x23= 3.579816
 x24= 8.875432
 x25=-6.273684
 x26= 13.58612
 x27= 9.645986
 x28=-3.098981
 x29= 4.459523
 x30=6.934322
 x31=-7.890655
 x32=1.5644651
 x33=2.5861816
 x34= 6.875432
 x35=-6.875984
 x36= 14.586123
 x37= 2.6612868
 x38=-5.278981
 x39= 4.459823
 x40=5.945722
 x41=-4.864955
 x42= -5.086549
 x43= 3.879876
 x44= 9.875432
 x45=-6.283684
 x46= 13.58612
 x47= 9.661986
 x48=-5.097981
 x49= 4.459523
 x50=5.935722
 x51=-4.896655
 x52= -5.026549
 x53= 4.579816
 x54= 8.879432
 x55=-6.273684
 x56= 18.58612
 x57= 9.645986
 x58=-3.098981
 x59= 4.459523
 x60=6.934322
 x61=-7.890655
 x62=1.5644651
 x63=2.5861816



do k=1,10

call cpu_time(t1)

do i=1,100000000
x=(f-a2*x2-a3*x3-a4*x4-a5*x5-a6*x6-a7*x7-a8*x8-a9*x9
     & -a10*x10-a11*x11-a12*x12-a13*x13-a14*x14-a15*x15
     & -a16*x16-a17*x17-a18*x18-a19*x19-a20*x20-a21*x21
     & -a22*x22-a23*x23-a24*x24-a25*x25-a26*x26-a27*x27
     & -a28*x28-a29*x29-a30*x30-a31*x31-a32*x32-a33*x33
     & -a34*x34-a35*x35
     & -a36*x36-a37*x37-a38*x38-a39*x39-a40*x40-a41*x41
     & -a42*x42-a43*x43-a44*x44-a45*x45-a46*x46-a47*x47
     & -a48*x48-a49*x49-a50*x50-a51*x51-52*x52-a53*x53
     & -a54*x54-a55*x55-a56*x56-a57*x57-a58*x58-a59*x59
     & -a60*x60-a61*x61-a62*x62-a63*x63)/a1
```

```
end do


call cpu_time(t2)

print *, 'time to process = ', (t2-t1)

end do

end program gstest
```

B.2 Correction Efficiency Test Program

```
program correctiontest

real ::x
real ::f
real ::a1
real ::a2
real ::a3
real ::a4
real ::a5
real ::a6
real ::a7
real ::x2
real ::x3
real ::x4
real ::x5
real ::x6
real ::x7


x1= -2.458456
f=  5.463667
a1= 3.254534
a2= 5.979375
a3= -37.85091
a4= 4.235096
a5= -15.49078
a6= 7.094582
a7= 9.430092
a8= 4.687798
a9= 3.068973
a10=-6.975475
a11=3.254534
a12= 5.979375
a13= -37.85091
a14= 4.235096
a15= -15.49078
a16= 7.094582
a17= 9.432092
a18= 4.687798
a19= 6.045973
a20=-6.978475
a21=3.254534
a22= 5.979375
a23= -3.550912
a24= 4.735056
a25= -15.49078
```

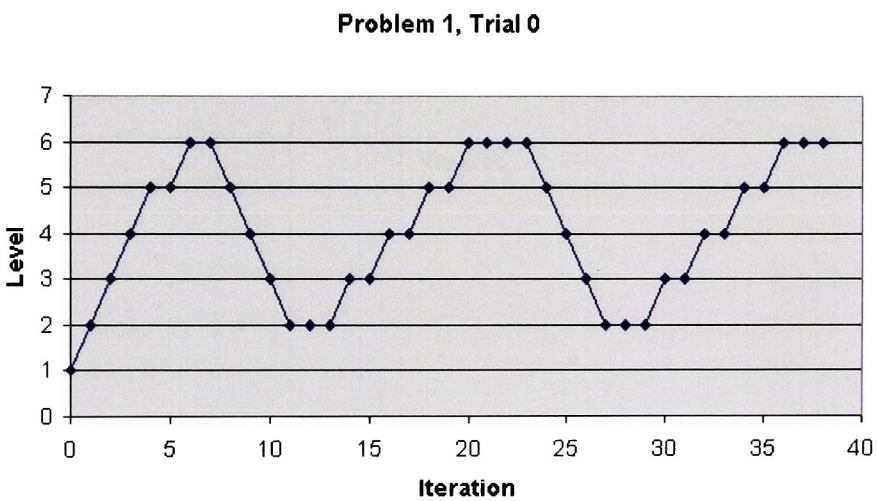
```
a26= 7.094582
a27= 9.439092
a28= 4.687798
a29= 5.045973
a30=-6.978475
a31= 2.655665
a32= 7.484155
a33= -2.342189
a34= 4.235096
a35= -15.49078
a36= 8.094582
a37= 9.430092
a38= 4.647798
a39= 2.068973
a40=-6.975475
a41=3.554534
a42= 5.979375
a43= -37.87091
a44= 4.235096
a45= -15.49078
a46= 7.094582
a47= 9.432092
a48= 4.687798
a49= 6.045973
a50=-6.978475
a51=3.254534
a52= 5.979375
a53= -3.550912
a54= 4.735056
a55= -15.49078
a56= 7.098582
a57= 9.439092
a58= 5.687798
a59= 5.045973
a60=-6.978475
a61= 2.555665
a62= 3.484155
a63= -2.347189

x2= -9.086549
x3= 3.576876
x4= 6.875432
x5=-6.873684
x6= 14.586123
x7= 9.6612868
x8=-5.298981
x9= 4.459823
x10=8.945722
x11=-4.864655
x12= -5.086549
x13= 3.579876
x14= 6.875432
x15=-6.273684
x16= 13.58612
x17= 9.661986
x18=-5.098981
x19= 4.459523
x20=6.935722
x21=-4.896655
x22= -5.026549
x23= 3.579816
x24= 8.875432
x25=-6.273684
x26= 13.58612
```

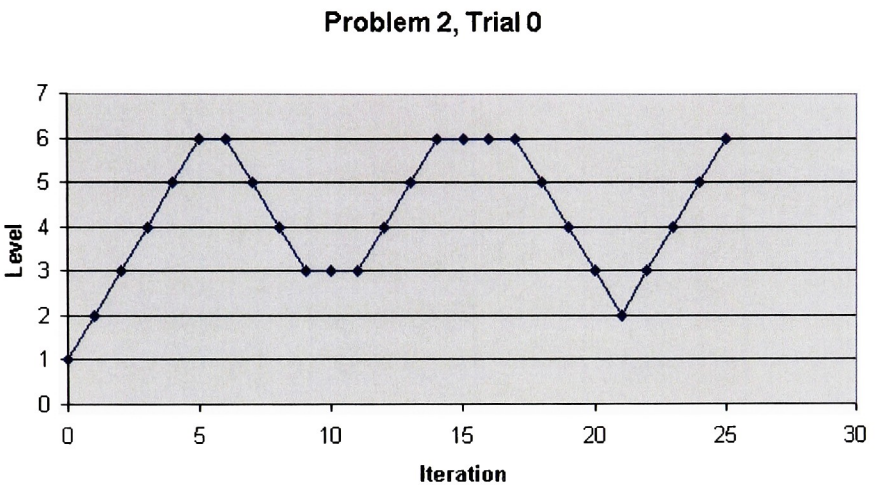
```
 x27= 9.645986
 x28=-3.098981
 x29= 4.459523
 x30=6.934322
 x31=-7.890655
 x32=1.5644651
 x33=2.5861816
 x34= 6.875432
 x35=-6.875984
 x36= 14.586123
 x37= 2.6612868
 x38=-5.278981
 x39= 4.459823
 x40=5.945722
 x41=-4.864955
 x42= -5.086549
 x43= 3.879876
 x44= 9.875432
 x45=-6.283684
 x46= 13.58612
 x47= 9.661986
 x48=-5.097981
 x49= 4.459523
 x50=5.935722
 x51=-4.896655
 x52= -5.026549
 x53= 4.579816
 x54= 8.879432
 x55=-6.273684
 x56= 18.58612
 x57= 9.645986
 x58=-3.098981
 x59= 4.459523
 x60=6.934322
 x61=-7.890655
 x62=1.5644651
 x63=2.5861816



 do k=1,10

 call cpu_time(t1)

