Application of Multigrid Methods to the Boltzmann Equation

School of Physical Sciences, Dublin City University, Dublin, Ireland

Study for the higher degree of

Master of Science in Computational Physics

Candidate **P.R.Hayden** *Supervisor* **M.M.Turner**

July, 1995

I hereby certify that the this material, which I now submit for the assessment on the programme of study leading to the award of Masters Degree by Research in Computational Physics is entirely my ownwork and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work

Signed _____ Date _____

Candidate

ACKNOWLEDGEMENTS

I would like to thank Dr Miles Turner, for the guidance he has given me for the duration of my study

Application of Multigrid Methods to the Boltzmann Equation

P.R.Hayden and M.M.Turner

School of Physical Sciences, Dublin City University, Dublin, Ireland

ABSTRACT

A multigrid solver was applied to the simple 1-D Boltzmann equation [21] for the linear case with no electron-electron collisions The findings from this were compared to those of an existing direct solver for the same problem The Boltzmann problem was defined for a N_2 plasma A basic Gauss-Seidel iteration was found to act as the best smoother for the multigrid solver The Galerkin method of restriction was found to work, while the direct method failed Reasons for this are suggested An adaptive method was developed which may slightly improve performance in some cases It was found for 256 points that multigrid only used 20% of the Work Units required by the direct solver Given that the direct LU-decomposition solver requires $^1/_3N^3$ manipulations, and the multigrid Vcycle uses ON^2logN, the method has increasing advantage for larger systems The efficiency also improves with increasing dimension Another important advantage of multigrid is that there should be no considerable loss of efficiency when solving the non linear case which includes electron-electron collisions

Contents

Introduction (Ch 1) 1
- Origin and Form of the Boltzmann Equation 1
- Motivation for using Multigrid 7

General Principles of Multigrid (Ch 2) 8
- Poisson Equation 8
- Boundary Condition 8
- Discretization Error 9
- Iterative Methods 10
- Fourier Discussion 14
- Smoothing Properties 16
- Course Grid Correction Scheme 17
- The Vcycle and other Multigrid Cycles 22
- Performance of the Poisson Equation 27

Performance Results for the Boltzmann Case (Ch 3) 34
- Multigrid Program for the Boltzmann Equation 34
- The Main Program Procedural Flow 34
- Code Testing 35
- The Residual 36
- Efficiency 56

Conclusion 59

Appendix 62
- Appendix A 62
  - List of Multigrid Output Data Files 62
  - List of C Macros used in the Program 63
  - List of C function Prototypes 64
- Appendix B 81
  - Boltzmann Equation 81
- Appendix C 83
  - Multigrid Parameters or Settings 83
  - Program Details 85
  - Program Flow 86
  - Code Testing of the Fourier Function 89
- Appendix D 90
  - Boltzmann Equation 90

References

Introduction (Ch 1)

Simulations of a high pressure gas discharge devices require the electron energy distribution The equation that describes this electron distribution for an isotropic plasma, with a constant applied electric field, and no applied magnetic field, is the 1-D dc Boltzmann equation (shown later as equation -(1 9)) The origin of this form of the Boltzmann equation as in equation -(1 9) will be briefly looked at When the Boltzmann equation is introduced, it is then the task to discuss how the discrete form of the Boltzmann equation can be more efficiently solved using a multigrid solver, as opposed to the previous direct method Thus the motivation behind multigrid theory will be looked at

Origin and Form of the Boltzmann Equation

The Maxwell-Boltzmann distribution, uses the probability distribution function, f(r,v,t) such that it describes either the number or probability of electrons at a specific position vector r(x,y,z), with velocity v(x,y,z), at time t The distribution function is a function of speed in an isotropic gas system, and thus we obtain an energy distribution for the electrons in a gas However, when external forces are applied, an equation of motion for the distribution function is now needed This is where the Boltzmann distribution comes into play This is now a six dimensional problem with time, since with time the number of electrons at dr^3dv^3 will change in accordance to the flow across the walls of this element in phase space The following equation describes the continuity of electrons in six dimensional phase space,

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial r}\frac{\partial r}{\partial t} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial t} = 0 \qquad -(1\ 1)$$

Rewriting, and also considering changes in f which arise from collisions, we obtain the general form of the Boltzmann equation, more commonly written as,

$$\frac{\partial f}{\partial t} + \underline{v}\,\nabla_r f + \frac{\underline{F}}{m}\,\nabla_v f = S \qquad -(1\ 2)$$

It is now necessary to determine the form of the RHS term referred to as the collisional integral Considering the element dr^3dv^3, we know that at equilibrium that the rate of

processes into dr^3dv^3 is the same as the rate of processes out of this element Using the differential scattering cross section $d\sigma(\theta)/d\Omega$, we can determine the flux in and out of dr^3dv^3 The rate of change of electrons in this element can then be integrated over all position and velocity to give the implicit form of the collisional integral as [8]

$$S = \int d^3v_1 \int d\Omega \left[f(r,v',t)F(r,v',t) - f(r,v,t)F(r,v,t)\right] \times |v - v_1| \frac{d\sigma(\theta)}{d\Omega} \qquad -(1\ 3)$$

In this equation, v is the velocity of the particle under consideration and v_1 is the velocity of the other before collision, v' and v_1' the velocities after F is then the distribution function for the second particle This uses the assumption that only binary collisions are important and that the mean distance between collisions is much larger than the range of inter particle forces This is valid for all case except electron-electron collisions

An applied external electric field is then treated as a perturbation Rewriting the Boltzmann equation in terms of polar co-ordinates, and setting z axis as the direction of the applied electric field, the distribution function is then expanded using Legendre Polynomials

$$f(r,v,t) = \Sigma_{k=0} P_k(\cos\theta_l)\ f_k(v_a,r,t) \qquad -(1\ 4)$$

On the RHS, f_k has speed v_a as opposed to velocity Substituting this back into the Boltzmann equation results in an equation with an infinite number of terms Using the treatment by Cherrington[8], these can be reduced to an infinite number of equations, the general term is given in the appendix B Another Legendre Polynomial expansion is used for the collision integral on the RHS Some discussion of the manipulation of this set of infinite equations by Cherrington[8] is shown in appendix B Rewriting the distribution in terms of energy, and normalizing by

$$\int \varepsilon^{1/2} f_0(\varepsilon) d\varepsilon = 1 \qquad -(1\ 5)$$

we get

$$\frac{d}{d\varepsilon}\left[\frac{e^2E^2\varepsilon}{3N\sigma_m}\frac{df}{d\varepsilon}\right] + \frac{2md}{M}\frac{d}{d\varepsilon}\left[\varepsilon^2 N\sigma_m(f_o + kT\frac{df}{d\varepsilon})\right] = 0 \qquad -(16)$$

where N is the density of the neutrals in question, and σ_m is the cross section for elastic collisions As only the average velocity was taken into consideration, and as the plasma is assumed to have a uniform distribution in space, the six dimensional problem of equation -(3 3a) may be reduced to a one dimensional problem in average velocity (or energy) The above form of the Boltzmann equation applies to the case when elastic collisions dominate, and we have ignored inelastic collisions We can formulate the above equation in the similar manner, except this time include the appropriate collision integral term for the inelastic processes However we have assumed that electron-electron collisions have a negligible effect The equation is then given by Lowke, Phelps and Irwin[16] as

$$\frac{d}{d\varepsilon}\left[\frac{e^2E^2\varepsilon}{3N\sigma_m}\frac{df}{d\varepsilon}\right] + \frac{2md}{M}\frac{d}{d\varepsilon}\left[\varepsilon^2 N\sigma_m(f_o + kT\frac{df}{d\varepsilon})\right]$$

$$+ \Sigma_j\left[(\varepsilon+\varepsilon_j)f(\varepsilon+\varepsilon_j)N\sigma_j(\varepsilon+\varepsilon_j) - \varepsilon f(\varepsilon)N\sigma_j(\varepsilon)\right]$$

$$+ \Sigma_j\left[(\varepsilon-\varepsilon_j)f(\varepsilon-\varepsilon_j)N^*\sigma_j(\varepsilon-\varepsilon_j) - \varepsilon f(\varepsilon)N^*\sigma_j(\varepsilon)\right] = 0 \qquad -(17)$$

where N* is the excited state density Equation -(1 7) can be rewritten using a different normalisation given by the relation

$$n(\varepsilon) = f(\varepsilon)\varepsilon^{1/2} n_e \qquad -(18)$$

Obtaining a steady state solution for f(ε) then enables other plasma parameters such as swarm parameters to be obtained These are necessary in order to generate a self consistent set of cross sections from this transport data following the approach of Rockwood [21] After further manipulation we obtain the one dimensional dc Boltzmann equation in finite difference form given by Rockwood [21]

$$\frac{dn_k}{dt} = a_{k-1}n_{k-1} + b_{k+1}n_{k+1} - (a_k + b_k)n_k + \sum_{s\,j} N_s \left[R_{sjk+m}\, n_{k+m_{sj}} + R'_{sjk-m_{sj}}\, n_{k-m_{sj}} \frac{N^j_s}{N_s} \right. \quad \text{-(1 9)}$$

$$\left. + R^{i}_{sk+m_{si}}\, n_{k+m_{si}} + \delta_{1k} \sum_m R^{i}_{sm}\, n_m - (R_{sjk} + R'_{sjk} + R^{i}_{sk})\, n_k \right]$$

with

$$a_k = \frac{2Ne^2}{3m}\left(\frac{E}{N}\right)^2 \frac{1}{v^+_k/N}\left(\frac{1}{\Delta\varepsilon}\right)\left(\varepsilon^+_k + \frac{\Delta\varepsilon}{4}\right) + \frac{\bar{v}_k}{2\Delta\varepsilon}\left(\frac{KT}{2} - \varepsilon^+_k + \frac{2KT}{\Delta\varepsilon}\varepsilon^+_k\right) \quad ,$$

$$b_k = \frac{2Ne^2}{3m}\left(\frac{E}{N}\right)^2 \frac{1}{v^+_k/N}\left(\frac{1}{\Delta\varepsilon}\right)\left(\varepsilon^+_k - \frac{\Delta\varepsilon}{4}\right) + \frac{\bar{v}_k}{2\Delta\varepsilon}\left(\varepsilon^+_k - \frac{KT}{2} + \frac{2KT}{\Delta\varepsilon}\varepsilon^+_k\right) \quad ,$$

-(1 9a)

where $\varepsilon^+_k = k\Delta\varepsilon$, $\varepsilon^-_k = \varepsilon^+_{k-1}$ -(1 9b)

$$\frac{v^+_k}{N} = \left(\frac{2\varepsilon^+_k}{m}\right)^{1/2} \sum_s q_s \sigma_s(\varepsilon^+_k) \quad , \qquad \bar{v}^+_k = 2mN\left(\frac{2\varepsilon^+_k}{m}\right)^{1/2} \sum_s \frac{q_s \sigma_s(\varepsilon^+_k)}{M_s} \quad ,$$

-(1 9c)

Equation -(1 9) gives a set of k-coupled ordinary differential equations which were obtained by finite differencing the continuous form Energy space has been divided up into 2^q - 1 points, where each k^{th} point is separated by the energy interval $\Delta\varepsilon$ We can interpret a_k as the rate at which electrons of mass m, charge e and energy ε_k are promoted to energy ε_{k+1}, while b_k is the rate for demotion from ε_k to ε_{k-1}, due to applied constant field E The summation term on the right involves collisions R_{sjk}, R'_{sjk} and R^{i}_{sk} are rates at which electrons at energy ε_k suffer elastic collisions, super elastic collisions and creation through ionisation of the neutral molecules in the plasma The subscript j denotes excitation or de-excitation of the molecules N^j_s of species s to state j The cross section for momentum transfer from electrons at energy ε_k^+ to molecules N_s of species s is denoted by $\sigma_s(\varepsilon_k^+)$, and q_s is the mole fraction of species s M_s is the mass of species s and T is the gas temperature

This ignores electron-electron collisions The distribution function n(ε,t) then gives the number of electrons at energy ε and time t Obtaining a steady state solution for n(ε) then enables other plasma parameters such as swarm parameters to be obtained Among these include electron temperature, characteristic energy, drift velocity, diffusion coefficient and mobility From these it is then possible to generate a self consistent set of cross sections from this transport data following the approach of Rockwood [21]

The set of k-coupled equations can be written in matrix form as

$$\mathbf{An} = \mathbf{n}' \qquad \text{-(1 10)}$$

where **A** is an NxN sparse matrix, **n** = **n**(ε) is an Nx1 vector giving the probability distribution, and **n'** is an Nx1 vector giving the rate of change of **n** for different energies on the grid The existing solver, written in C on UNIX work stations, uses an implicit Euler algorithm, which effectively integrates forward in time to a steady state solution This uses the direct method of LU-Decomposition (LUD) However, multigrid grid can be applied to the steady state problem

$$\mathbf{An} = \mathbf{n}' = 0 \qquad \text{-(1 11)}$$

Because the RHS is zero, there is an infinite series of solutions that differ to within a constant The direct LUD solver gives a solution as **n**(ε)=0, for every point on the grid It cannot give a non zero solution unless there exists a non zero element on the RHS It is physically possible that **n**(ε)=0 is a solution However, if the problem is given in terms of the normalization **f**(ε) instead of **n**(ε), it is not physically possible that **f**(ε)=0 is a solution, because even thought there are no electrons in this system, there still exists a probability distribution, regardless of the number of free electrons in the plasma Thus for **Af**(ε) = 0, there must be some non zero element on the RHS that prevents the non realistic LUD solution **f**(ε)=0 from occurring The boundary conditions for the lower energy boundary of the grid must be non zero if **f**(ε) is chosen However, the grid system only takes into consideration the 2^q-1 internal grid points, and not the actually boundaries of **f**(ε) Appendix D gives the new version of equations -(1 9, 1 9a, 1 9b, 1 9c) written in terms of **f**(ε) instead of **n**(ε)

The first row by column multiplication of **Af** would be:

$$a_0f_0 + b_2f_2 - a_1f_1 - b_1f_1 + T = 0 \qquad \text{-(1 12)}$$

where T is all other off diagonal elements associated with collisions

If we consider the k=1 rate equation under the new normalisation, the very first term is $a_0 f_0$, which is not taken into consideration by the matrix or the boundaries, since equation -(D4a) (given in appendix D) for $\mathbf{f}(\varepsilon)$, like -(1 9a) for $\mathbf{n}(\varepsilon)$ gives $a_0=0$ However, since electrons exist with no energy, or energies between the 0th and the 1st energy points, then it is physically possible that a_0 may be non zero, ignoring the definition from equation -(D4a) This term can thus be transferred over to the RHS, making this non zero Equation -(1 12) would then become

$$b_2 f_2 - a_1 f_1 - b_1 f_1 + T = -a_0 f_0 \qquad -(1\ 13)$$

Since this contains f_0, we can set this boundary condition, and so fix $\mathbf{f}(\varepsilon)$ such that it is non zero The mathematical effect of setting the first term on the right hand side to a non zero value is such to give a non zero solution Thus there are two possible systems to solve, the $\mathbf{n}(\varepsilon)$ and the $\mathbf{f}(\varepsilon)$ However, the difference between the results from these is quite small, as will be observed later

Motivation for using Multigrid

Simulations of high pressure gas discharge devices require a solution to the 1-D dc Boltzmann equation in finite difference form (given by equation -(1 9)) This equation, in matrix form as equation -(1 10) **An = n'** , is solved by an existing program which uses a direct method called LU-Decomposition Direct methods, of which Gaussian elimination is the prototype, determines a solution exactly (up to machine precision) in a finite number of arithmetic steps The main objective of the multigrid approach is to reduce the computational work, and therefore reduce the time required to solve partial differential equations, exploiting the fact that the solution need not be exact In a finite difference problem, there is no advantage in reducing the error in the solution to smaller than that of the discretization error In the direct method, the error is reduced to below the size of the discretization error Relaxation methods however, as they converge onto the solution, gradually become more accurate with increasing iteration or cycle, and the method can be stopped when the error reaches the size of the discretization error Given that this is the main reason for the employment of a relaxation method, we will now introduce the multigrid method, and the principles behind its use

General Principles of Multigrid (Ch 2)

Poisson Equation

Many textbooks on multigrid illustrate the method using the simple Poisson equation in one [7,12,18,23] or two [11,14,19] dimensions In order to discuss and introduce the method, it is better to briefly outline the Poisson system in one dimensions as

$$-u(x)'' = f(x) = 0 \qquad -(2\ 0)$$

For simplicity, it is possible to divide the interval in x between boundary points u(0) and u(1) into N-1 internal mesh points each of separation $\Delta x=1/N$ The finite difference equivalent to equation -(2 0) is

$$-v_{i-1} + 2v_i - v_{i+1} = f(x_i) \qquad (0 < i < N) \qquad -(2\ 1)$$

with $x_i = i\Delta x$ and v_i as the approximation to the solution u_i at mesh position x_i For simplicity we can set both boundaries u_0 and u_N to zero This system of i equations can be written as

$$\mathbf{Av} = \mathbf{f} \qquad -(2\ 2)$$

This symmetric block tridiagonal sparse matrix can then be solved by several direct methods such as Gauss-Jordan Elimination, Gauss Elimination and LU Decomposition [10,2] among others However, multigrid uses iterative methods, which will now be discussed

Boundary Condition

As stated the simple boundary conditions $u_0 = u_N = 0$ are chosen first for simplicity However, other conditions can be considered The commonly seen Dirichlet boundary condition is $u_N = g$ or $u_0 = g$ The Neumann boundary condition involves the slope at the boundary, $du/dx = g$ A mixed boundary condition may also be considered

$$au_N + du/dx|_N = g \qquad -(2\ 3)$$

where a is a constant

Discretization Error

The global error E can be defined as the difference between exact continuous solution and the solution to the finite difference problem

$$E = \|u(x) - u_i\| \qquad -(2\ 4)$$

Equation -(2 1) assumes that the higher terms in the Taylor expansion of equation -(2 0) can be ignored Hackbusch [12] has discussed the size of this in depth, but this agrees in general with simpler approach of Briggs and McCormick [18], which involves the truncation error t_i given in the continuous equation,

$$Au(x) = \Delta x^{-2}\{ -u(x_{i-1}) + 2u(x_i) - u(x_{i+1}) \} + t_i \qquad -(2\ 5)$$

A is the operator taken at position i on the continuous system If we assume that the exact solution u(x) admits a Taylor series expansion about x_i, then t_i may be written as [18]

$$t_i = \Delta x^2/12(-f''(x'_i)) \qquad -(2\ 6)$$

where x'_i denotes a point between x_{i-1} and x_{i+1} It is possible to bound f''(x) using B by,

$$\| f''(x) \| <= B \| f \| \qquad -(2\ 7)$$

We must assume f is sufficiently smooth Then it is possible to bound the truncation error by

$$\| t_i \| <= (\Delta x^2/12) B \| f \| \qquad -(2\ 8)$$

Equation -(2 4) gives a relationship between the discretization error E and this truncation error t_i as

$$AE = t_i \qquad -(2\ 9)$$

which then gives $E <= A^{-1} t_i$, which has been shown [18] to give

$$E <= C \| f \| \Delta x^2/(12\Pi^2) \qquad \text{-(2 10)}$$

where C ~ B This gives the bound for E as [1,7,18,23]

$$E <= O(\Delta x^2) \qquad \text{-(2 11)}$$

The general bound can be $E <= O(\Delta x^p)$ [7] where p is the order of the differential equation in question

Iterative Methods

The two most commonly used 'classical' iterative methods can be found in most numerical, statistical and matrix mathematics texts These are the Jacobi and Gauss-Seidel iterations, which for about one century were the only tools for solving small linear systems iteratively These can be described as operating on a one dimensional system of matrix form **Av** = **b**, giving the i^{th} term as

$$v_i = \frac{1}{a_{ii}} \{ b_i - \sum_{j=1}^{i-1} a_{ij} v_j - \sum_{j=i+1}^{N} a_{ij} v_j \} \qquad (0 < i < N) \qquad \text{-(2 12)}$$

They are also called linear stationary methods [1,12,18] However, the difference between these methods is that the Jacobi method employs two separate **v** arrays and doesn't overwrite existing elements in the **v** array but the Gauss-Seidel does overwrite the previous elements in the **v** array as we sweep over i The storage required for Gauss-Seidel is only one **v** array and is thus less than that of the Jacobi These methods are also known as relaxation methods and involve an initial guess at the solution They then proceed to improve the current approximation by a succession of simple updating steps or iterations, e g sweeps over i for equation -(2 12) The sequence of approximations **v** which is generated, should converge to the exact solution **u** of the finite difference problem This convergence depends on whether a physical solution exists to the corresponding analytical system, which can be tested by the existence of $\mathbf{A}^{-1}$ Matrix **A** must be invertable if a solution is to exist Splitting or decomposing matrix **A** into upper, lower and diagonal parts giving **U**, **L** and **D** respectively, we have

$$\mathbf{Av} = (\mathbf{D} - \mathbf{L} - \mathbf{U})\mathbf{v} = \mathbf{b} \qquad \text{-(2 13)}$$

Then the Jacobi condition for convergence can be found from

$$\mathbf{v}_{new} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{v}_{old} + \mathbf{D}^{-1}\mathbf{b} \qquad -(2\ 14)$$

with

$$\mathbf{P}_J = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}) \qquad -(2\ 15)$$

where $\mathbf{P}_J$ is the Jacobi matrix If written in the form of

$$\mathbf{v}_{m+1} = \mathbf{v}_m - \mathbf{B}(\mathbf{A}\mathbf{v}_m - \mathbf{b}) \qquad -(2\ 16)$$

then **B** is referred to as a preconditioner The subscript m denotes the iteration number The choice of preconditioner can then dictate the convergence properties [1] Equation -(2 15) must satisfy the condition [1,2,12,14,18,20,23]

$$\| \mathbf{P}_J \| < 1 \qquad -(2\ 17)$$

where the Euclidean norm of the Jacobi matrix $\mathbf{P}_J$ is taken,

$$\| \mathbf{P}_J \| = (\Sigma\Sigma | \mathbf{P}_{ij} |^2)^{1/2} \qquad -(2\ 18)$$

Brambles [1] has rigourously shown using the smallest and largest eigenvalues that for this condition to be satisfied, the eigenvalues of $\mathbf{P}_J$, $\lambda(\mathbf{P}_J)$ must satisfy

$$|\lambda(\mathbf{P}_J)|_{max} < 1 \qquad -(2\ 19)$$

where $|\lambda(\mathbf{P}_J)|_{max}$ is the largest absolute eigenvalue and is called the spectral radius or the convergence factor [2,7] In a Similar fashion, the convergence condition for Gauss-Seidel is found from rewriting equation -(2 12) as

$$\mathbf{v} \leftarrow (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\,\mathbf{v} + (\mathbf{D} - \mathbf{L})^{-1}\mathbf{b}$$

or

$$\mathbf{v} \leftarrow \mathbf{P}\mathbf{v} + \mathbf{g} \qquad -(2\ 20)$$

with

$$\mathbf{P}_g = (\mathbf{D} - \mathbf{L})^{-1}\,\mathbf{U} \qquad \text{-(2 21)}$$

where the arrow [1,2,7] indicates replacement of v_i as we sweep over i in an iteration given by equation -(2 1) The requirement is again that

$$\|\mathbf{P}_g\| < 1 \qquad \text{-(2 22)}$$

or

$$|\lambda(\mathbf{P}_g)|_{max} < 1 \qquad \text{-(2 23)}$$

A hand waving argument for the conditions in equations -(2 17) and -(2 22) is given by Briggs [7] from the general form of equation -(2 20) which is, $\mathbf{v}^{(1)} = \mathbf{P}\mathbf{v}^{(0)} + \mathbf{g}$ where the superscript denotes the number of iterations This is the general form of methods referred to as 'basic iterative methods', to which the Jacobi and Gauss-Seidel belong The exact solution is **u** and $\mathbf{u} - \mathbf{v}^{(n)} = \mathbf{e}^{(n)}$ is the error after n iterative steps Using equation -(2 20), we can write

$$\mathbf{u} = \mathbf{P}\mathbf{u} + \mathbf{g} \qquad \text{-(2 24)}$$

Subtracting -(2 20) from -(2 24) gives $\mathbf{e}^{(1)} = \mathbf{P}\mathbf{e}^{(0)}$, which after n iteration sweeps becomes,

$$\mathbf{e}^{(n)} = \mathbf{P}^n\,\mathbf{e}^{(0)} \qquad \text{-(2 25)}$$

and thus we can write

$$\|\mathbf{e}^{(n)}\| <= \|\mathbf{P}\|^n\,\|\mathbf{e}^{(0)}\| \qquad \text{-(2 26)}$$

This leads to the conclusion that if $\|P\| < 1$, then the error is forced to zero as the iteration proceeds This leads to the definition of the contraction number z [1,12,14,18,20,23], which is a function of iteration number n,

$$\|\mathbf{e}^{(n+1)}\| <= z^{(n)}\,\|\mathbf{e}^{(n)}\| \qquad \text{-(2 27)}$$

If the limit of the norm $\|\mathbf{P}\|$ with n goes to zero, then the method is convergent However, this is only a sufficient condition [2,23] A necessary condition is that the spectral radius of **P** is less than 1 [2,7,23] Other sufficient conditions are

1) Diagonal dominance [2]

$$|a_{ii}| > \Sigma |a_{ij}| \qquad -(2\ 28)$$

2) **A** is real and positive definite

$$\mathbf{x}^T\mathbf{A}\mathbf{x} > 0 \qquad -(2\ 29)$$

3) **A** is an M-matrix [2] (This was not used)

The Poisson matrix **A** wasn't diagonally dominant, was real positive definite and more importantly it's spectral radius was found to be less than 1 (as expected) The error in the approximation can be calculated after each iteration via $\mathbf{u} - \mathbf{v}^{(n)} = \mathbf{e}^{(n)}$, and we can obtain an error norm

$$\|\mathbf{e}\|^2 = (\Sigma |\mathbf{e}_i|^2)^{1/2}$$

and plot this against iteration as in figure -(2 0)

However, since we don't normally possess the solution u, it is more practical to obtain the residual norm $\|\mathbf{r}\|^2$ where the residual **r** is defined in the residual equation

$$\mathbf{r} = \mathbf{A}\mathbf{u} - \mathbf{A}\mathbf{v} \qquad -(2\ 30)$$

Hackbusch [1] has demonstrated using the iteration matrix M, $\mathbf{v}^{m+1} = M(\mathbf{v}^m)$, that there is no fundamental distinction between convergence estimates for the error and those for the residual, thus calculation of the residual will suffice Different types of Jacobi or Gauss-Seidel iterations appear if we sweep over i backwards from N-1 to 1, this will be referred to as a backward iteration, and the initial sweep as a forward iteration Alternating between these gives a symmetric iteration Another effective alternative in the case of the Gauss-Seidel is to update all the even components first by the expression

$v_{2i} = 1/a_{2i2i}\{b_{2i} - \Sigma a_{2ij}v_j - \Sigma a_{2ij}v_j\}$ and later sweeping over the odd points,

$v_{2i+1} = 1/a_{2i+1\,2i+1}\{b_{2i+1} - \Sigma a_{2i+1j}v_j - \Sigma a_{2i+1j}v_j\}$

Thus all even points are updated first, then all the odd points are updated This is commonly referred to as a Red-Black iteration This has a clear advantage in terms of implementation on a parallel computer Thus forward, backward, Red-Black and ordinary/simple Gauss Seidel iterations can be combined together to give the optimum results

Forward Red-Black Gauss-Seidel gives the best reduction in the residual norm or error norm with increasing iterations Gauss-Seidel was chosen instead of Jacobi, because of lower storage cost, and other reasons that will be discussed later

Plotting the residual norm for 100 iterations (figure -(2 1) we see that after around 10 or less iterations, the residual flattens, as does the error norm in figure -(2 0)

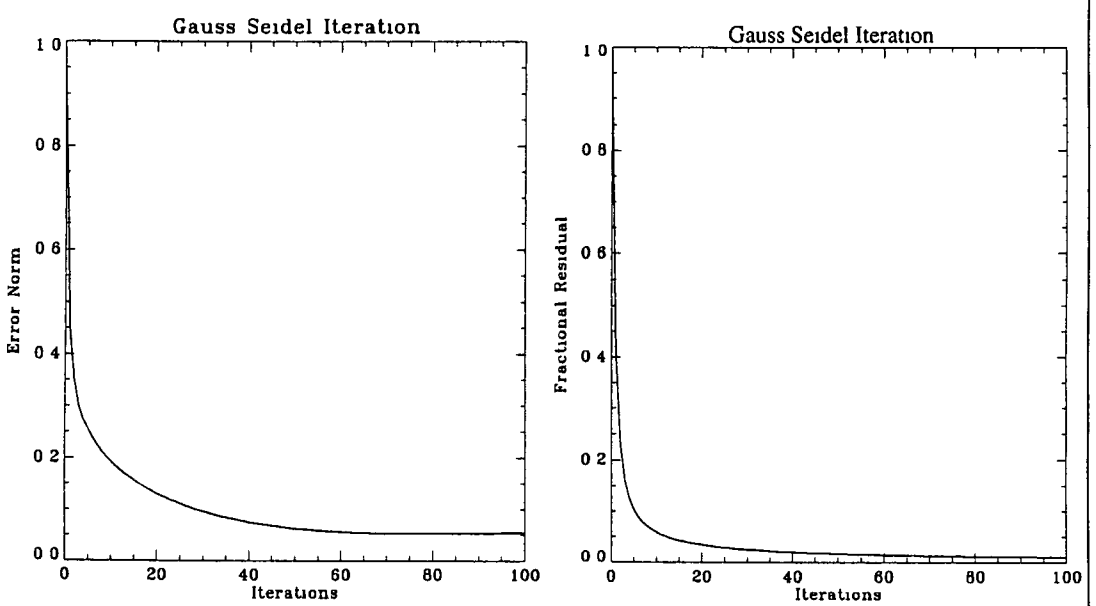


Figure -(2 0) shows the reduction in error norm and Figure -(2 1) shows the reduction in the fractional residual, both for 100 iterations

This is a limitation in most iterative methods By considering the Fourier aspects of the error as in the next section, we can develop ways to overcome this

Fourier Discussion

If we consider the basis set of eigenvectors of the difference equation matrix **A**, any function in the space defined by the problem can be written in terms of this basis The error or residual which are both linear, can be written in terms of these basis

Nearly all multigrid texts [1,7,12,14,18,20,23] choose the Poisson case because it has the advantage that its basis vectors are well known, and most importantly they can be written as

$$v(x_l) = \Sigma_k \sin(^{lk\pi}/_N) \qquad -(2\ 31)$$

where l is the point on the grid between 0 and N, and k is the mode This is in fact a straight forward sine series, and can be used to describe the error Fortunately, this corresponds to a sine Fourier analysis and from this we can see how the error behaves with iteration, or more precisely, the relative strength of each of these eigenvector elements can be observed with increasing iteration, due to the coincidence that these eigenvectors for the Poisson case are in fact a sine series Another reason the sine Fourier transform was chosen as opposed to the cosine or fast Fourier transform was because of the zero periodic boundary conditions, $v_0 = v_N = 0$ For the simple Poisson case, Briggs and McCormick [7,18] have exploited the fact that the exact solution is u(x)=0 for all mesh points Thus plotting $v(x_l)$ gives the error at each mesh point, shown in figure -(2 2), after 100 iterations (notice only the lowest mode remains, e g a sine wave)

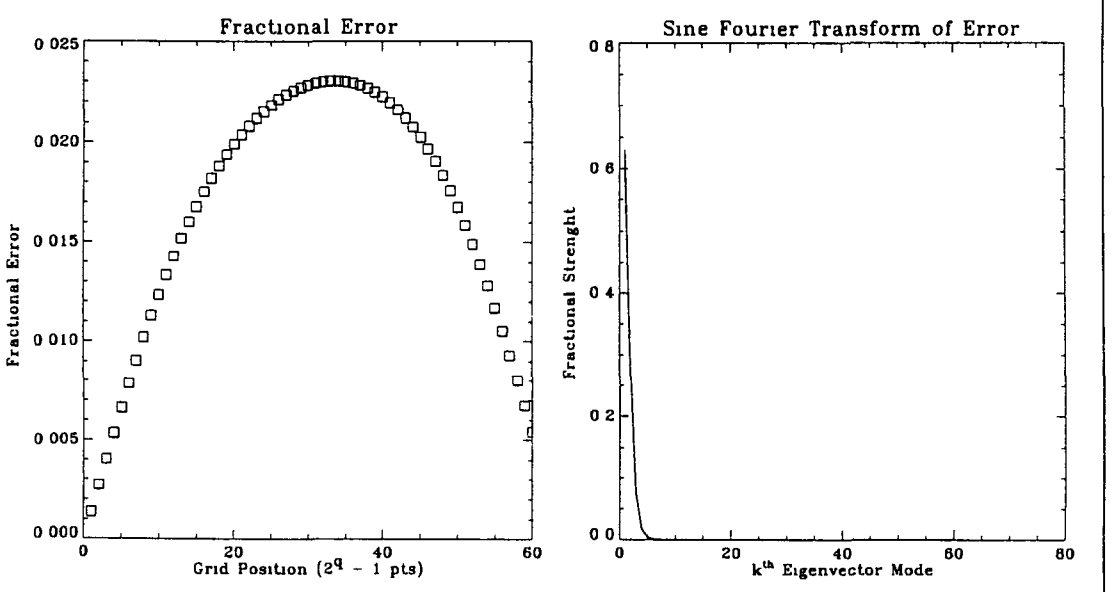


Figure -(2 2) shows the error at each grid point, and figure -(2 3) shows the sine Fourier transform of the error, both after 100 iterations Both suggest that only the low frequency errors remain, the lowest being the strongest, (a simple sine wave)

The magnitude of each mode k is given by the sine Fourier transform function [10] (given in appendix A), and the result after 100 iterations is plotted in figure -(2 3), where an equal amount of each mode was initially present (seen in the Fourier

transform as a straight line) It is then possible to control the initial guess in terms of frequency content We can then set the initial guess to contain only one frequency We can then repeat this for all the frequencies or modes, and observe how each is reduced with increasing iterations Figure -(2 4) shows the number of iterations required to reduce the residual to 1% of the original

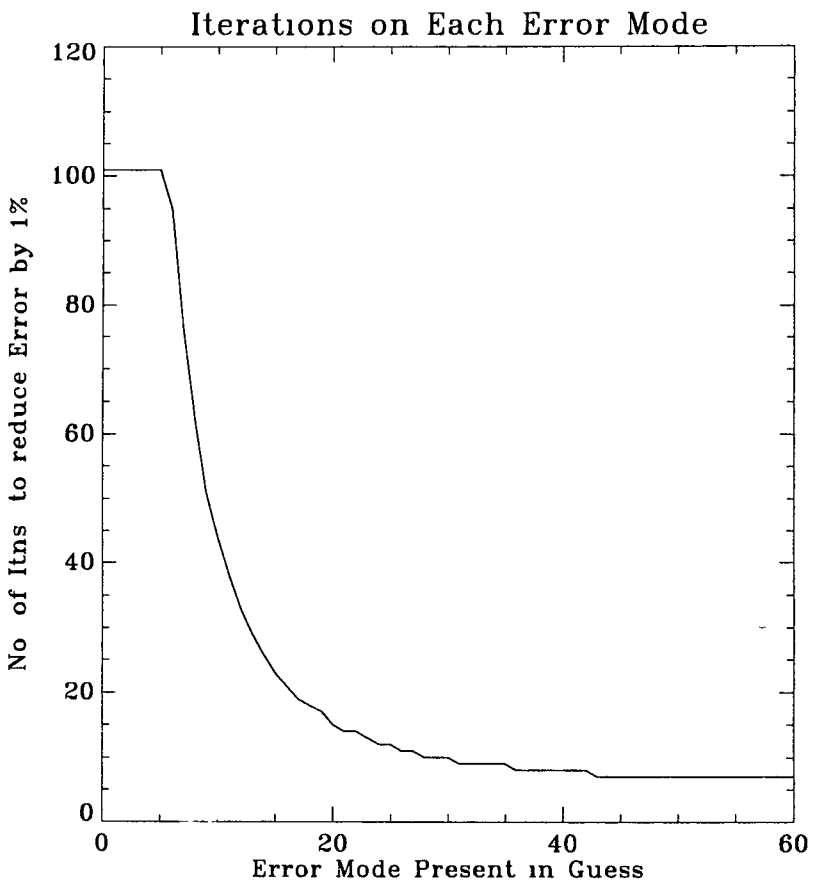


Figure -(2 4) shows the number of iterations required to reduce the error, for each individual error mode separately Less iterations are required to remove the high frequency error modes, where as more are needed for the lower frequencies

From these figures, it becomes evident that the lower k modes or 'smooth' modes are difficult to remove, and this is in fact the reason why the residual or error fails to be reduced further, once the higher k modes or 'oscillatory' modes have been removed

Smoothing Properties

This property that removes the high frequency modes but leaves the smooth, is referred to as the smoothing property Thus our concern would be to find a mechanism to reduce these smooth modes As we shall see later, this is achieved by the Coarse

Grid Correction Scheme Any iterative method that reduces the high frequency error more so than lower frequencies is referred to as a 'smoother' The Gauss Seidel is the most convenient and is often a very effective smoother [12] In order for the Jacobi method to perform as a smoother, it must be weighted or damped [7,12,18], that is, instead of replacing each new point v_i with the new calculation, we add some of the new to the old approximation for this point It will be explained later why it is essential to have a smoother in a multigrid method Hackbusch [12] has defined a smoothing number σ_L which is close to the spectral radius, and can thus be used to approximate this for small iterations, as the eigenvalues are not readily known, and there calculation may prove as costly as the original problem This is a function of iteration number n,

$$\sigma_L(n) = \sup {}^{\|AS\|}/_{\|A\|} \qquad -(2\ 32)$$

Where **S** is a linear transform matrix such that, $e^{m+1} = S(e^m,b)$ Initially, $\sigma_L(0) = 1$ **S** can be obtained after several iterations, thus this should indicate after several iterations how the smoothing number decreases, as less high frequency modes are present A smoothing rate σ_b was introduced by Brandt [3], in terms of the k^{th} eigenvalue λ_k of **S**,

$$\sigma_b(n) = \sup_{max}\{\ |\lambda_k|(n) \quad {}^{N}/_{2} + 1 < k < N-1\ \} \qquad -(2\ 33)$$

This also has an original value of 1 and decreases with iteration

Coarse Grid Correction Scheme

So far we have established that many standard iterative methods posses the smoothing property However the objective is to eliminate all frequency elements in the error (and thus the entire error) A good initial guess could improve the relaxation scheme, and a well known technique for obtaining this is to perform some preliminary iterations on a coarse grid, and then use this as the initial guess on the fine grid

Instead of the sine fourier transform, the fast fourier transform was used to described the error after m iterations as

$$e^m_j = \sum_{\alpha=0}^{2n-1} c^m_\alpha e^{i(j\theta\alpha)} \qquad \text{with} \qquad \theta_\alpha = \pi\alpha / n \qquad -(2\ 34)$$

Orthogonality of $\{e^{l(j\theta\alpha)}\}$ [1,7,12,14,18,20,23] allows us to define an amplification factor [23] $g(\theta_\alpha)$, between each m^{th} iteration as

$$c^m{}_\alpha = g(\theta_\alpha)c^{m-1}{}_\alpha \quad \text{and}$$

$$g(\theta_\alpha) = e^{l(j\theta\alpha)}/(2- e^{l(j\theta\alpha)}) \qquad -(2\ 35)$$

Wesseling [23] reported that

$$|g(\theta_\alpha)| = (5 - 4\cos\theta_\alpha)^{1/2} \qquad -(2\ 36)$$

and giving the interval as $h = 1/N$

$$_{max}|g(\theta_\alpha)| = |g(\theta_1)| = (1 + 2\theta_1^2 + O(\theta_1^4))^{1/2}$$

$$= 1 - 4\pi^2 h^2 + O(h^4) \qquad -(2\ 37)$$

this is in general agreement with the spectral radius or convergence factor behaving as [7,12,18] $1-O(h^2)$ It is possible that Successive Over Relaxation (SOR) method may achieve $_{max}|g(\theta_\alpha)| \sim 1-O(h)$ for the Gauss-Seidel and Jacobi methods Both would then suggest increasing h would give better convergence Also, iterating on a coarser grid (with increased h) would prove less expensive since there are fewer unknowns to be updated Recalling that most relaxation schemes discussed, have the common flaw of preserving the low frequencies or smooth components in the error Assume that only modes below $k = N/2$ are left The even points on the fine grid would be described on the coarse grid as

$$e_i = \sin\left(\frac{2ik\pi}{N}\right) = \sin\left(\frac{ik\pi}{N/2}\right) \qquad 0 < k < N/2 \qquad -(2\ 38)$$

This states that the k^{th} mode on the fine grid (denoted by grid h) appears as low frequency or smooth on this grid, but the same k^{th} mode appears as a higher frequency mode on the coarse grid (denoted by 2h) There was no reported advantage in using grid spacings with ratios other than two [7,23], between fine and coarse grids This ratio of two was found to be almost universal practice These higher frequency modes on the coarse grid can then be eliminated by iterating on that grid.