 !residual calculation
 do i=1,100000000

 x=f-a1*x1-a2*x2-a3*x3-a4*x4-a5*x5-a6*x6-a7*x7-a8*x8-a9*x9
      & -a10*x10-a11*x11-a12*x12-a13*x13-a14*x14-a15*x15
      & -a16*x16-a17*x17-a18*x18-a19*x19-a20*x20-a21*x21
      & -a22*x22-a23*x23-a24*x24-a25*x25-a26*x26-a27*x27
      &   -a28*x28-a29*x29-a30*x30-a31*x31 -a32*x32-a33*x33
      & -a34*x34-a35*x35
      & -a36*x36-a37*x37-a38*x38-a39*x39-a40*x40-a41*x41
      & -a42*x42-a43*x43-a44*x44-a45*x45-a46*x46-a47*x47
      & -a48*x48-a49*x49-a50*x50-a51*x51-52*x52-a53*x53
      & -a54*x54-a55*x55-a56*x56-a57*x57-a58*x58-a59*x59
      & -a60*x60-a61*x61-a62*x62-a63*x63


 end do

 !comparison
 do l=1,98412698   !100000000*62/63  !62 comparisons are made for every 63 runs
```

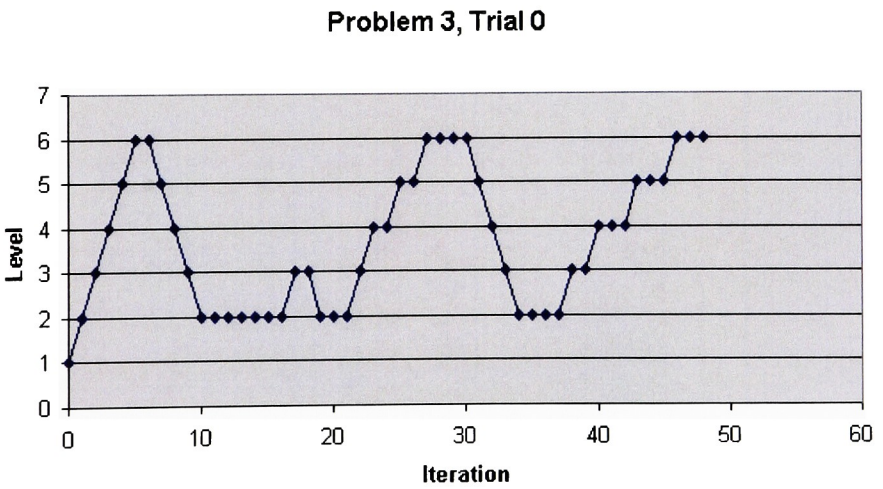
```
if(x>2) then
x=1
end if
end do

call cpu_time(t2)

print *, 'time to process = ', (t2-t1)