This reasoning gives the advantage of either starting on, or transferring down to a lower grid (restriction), and then transferring back to the fine grid (interpolation or

prolongation) Both of these terms are common in numerical analysis However, should any high frequency modes remain after iterations on the fine grid, ($k > N/2$), they are misrepresented on the coarse grid 2h as the $(N - k)^{th}$ mode, in a phenomenon known as aliasing The effects of this can be later removed on the fine grid

We now make use of a very important relationship between the original problem, and the residual equation, which is that, relaxation on the original equation $\mathbf{Av} = \mathbf{f}$ with an arbitrary initial guess is equivalent to relaxing on the residual equation $\mathbf{Ae} = \mathbf{r}$ with the specific initial guess $\mathbf{e} = 0$ [7,12,18,23] It was this connection that prompted the combination of this correction scheme and the transfer to the coarse grid

First, we iterate to remove high frequency modes then obtain the residual from $\mathbf{f-Av} = \mathbf{r}$ We then proceed to transfer the fine grid equation $\mathbf{A}^h\mathbf{e}^h = \mathbf{r}^h$ to the coarse grid to give $\mathbf{A}^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ Further iterations on this equation will eliminate the high frequency modes on this coarse grid We then interpolate the error $\mathbf{e}^{2h}$ back onto grid h as $\mathbf{e}^h$, and add it to the approximation $\mathbf{v}$ giving

$$\mathbf{v}^h = \mathbf{v}^h + \mathbf{e}^h \qquad -(2\ 39)$$

It now remains to be seen how we express or define grid transfer, both restriction to lower and interpolation to higher grids Linear or direct restriction would simply involve transferring all the even points of $\mathbf{e}^h$ on the fine grid h to the points on the coarse grid 2h, to give $\mathbf{e}^{2h}$, diagramatically shown in figure -(2 5), taken from Briggs [7]

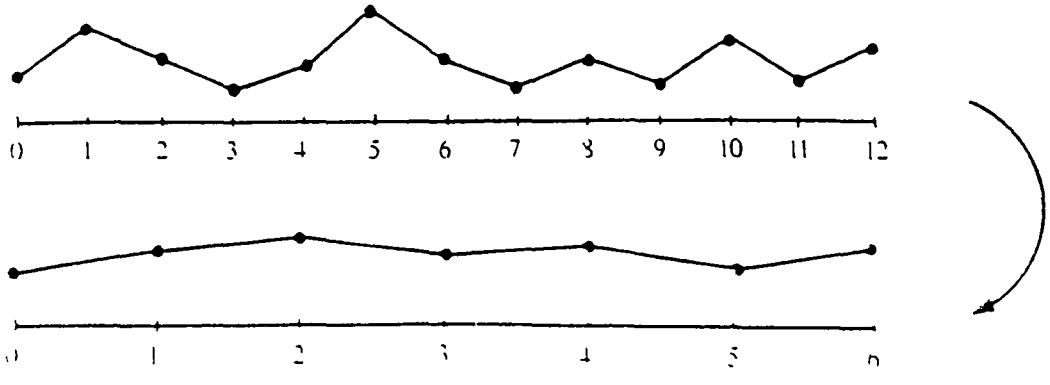


Figure -(2 5) shows restriction by full weighting of a fine vector to the coarse grid

The general routine for this procedure is given by the C function **Restrict**() in appendix A, which allows for the full weighting, or averaging between surrounding points on the fine grid h Two possible approaches are possible, and the differences will be discussed in the appendix C,

$$e^{2h}_{i} = 1/4(v^{h}_{2i-1} + 2v^{h}_{2i} + v^{h}_{2i+1}) \qquad 0 < i < N/2 \qquad -(2\ 40)$$

We can vary the degree of averaging of each of the surrounding points and this is known as weighted restriction

For transferring the error e^{2h} from coarse to fine grid, the function **Interpolation**() shown in appendix A, uses linear interpolation given by

$$e^{h}_{2i} = e^{2h}_{i} \qquad 0 <= i <= N/2 - 1 \qquad -(2\ 41)$$

$$e^{h}_{2i+1} = 1/2(e^{2h}_{i} + e^{2h}_{i+1}) \qquad -(2\ 42)$$

Briggs [7] has graphically illustrated this by figure -(2 6), where the black dots are e^{2h}, and the white dots are the interpolated points on grid h

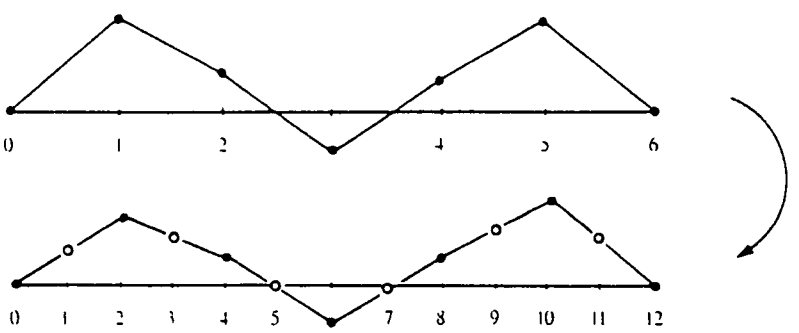


Figure -(2 6) shows interpolation of a vector on the coarse grid to the fine grid The white circles are the interpolated points

In order for this linear interpolation to work, it is important that e^{h} should appear as a smooth vector on grid h with smooth modes as in figure -(2 7a), (black and white dots), since high frequency modes cannot be represented on the coarse grid 2h except

by aliasing, and thus interpolation (between black points) would give a bad approximation (for where the white dots should be), as shown in figure -(2 7b) from Briggs [7]

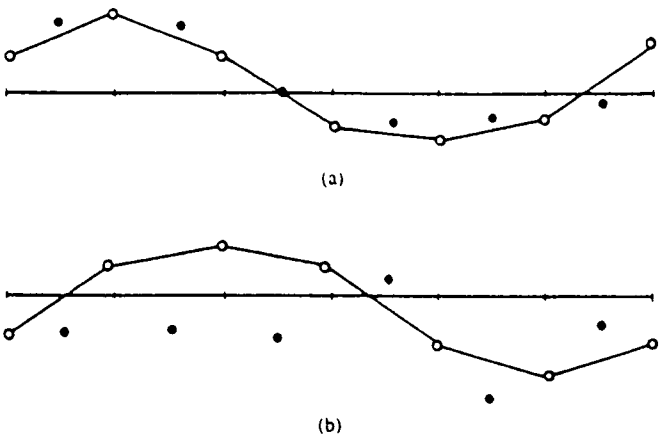


Figure -(2 7a) shows good interpolation, when the error on the fine grid is smooth The interpolated points (white circles) are a good approximation Figure -(2 7b) shows a bad interpolation, occurring because the error on the fine grid is oscillatory

We now need to define the coarse grid matrix operator $\mathbf{A}^{2h}$, which describes the finite difference Poisson problem on the coarse grid 2h From equation -(2 2), which describes the fine grid operator $\mathbf{A}^{h}$, it becomes apparent that the interval Δx should change from h to 2h, e g effectively double, since there are now only $N/2 - 1$ internal mesh points on the grid 2h The grid on which a particular vector applies to is denoted in the superscript This approach simply calculates $\mathbf{A}^{2h}$ as if it were the initial problem, except with interval 2h and $N/2-1$ internal grid points This is referred to as the direct approach Another way of representing **A** on grid 2h is by the Galerkin [1,7,12,18,19,23] representation This can be written in terms of functions as

$$\mathbf{A}^{2h} = \mathbf{Restrict}(\mathbf{A}^{h})\mathbf{Interpolate}() \qquad -(2\ 43)$$

where the open brackets indicate that this operator will work on vector to the right This is more commonly written as

$$\mathbf{A}^{2h} = \mathbf{I}_{h}^{2h}\,\mathbf{A}^{h}\,\mathbf{I}_{2h}^{h} \quad \text{or} \qquad -(2\ 44)$$

$$\mathbf{A}^{2h} = \mathbf{R}\mathbf{A}^{h}\mathbf{P} \qquad -(2\ 45)$$

where we have defined the two operators in matrix form, **R** and **P** for restriction and interpolation respectively This Galerkin method involves defining two matrices **P** and **R**, thus using up extra storage, but does not require redefining the whole problem matrix **A** on each level The Galerkin method may therefore costs less CPU time Since almost all of the non diagonal elements in the matrices **P** and **R** are zero, it may also be possible to make the product **RAP** an operation as opposed to matrix multiplication, thus further saving on CPU time

It is now possible to present a general form for the coarse grid correction scheme [1,2,8,52,53,54] using the **Restrict()** and **Interpolation()** operators

1) Iterate ν_1 times on $\mathbf{A}^h\mathbf{f}^h = \mathbf{v}^h$ on the fine grid h, with the initial guess $\mathbf{v}^h$

2) Compute the residual for coarse grid 2h, $\mathbf{r}^{2h} = \mathbf{Restrict}(\mathbf{r}^h = \mathbf{f}^h - \mathbf{A}^h\mathbf{v}^h)$

3) Iterate ν_2 times (or in fact solve) $\mathbf{A}^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ on the coarse grid 2h

4) Correct fine grid approximation, $\mathbf{v}^h = \mathbf{v}^h + \mathbf{Interpolate}(\mathbf{e}^{2h})$

5) Relax ν_3 times on $\mathbf{A}^h\mathbf{v}^h = \mathbf{f}^h$ on the fine grid h

The three parameters ν_1, ν_2 and ν_3 can be chosen to give optimum performance as discussed later It is important to consider the advantage of such a scheme again Iterating on the fine grid will eliminate the oscillatory components of the error, leaving a relatively smooth error Assume that we cannot solve the residual equation exactly on the coarse grid 2h, but do obtain a good approximation at the error Since this error is smooth, interpolation will work very well and the error should be represented accurately on the fine grid

A program was written to implement a coarse grid correction scheme, and the results after three or more iterations on both grids, agreed with those of Briggs [7], if equal amounts of two initial modes are present

The Vcycle and other Multigrid Cycles

While on the coarse grid 2h, it is possible to perform another nested coarse grid correction scheme, first finding the residual $\mathbf{r}^{2h}$, transferring this onto a coarser grid 4h, via the restriction operator, **Restrict**($\mathbf{r}^{2h}$) We then perform several iterations on the equation $\mathbf{A}^{4h}\mathbf{e}^{4h} = \mathbf{r}^{4h}$ on the grid 4h We then interpolate the error, interpolate($\mathbf{e}^{4h}$) and add this to the approximation $\mathbf{e}^{2h}$, $\mathbf{e}^{2h} = \mathbf{e}^{2h} + \mathbf{Interpolate}(\mathbf{e}^{4h})$ It may also be possible to solve $\mathbf{e}^{4h}$ exactly, however this may not be necessary If nested coarse grid corrections are applied in this manner until we reach the coarsest grid we can then proceed upwards, continuously correcting the error on each level as above, until the fine grid is reached again This basic idea is referred to as the Vcycle

As discussed later, for large meshes, this will prove more efficient than solving exactly on the grid 2h Also it may not be necessary to descend to the coarsest grid Given a mesh with $N = 2^q$, having $2^q - 1$ internal grid points, each again of separation h, the coarser grid below this will have $2^{(q-1)}-1$ points and a grid separation 2h, and so on as we descend down the grids The coarsest grid is given by Lh, with $L = 2^{(Q-1)}$, where $Q > 1$ The Vcycle procedure is the fundamental cycle in multigrid and can be found in most multigrid texts Briggs [7] has summarised the Vcycle procedure as follows,

Iterate on $\mathbf{A}^h\mathbf{v}^h = \mathbf{f}^h$ ν_1 times on the fine grid h with the initial guess $\mathbf{v}^h$
Compute $\mathbf{r}^{2h} = \mathbf{Restrict}(\mathbf{r}^h)$
Iterate on $\mathbf{A}^{2h}\mathbf{e}^{2h} = \mathbf{r}^{2h}$ ν_1 times on the grid 2h with the initial guess $\mathbf{e}^{2h}=0$
Compute $\mathbf{r}^{4h} = \mathbf{Restrict}(\mathbf{A}^{2h}\mathbf{e}^{2h} - \mathbf{r}^{2h})$
Iterate on $\mathbf{A}^{4h}\mathbf{e}^{4h} = \mathbf{r}^{4h}$ ν_1 times on the grid 4h with the initial guess $\mathbf{e}^{4h}=0$
Compute $\mathbf{r}^{8h} = \mathbf{Restrict}(\mathbf{A}^{4h}\mathbf{e}^{4h} - \mathbf{r}^{4h})$

Solve on $\mathbf{A}^{Lh}\mathbf{e}^{Lh} = \mathbf{r}^{Lh}$ on the coarsest grid Lh with the initial guess $\mathbf{e}^{Lh}=0$

Correct $\mathbf{e}^{4h} = \mathbf{e}^{4h} + \mathbf{Interpolate}(\mathbf{e}^{8h})$
Iterate on $\mathbf{A}^{4h}\mathbf{e}^{4h} = \mathbf{r}^{4h}$ ν_2 times on the grid 4h with the corrected guess $\mathbf{e}^{4h}$
Correct $\mathbf{e}^{2h} = \mathbf{e}^{2h} + \mathbf{Interpolate}(\mathbf{e}^{4h})$

Iterate on $A^{2h}e^{2h} = r^{2h}$ ν_2 times on the grid 2h with the corrected guess e^{2h}

Correct $v^h = v^h + \textbf{Interpolate}(e^{2h})$
Iterate on $A^h v^h = f^h$ ν_2 times on the fine grid h with the corrected approximation v^h

It is also worth noting that any given number of iterations can be performed on any particular grid, not just ν_1 while descending, and ν_2 while ascending It may be possible to find the best performance by comparing the results from the different combinations Before the Vcycle can be written recursively, the notation f^{2h} will be used instead of r^{2h}, and v^{2h} instead of e^{2h} We have now removed both **e** and **r** from the notation Defining the Vcycle function $v^h = \textbf{Vcycle}(v^h, f^h, A^h)$, recursively the Vcycle has been written by Briggs [7],

$v^h = \textbf{Vcycle}(v^h, f^h, A^h)$

1) **Iterate** ν_1 times on $A^h v^h = f^h$ with a given initial guess v^h

2) If this grid is the coarsest grid, then go to 4
   Else $f^{2h} = \textbf{Restrict}(f^h - A^h v^h)$
      Set $v^{2h} = 0$ as initial guess
      repeat ν times ($v^{2h} = \textbf{Vcycle}(v^{2h}, f^{2h}, A^{2h})$)

3) Correct $v^h = v^h + \textbf{Interpolate}(v^{2h})$

4) **Iterate** ν_2 times on $A^h v^h = f^h$ with initial correction v^h

If ν is 1, then a Vcycle results However if ν is 2, a Wcycle results, so called because of its shape as it goes up and down the grids with time If we defined a four level grid system, the number of grid points would be given by figure -(2 8a) For this four level system, figures -(2 8b, c, and d) shows the Vcycle, Wcycle, and the Full Multigrid Cycle, which we will now introduce

This Full Multigrid Cycle (or FMcycle) starts on the coarsest level, and consists of a series of Vcycles at each level as we ascend to the fine grid This utilizes the idea of starting on the coarsest grid, so as to obtain a rough initial guess for the higher grids Brandt [5] put forward this idea and has described the many advantages of this approach Briggs [7] has written the algorithm as

$\mathbf{v}^h = \mathbf{FMcycle}(\mathbf{v}^h,\mathbf{f}^h,\mathbf{A}^h)$

COMPUTE $\mathbf{f}^h,\mathbf{f}^{2h}$, ,
SET $\mathbf{v}^h,\mathbf{v}^{2h}$, to zero

Solve or **iterate** on the coarsest grid

$\mathbf{v}^{4h} = \mathbf{v}^{4h} + \mathbf{Interpolate}(\mathbf{v}^{8h})$
$\mathbf{v}^{4h} = \mathbf{Vcycle}(\mathbf{v}^{4h},\mathbf{f}^{4h},\mathbf{A}^{4h})$
$\mathbf{v}^{2h} = \mathbf{v}^{2h} + \mathbf{Interpolate}(\mathbf{v}^{4h})$
$\mathbf{v}^{2h} = \mathbf{Vcycle}(\mathbf{v}^{2h},\mathbf{f}^{2h},\mathbf{A}^{2h})$
$\mathbf{v}^h = \mathbf{v}^h + \mathbf{Interpolate}(\mathbf{v}^{2h})$
$\mathbf{v}^h = \mathbf{Vcycle}(\mathbf{v}^h,\mathbf{f}^h,\mathbf{A}^h)$

This can be written recursively as

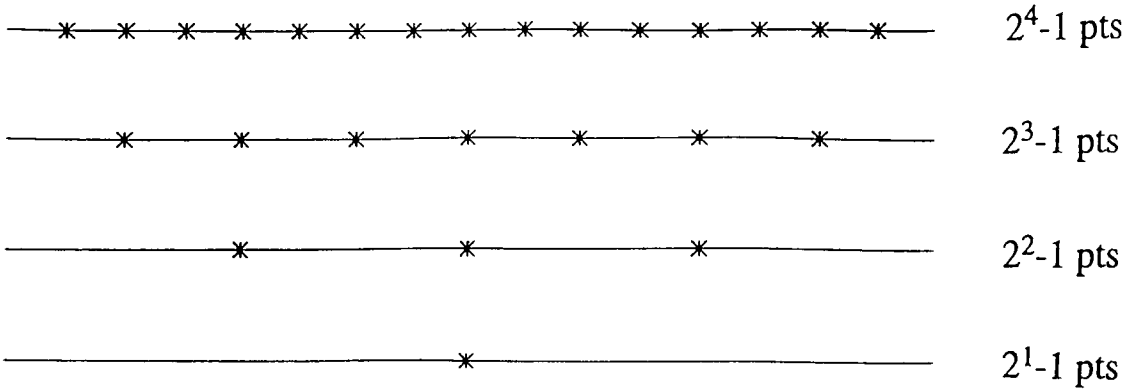
$\mathbf{v}^h = \mathbf{FMcycle}(\mathbf{v}^h,\mathbf{f}^h,\mathbf{A}^h)$

1) If this grid is the coarsest grid go to step 3
   Else $\mathbf{f}^{2h} = \mathbf{Restrict}(\mathbf{f}^h - \mathbf{A}^h\mathbf{v}^h)$
   Set $\mathbf{v}^{2h}$ =0 as initial guess
   $\mathbf{v}^{2h} = \mathbf{FMcycle}(\mathbf{v}^{2h},\mathbf{f}^{2h},\mathbf{A}^{2h})$

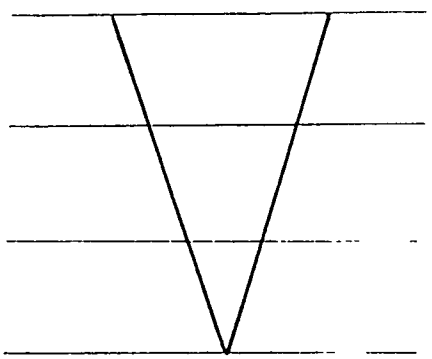
2) Correct $\mathbf{v}^h = \mathbf{v}^h + \mathbf{Interpolate}(\mathbf{v}^{2h})$

3) $\mathbf{v}^h = \mathbf{Vcycle}(\mathbf{v}^h,\mathbf{f}^h,\mathbf{A}^h)$, ν times

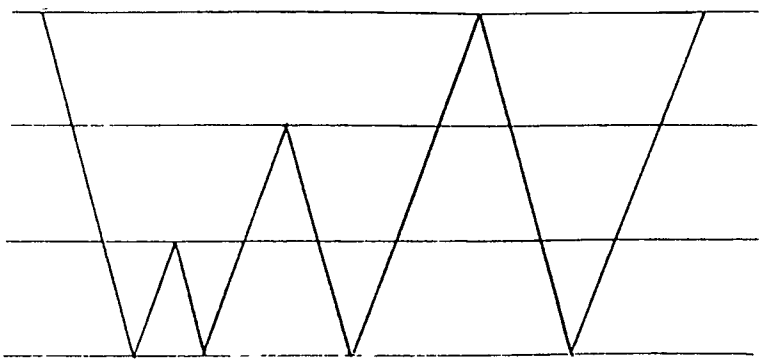
Notice that this calls the Vcycle() function, as well as itself recursively For any discretized stationary PDE problem, a full multigrid cycle FMcycle() with ν_1 and ν_2 = 3 is enough [19] to solve the equation $\mathbf{Av} = \mathbf{f}$ with ν =1, each time Vcycle() is called It is now apparent that the coarse grid correction scheme is the key part in the Vcycle(), and the Vcycle() is the fundamental cycle in all multigrid cycles The way in which each of the different cycles ascends and descends the grids is referred to as the multigrid schedule [23] There is no universal 'Black Box' multigrid algorithm that works for all systems [12], so each of these methods may be tested, for any particular given problem in question, to observe which performs best



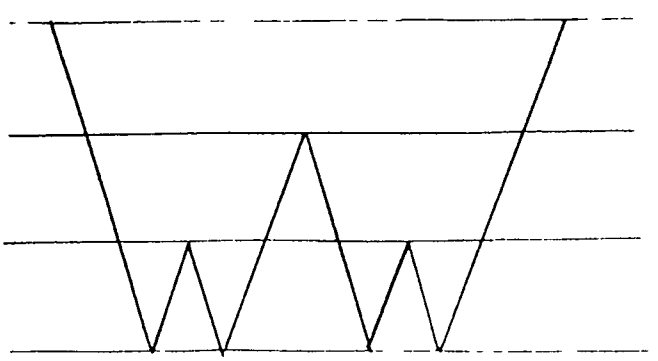
Four Level Grid System a)



V-Cycle b)



Full Multigrid Cycle c)



W-Cycle d)

Performance of the Poisson Equation

In order to discuss and compare the efficiency of all the above methods, both the direct and the iterative, we must consider how much CPU time each of this would roughly use It is common in multigrid literature to define a Work Unit (WU) [4,7,19] as the amount of computational work done, or CPU processing time, as the work required to compute one iteration or relaxation on the fine grid This has been put forward as the golden rule of computation by Brandt (1982) [3,4,6], although the WU can be defined differently As has been put forward a number of times by Brandt in a number of publications, the WU depends not only on the type of method to be used, but the choice of mathematical model used In general, many publications show that for a given equation **Av = f**, it will cost at least several WU's to solve [23] If we consider the Poisson equation for the simplest case, **Av=0**, with both boundaries set to zero, we can plot the log of the error norm or residual norm against cycle, or now in this case against the number of WU's performed

We will consider the computational cost of the WU's of for the Vcycle, Wcycle and the FMcycle in accordance to the rough guide given by Briggs[7] Consider a Vcycle with v iterations on each grid while descending and ascending It is customary to neglect the cost of intergrid transfer operations which could amount to 15-20 percent of the cost of the entire cycle (for the Poisson case) On the fine grid there are v iterations, or v WU's, then on the next coarse grid 2h there are 2^{-d} times less, where d is the dimension of the mesh Similarly on the grid 4h there are 4^{-d} times less The grid ph has p^{-d} times less Adding these together as an upper bound geometric series we get

$$1 \text{ Vcycle costs } = 2v\{1+2^{-d}+2^{-2d}+ \quad +2^{-nd}\} < 2v\left(\frac{1}{1 - 2^{-d}}\right) \text{ WU's} \qquad -(2\ 46)$$

The factor of two arises because we stop on each level twice Similarly the cost of a FMcycle with v iterations can be roughly given from this guide If the cost of a Vcycle starting on the grid 2h is 2^{-d} of the cost on the fine grid, then starting on grid 4h would cost 4^{-d} times as many WU's Any grid ph would then cost p^{-d} times less This results in the series

$$1 \text{ FMcycle} = 2v\left(\frac{1}{1 - 2^{-d}}\right)\{1+2^{-d}+2^{-2d}+ \quad +2^{-nd}\} < 2v\left(\frac{1}{1 - 2^{-d}}\right)^2 \text{ WU's} \qquad -(2\ 47)$$

It now becomes apparent that as the dimensions of the problem increase, we increase the performance of multigrid cycles Using the above guide, a FMcycle in 1-D would cost 8 WU In 2-D the cost is about $^{7}/_{2}$ and in 3-D $^{5}/_{2}$ In each case the definition and the amount of work by each WU is different, but WU can be used to compare different systems in this way

Using the above reasoning, it can be shown, that the series for the more complicated Wcycle is given by

$$1\ \text{Wcycle} = 2\nu\{1+2^{d}(2^{-d})+2^{2d}(2^{-2d})+ \quad +2^{nd}(2^{-nd})\} = 2\nu\{n\}\ \text{WU's} \quad -(2\ 48)$$

For six levels, in 1-D, with a total of 2^{6}-1 points, we roughly surmise,

Vcycle WU < 4v
Wcycle WU ~ 12v
FMcycle WU < 8v

The above has defined a work function as one iteration, which in turn is proportional to N, for the Poisson case For the Poisson equation, it is not necessary to operate on all the elements, only the tridiagonal elements It is known that all other elements are zero Thus the iteration is given by

$$v_{i} = {}^{1}/_{2}(f(x_{i}) + v_{i-1} + v_{i+1}) \qquad (0 < i < N) \qquad -(2\ 49)$$

which is obtained from equation -(2 1) If this is performed for every i^{th} point on the grid, then it can be easily seen that for one iteration, we have two additions and one division, e g approximately 3N operations, counting multiplication, division, addition and subtraction as one operation

However, if it is not known what elements are zero throughout the matrix, or if there is no definite pattern, e g certain off diagonal lines are known to be non zero, then the general matrix form of the Gauss-Seidel iteration must then be used as given by equation -(2 12)

$$v_{i} = {}^{1}\!/_{a_{ii}}\{ b_{i} - \sum_{j=1}^{i-1} a_{ij}v_{j} - \sum_{j=i+1}^{N-1} a_{ij}v_{j} \} \qquad (0 < i < N) \qquad -(2\ 50)$$

Using this equation, and looking in more detail at the number of mathematical manipulations required, one iteration now consists of

1) $\sum_{j=1}^{i-1} a_{ij} v_j$ having $N/2$-1 multiplication and additions, gives N-2 manipulations

2) $\sum_{j=i+1}^{N-1} a_{ij} v_j$ having $N/2$-1 multiplication and additions, gives N-2 manipulations

3) $b_i - \sum_{j=1}^{i-1} a_{ij} v_j - \sum_{j=i+1}^{N-1} a_{ij} v_j$ having two subtraction's for each i^{th} term

4) $1/a_{ii} \{ b_i - \sum_{j=1}^{i-1} a_{ij} v_j - \sum_{j=i+1}^{N-1} a_{ij} v_j \}$ giving one division for each i^{th} term

5) the above repeated for each i^{th} term, (repeated N times)

Summing from 1) to 4) gives 2N-1 ~ 2N operations, and this procedure is then repeated for each i^{th} term, e g N times, indicating that one iteration is now proportional to N^2, instead of the previous N for the Poisson case Equations -(2 46 - 2 48) now give

Vcycle cost in WU's $< 8\nu N^2$ manipulations

Wcycle cost in WU's $\sim 24\nu N^2$ manipulations

FMcycle cost in WU's $< 16\nu N^2$ manipulations

The storage considerations involve storage for the problem matrix **A**, the solution **v** and the RHS **f** on all levels In this case, even though most of the elements in **A** are zero, we cannot identify a pattern as to which diagonals exist, why and where these will be in **A**, and thus cannot represent **A** by its diagonal elements Also, ionization causes a further problem, as its elements, unlike all others are placed in a row For these reasons, **A** was not represented by its diagonals, but left, as a sparse matrix

For a d dimensional problem, **A** has $(N^2)^d$ or N^{2d} elements, **f** and **v** have both N^d elements The total is then $N^{2d}+2N^d$, on the fine grid On the grid below this, this

quantity is reduced to $(N/2)^{2d} + 2(N/2)^{d}$, or $2^{-2d}(N)^{2d} + 2(2^{-d})(N)^{d}$ The storage cost for all levels can then be written as,

$$N^{2d}\{1+2^{-2d}+2^{-4d}+\quad+2^{-2nd}\} \;+\; 2N^{d}\{1+2^{-d}+2^{-2d}+\quad+2^{-nd}\}$$

$$< \; N^{2d}\left(\frac{1}{1-2^{-2d}}\right) + 2N^{d}\left(\frac{1}{1-2^{d}}\right) \qquad -(2\ 51)$$

If we consider the Poisson equation for the simplest case, **Av=0**, with both boundaries set to zero, we can plot the log of the error norm or residual norm against cycle, or now in this case against the number of WU's performed (figures -(2 9a)) The direct method of restriction was used We also observe from this that the Vcycle was most efficient, the Wcycle next and then the FMcycle

However, such plots don't give any details of the norms during the cycle Briggs [7] has shown a very detailed plot of how these norms behave throughout a vcycle with ν=1 iterations at each level (figure -(2 9b))

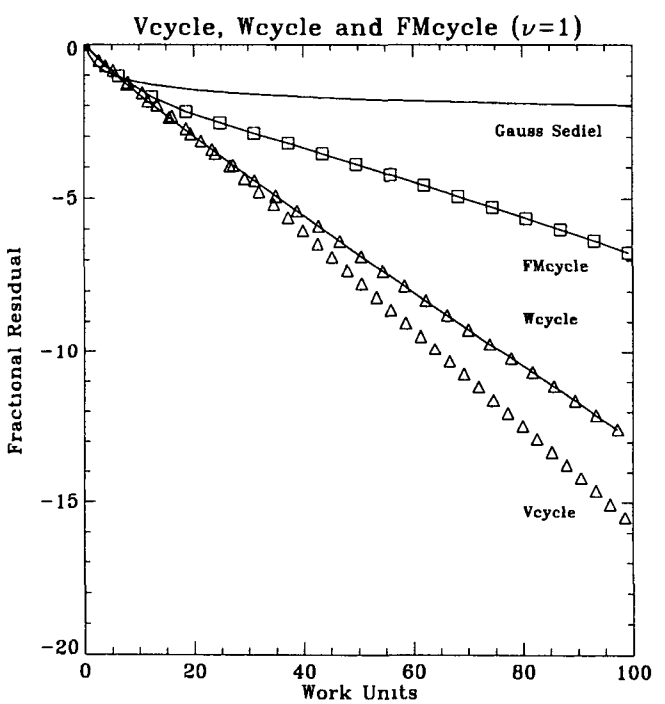


Figure -(2.9a) shows how the fractional residual decreases for different cycles The direct method of restriction is used

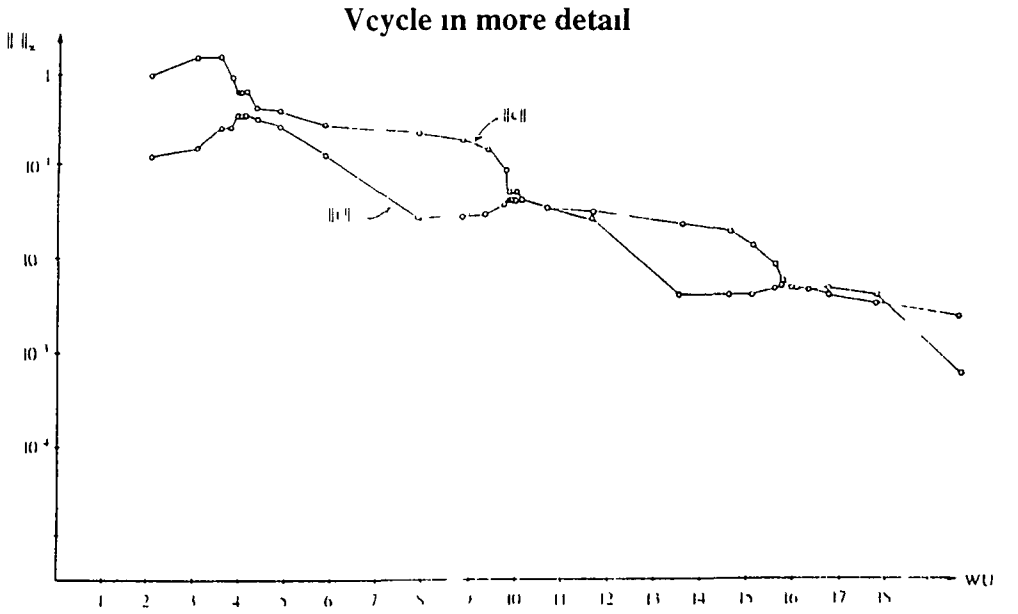
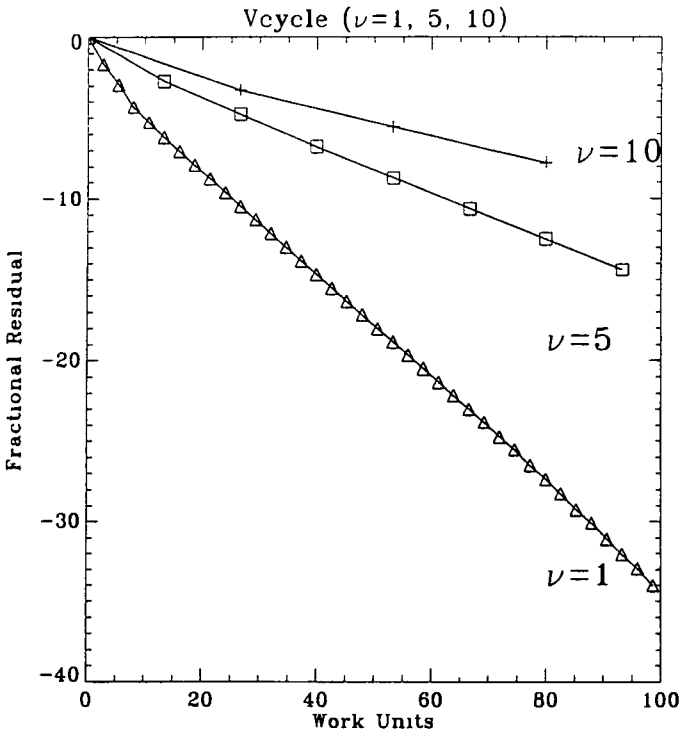
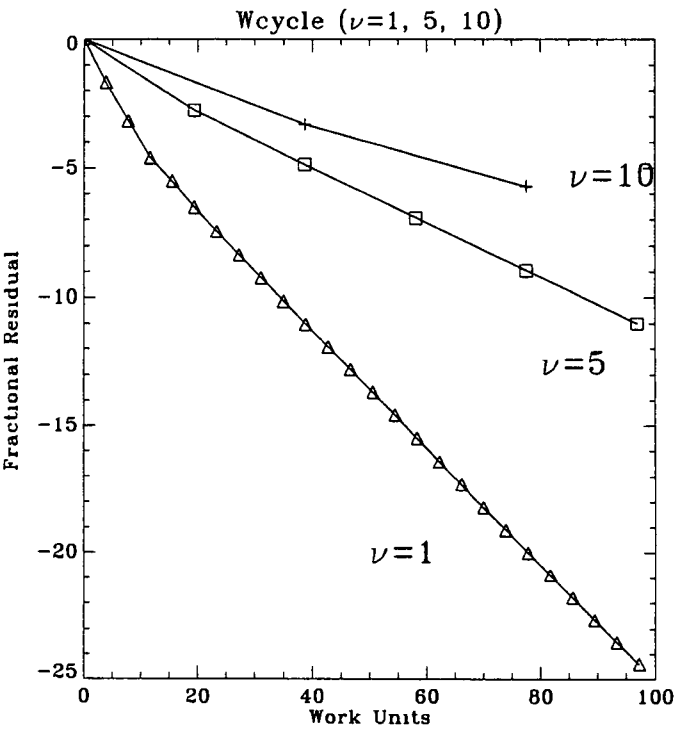


Figure -(2 9b) shows how the error and residual norms behave during each Vcycle, with v=1

In general agreement with this for the same simple Poisson case, figures 2 10a-c, show that if we increase n to 5, and then 10, the performance, or slope of the curves decreases The Galerkin method of restriction was used, and this converges faster than figure -(2 9a) using the direct method (for the Poisson case)

As mentioned before, when writing the code for the Restrict() function, it became apparent that there were two possible ways to achieve this (ignoring the weighting effect) These are to be referred to as Restrict(Ah) and Restrict(A2h) The difference between them is discussed in the appendix C They give slightly different results, and this will be discussed later We have now looked at how the multigrid program works for our test Poisson problem Chapter one has introduced the form of the Boltzmann problem Now we consider the application of the multigrid method The same multigrid algorithms will now be applied to the Boltzmann problem, and the results observed





Figures -(2 10a,b) show the effects of increasing v to 5 and 10, for the Vcycle and the Wcycle

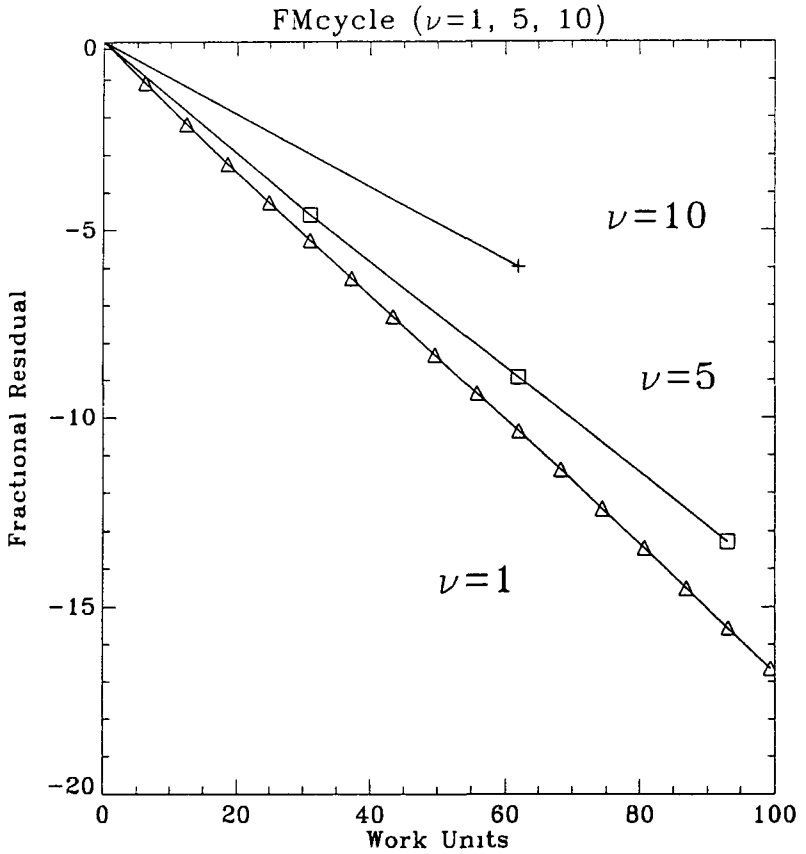


Figure -(2 10c) shows the effects of increasing ν to 5 and 10, for the FMcycle

Performance Results for the Boltzmann Case (Ch 3)

Multigrid Program for the Boltzmann Equation

The program Multigrid_Study c was developed from existing code which solved the same plasma problem, except with a direct method, and by integrating forward in time steps until a steady state solution was found This program solved the steady state equations in matrix form given by **Af=0** The program also explores the results of setting different multigrid parameters or multigrid settings, to observe the best convergence obtainable A more detailed listing of these settings is given in the appendix C Such multigrid parameters would be for example, whether the mechanism for restriction was weighted or linear, or if this was of type AH or A2H (Chapter 2) The type of cycle used could be Vcycle, Wcycle or FMcycle The level to which this cycle descends to is referred to as the level, and how many iterations v are performed on each level will all effect how the multigrid method performs

The definition for the problem matrix **A**, can be given by either the direct approach, or the Galerkin method The direct approach involves redefining the matrix **A** at each level for that given number of grid points, but with a different energy interval The Galerkin method calculates **A** on the lower grids by the expression **RAP** given in chapter 2 by the equations -(2 43,44,45)

The program contains macros for each of the corresponding settings These are all given in the appendix A If these macros are true, then the program adopts these settings If they are false, then the settings in question are not used

The program can repeatedly solve the problem for each different combination of multigrid settings or parameters and present the best results obtained This is achieved by the Multigrid Operations section of the code (discussed in appendix C)

The Main Program Procedural Flow

A list of all the macro's related to the multigrid aspects of the program is given in the appendix A A list of input and output files is given also It is possible to set exactly what the program does by setting macro's (most of which are either set to 1 or 0). Originally a third data file was used for this, but it was found more simple and

workable to use macro's for this purpose Appendix A also gives a listing and brief explanation of the prototypes of all the functions in Multigrid_Study c and Cycles c

An overview of the program flow is given here, and a more detailed account is presented in the appendix C The first of the two data files, b17 dat is read, which contains information regarding plasma parameters, and the size of the problem q, which would give 2^q -1 internal grid points Memory is then allocated for the vectors and matrices involved, whose size is now given The second data file xs dat is then read This contains the cross-sections for all the collisional processes between electrons and the neutrals, both elastic and inelastic The elements of the matrix **A** are now calculated, for each grid, in accordance to the multigrid settings The boundary conditions are also set, where the RHS can be made non zero as in equation -(1 13) The problem is now defined, and the multigrid solver now steps into play The set problem can be solved again for a range of multigrid parameters given The performance is then written to output files, indicating which multigrid settings for the same given problem have performed the best Thus multigrid methods are being compared, by experimentation, to find the optimum settings The simple flow diagram given in appendix C shows how the above has been split into 7 blocks or sections of code These blocks are discussed in more detail in the appendix C