end program correction test
```

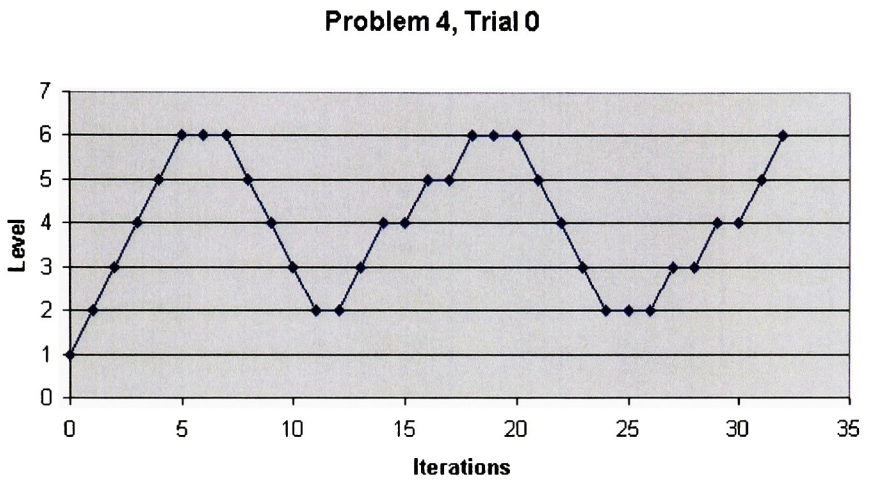
Appendix C

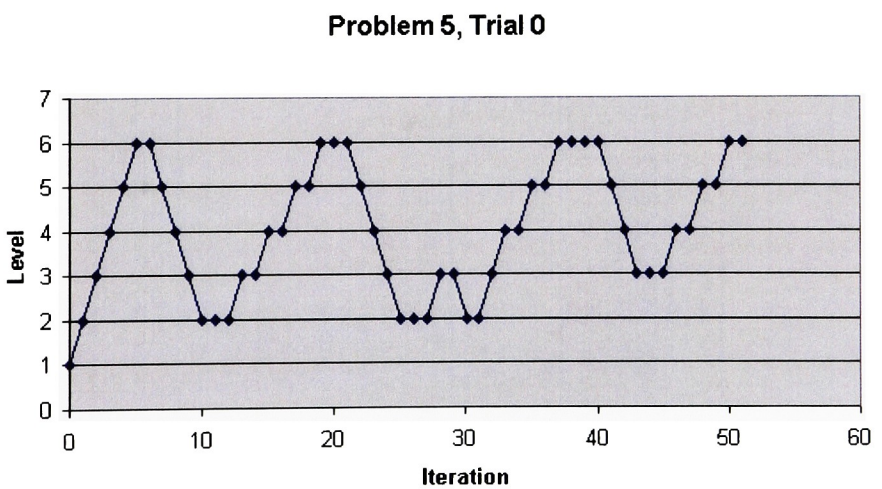
Accomodative Algorithm Runs

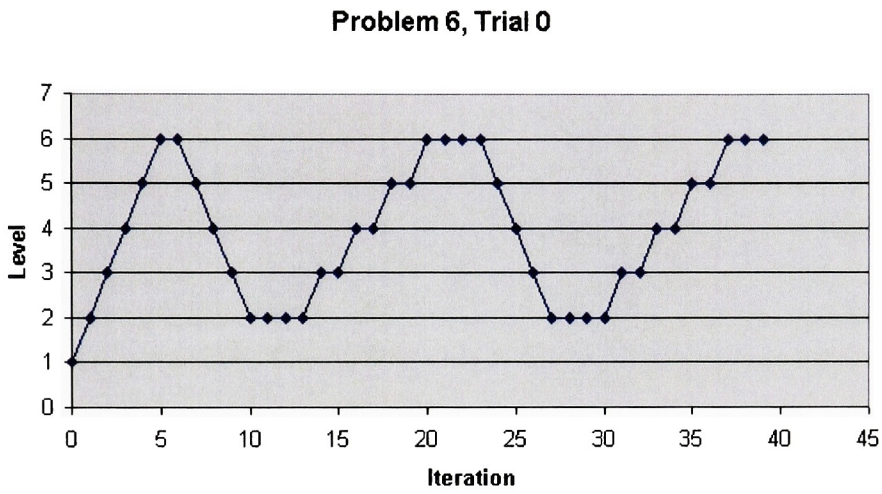


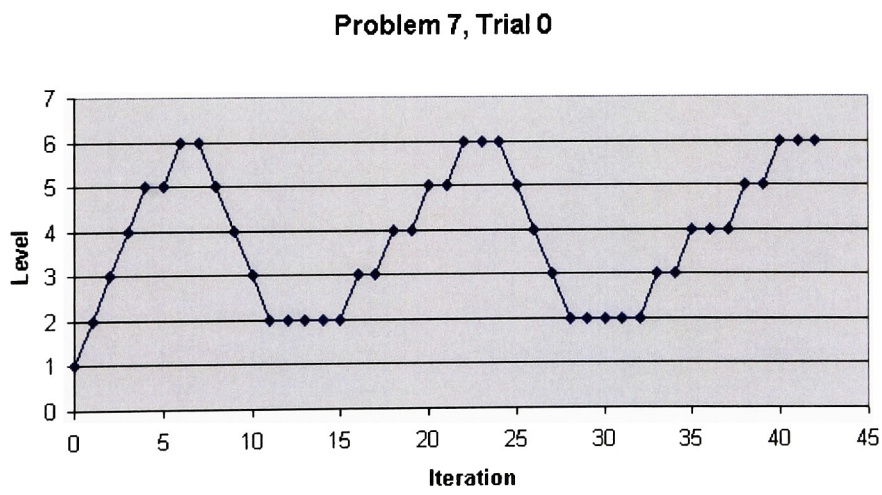

Problem 2, Trial 0
Level
Iteration
0
1
2
3
4
5
6
7
0
5
10
15
20
25
30

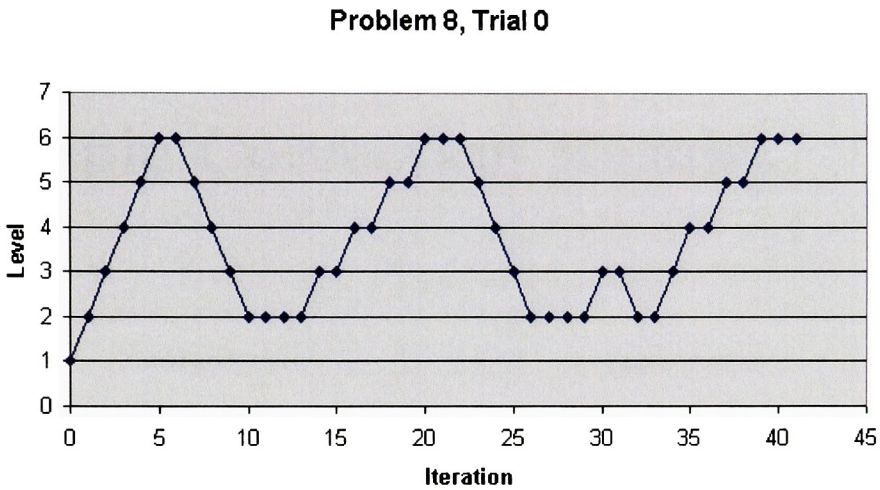

Problem 3, Trial 0
Level
Iteration
0
1
2
3
4
5
6
7
0
10
20
30
40
50
60

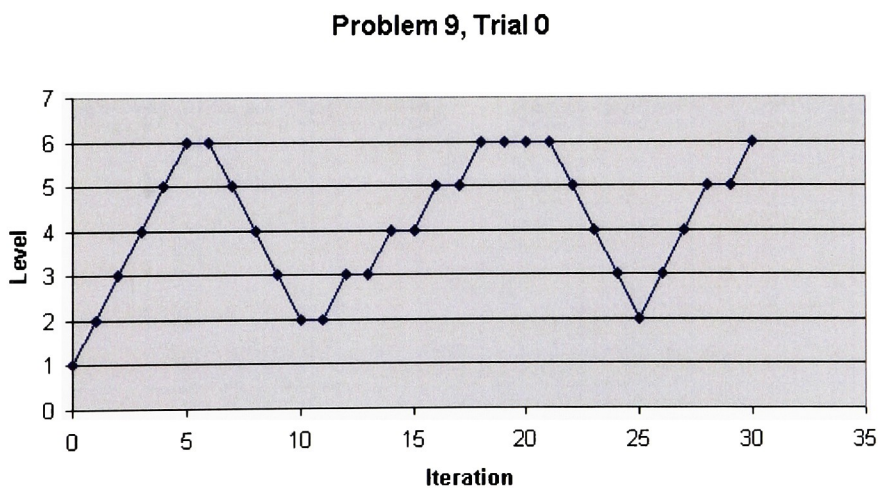

Problem 4, Trial 0
Level
Iterations
0
1
2
3
4
5
6
7
0
5
10
15
20
25
30
35


Problem 5, Trial 0
Level
Iteration
0
1
2
3
4
5
6
7
0
10
20
30
40
50
60


Problem 6, Trial 0
Level
Iteration
0
1
2
3
4
5
6
7
0
5
10
15
20
25
30
35
40
45

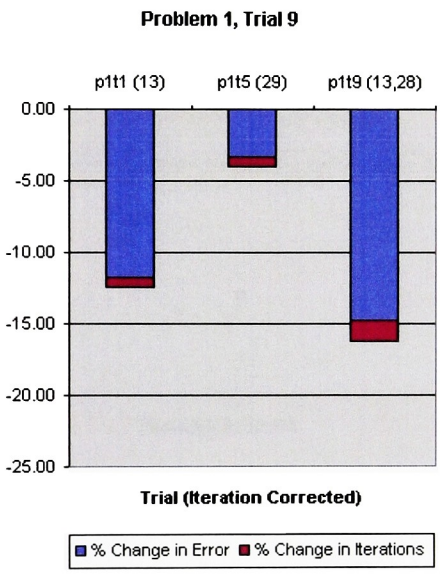

Problem 7, Trial 0
Level
Iteration
0
1
2
3
4
5
6
7
0
5
10
15
20
25
30
35
40
45

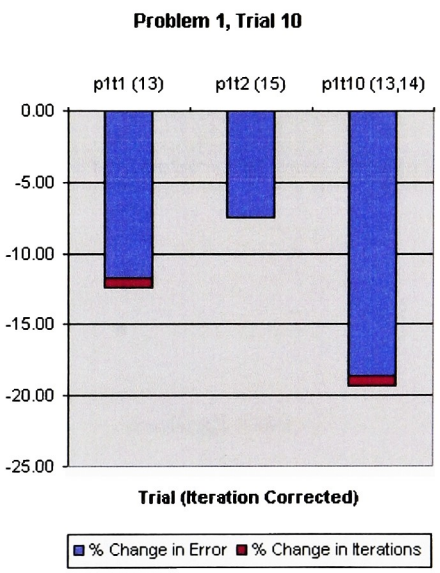

Problem 8, Trial 0
Level
Iteration
0
1
2
3
4
5
6
7
0
5
10
15
20
25
30
35
40
45

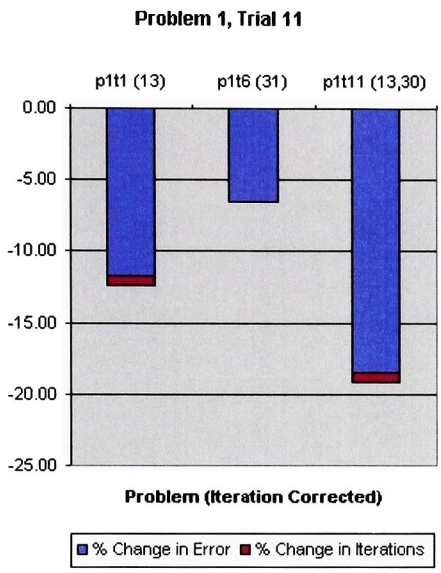

Problem 9, Trial 0
Level
Iteration
0
1
2
3
4
5
6
7
0
5
10
15
20
25
30
35

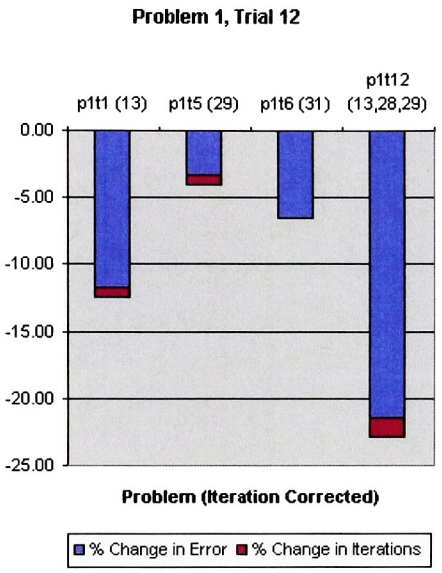
Appendix D

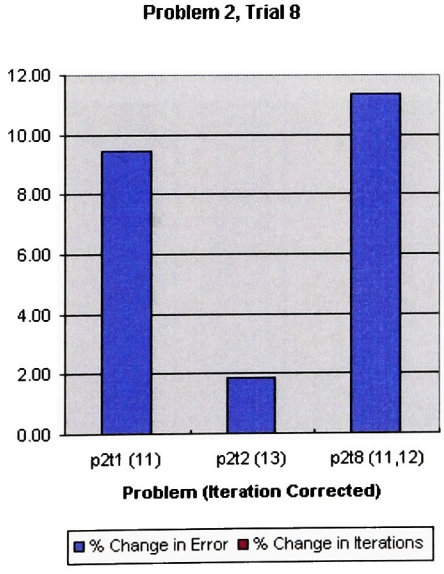
Multiple Correction Results

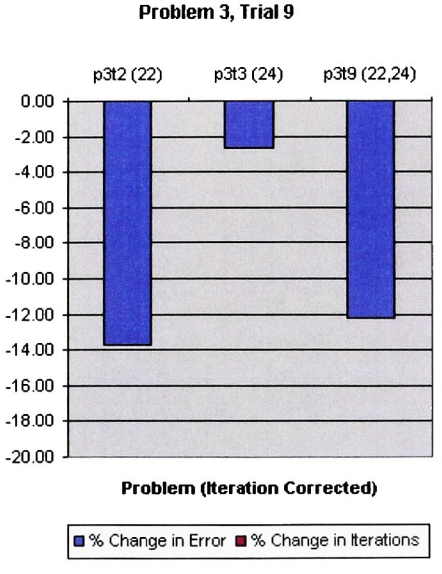


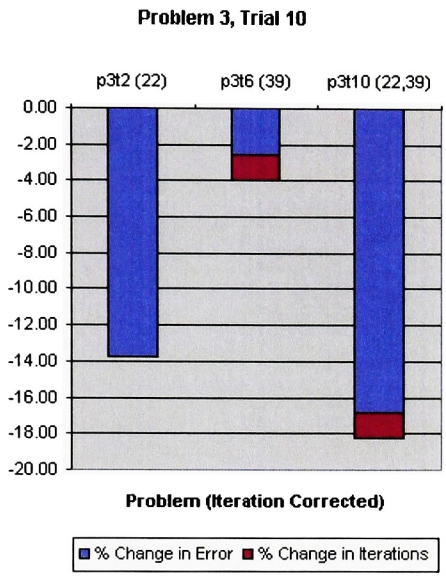


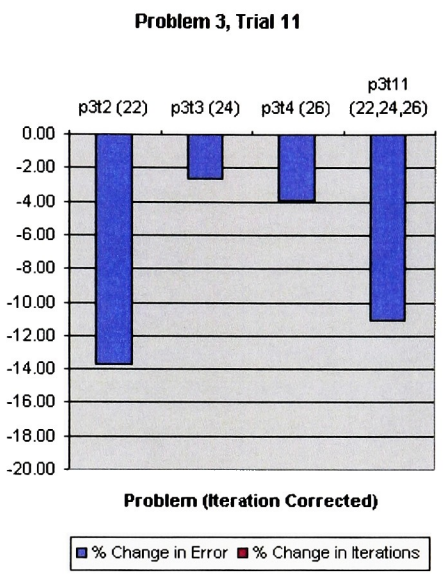

Problem 1, Trial 11
p1t1 (13)
p1t6 (31)
p1t11 (13,30)
0.00
-5.00
-10.00
-15.00
-20.00
-25.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

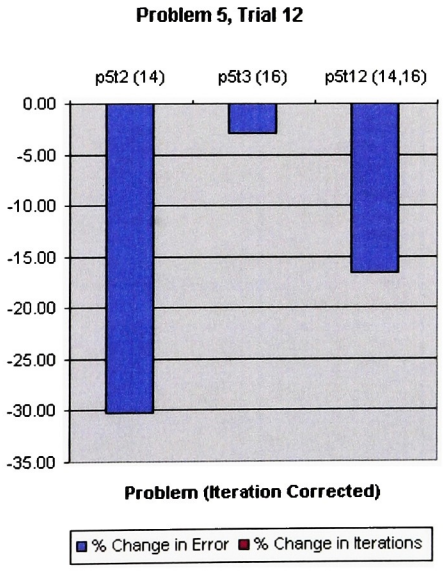

Problem 1, Trial 12
p1t1 (13)
p1t5 (29)
p1t6 (31)
p1t12 (13,28,29)
0.00
-5.00
-10.00
-15.00
-20.00
-25.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

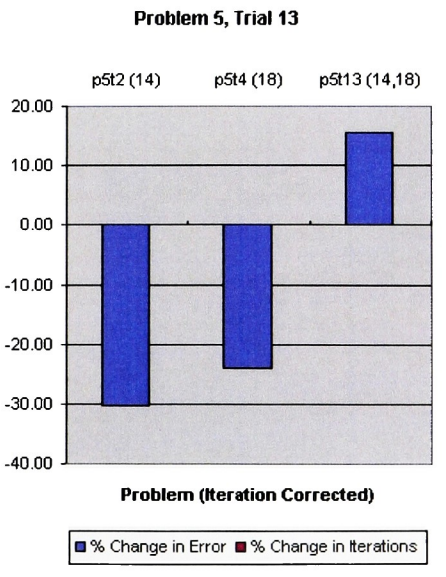

Problem 2, Trial 8
12.00
10.00
8.00
6.00
4.00
2.00
0.00
p2t1 (11)
p2t2 (13)
p2t8 (11,12)
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

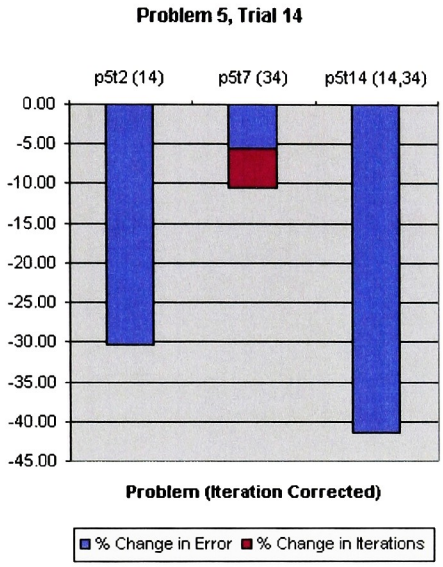

Problem 3, Trial 9
p3t2 (22)
p3t3 (24)
p3t9 (22,24)
0.00
-2.00
-4.00
-6.00
-8.00
-10.00
-12.00
-14.00
-16.00
-18.00
-20.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

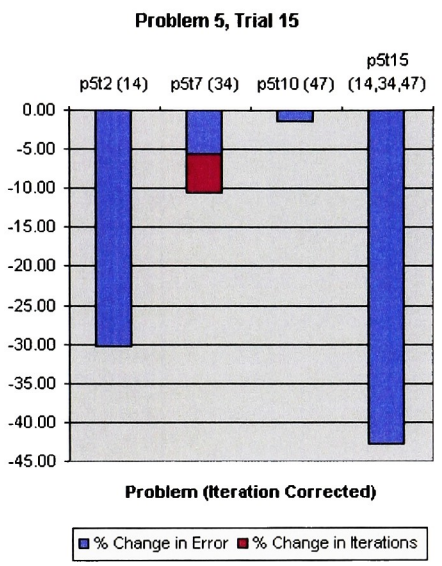

Problem 3, Trial 10
p3t2 (22)
p3t6 (39)
p3t10 (22,39)
0.00
-2.00
-4.00
-6.00
-8.00
-10.00
-12.00
-14.00
-16.00
-18.00
-20.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

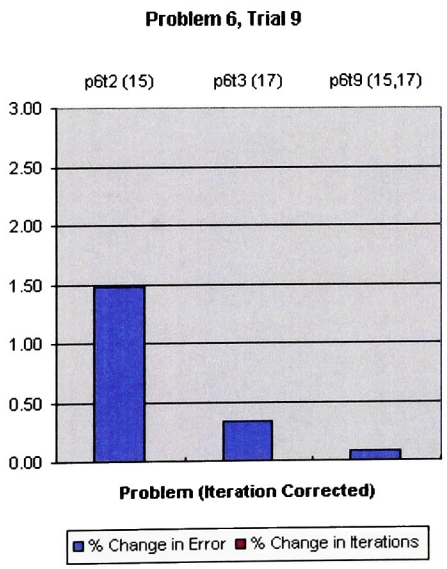

Problem 3, Trial 11
p3t2 (22)
p3t3 (24)
p3t4 (26)
p3t11 (22,24,26)
0.00
-2.00
-4.00
-6.00
-8.00
-10.00
-12.00
-14.00
-16.00
-18.00
-20.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

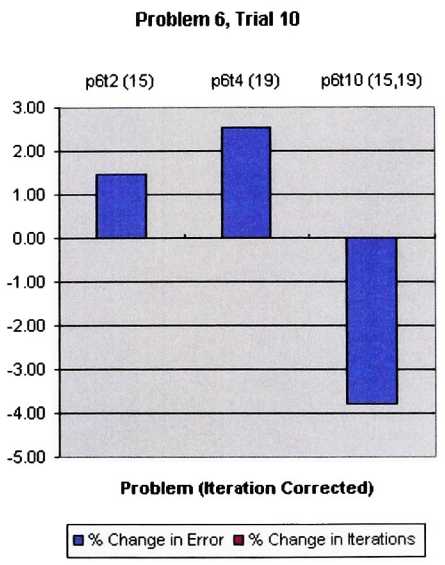

Problem 5, Trial 12
p5t2 (14)
p5t3 (16)
p5t12 (14,16)
0.00
-5.00
-10.00
-15.00
-20.00
-25.00
-30.00
-35.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

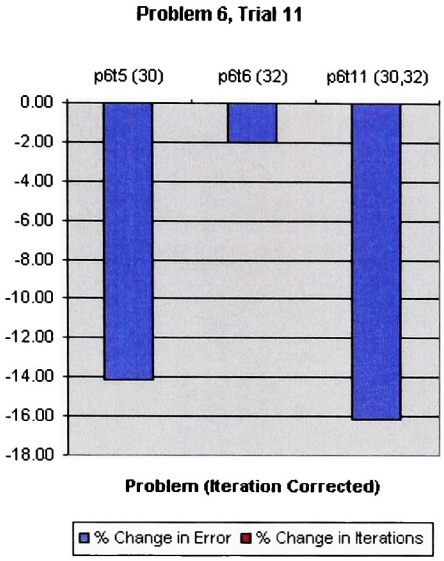

Problem 5, Trial 13
p5t2 (14)
p5t4 (18)
p5t13 (14,18)
20.00
10.00
0.00
-10.00
-20.00
-30.00
-40.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

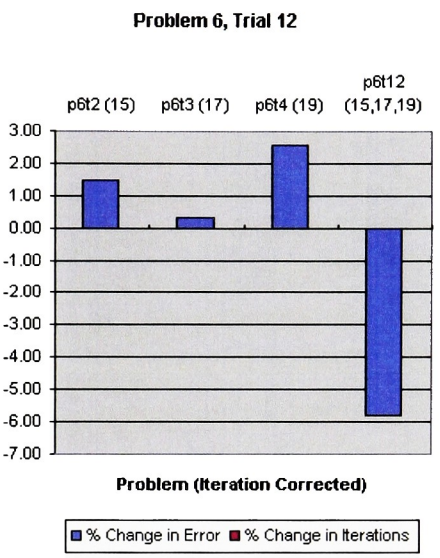

Problem 5, Trial 14
p5t2 (14)
p5t7 (34)
p5t14 (14,34)
0.00
-5.00
-10.00
-15.00
-20.00
-25.00
-30.00
-35.00
-40.00
-45.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

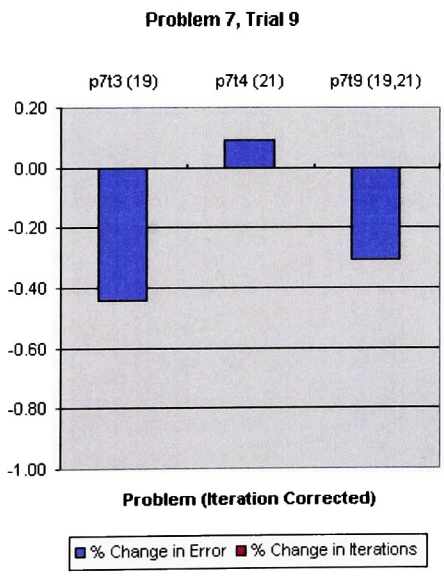

Problem 5, Trial 15
p5t2 (14)
p5t7 (34)
p5t10 (47)
p5t15 (14,34,47)
0.00
-5.00
-10.00
-15.00
-20.00
-25.00
-30.00
-35.00
-40.00
-45.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

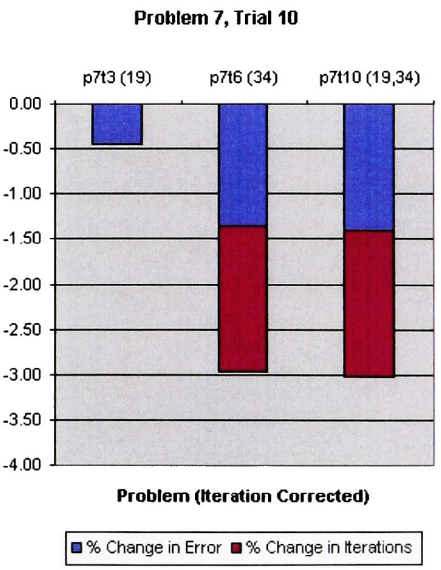

Problem 6, Trial 9
p6t2 (15)
p6t3 (17)
p6t9 (15,17)
3.00
2.50
2.00
1.50
1.00
0.50
0.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

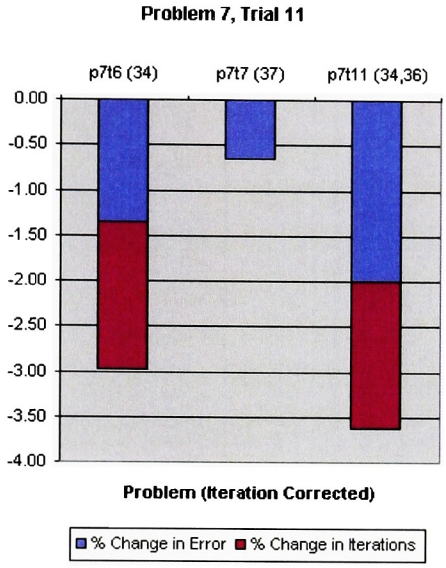

Problem 6, Trial 10
p6t2 (15)
p6t4 (19)
p6t10 (15,19)
3.00
2.00
1.00
0.00
-1.00
-2.00
-3.00
-4.00
-5.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

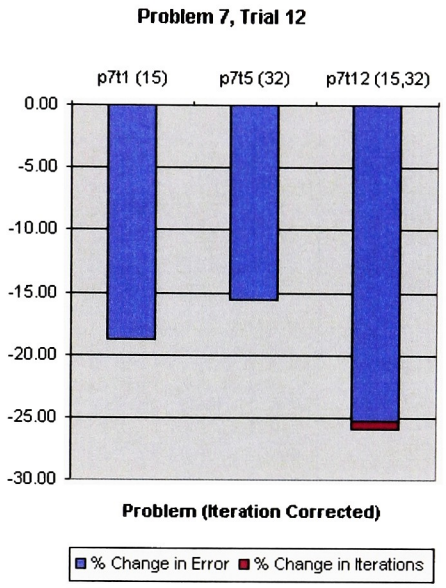

Problem 6, Trial 11
p6t5 (30)
p6t6 (32)
p6t11 (30,32)
0.00
-2.00
-4.00
-6.00
-8.00
-10.00
-12.00
-14.00
-16.00
-18.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

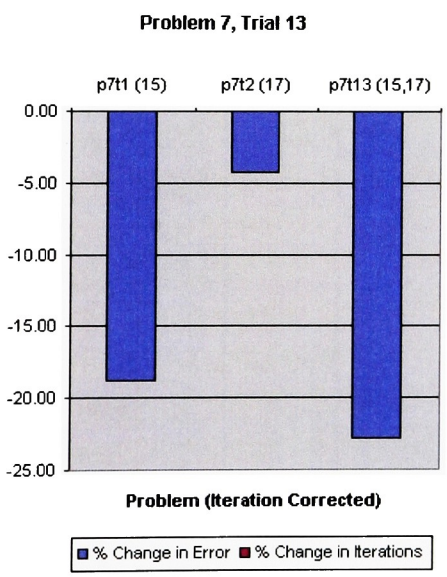

Problem 6, Trial 12
p6t2 (15)
p6t3 (17)
p6t4 (19)
p6t12 (15,17,19)
3.00
2.00
1.00
0.00
-1.00
-2.00
-3.00
-4.00
-5.00
-6.00
-7.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

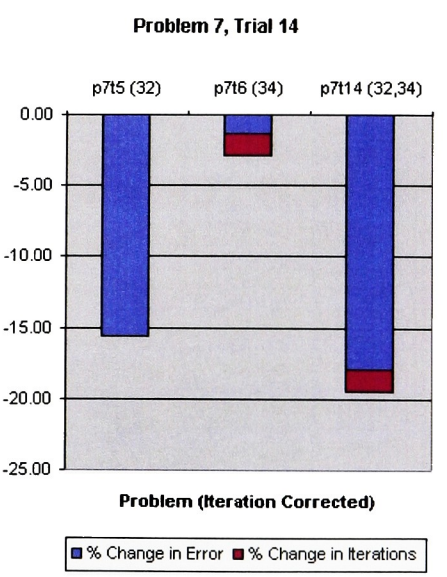

Problem 7, Trial 9
p7t3 (19)
p7t4 (21)
p7t9 (19,21)
0.20
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

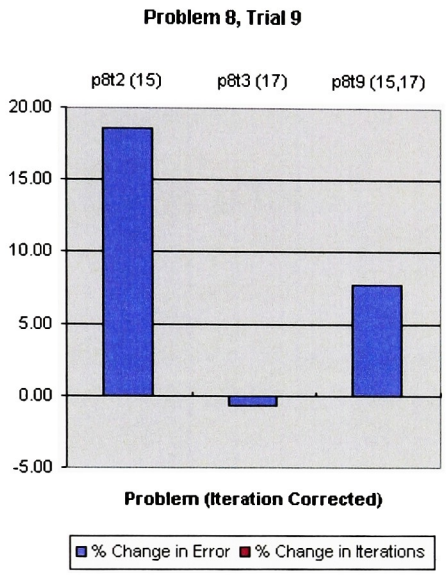

Problem 7, Trial 10
p7t3 (19)
p7t6 (34)
p7t10 (19,34)
0.00
-0.50
-1.00
-1.50
-2.00
-2.50
-3.00
-3.50
-4.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

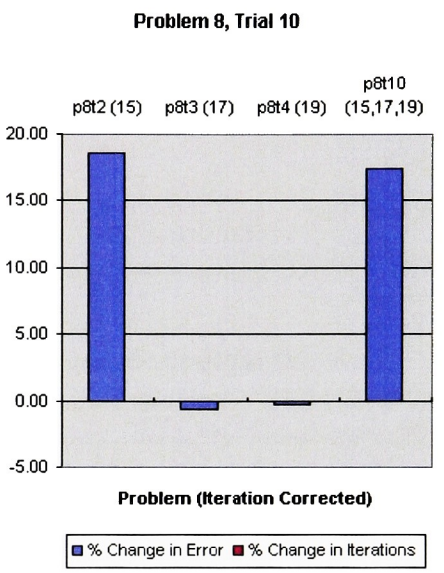

Problem 7, Trial 11
p7t6 (34)
p7t7 (37)
p7t11 (34,36)
0.00
-0.50
-1.00
-1.50
-2.00
-2.50
-3.00
-3.50
-4.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

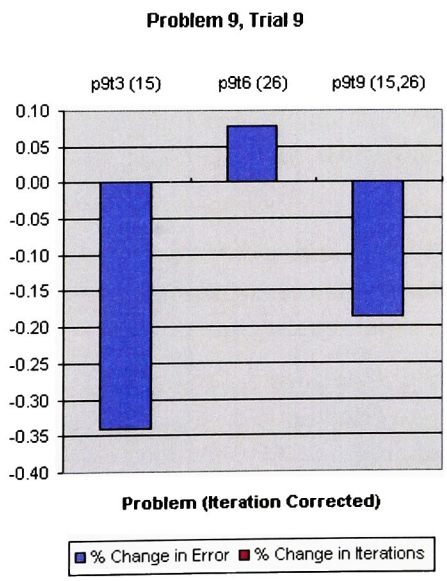

Problem 7, Trial 12
p7t1 (15)
p7t5 (32)
p7t12 (15,32)
0.00
-5.00
-10.00
-15.00
-20.00
-25.00
-30.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

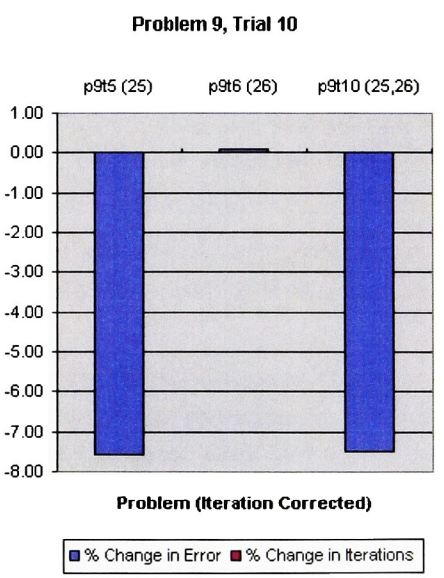

Problem 7, Trial 13
p7t1 (15)
p7t2 (17)
p7t13 (15,17)
0.00
-5.00
-10.00
-15.00
-20.00
-25.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations


Problem 7, Trial 14
p7t5 (32)
p7t6 (34)
p7t14 (32,34)
0.00
-5.00
-10.00
-15.00
-20.00
-25.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations


Problem 8, Trial 9
p8t2 (15)
p8t3 (17)
p8t9 (15,17)
20.00
15.00
10.00
5.00
0.00
-5.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations


Problem 8, Trial 10
p8t2 (15)
p8t3 (17)
p8t4 (19)
p8t10 (15,17,19)
20.00
15.00
10.00
5.00
0.00
-5.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations


Problem 9, Trial 9
p9t3 (15)
p9t6 (26)
p9t9 (15,26)
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
-0.30
-0.35
-0.40
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations


Problem 9, Trial 10
p9t5 (25)
p9t6 (26)
p9t10 (25,26)
1.00
0.00
-1.00
-2.00
-3.00
-4.00
-5.00
-6.00
-7.00
-8.00
Problem (Iteration Corrected)
% Change in Error
% Change in Iterations

Appendix E