Code Testing

The LUD solver, (achieved by the function Solve(), see appendix C), was tested on a 3x3 matrix It was then tested for the unique Poisson case, $\mathbf{An} = \mathbf{n}' = 0$, with only one solution possible, $\mathbf{n}=0$ only, given the two boundary conditions, $n_0=0$ and $n_N=0$ This was tested for different sized meshes In the multigrid case, where a solution exists on each level, it was tested also The LUD solver was then tested for different boundary conditions, and was found to function correctly

An effective method for testing the multigrid code, is using cyclic reduction [2] This is possible for the Poisson case, and if full weighted restriction is used, the solution can be found by proceeding down and up the grids with zero error, or exactly zero residual There is no reason why any multigrid solver should ever give such an exact result At first, it appeared that multigrid was performing extremely successfully, but by coincidence, the mechanism of descending to the coarsest grid with only one point, and then ascending back to the fine grid, using only red-black (one is sufficient), (forward or backward), Gauss-Seidel iteration on each level, and using weighted

restriction of type A2H, corresponds, mathematically to the process of cyclic reduction This caused confusion However, this fact was used to test the multigrid code, since successful cyclic reduction indicated that the code is working correctly, in so far as the method of descending and ascending to grids is correctly handled This was found to be a useful test, when ever important fundamental changes were made to the code The LUD solver could also be tested by inserting it at any level in the cyclic reduction process, and solving on that level instead of proceeding down further to lower levels The residual still remained exactly zero The only explanation for this is that both LUD and multigrid processes were functioning correctly

Several functions have been tested, particularly Residual() (see appendix C) which calculates the residual in equation -(2 30), as this was vital in viewing the performance of multigrid Similar functions were tested which calculated the error norms These were rewritten several times, using different methods, all of which gave the same result

The following notation will be used in the graphs, the number of iterations on each grid v, and the lowest level (referred to as the level) which the given cycle descends to, will be denoted by, v, level (e g 3,4 corresponds to v=3, level=4) The fine grid is level 6, which contains 2^6-1 internal grid points

If the type of restriction is not mentioned, it is to be assumed that it is of types AH and LINEAR, since these generally give the best results for the Boltzmann case Unless otherwise stated, it is to be assumed that the type of Gauss-Seidel iteration is forward, and basic/ordinary (not red-black) The figures produced, unless otherwise stated, are the output of the program Multigrid_Study c, which have been plotted by various IDL software programs These programs read in the data from the output files, and plot them accordingly

The Residual

It is almost universal practice in multigrid texts to use the residual as a measure of the performance of the given solver in question Equation -(2 30) gives the expression for the residual However, if we consider the matrix problem $\mathbf{Av} = \mathbf{f} = 0$, the absolute value of the residual is a factor of the size of the elements of **A**, and **f** If the boundary condition is set by making the first element f_1 on the right non zero, then the size of this also effects the size of the absolute residual

However, since the whole equation can be multiplied by any factor, then the size of the absolute residual itself is of no significance to the success or failure of the method This was the incentive for using a fractional residual, simply dividing the resulting absolute residual by the initial residual At first glance one could assume that there should not be any problem with this approach of using fractional residual to monitor performance

The problem arises due to the observation that the residual results are not always reproducible after the same number of multigrid cycles, only because of the arbitrary boundary conditions and initial guess may be different It is not how close to the solution the initial guess is, as clearly this will mean less work for the solver, but the actual size or summation of the elements in the initial guess It is found that fractional residual may be a function of the boundary condition and the initial guess, but there is nothing to connect these two parameters, e g , for any given chosen boundary condition, there could be any arbitrary initial guess size (Σf_{guess}), and visa versa By setting a boundary condition, this forces a unique solution, which in turn suggests that the solution should sum to a given constant, Σf_{sol} (LUD) For the Boltzmann case, any such problem does not arise unless f_1 is non zero, and the boundary condition is large enough such that

$$\Sigma f_{sol} \text{ (LUD)} > \Sigma f_{guess} \qquad \text{-(3 0)}$$

Multigrid however, for the Boltzmann case, even when a boundary condition is set, proceeds to solve the problem to within a constant of the actual solution, given by LUD This is easily seen from figure -(3 0), which plots the multigrid and LUD solution This is the case if the boundary conditions at both ends are equal [23], but the left boundary has been set to non zero This is the case when both normalization's $\mathbf{n}(\varepsilon)$ and $\mathbf{f}(\varepsilon)$ are solved for For the time being the $\mathbf{n}(\varepsilon)$ normalisation will be chosen Figures -(3 1a-d) show that for the Boltzmann case, as the size of the initial guess becomes smaller than that of the solution, from 1E+0 down to 1E-6 of its size, the results become markedly worse (this is only seen in the levels lower than 5) The multigrid solution is no longer solving to within a constant, as seen in figure -(3 0) When the initial guess is 1E-6 smaller than the solution, the shape of this curve as seen in figure -(3 0) cannot be superimposed onto the LUD shape, as it can when the initial guess is 1E+0 and 1E+6 the size of the LUD solution If the initial guess is 1E+6 larger than the size of the LUD solution, multigrid is seen to solve to within a constant, as figure -(3 0) clearly shows, and the corresponding fractional residual suggests that the

solver is converging just as successfully as it did when the initial guess was the same size as the LUD solution This confirms equation -(3 0)

The problem is that if the residual is not reproducible for different guess size, and since it depends on the relationship between two arbitrary parameters, the guess size and the solution size (dictated by the boundary condition), then from this we cannot depend on the residual to give a reliably account of the performance However, equation -(3 0) gives that the residual is reproducible for the Boltzmann case, once the arbitrary guess size is larger than that of the solution This is criteria by which the performance results will proceed

Another indication of performance is by setting a convergence criteria This will be discussed later, but this has its limitations

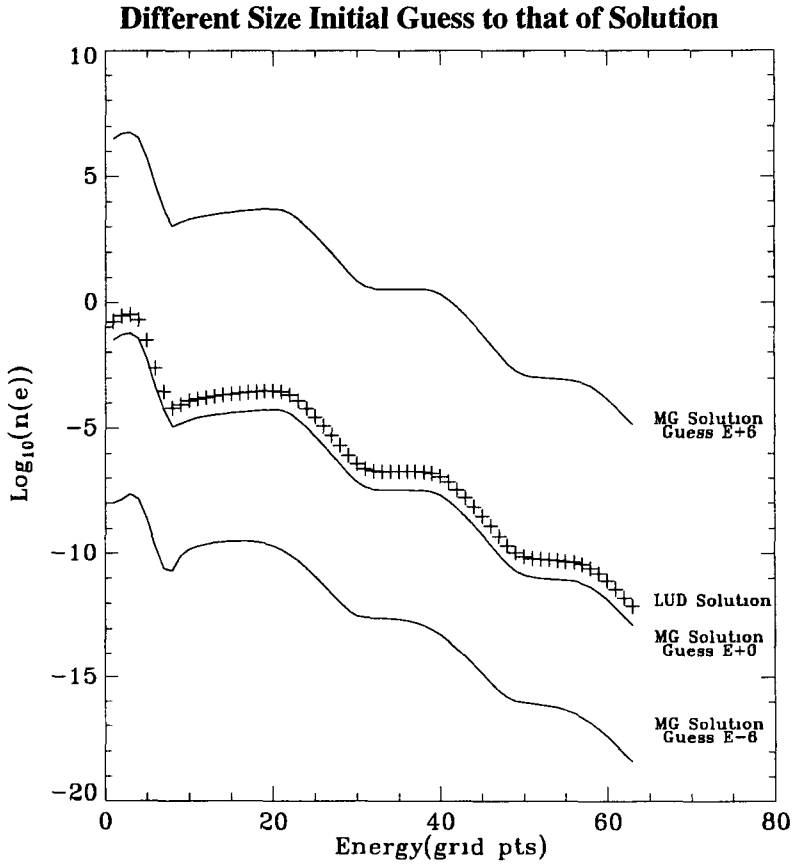
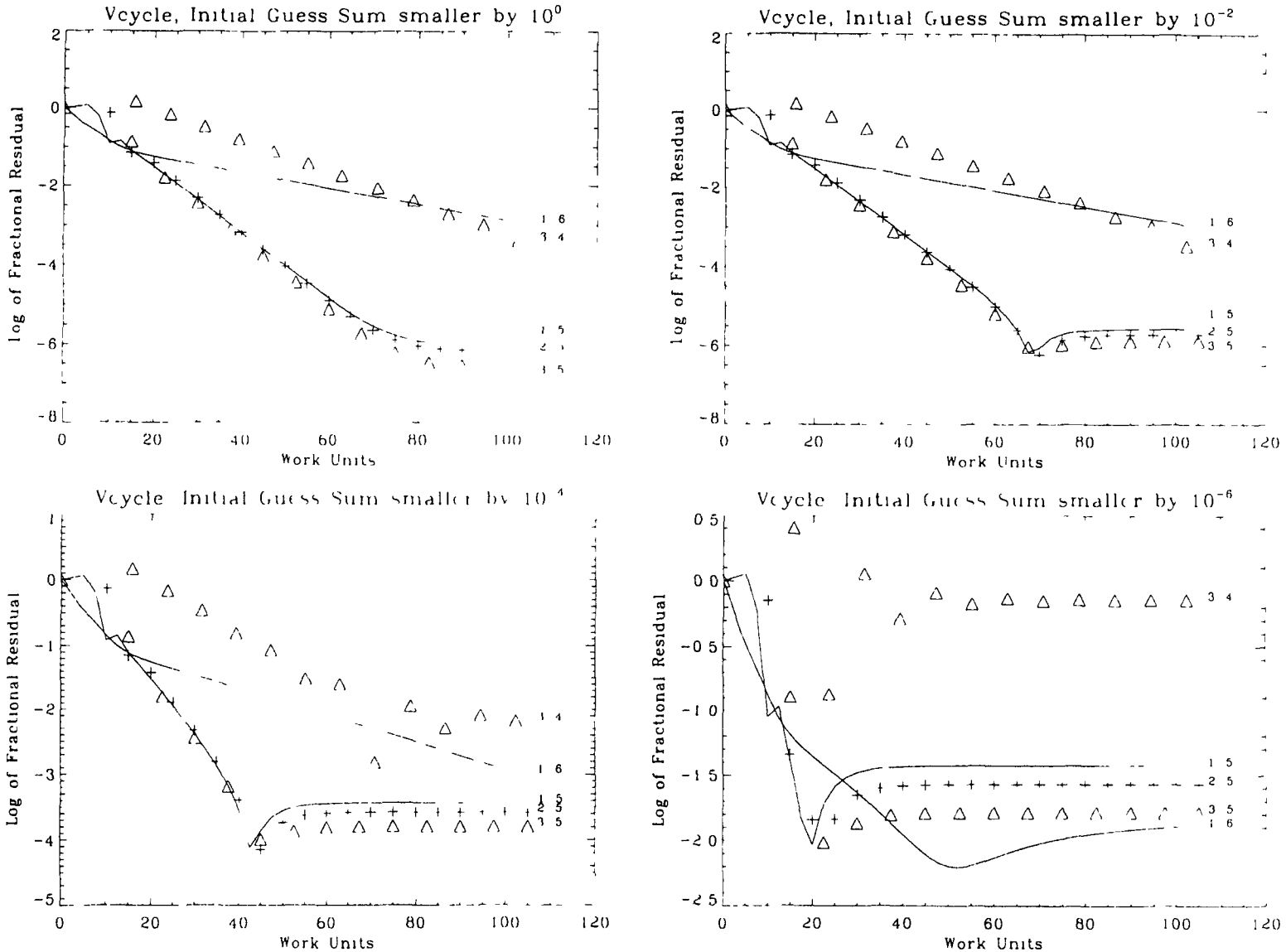


Figure -(3 0) shows the multigrid solution when the initial guess size (Σf_{guess}) is E+6, E+0, and E-6 that of the LUD solution size (Σf_{sol}(LUD)) For E-6, the multigrid solution visibly appears different, and not within a constant to the LUD solution



Figures -(3 1a-d) show that for the Boltzmann case, as the size of the initial guess becomes smaller than that of the solution, from 1E+0 down to 1E-6 of its size, the results become markedly worse (this is only seen in the levels lower than 5)

This size indifference between the guess and solution, is solved by normalizing the size of the approximation to that of the solution after each cycle However, this defeats the purpose of the objective, in that the solution is unknown to the multigrid solver, as is its size (Σf_{sol}(LUD)) It is found that by doing this, the residual will continue to decrease down to the order of 1E-15/-16 This will be discussed later

The Vcycle is the fundamental cycle in multigrid, from which all the other cycles have be constructed from For this reason, we first consider its performance, as if this is bad, then the other cycles will tend to follow suit

Figure -(3 2) shows results for the different types of restriction (AH or A2H, LINEAR or WEIGHTED) From this we see that the CGCS (Coarse Grid Correction Scheme, e g Vcycle to level 5), denoted by 1,5, 2,5, and 3,5, gives the best performance If the multigrid method was working correctly, the cycles to lower levels should at least perform better than the CGCS However, level 4, 1,4, and 2,4 both fail, and 3,4 barely achieves anything better than that given by the Gauss-Seidel iteration (denoted by GS) This suggests that at least ν=3 is required to give a reasonable improvement on the convergence of the Gauss Seidel This is not in accordance with the general multigrid theory The Vcycles 3,3 only converges if the restriction types are AH and WEIGHTED All other 3,3 Vcycles fail or tend to perform worse than the Gauss Seidel, and are thus clearly failing All cycles to lower levels fail Vcycles 3,4 A2H, LINEAR and 3,4 A2H, WEIGHTED also fail AH 3,4 LINEAR and WEIGHTED both converge, but not very convincingly The fact that the CGCS performance shown in figure -(3 3) is better than the previous figure -(3 2), and that the Vcycle fails on all lower levels indicates that multigrid is not performing in the correct expected manner

At this point, we could speculate that perhaps something is occurring wrong on the lower grids The code was tested, and was found to work for the Poisson, and the cyclic reduction testing as mentioned before This testing suggests that it is not the multigrid code mechanism that is at fault This code is in fact doing what is asked of it

Figure -(3 3) shows that AH, LINEAR 2,5 gives almost an identical result as AH, WEIGHTED 2,5, thus indicating that for CGCS, the types LINEAR and WEIGHTED have little or no significant effect on the results The type A2H however may give a different result than that of AH Another concerning issue is the manner in which the residual seems to flatten out in figure -(3 3), at around fres=10^{-6}. This is not small enough to be of the order of the round off error that the double precision code uses The residual for the Poisson case has been observed to converge to 10^{-25} or more, and

has not been seen flattening out in this manner The Gauss-Seidel iteration, if continued for several hundred iterations, will give a continuously decreasing residual, where flattening has not been observed as in the case of CGCS or Vcycles to lower grids Thus the problem is definitely associated with the lower grids As the coding mechanism is functioning correctly, perhaps the representation of the problem on the coarse grids is at fault in some way This was checked in great detail several times, and is believed to be representing the problem in the required manner in accordance to multigrid procedure

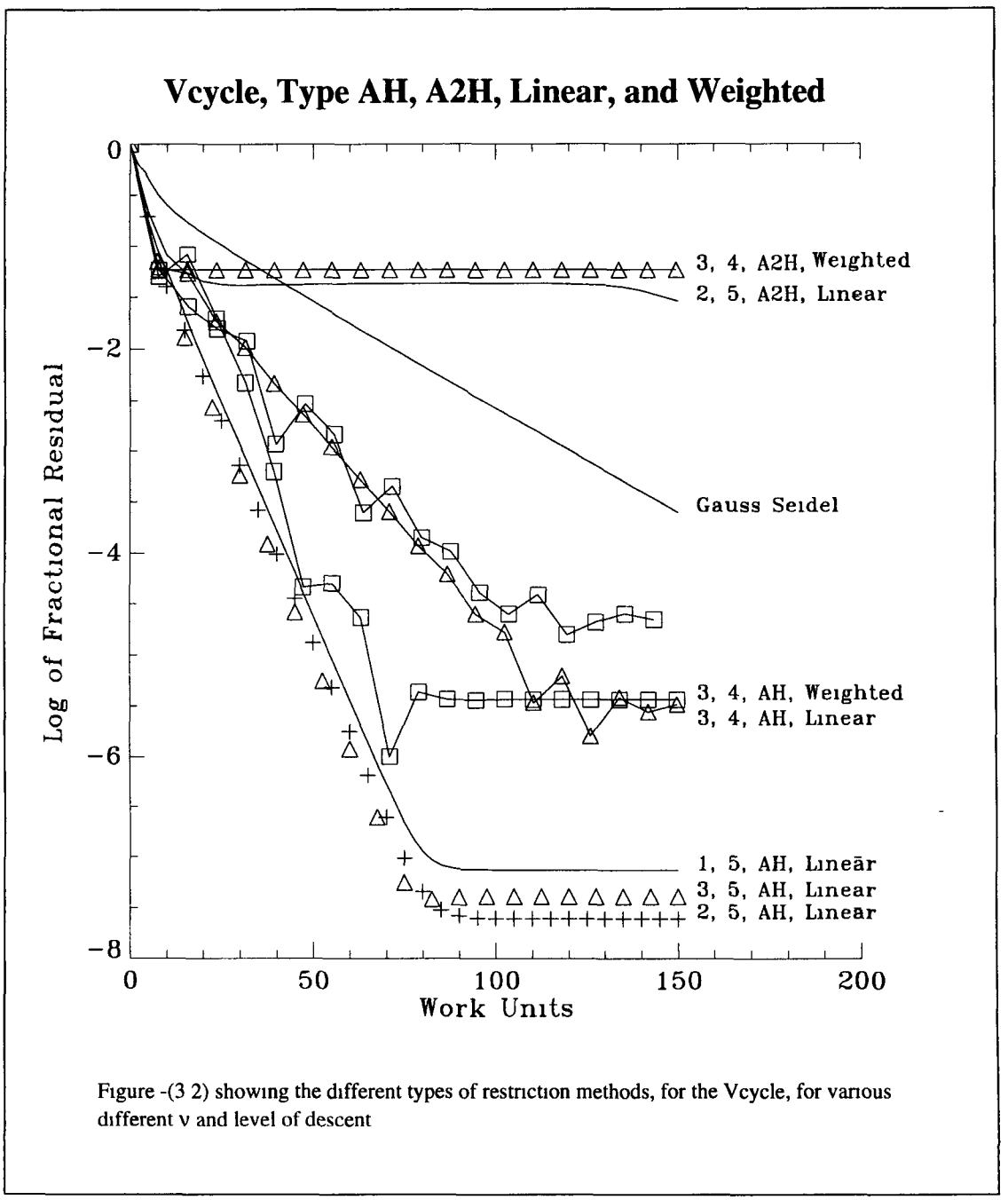


Figure -(3 2) showing the different types of restriction methods, for the Vcycle, for various different v and level of descent

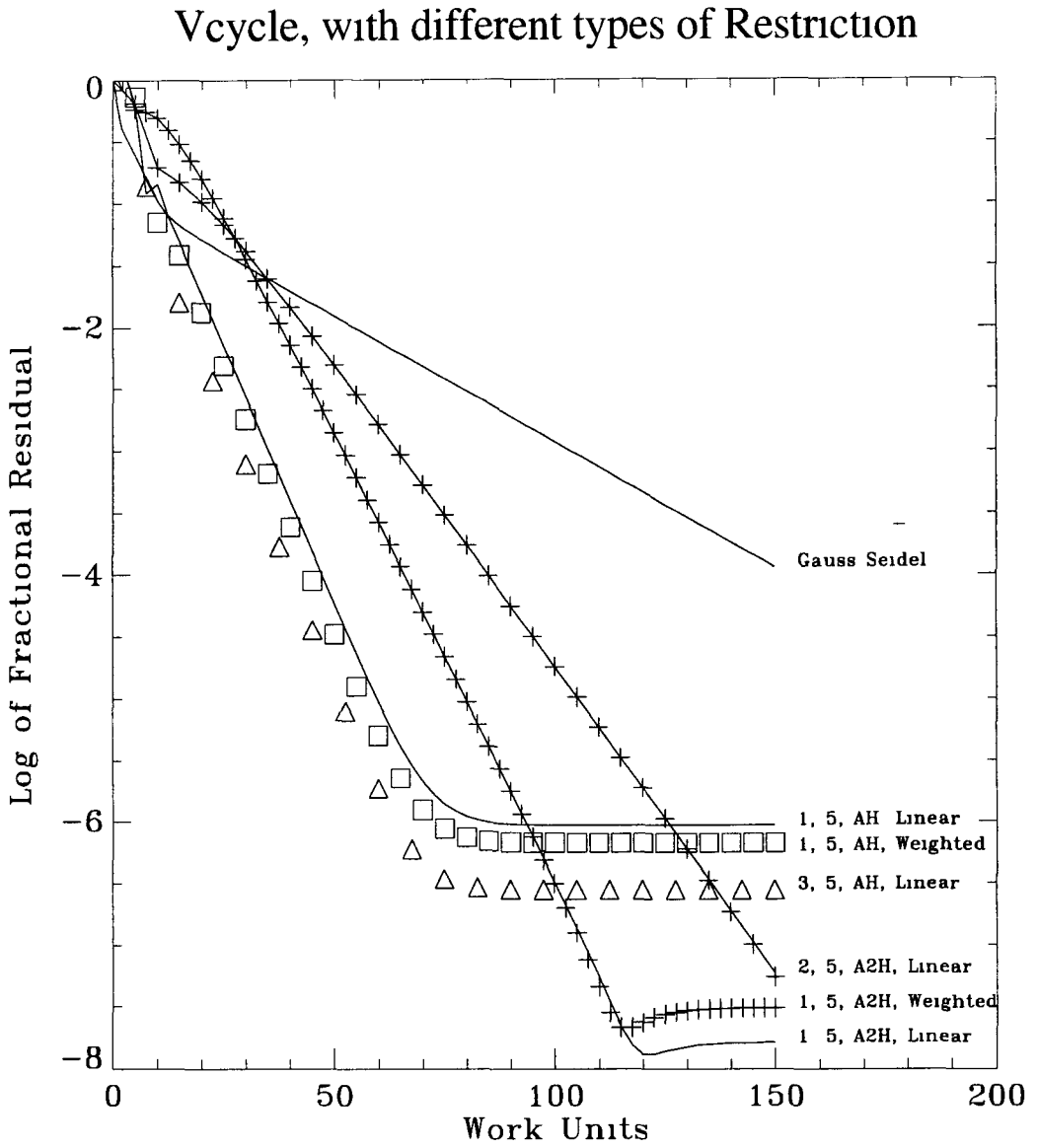


Figure -(3 3) showing the different types of restriction methods, for the Vcycle, for various different v and level of descent AH, LINEAR 2,5 gives almost an identical result as AH, WEIGHTED 2,5, thus indicating that for CGCS, the types LINEAR and WEIGHTED have little or no significant effect on the results