Trial Results

The tables below show the results of all of the trials executed for this study. The letter p indicates the problem number, t indicates the trial number. Trial 0 is the multigrid run performed without any corrections. The trials which are listed on multiple lines have multiple corrections, the net result of the trial is shown on the first line.

The 'Iteration' column indicates the Gauss Seidel iteration the correction was performed after. 'Point in Cycle' is the ratio of the iteration number to the total number of iterations multiplied by a factor of 100. 'Level' indicates the grid level that the correction was made on. 'Direction' refers to the error spike direction (see section 8.1). 'Grid Point' is the grid point which was corrected. 'Location' is the ratio of 'Grid Point' to the total number of grid points on the level multiplied by a factor of 100. 'Residual/Norm' is the ratio of value of the residual of the corrected point prior to the correction to the residual norm (4.6).

'Total Iterations' is the total number of iterations required for convergence (convergence criteria are given in Table 4.2. 'Weighted Iterations' is the equivalent number of level 6 iterations (see section 4.6). '% Δ Iterations' is the percentage change in number of iterations required for convergence (7.2). A negative value indicates a decrease in the total iterations.

'|Error|' is the average of the absolute value of the error of the solution. '% Δ Error' is the percentage difference in error between the trial and trial 0 (7.1). A negative value indicates an increase in accuracy.

'Δ Work' is obtained from subtracting the weighted iterations of the trial from the weighted iterations of trial 0 and adding the value of one iteration on the level of each correction (each correction takes approximately the same time to process as a Gauss Seidel iteration, section 5.2). A positive value indicates an increase in computational work. It should be noted that this is a calculated value and not an actual measure of computation time.

Problem, Trial	Iteration	Point in Cycle (%)	Level	Direction	Grid Point	Location (% span)	Residual/ Norm	Total Iterations	Weighted Iterations	% Δ Iterations	|Error|	% Δ Error	Δ Work (iterations)
p1t0	-	-	-	-	-	-	-	38	16.06	-	4.44E-04	-	-
p1t1	13	35.1	3	Correct	5	71.4	5.1	37	15.94	-0.78	3.92E-04	-11.73	0.00
p1t2	15	39.5	4	Correct	13	86.7	8.1	38	16.06	0.00	4.11E-04	-7.46	0.25
p1t3	17	44.7	5	Correct	27	87.1	10.2	38	16.06	0.00	4.37E-04	-1.49	0.50
p1t4	19	50.0	6	Correct	57	90.5	16.3	38	16.06	0.00	4.42E-04	-0.50	1.00
p1t5	29	78.4	3	Correct	5	71.4	5.0	37	15.94	-0.78	4.29E-04	-3.31	0.00
p1t6	31	81.6	4	Correct	13	86.7	7.7	38	16.06	0.00	4.15E-04	-6.58	0.25
p1t7	33	86.8	5	Correct	29	93.5	10.8	38	16.06	0.00	4.39E-04	-1.06	0.50
p1t8	35	92.1	6	Incorrect	52	82.5	14.9	38	16.06	0.00	4.45E-04	0.23	1.00
p1t9	13	36.1	3	Correct	5	71.4	5.1	36	15.81	-1.58	3.78E-04	-14.75	0.00
p1t9	28	77.8	3	Correct	5	71.4	5.0						
p1t10	13	35.1	3	Correct	5	71.4	5.1	37	15.94	-0.78	3.61E-04	-18.61	0.25
p1t10	14	37.8	4	Correct	13	86.7	8.1						
p1t11	13	35.1	3	Correct	5	71.4	5.1	37	15.94	-0.78	3.62E-04	-18.43	0.25
p1t11	30	81.1	4	Correct	13	86.7	8.4						
p1t12	13	36.1	3	Correct	5	71.4	5.1	36	15.81	-1.58	3.49E-04	-21.44	0.25
p1t12	28	77.8	3	Correct	5	71.4	5.0						
p1t12	29	80.6	4	Correct	13	86.7	8.6						
p2t0	-	-	-	-	-	-	-	27	12.38	-	1.85E-04	-	-
p2t1	11	40.7	4	Incorrect	13	86.7	9.0	27	12.38	0.00	2.03E-04	9.47	0.25
p2t2	13	48.1	5	Correct	29	93.5	16.5	27	12.38	0.00	1.89E-04	1.88	0.50
p2t3	15	55.6	6	Correct	59	93.7	23.9	27	12.38	0.00	1.86E-04	0.52	1.00
p2t4	23	85.2	3	Correct	3	42.9	5.4	27	12.38	0.00	1.69E-04	-8.56	0.13
p2t5	24	88.9	4	Incorrect	13	86.7	6.7	27	12.38	0.00	1.88E-04	1.57	0.25
p2t6	25	92.6	5	Incorrect	29	93.5	15.0	27	12.38	0.00	1.86E-04	0.53	0.50
p2t7	26	96.3	6	Correct	54	85.7	21.6	27	12.38	0.00	1.85E-04	-0.12	1.00
p2t8	11	40.7	4	Incorrect	13	86.7	9.0	27	12.38	0.00	2.06E-04	11.38	0.75
p2t8	12	44.4	5	Correct	29	93.5	16.5						

Problem, Trial	Iteration	Point in Cycle (%)	Level	Direction	Grid Point	Location (% span)	Residual/ Norm	Total Iterations	Weighted Iterations	% Δ Iterations	|Error|	% Δ Error	Δ Work (iterations)
p3t0	-	-	-	-	-	-	-	48	16.94	-	1.0E-04	-	-
p3t1	21	43.8	3	Incorrect	2	28.6	3.3	48	16.94	0.00	9.5E-05	-7.89	0.13
p3t2	22	45.8	4	Correct	7	46.7	6.3	48	16.94	0.00	8.9E-05	-13.70	0.25
p3t3	24	50.0	5	Correct	13	41.9	11.8	48	16.94	0.00	1.0E-04	-2.68	0.50
p3t4	26	54.2	6	Correct	31	49.2	17.0	48	16.94	0.00	1.0E-04	-3.97	1.00
p3t5	37	77.1	3	Incorrect	2	28.6	3.1	48	16.94	0.00	1.2E-04	11.89	0.13
p3t6	39	83.0	4	Correct	3	20.0	8.2	47	16.69	-1.50	1.0E-04	-2.56	0.00
p3t7	42	87.5	5	Correct	7	22.6	12.3	48	16.94	0.00	1.0E-04	-1.19	0.50
p3t8	45	93.8	6	Correct	15	23.8	19.5	48	16.94	0.00	1.0E-04	-0.21	1.00
p3t9	22	45.8	4	Correct	7	46.7	6.3	48	16.94	0.00	9.1E-05	-12.24	0.75
p3t9	24	50.0	5	Correct	13	41.9	11.6						
p3t10	22	46.8	4	Correct	7	46.7	6.3	47	16.69	-1.50	8.6E-05	-16.84	0.25
p3t10	39	83.0	4	Correct	3	20	8.6						
p3t11	22	45.8	4	Correct	7	46.7	6.3	48	16.94	0.00	9.2E-05	-11.11	1.75
p3t11	24	50.0	5	Correct	13	41.9	11.6						
p3t11	26	54.2	6	Correct	31	49.2	17.0						
p4t0	-	-	-	-	-	-	-	32	12.88	-	3.76E-04	-	-
p4t1	12	37.5	3	Correct	3	42.9	5.3	32	12.88	0.00	3.76E-04	-0.10	0.13
p4t2	13	40.6	4	Incorrect	13	86.7	7.1	32	12.88	0.00	3.80E-04	0.93	0.25
p4t3	15	46.9	5	Incorrect	29	93.5	13.0	32	12.88	0.00	3.76E-04	-0.07	0.50
p4t4	17	53.1	6	Incorrect	61	96.8	17.9	32	12.88	0.00	3.76E-04	-0.08	1.00
p4t5	26	81.3	3	Correct	7	100.0	4.7	32	12.88	0.00	3.59E-04	-4.45	0.13
p4t6	28	87.5	4	Correct	15	100.0	7.2	32	12.88	0.00	3.71E-04	-1.39	0.25
p4t7	30	93.8	5	Correct	19	61.3	11.1	32	12.88	0.00	3.75E-04	-0.26	0.50
p4t8	31	96.9	6	Correct	49	77.8	18.6	32	12.88	0.00	3.76E-04	-0.10	1.00

Problem, Trial	Iteration	Point in Cycle (%)	Level	Direction	Grid Point	Location (% span)	Residual/ Norm	Total Iterations	Weighted Iterations	% Δ Iterations	|Error|	% Δ Error	Δ Work (iterations)