Figure -(3 4) shows that the nearest performance to the CGCS is the CCC1 cycle, 3,4 which involve a Vcycle to level 4, followed by a Vcycle to level 5 Again this indicates that the problem of the lower grids is apparent on level four

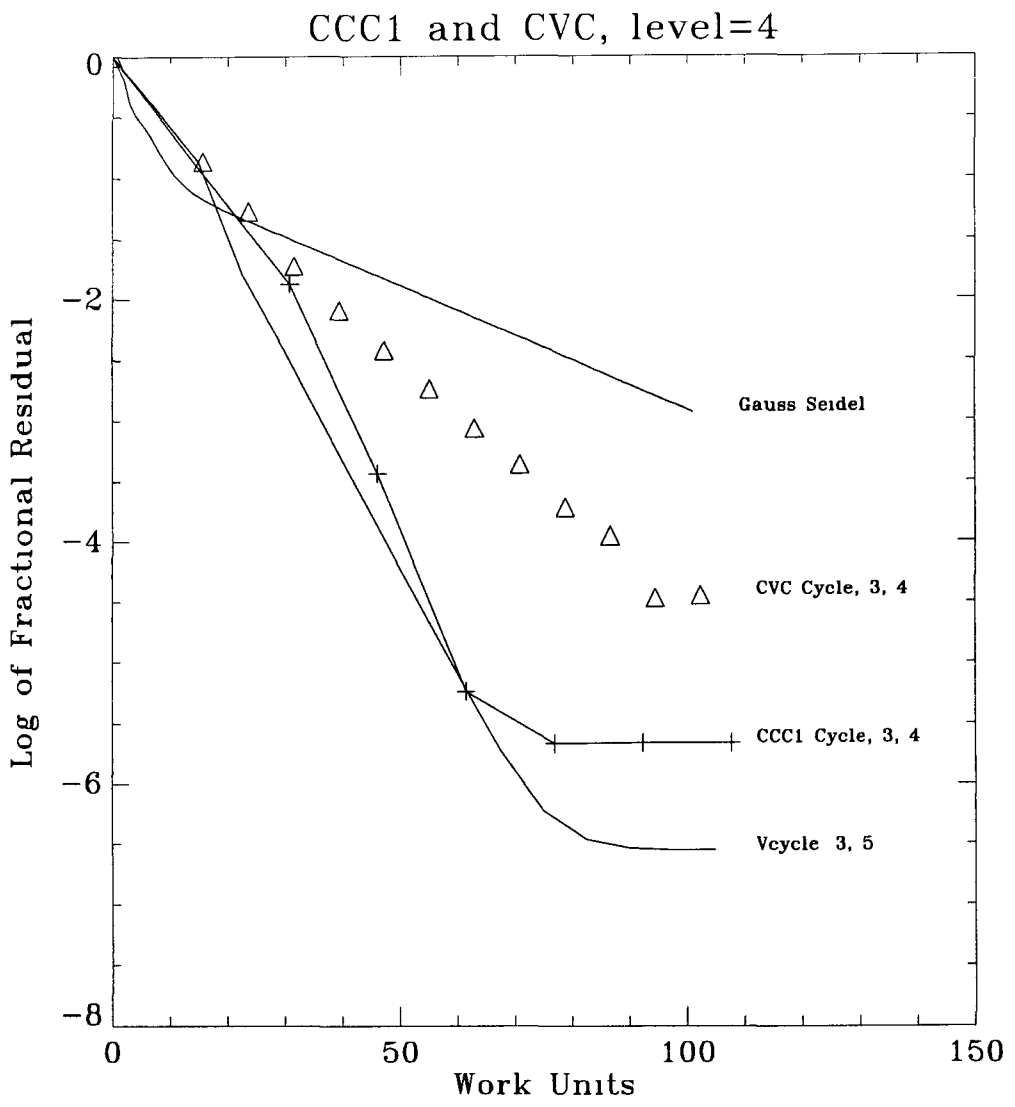


Figure -(3 4) shows the Vcycle, CGCS cycle and the CCC1 cycle The Vcycle gives the best results

Previously it was suggested that one solution to the residual problem is to normalize the approximation after each cycle to that of the solution This defeats the purpose of a black box solver, but, unexpectedly, this particular normalisation is seen to be extremely effective We could normalize to any arbitrary constant, but the residual and (the observed result) is not as good as normalizing to the solution, as shown in figures -(3 5) and -(3 6) Figure -(3 5) shows that the cycles for 3,4 converge much more rapidly, especially the CCC1 cycle, which now converges better than the CGCS, shown in figure -(3 5), (Vcycle 3,5)

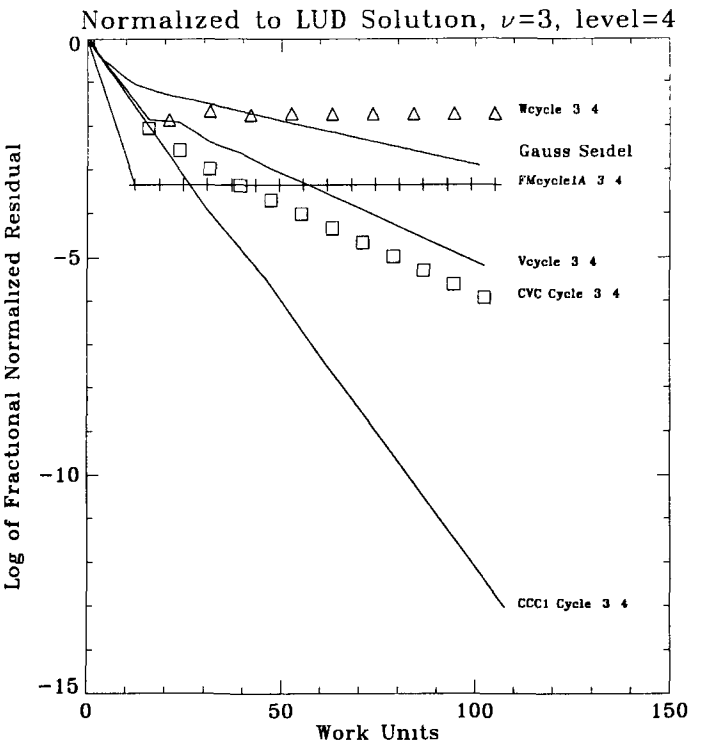


Figure -(3 5) shows the different cycles with normalization after each cycle This shows the failure of the FMcycle and the Wcycle

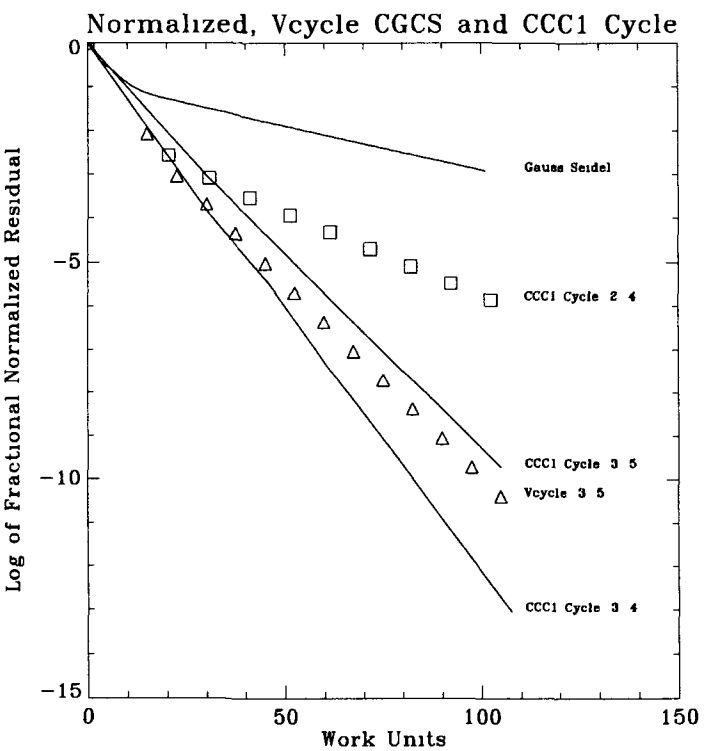


Figure -(3 6) shows the CCC1 Cycle, for different v and level, with normalization after each cycle

These results are not possible however, unless the solution (or its size is known) It is however not possible to find the size (Σf_{sol}) without knowing the solution These results tell us that the multigrid solver is converging to within a constant, or that the solution is of the wrong size to that dictated by the boundary condition The LUD size ($\Sigma f_{sol}(LUD)$) is the exact size dictated to it by the boundary condition, and thus the size of its solution gives the lowest possible residual, lower than any other size (Σf_{sol})

If the multigrid solver solves to within a constant, then if we test the residual of its result with boundary condition equal to zero (giving the infinite solution problem), or having f_0=0, it could be expected to be better However, the residual result was identical to the case when f_0 is non zero, because of the negligible effect that a non zero f_0 had in the calculation of the residual This suggests that the presence of a non zero f_0 has little or no effect on the multigrid solution (provided that we conform with equation -(3 0), and as a result, we cannot force the desired multigrid solution towards any size by setting the boundary conditions Thus the fractional residual can be changed by normalizing the solution This tells us that the actual values of fractional residual obtained, don't in fact indicate an exact measurement of performance, but act just as a guide line However, following the criteria given by equation -(3 0), all fractional residual plots are reproducible, and their relative performances can be given by their relative values of fractional residual in the plots Since multigrid can only solve to within a constant of the LUD solution (which offers the best results), the multigrid solution must be normalized to that of the LUD in order to obtain the error norm, or maximum error The absolute value of the error norm (or maximum) gives a realistic and quantitative assessment of the performance

The above results, which indicate problems on the lower grids, acted as an incentive to recode the problem, such that it would solve for $\mathbf{f}(\varepsilon)$, instead of the previous normalisation $\mathbf{n}(\varepsilon)$ Figure -(3 7) shows that when solving for $\mathbf{f}(\varepsilon)$, the performance of the CGCS is identical as is the CCC1 cycle 3,4, to the $\mathbf{n}(\varepsilon)$ choice However, one small insignificant difference was observed in that the CVC cycle 3,4, for $\mathbf{f}(\varepsilon)$ is not as effective as it is for $\mathbf{n}(\varepsilon)$ After observing other results, it was apparent that this recoding did not appear to remove the problem on the lower grids, as this problem was still present with the $\mathbf{f}(\varepsilon)$ normalisation

In an attempt to improve the unexpected bad performance results so far, several different adaptive iterations were considered, and the choice of ascending (forward) Gauss-Seidel was revised Red-black Gauss-Seidel is not as good a smoother as the basic or simple Gauss-Seidel, and ascending through the unknowns (forward) always

appeared to give better results than the choice of descending through the unknowns (backward)

If we consider the residual equation, this equation -(2 30) **r** = **A**v - **f**, depends on the size of the solution **v** The exact LUD solution for **v**, for the choice of energy interval dε = 4ev with 2^6-1 internal points, (which has been used so far), scales 11 orders of magnitude

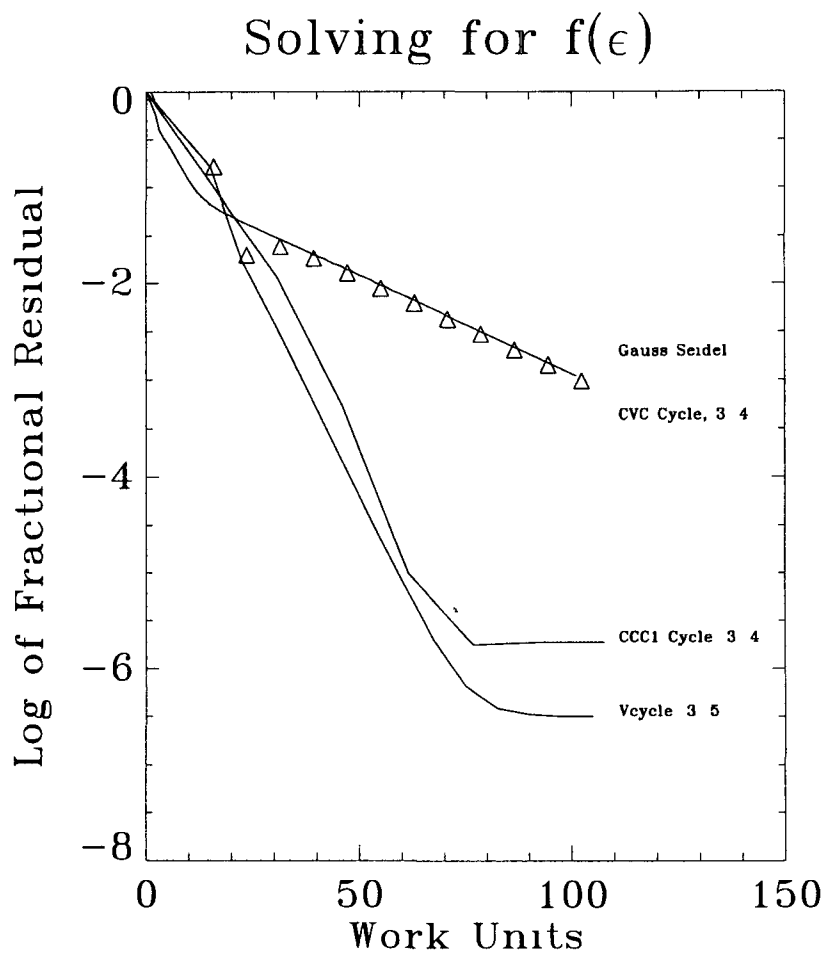


Figure -(3 7) solves for f(ε) instead of n(ε) The problem on the lower grid still remains

Figure -(3 0) shows that the first sixth of the curve contains the main peak of electrons, and it is this larger part of the solution that is more important in the residual equation, as this larger part dictates whether the fractional residual appears good or bad If we obtained a better solution for this region, the fractional residual would appear good, almost regardless of how the multigrid solution is behaving for the rest of the curve This point was tested by creating an iteration that updated this the first one sixth of the multigrid approximation, and then proceeded to update the entire approximation Thus more emphasis was placed on this larger region of the curve Figures -(3 8) and -(3 9)

give the best illustration of the adaptive case, for the case of some failing Vcycles 2,3, 2,4, and 3,3, the adaptive iteration can improve the error norms (the fractional residual was in fact greater than unity, which clearly suggests failure in these cases) The adaptation of updating approximately one sixth of the points was found to give the best results This adaptation was only found to be really effective if implemented as the Vcycle ascends, and omitted as the Vcycle descends Also, a similar adaptation on the lower grids was found to give slightly better results, if one iteration consists of updating about one quarter of the points, and then all the points, again putting more emphasis on the larger points, which in fact turn out to be in the same region of the curve as those on the fine grid More results from the adaptive case will be discussed later, but the adaptive case has not in any way eliminated the main problem as to why the multigrid solver fails when it incorporates the lower grids into the cycle

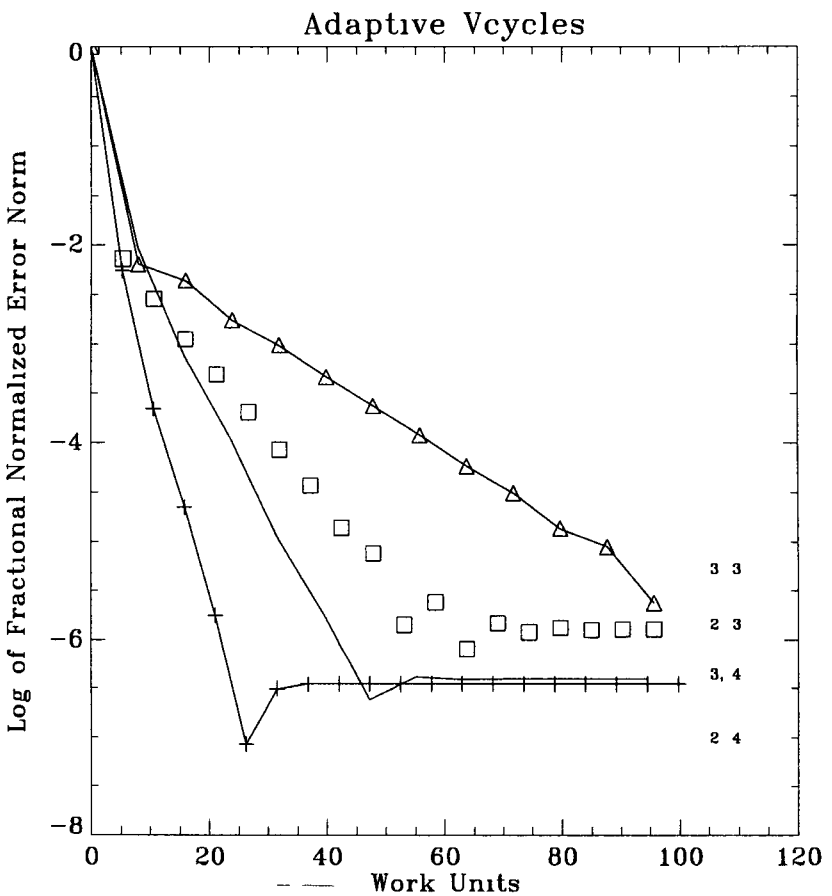


Figure -(3 8) shows the Vcycles with the adaptive iteration , improving results, (compared to figure -(3 9) below) but not removing the problem occurring on the lower grid

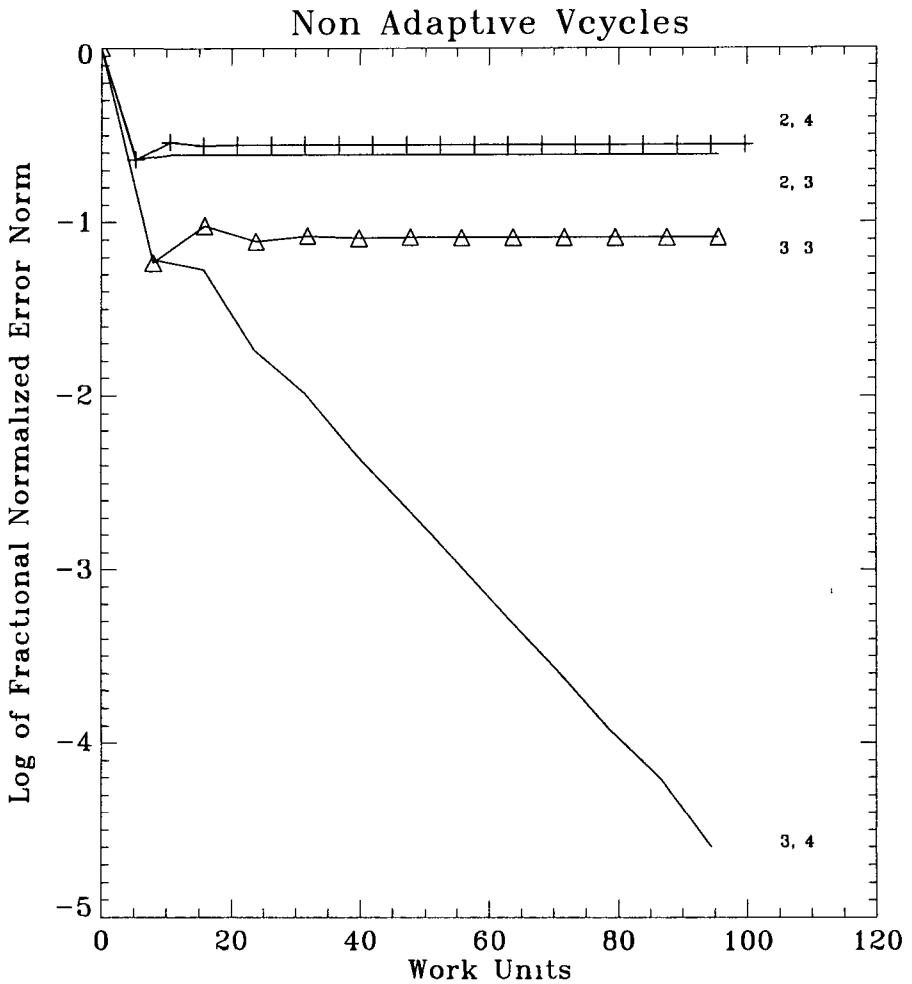


Figure -(3 9) shows that the same Vcycles as in figure -(3 8), but without the adaptive iteration