p5t0	-	-	-	-	-	-	-	51	20.44	-	5.01E-06	-	-
p5t1	12	23.5	3	Incorrect	3	42.9	3.3	51	20.44	0.00	2.40E-06	-52.05	0.13
p5t2	14	27.5	4	Correct	5	33.3	9.0	51	20.44	0.00	3.5E-06	-30.25	0.25
p5t3	16	31.4	5	Incorrect	13	41.9	13.0	51	20.44	0.00	4.86E-06	-2.97	0.50
p5t4	18	35.3	6	Correct	30	47.6	19.1	51	20.44	0.00	3.80E-06	-24.14	1.00
p5t5	31	60.8	3	Incorrect	2	28.6	3.5	51	20.44	0.00	4.99E-06	-0.26	0.13
p5t6	32	62.7	4	Correct	3	20.0	5.6	51	20.44	0.00	5.0E-06	-0.27	0.25
p5t7	34	68.0	5	Incorrect	4	12.9	10.4	50	19.44	-5.14	4.7E-06	-5.58	-0.50
p5t8	36	70.6	6	Correct	35	55.6	18.4	51	20.44	0.00	4.02E-06	-19.69	1.00
p5t9	45	88.2	4	Correct	2	13.3	7.5	51	20.44	0.00	4.80E-06	-4.11	0.25
p5t10	47	92.2	5	Correct	2	6.5	12.2	51	20.44	0.00	4.94E-06	-1.44	0.50
p5t11	49	96.1	6	Correct	3	4.8	22.0	51	20.44	0.00	5.03E-06	0.38	1.00
p5t12	14	27.5	4	Correct	5	33.3	9.0	51	20.44	0.00	4.2E-06	-16.59	0.75
p5t12	16	31.4	5	Incorrect	13	41.9	12.9						
p5t13	14	27.5	4	Correct	5	33.3	9.0	51	20.44	0.00	5.8E-06	15.65	1.25
p5t13	18	35.3	6	Correct	30	47.6	18.9						
p5t14	14	27.5	4	Correct	5	33.3	9.0	51	20.44	0.00	2.9E-06	-41.35	0.75
p5t14	34	66.7	5	Correct	3	9.7	10.6						
p5t15	14	27.5	4	Correct	5	33.3	9.0	51	20.44	0.00	2.87E-06	-42.75	1.25
p5t15	34	66.7	5	Correct	3	9.7	10.6						
p5t15	47	92.2	5	Correct	2	6.5	11.0						

Problem, Trial	Iteration	Point in Cycle (%)	Level	Direction	Grid Point	Location (% span)	Residual/ Norm	Total Iterations	Weighted Iterations	% Δ Iterations	|Error|	% Δ Error	Δ Work (iterations)
p6t0	-	-	-	-	-	-	-	39	15.69	-	3.93E-04	-	-
p6t1	13	34.2	3	Correct	5	71.4	4.7	38	15.56	-0.80	3.86E-04	-1.83	0.00
p6t2	15	38.5	4	Correct	13	86.7	6.7	39	15.69	0.00	3.99E-04	1.47	0.25
p6t3	17	43.6	5	Correct	29	93.5	8.8	39	15.69	0.00	3.95E-04	0.33	0.50
p6t4	19	48.7	6	Correct	47	74.6	18.3	39	15.69	0.00	4.03E-04	2.55	1.00
p6t5	30	76.9	3	Correct	5	71.4	4.3	39	15.69	0.00	3.38E-04	-14.14	0.13
p6t6	32	82.1	4	Correct	13	86.7	5.7	39	15.69	0.00	3.85E-04	-2.02	0.25
p6t7	34	87.2	5	Correct	13	41.9	9.0	39	15.69	0.00	3.92E-04	-0.36	0.50
p6t8	36	92.3	6	Correct	43	68.3	13.7	39	15.69	0.00	3.93E-04	-0.05	1.00
p6t9	15	38.5	4	Correct	13	86.7	6.7	39	15.69	0.00	3.94E-04	0.08	0.75
p6t9	17	43.6	5	Correct	29	93.5	8.7						
p6t10	15	38.5	4	Correct	13	86.7	6.7	39	15.69	0.00	3.78E-04	-3.79	1.25
p6t10	19	48.7	6	Correct	47	74.6	18.2						
p6t11	30	76.9	3	Correct	5	71.4	4.3	39	15.69	0.00	3.30E-04	-16.15	0.38
p6t11	32	82.1	4	Correct	13	86.7	5.6						
p6t12	15	38.5	4	Correct	13	86.7	6.7	39	15.69	0.00	3.70E-04	-5.80	1.75
p6t12	17	43.6	5	Correct	29	93.5	8.7						
p6t12	19	48.7	6	Correct	47	74.6	18.2						

Problem, Trial	Iteration	Point in Cycle (%)	Level	Direction	Grid Point	Location (% span)	Residual/ Norm	Total Iterations	Weighted Iterations	% Δ Iterations	|Error|	% Δ Error	Δ Work (iterations)
p7t0	-	-	-	-	-	-	-	42	15.56	-	2.29E-03	-	-
p7t1	15	35.7	3	Correct	5	71.4	4.8	42	15.56	0.00	1.86E-03	-18.76	0.13
p7t2	17	40.5	4	Correct	11	73.3	5.4	42	15.56	0.00	2.19E-03	-4.29	0.25