The Galerkin Restriction method however, if used instead of the previous direct method, shows more promising results The main difference being that this method doesn't require that the level be as high as 3 or 4, or that v be as high as 2 or 3 Unlike the direct method, the CGCS is not the most or one of the most effective results, e g Vcycles with level=5 Multigrid theory clearly states that it is more beneficial to descend down to lower grids Figure -(3 10) shows that going down to level four or lower achieves a much better performance than the CGCS This was perhaps the major problem with the direct method There seemed no reason why a Galerkin method should work, while this should fail

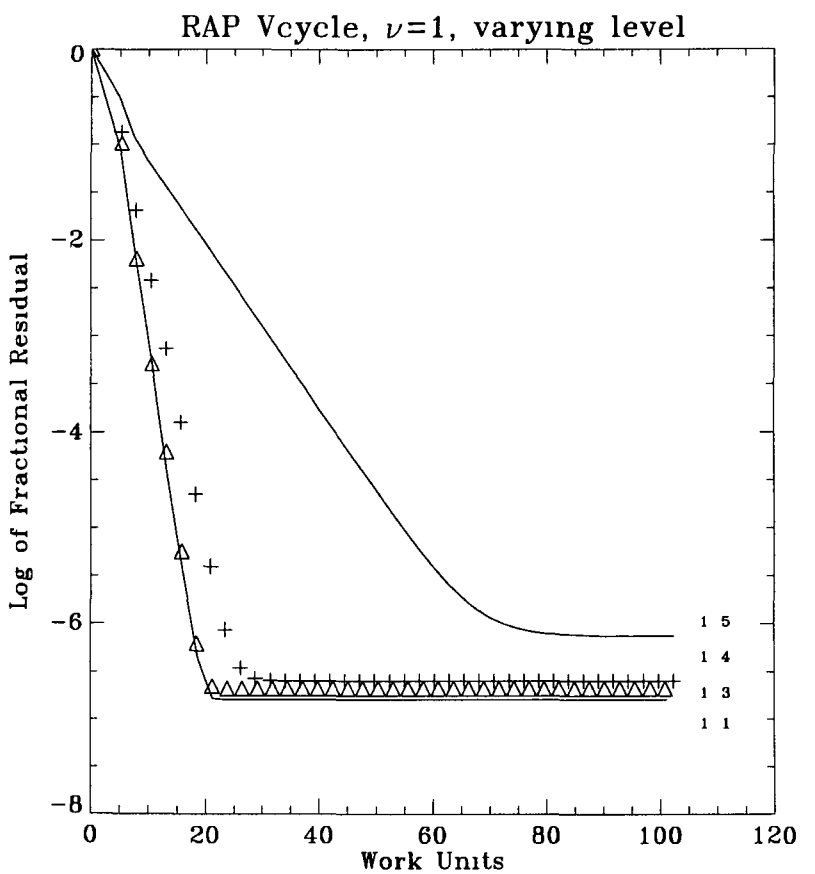


Figure -(3 10) shows that going down to level four or lower achieves a much better performance than the CGCS

When v=1, level=1, figure -(3 10) shows that the Galerkin method produces the best results This, (in contrast with the direct method), agrees with multigrid in general There is only the problem or question regarding the flattening out of the residual, somewhere between 1E-6 and 1E-7 The error norm and maximum error also flatten out after the same number of WUs This will be addressed later The more appropriate question to ask

now, would be as to why the direct method appears to fail, while the Galerkin method succeeds The fact that both succeed as expected for the Poisson case, suggests that the code is functioning correctly This was looked into further, and this was in fact the case

No preferred method for restriction between either of these had been previously suggested in any multigrid text or publications, only that both would be expected to succeed for all problems, not one or the other

If we consider the direct method first, the problem matrix is defined for level q, for 2^q -1 points It is then defined on level q-1 for 2^{q-1} -1 points, and so on down to level 1 The data file, containing the cross sections, also contains the threshold energies, of which there are, for example, four in the case of a plasma consisting of N_2 molecules, two electronic excitations, one vibrational and one ionisation The cross sections are given in eV, and must be converted to points on the grid, which are equally spaced Thus, if the problem matrix is to be redefined on each level, corresponding thresholds must also be redefined for each level This was achieved in such a manner so as to obtain thresholds similar to those obtained if the problem had been defined initially on that lower grid Given that there are four threshold energies in the data file, a one, three or seven grid system would only contain 1 or 2 of those four thresholds, thus the system would be different Also, the whole physical concept of redefining the system with 1, 3 and 7 energy points is not quite convincing Clearly there are not enough energy points to make up a system which in any way represents the original one By redefining the problem with less points, because of the thresholds, it is as though we are defining a different system In contrast to this, the Galerkin method simply manipulates the fine grid matrix **A**, by finding **RAP** (equation -(2 45)) to give **A** for the next lowest grid, and so on down the lower grids The matrices **R** and **P** contain simple fractions Threshold information is not lost in this manner Nothing is redefined for a smaller number of points, and all the information that exists in the original problem is still preserved in **A** for the lower grids This is the suggested reason for the success of the Galerkin method, and the failure of the direct method

Figure -(3 10), also shows that there is little or no difference between descending to level 1, 2 or level 3, although level 1 is slightly better These however perform better than level 4 This seems to indicate that work done on grids 1 and 2 don't seem to contribute much to the general scheme of things This does comply with multigrid theory

A sub routine was incorporated into the program from Numerical Recipes in C [5], in order to obtain the eigenvalues for the matrix **A** It first finds the inverse, using the Gauss Jordan method, and then proceeds to numerically calculate the eigenvalues Each eigenvalue corresponds to an eigenvector, and the set of eigenvectors can define any error vector **e**, which gives the difference between the exact solution **u**, and the approximation **v** The effect of continuous Gauss Seidel iterations applied on the approximation **v**, for the problem **Av=f**, effectively remove the oscillatory or higher frequency eigenvector elements of the error **e** This is the smoothing effect of the Gauss Seidel iteration By calculating the eigenvalues for the problem matrix on all levels, it was found that for the particular choice of 2^6-1 points, and with dε= 4eV, the maximum eigenvalues for levels 4 or lower were in fact greater than unity This does not conform to the required convergence condition given by equation -(2 23), $|\lambda(\mathbf{P}_g)|_{max} < 1$

This indicates that the Gauss Seidel iteration cannot converge on these grids, since the maximum eigenvalue is greater than unity Although it was not possible to obtain plots of the eigenvalues for each mode present in the error, or study the actual size of each eigenvalue element of the error, we know that for the Gauss Seidel case the largest eigenvalue is given by the lowest frequency mode [7] This is probably the case for the Boltzmann case, and if any of the lower eigenvector modes have corresponding eigenvalues larger than one, this means that these eigenvector modes (error elements) cannot be removed by the Gauss-Seidel iteration However, multigrid does not require that the lower frequency modes be completely removed, only on the coarsest grid, and this is perhaps the reason why multigrid works even though convergence is not possible on levels 4 or lower Recalling the observation that for the Boltzmann case, multigrid, regardless of the fact that the boundary conditions force a unique solution, only solves the problem to within a constant, we can speculate that if the lowest frequency could not be removed by multigrid, then it would perhaps be observed as a smooth component, (we don't know the form or shape of this mode), and on a Log scale this could possibly appear as a constant value error, e g the mulitgrid solution appears to have solved to within a constant, but perhaps this is in fact the lowest frequency mode, or a combination of some low frequency modes which could not be removed by multigrid because the maximum eigenvalues on levels four or lower are greater than unity

Figure -(3 11) shows the Galerkin method (using **RAP**, as in equation -(2 45)) for the Poisson case, with ν=1, and level=1-5 There is definitely an advantage in descending down to the lower levels, in particular, the lowest level

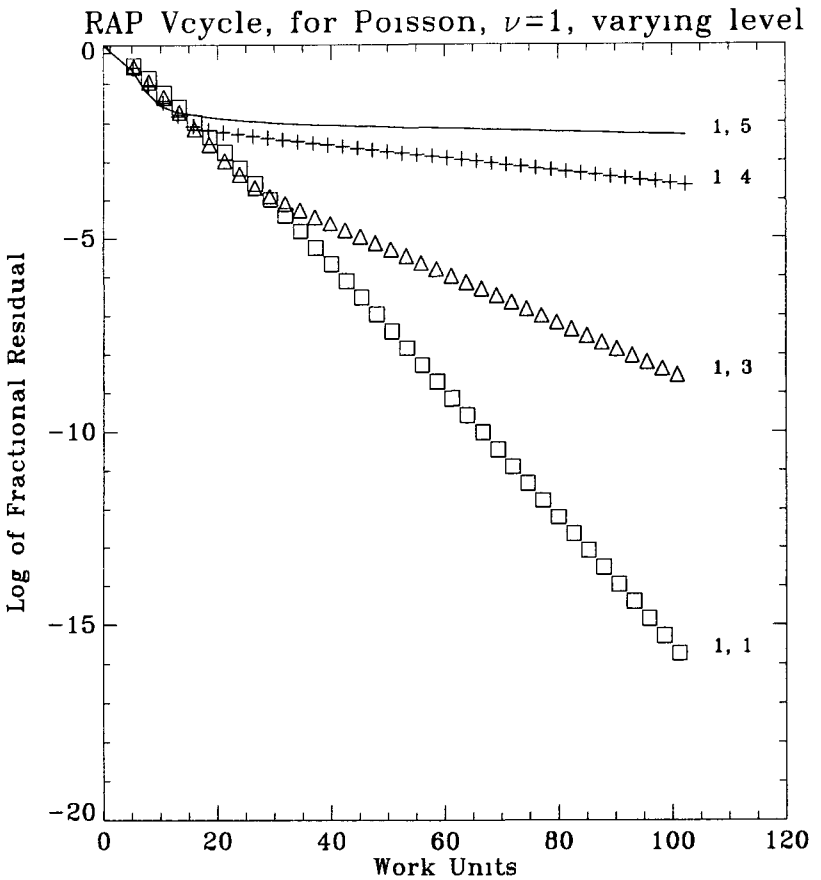


Figure -(3 11) shows the **RAP** (Galerkin method) for the Poisson case, with v=1, and level=1 to 5

This is the type of multigrid behaviour we would expect in figure -(3 10), but there is no real advantage in descending below level 3 However, the slope of these performances are better for the Boltzmann, for the first 25 WU or so, and then the fractional residual flattens out This behaviour is not seen in the Poisson case, again in this respect, we would not expect this flattening out for multigrid behaviour Could it be possible to attribute these bad elements to the eigenvalue size on lower levels?

Figure -(3 12) shows that LINEAR restriction worked better than WEIGHTED, and when restriction type A2H is used, the method fails completely The best results so far are then given in figure -(3 10), with 1,1 (v=1, level=1)

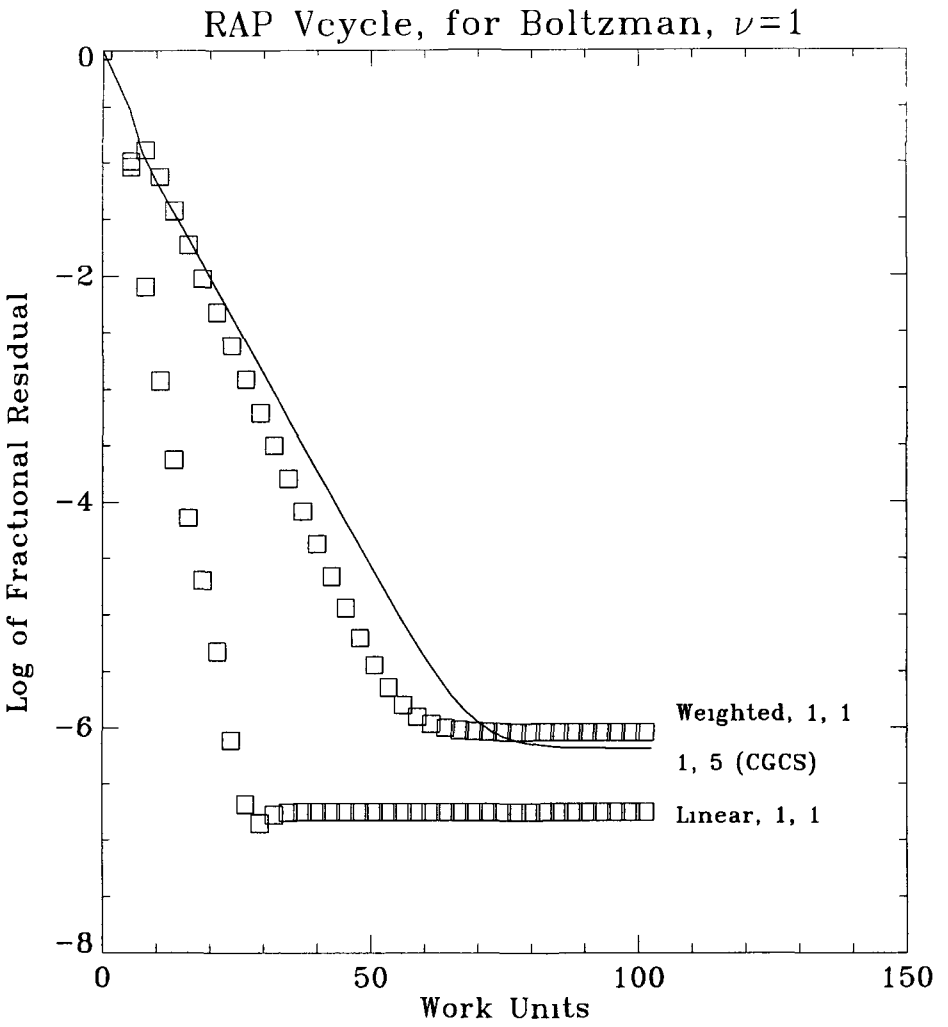


Figure -(3 12) shows that for the Boltzmann case, the Vcycle with **RAP** (Galerkin method of restriction) , LINEAR restriction worked better than WEIGHTED

Instead of using either type A2H or AH for restriction, the matrix operator **R** can be employed [7,18,23] The matrix operator **P** can be used for interpolation Since, however most of the elements in both these operators are zero, there is a lot of wasted calculation, compared to the previous AH and A2H Figure -(3 13) shows that the results of this are slightly better than figure -(3 10), where the fractional residual flattens out around 20 WU as opposed to 25

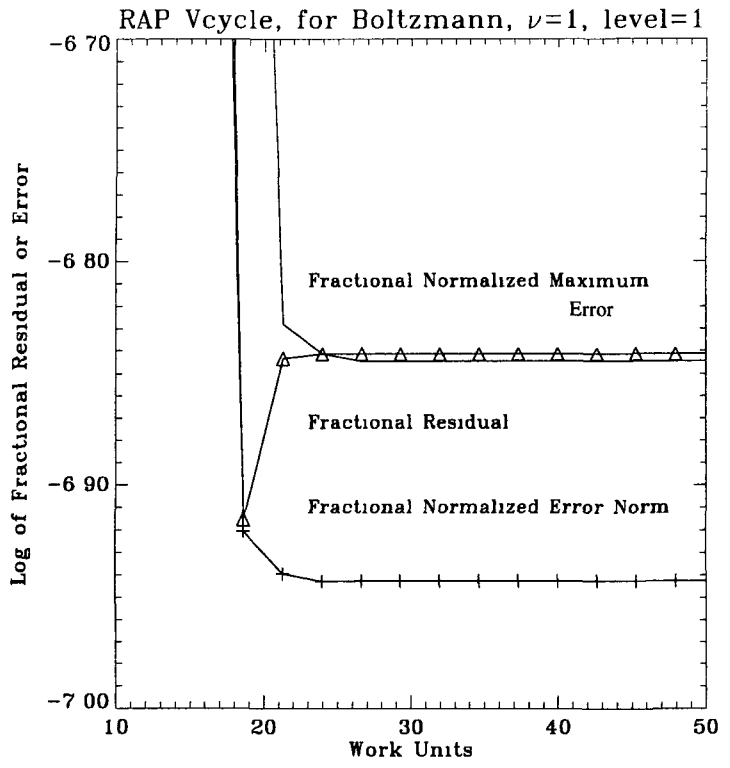


Figure -(3 13) shows **RAP** (Galerkin method) Vcycle, using **R** and **P** instead of AH or A2H methods

If the error norm or the maximum error is plotted instead of fractional residual, as in figure -(3 13), we see that the magnitude is of the same order of fractional residual Also the behaviour is identical, e g both errors flatten out after exactly the same number of WUs After this point, there seems to be no advantage in continuing the method There is also no advantage in reducing the order of the error smaller than the discretization error E <= $O(h^p)$, [1,7,18,23], where p is the order of the differential equation, 2, and h is the size of the interval, such that all intervals add to 1 For the present q=6 case, h=1/63, and E = O 10^{-4}-10^{-5} Figure -(3 15) shows that to reduce the order of the error, to 10^{-4}-10^{-5}, requires around 6-9 WU It is not necessary to progress further than this, and from this point of view, the method is a success The problem of the flattening out of fractional residual or error does not effect the practical application of the result

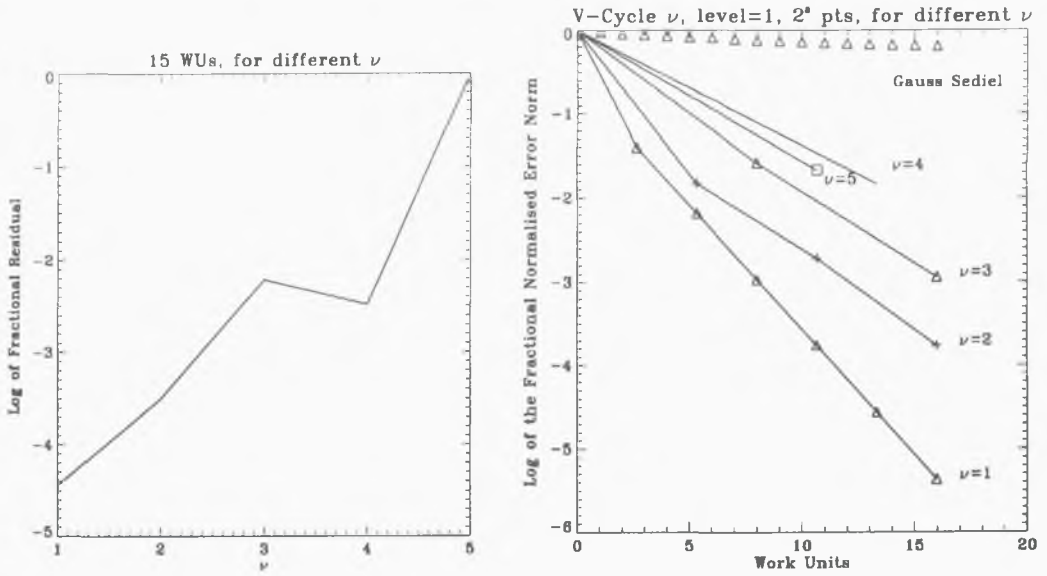


Figure -(3.14) shows the fractional residual after 15 WU's, for different ν. Figure -(3.15) shows (more clearly for 2^8-1 points), how ν=1 is the best choice in agreement with the previous figure.