p7t3	19	45.2	5	Correct	13	41.9	9.8	42	15.56	0.00	2.28E-03	-0.44	0.50
p7t4	21	50.0	6	Correct	45	71.4	16.4	42	15.56	0.00	2.29E-03	0.09	1.00
p7t5	32	76.2	3	Correct	3	42.9	3.8	42	15.56	0.00	1.93E-03	-15.62	0.13
p7t6	34	82.9	4	Correct	5	33.3	8.5	41	15.31	-1.63	2.26E-03	-1.36	0.00
p7t7	37	88.1	5	Correct	11	35.5	13.1	42	15.56	0.00	2.28E-03	-0.65	0.50
p7t8	39	92.9	6	Incorrect	16	25.4	16.5	42	15.56	0.00	2.29E-03	0.07	1.00
p7t9	19	45.2	5	Correct	13	41.9	9.8	42	15.56	0.00	2.28E-03	-0.31	1.50
p7t9	21	50.0	6	Correct	45	71.4	16.4	42					
p7t10	19	46.3	5	Correct	13	41.9	9.8	41	15.31	-1.63	2.26E-03	-1.41	0.50
p7t10	34	82.9	4	Correct	5	33.3	7.9	41					
p7t11	34	82.9	4	Correct	5	33.3	8.5	41	15.31	-1.63	2.25E-03	-2.01	0.50
p7t11	36	87.8	5	Correct	11	35.5	12.9	41					
p7t12	15	36.6	3	Correct	5	71.4	4.8	41	15.44	-0.81	1.71E-03	-25.23	0.13
p7t12	32	78.0	3	Correct	3	42.9	3.3	41					
p7t13	15	35.7	3	Correct	5	71.4	4.8	42	15.56	0.00	1.77E-03	-22.79	0.38
p7t13	17	40.5	4	Correct	11	73.3	5.2	42					
p7t14	32	78.0	3	Correct	3	42.9	3.8	41	15.31	-1.63	1.88E-03	-17.95	0.13
p7t14	34	82.9	4	Correct	5	33.3	8.3	41					

Problem, Trial	Iteration	Point in Cycle (%)	Level	Direction	Grid Point	Location (% span)	Residual/ Norm	Total Iterations	Weighted Iterations	% Δ Iterations	|Error|	% Δ Error	Δ Work (iterations)
p8t0	-	-	-	-	-	-	-	41	14.94	-	1.17E-04	-	-
p8t1	13	31.7	3	Incorrect	3	42.9	3.3	41	14.94	0.00	1.78E-04	51.60	0.13
p8t2	15	36.6	4	Incorrect	5	33.3	8.7	41	14.94	0.00	1.4E-04	18.60	0.25
p8t3	17	41.5	5	Incorrect	13	41.9	13.0	41	14.94	0.00	1.16E-04	-0.63	0.50
p8t4	19	46.3	6	Correct	30	47.6	18.7	41	14.94	0.00	1.17E-04	-0.32	1.00
p8t5	33	80.5	3	Incorrect	2	28.6	3.6	41	14.94	0.00	8.70E-05	-25.81	0.13
p8t6	34	82.9	4	Incorrect	7	46.7	6.6	41	14.94	0.00	1.5E-04	23.88	0.25
p8t7	36	87.8	5	Correct	16	51.6	11.8	41	14.94	0.00	1.12E-04	-4.40	0.50
p8t8	38	92.7	6	Incorrect	35	55.6	19.4	41	14.94	0.00	1.18E-04	0.74	1.00
p8t9	15	36.6	4	Incorrect	5	33.3	8.7	41	14.94	0.00	1.3E-04	7.70	0.75
p8t9	17	41.5	5	Incorrect	13	41.9	12.9						
p8t10	15	36.6	4	Incorrect	5	33.3	8.7	41	14.94	0.00	1.4E-04	17.44	1.75
p8t10	17	41.5	5	Incorrect	13	41.9	12.9						
p8t10	19	46.3	6	Correct	30	47.6	18.3						
p9t0	-	-	-	-	-	-	-	30	13.00	-	2.04E-04	-	-
p9t1	11	36.7	3	Correct	3	42.9	4.5	30	13.00	0.00	2.76E-04	35.33	0.13
p9t2	13	43.3	4	Correct	7	46.7	6.9	30	13.00	0.00	2.29E-04	12.58	0.25
p9t3	15	50.0	5	Correct	2	6.5	9.2	30	13.00	0.00	2.03E-04	-0.34	0.50
p9t4	17	56.7	6	Correct	15	23.8	21.5	30	13.00	0.00	2.04E-04	0.23	1.00
p9t5	25	83.3	3	Correct	5	71.4	2.3	30	13.00	0.00	1.88E-04	-7.56	0.13
p9t6	26	86.7	4	Correct	2	13.3	6.8	30	13.00	0.00	2.04E-04	0.08	0.25
p9t7	27	90.0	5	Incorrect	5	16.1	13.9	30	13.00	0.00	2.05E-04	0.57	0.50
p9t8	29	96.7	6	Incorrect	9	14.3	20.7	30	13.00	0.00	2.04E-04	0.16	1.00
p9t9	15	50.0	5	Correct	2	6.5	9.2	30	13.00	0.00	2.03E-04	-0.19	0.75
p9t9	26	86.7	4	Correct	2	13.3	6.7						
p9t10	25	83.3	3	Correct	5	71.4	2.3	30	13.00	0.00	1.89E-04	-7.48	0.38
p9t10	26	86.7	4	Correct	2	13.3	6.8						