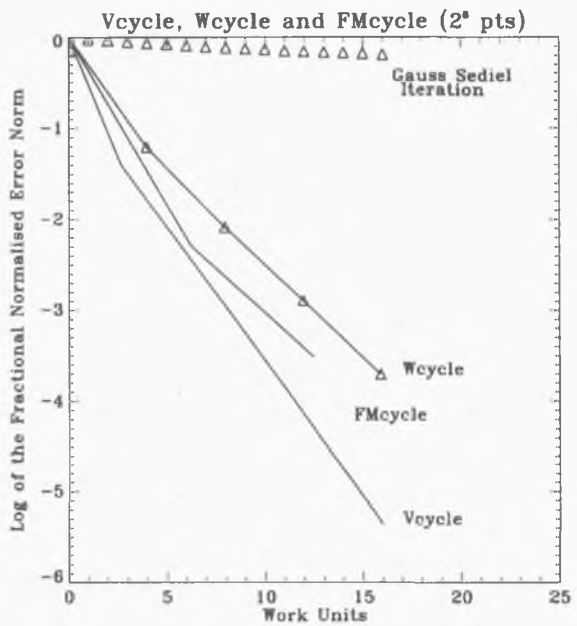


Figure -(3.16) shows again that the Vcycle offers a better performance than any other type of cycle (for 2^8-1 points).

Both the calculation of the maximum error and the error norm, involve knowledge of the solution, which, of coarse is not readily available Apart from calculating fractional residual, another, perhaps even less costly way of predicting the point where this flattening will occur is to calculate the fractional change in each f_l on the approximation from its previous value, given by $(f_l-f_{l-1})/f_l$, and continuing the method until this quantity is smaller than some suggested value,

$$(f_l-f_{l-1})/f_l < \rho \qquad -(3\ 1)$$

Figure -(3 14) shows the residuals of different v after 4 Vcycles, 15 WU's The choice of v=1 is clearly the best for levels=1,2,3,4 The choice of either level =1, 2 or 3 as mentioned before, appears to make little difference, only that all are better than descending to level 4 Figure -(3 15) also shows quite clearly that for around the first 30 WU's, with level=1, the best choice of v is 1 (more clearly seen for 2^8-1 points) This agrees with the Poisson case and that reported [1,7,18,23]

Figure -(3 16) shows that the Vcycle offers a better performance than any other type of cycle The poor performance of the fm1a and fm1b FMcycles (exact definitions given in appendix C) is concerning, as most reports suggest that the full multigrid cycles are more efficient than the Vcycle The Vcycle was also found the most effective for the Poisson case, again not agreeing with Briggs, Brandt or Wesseling [7,23,3]

Efficiency

The only solvers that in two or more dimensions can perhaps be comparable to the speed of the multigrid solvers is the direct solver based on the Fast Fourier Transform, but these are more limited than multigrid and cannot be compared for very large meshes McCormick maintains that the efficiency of full multigrid cycle attains the same efficiency for the general nonlinear, not necessarily elliptic problems as in the case of the two dimensional Poisson equation, but for any boundary condition problems, eigen problems, and for problems including free surfaces, shocks, reentrant corners, discountious coefficients and other singularities Unfortunately, the full multigrid cycle appears to be less effective than the Vcycle, so fast methods may be at least comparable to the multigrid

method for the Boltzmann case. The Vcycle must be reapplied again and again until the desired accuracy is maintained.

The required reduction in error is such to give an error norm of around the same size as the truncation error due to finite differencing with spacing h [1,7,18,23]

$$\mathbf{u} - \mathbf{v} = \mathbf{e} \sim Kh^2 = K(1/N)^2 = K(1/2^q)^2 = K2^{-2q} \qquad -(3.2)$$

where K is constant. The error norm **e** is to be reduced to around this amount for a q level system with 2^q points. Figure -(3.17) plots, (as we increase the number of points), the number of mathematical manipulations required to reduce the error norm in accordance to equation -(3.2). This curve is based on the calculation of $2N^2$ manipulations for one fine grid iteration (or 1WU), and just under 3WU's ($6N^2$) for one Vcycle, (ν=1, level=1), (A more accurate calculation will show that this is 2.66WU's, however, we have neglected intergrid transfer work estimated at around 15% or more of 1WU, which will bring this to around 3WU's). The choice of K has little effect on this curve for the multigrid case. It can be seen that for $q > 6$, multigrid becomes cheaper than the direct LUD method.

Figure -(3.15) shows that for 2-3 Vcycles with ν=1 and level=1 to 3, will reduce the error to the order of around 10^{-4}-10^{-5}. The direct LUD method requires $1/3N^3$ manipulations, as opposed to the $2N^2$ required per iteration, which gives the definition of 1WU. The number of WU's required for the direct method then becomes around $1/6N$, for q=6, this is around 10.5WU's. Two Vcycles costs less than 6WU's, and thus the multigrid method is cheaper once $1/6N$ WU's in the direct method is larger than 6. This is little improvement on the LUD case of 10.5WU's.

The 6WU's required for two Vcycles takes into consideration the work required for inter-grid transfer, which for the Boltzmann case is less than the estimate of around 15-20% of the work done per V-Cycle [7] for the Poisson problem, since the cost of one WU is more for the Boltzmann case. This is because of the there are more of diagonal matrix elements for the Boltzmann case.

If q=8, then $1/6N$ is around 42WU's. However, the truncation error is now $O(1/256)^2 \sim 10^{-5}$, and an error less than 10^{-5}-10^{-6} is now required, which in turn requires more WU's than for the q=6 case. The number of Vcycles required now becomes, 3-4, to reduce the error to 10^{-5}-10^{-6}. This costs around 9-12 WU's (where WU is defined as one iteration on the fine

grid with 2^8 - 1) For 2^{10} - 1 points, this error becomes of order 10^6 However, it is still possible to achieve this before the residual flattens out Thus for q > 6, multigrid has a clear advantage over the direct method, and more so as both the size N, and the number of dimensions of the problem increases

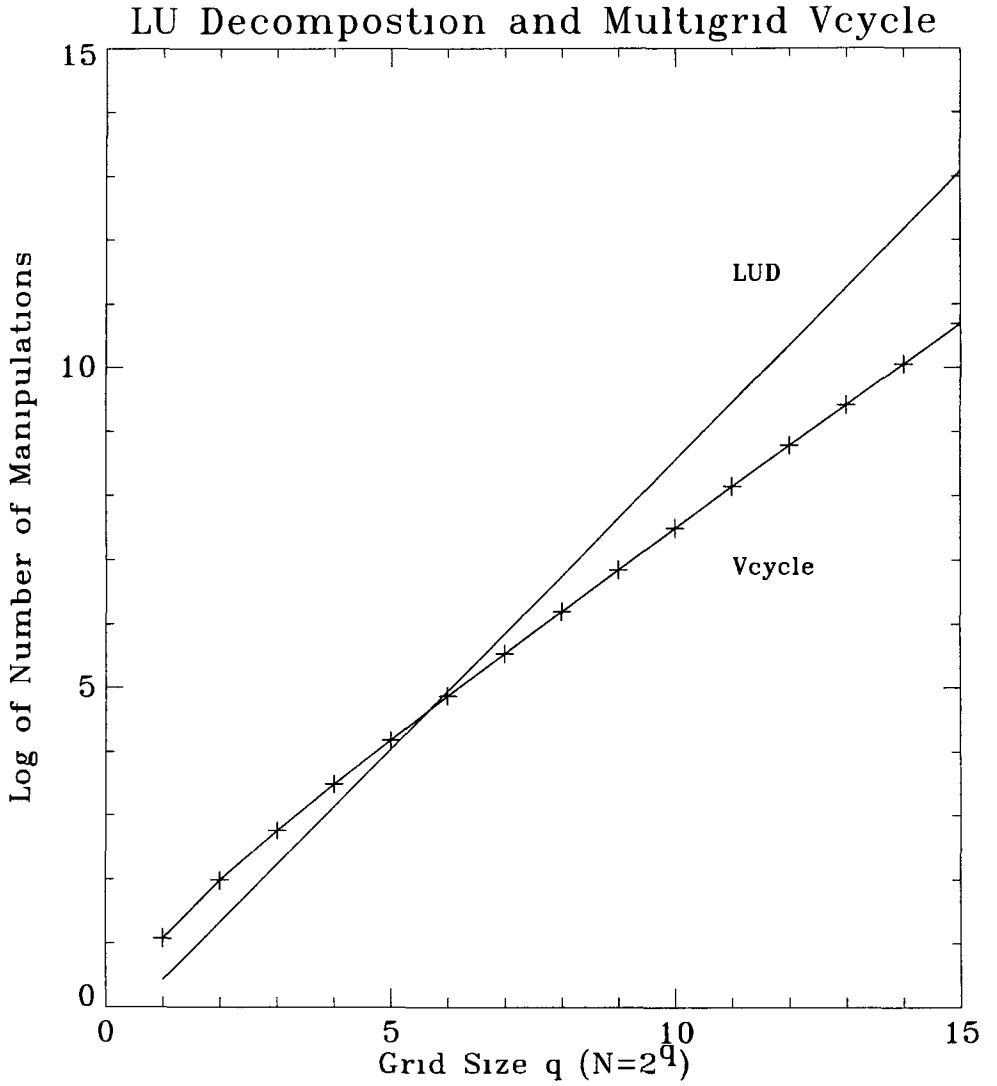


Figure -(3 17) shows, as we increase the number of points, the number of mathematical manipulations required to reduce the error norm in accordance to equation -(3 2)

Conclusions

The residual was not reproducible after the same number of multigrid cycles, only because of the size (Σf_{guess}) of the initial guess relative to the size (Σf_{sol}) of the solution given by the LUD solver The fractional residual was only found to be reproducible once equation -(3 0) was satisfied

$$\Sigma f_{sol}(LUD) > \Sigma f_{guess} \qquad -(3\ 0)$$

This is unlike the Poisson case Multigrid for the Boltzmann problem only solved to within a constant of the solution, regardless of whether the problem had an infinite series solution or a unique solution However, multigrid only does this if the equation -(3 0) is satisfied, otherwise it is clearly seen to fail for the Boltzmann case The direct method was looked at first for all four combinations of the restriction types AH or A2H and LINEAR or WEIGHTED The Coarse Grid Correction Scheme (CGCS, Vcycle descending only one level), was found to give the best performance Descending to lower grids decreased the performance, and for a 6 level system, descending below level 3, and in some cases 4 resulted in failure of the method A value of ν=1 also caused the method to fail, and neither of these observations agrees with multigrid theory All cycles appear to stop converging somewhere between a fractional residual of 10^{-3} and 10^{-6}

If the approximation is normalised to that of the LUD solution after each cycle, this results in much faster convergence However, the size Σf_{sol} (LUD) is then required, and thus the solution needs to be known, which then defeats the purpose Normalising to other sizes (unity for example) tended to give non reproducible results depending on both Σf_{sol}(LUD), Σf_{guess}, and did not result in the any reproducible improvement in convergence

Provided we conform to equation -(3 0), multigrid proceeds to converge to within a constant When the problem was defined for f(ε) instead of n(ε), the results were similar The problem of a failing method on the lower grids still occurred

The ordinary Gauss Seidel forward iteration was found to be a better smoother than Red-Black Gauss Seidel, or backward iterations

An adaptive iteration was developed which updated the first one sixth of the multigrid approximation, and then all the points on the approximation This focused more on the larger part of the electron distribution, as this larger part dictates the results in the form of the fractional residual and the error norms This adaptive iteration improved convergence, but did not remove the problem on the lower grids This adaptation was only found to be effective if implemented as the Vcycle ascends and omitted as it descends Similar adaptations to the basic iteration for the two lower grids also improved performance

The Galerkin method of restriction gives better convergence than the direct method for the Vcycle With the Galerkin method, the optimum choice of $\nu=1$ is in accordance with convergence estimates for similar second order differential problems Also, incorporating the lower grids into the cycle increases the performance for the Galerkin method, where as for the direct method, this reduces the performance, and if incorporating grid levels 3 or below, the method fails The Galerkin method is seen to behave as expected, apart from the flattening out of the residual (where convergence stops)

The Galerkin method employing **R** and **P** worked better than AH/A2H and LINEAR/WEIGHTED restriction methods, but uses more storage space

The value of $|\lambda(P_g)|_{max}$ for the iteration matrix P_g was found to be larger than unity for the Gauss Seidel iteration on the lower grids, indicating that the iteration can not converge on these levels It is suggested that $|\lambda(P_g)|_{max}$ is probably the k=1 eigenvalue or $|\lambda(P_g(k=1))|$ This corresponds to the lowest frequency eigenvector If this is the case, then it is this element of the error that cannot be removed This is the suggested reason as to why multigrid for the Boltzmann stops converging, and the error norms or the fractional residual flattens out

The Vcycle performs better than any other cycle, including the FMcycle This is concerning as it is not in agreement with Briggs, Brandt and other texts [3,4,6,7,23] We might however expect the Vcycle to perform better than the Wcycle

Fortunately, multigrid converges to the discretization error $E <= O(2^{-2q})$, before it stops or seizes to converge, for systems of point size 2^q-1, for $q = 6$ to 12 From this perspective, the method achieves what is asked of it before it seizes to converge, as there is no benefit in converging to below the size of the discretization error Thus the problem of the flattening out of the residual does not effect the practical application of the result

Multigrid methods are more favourable than direct methods given the accuracy required for the Boltzmann problem For q>6 multigrid becomes cheaper than the direct LUD method, when just enough Vcycles are applied to converge down to the same size as the discretization error For a 2^6 points problem, multigrid only requires around 60% of the time needed by LUD For a 2^8 points problem, multigrid requires around 20%, and for 2^{10} points around 5% The reduction in work load is also seen with increasing dimension

The multigrid method is less specialised than existing fast solvers, and in the case of the Boltzmann problem has the advantage that the method can be manipulated to cope with the non-linear case involving electron-electron collisions This can then be solved, with no substantial loss in efficiency

APPENDIX

Appendix A

This appendix contains more information about the C source code, and some guidelines for the program Multigrid_Study c A macro listing is given, and the purpose of each function is given beside each C function prototype A program listing of all the C source code, and header files is given on an attached disc, along with a makefile, used to compile these source programs The original program reads from two input files, b17 dat, which used the file pointer infile, and xs dat which used the file pointer xsfile The results file for the plasma properties is b17 res, which uses the file pointer outfile All other files begin with fp, followed by the file name, and have the extension dat All file pointers also begin with fp and contain the same name as the files, omitting the dat extension Also, Write() functions should usually indicate by their name what file pointer they are using, and output file they are writing to

List of Multigrid Output Data Files

Below is a list of multigrid output files, the purpose of which to view multigrid performance under different settings

fpbc dat	Records the area under the solution with the boundary condition
fpcoarse dat	Records the solution and approximation at every grid level
fpcycle dat	Records the fractional residual and other parameters with increasing cycles, for all cycles chosen
fpdfmax dat	Records the maximum difference between the previous i^{th} point and the new i^{th} point in the multigrid approximation, as the number of cycles increases
fpfour dat	Records the sine fourier transform of the error, (f_{sol}-f, where f_{sol} is the LUD solution and f is the approximation)
fpeedf dat	Records the LUD solution and multigrid approximation, with energy and grid point
fpeigen dat	Records the eigenvalues of the problem matrix A for each grid level It can also store this for different energy interval dε
fpminres dat	Records the best multigrid performance

List of C Macros used in the Program

The following is a macro list for multigrid macro's in the main program Multigrid_Study c Most macro's are tested for being either 1 (true), or 0 (false), by C statements The order of the macro's is the same as in the program Multigrid_Study c

BOLTZ	If true, then Boltzmann problem is set, if false, Poisson problem
LINEAR	If true, linear restrict() is used
WEIGHT	If true, weighted restrict() is used
WNO	If true, a range of different weighting factors are used
A2H	If true, A2H restrict() is used
AH	If true, AH restrict() is used
RAP	If true, then the Galerkin method is used If false, the direct method or redefinition method

The program will scan through the range given for v, and level

MEWSTART	The smallest number of iterations to perform on each grid
MEWFINISH	The largest number of iterations to perform on each grid
LEVELSTART	The lowest level to perform cycles to
LEVELFINISH	The highest level to perform cycles to

MINRES	If true, then the function Minres() is executed
GRIDSOLN	If true, then LUD solutions are found for every level

If any of the following are true, then the program executes the multigrid cycle function corresponding to each one If they are false, then the functions are ignored

VCYCLE
FMCYCLE1A
FMCYCLE1B
FMCYCLE2A
FMCYCLE2B
WCYCLE
CCC1
CVC
ITS (Gauss-Seidal iterations)

IN_ELASTIC If true, the problem is inelastic, if false, elastic

LUD If true, then the LUD method is employed on reaching the lowest level in

the cycle instead of Gauss-Seidel iterations If false, then the normal multigrid approach of performing iterations on the lowest level is then employed

GAUSS_JORDAN If true, then the eigenvalues of the problem matrix A are found using Gauss-Jordan elimination The Numerical Recipes [10] function gaussj() finds the inverse of the problem matrix A The functions hqr() and elmhes() [10], then proceed to find the eigenvalues of A If false, then none of this is executed

POSITIVE_DEFINITE Gives the option of calculating if A is a positive definite matrix

MODE If true, the function Mode_Sweep() is executed, which sweeps over all the modes for the initial conditions for the Poisson case The modes (or eigen functions) are not known for the Boltzmann case

DE_SWEEP If true, each time the program is run, the energy interval changes by a fixed amount, thus allowing for results at different energy intervals without manually changing the interval

WUCYCLES The maximum number of iterations or work cycles (WU's) performed each time the multigrid system is set

CYCLES2 The number of times the second cycle is repeated in the CCC1 cycle

List of C function Prototypes

The prototypes for the different functions in the program are given and an a brief explanation is given to the role of each function

double **New_de**(int total, double de),

Returns a new energy interval de, which has been incremented by 1/total

void **Mode_Sweep**(int qq, int NOofWs, int mode_start, int mode_finish, ITERATE *T),

This prints the residual results from a range of initial guess modes for the Poisson between mode_start and mode_finish The problem is set up for NOofWs different restriction weighting factors

void **Fourier**(int q, ITERATE *T),

Obtains the sine fourier transform of the error between the solution and the multigrid guess, and places back in the structure T

void **Alloc_Structures**(INELASTIC **xsi,
ELASTIC **xse,
ITERATE **T,
MINIMUM **mr,
FILL **F),

This function allocates memory for a pointer to each of the five arrays of structures in the argument list

void **Alloc_T**(int qq, int *ngrid, ITERATE *T),

Allocates memory for the contents of the structure T

void **Alloc_F**(int qq, ITERATE *T, FILL *F),

Allocates memory for the contents of the structure F

void **Alloc_mr**(int qq, ITERATE *T, MINIMUM *mr),

Allocates memory for the contents of the structure mr

void **Set_T_F_mr**(int qq, ITERATE *T, FILL *F, MINIMUM *mr, double source),

Set certain multigrid parameters in the structures in accordance to the macros

void **Write_fpminres**(MINIMUM *mr, double de),

Outputs the multigrid details of the smallest residuals to file fpminres dat This file will indicate the best cycles and settings, as the number of WU's are increased

void **Write_fpeedf**(int q, ITERATE *T),

Outputs the EEDF of the LUD solution and the multigrid approximation to file fpeedf dat

void **Write_fourier**(int q, ITERATE *T),

Outputs the sine fourier transform of the error between the LUD solution and the multigrid approximation to the file fpfour dat

void **Write_fpbc**(double frhs, double norm),

Outputs the boundary conditions, along with the area under the LUD solution (the size of the LUD solution, Σf_{sol})

void **Write_dfmax**(int i, double dfmax),

Outputs the maximum difference between the previous i^{th} point and the new i^{th} point in the multigrid approximation f, as the number of cycles increases

void **Set_fpcycle**(ITERATE *T),

Opens the file fpcycle dat, which will show results of the different cycles

void **Eigen**(int qq, int *ngrid, double ***a),

Calculates the eigen values of the matrix A on level qq, and prints the resulting eigenvalues to the screen

int **Cal_TypeNo**(void),

Returns the number of different cycles that the program will use

int **Set_Wexecute**(ITERATE *T),

Returns the number of times the for loop which contains it is repeated, depending on the two macros LINEAR and WEIGHT These give the option of weighted restriction or linear (or both)

void **Set_Weights**(int wr, ITERATE *T),

Sets the multigrid parameters inside the structure T, to either linear, weighted or both

int **Set_Aexecute**(ITERATE *T),

Sets the number of times the next for loop is repeated, depending on the two macros AH and A2H, which correspond to the type of restriction discussed earlier

void **Set_Restrict**(int ah2h, ITERATE *T),

Sets the type of restriction in the multigrid structure T

void **Set_Mew**(int mew, ITERATE *T),

Sets the number of iterations per grid ν *into the structure*

void **wgas**(FILE *file, ELASTIC *xse, int nxse),

Outputs some gas parameters of the plasma to file b17 res

void **wheader**(FILE *file),

Outputs the header for the file b17 res

void **wf**(FILE *file ,
double **f ,
double de ,
double ne ,
int *ngrid ,

int qq),

Outputs the EEDF, with increasing energy to b17 res

void **wtran**(FILE *file ,
double ebar ,
double ek ,
double diff ,
double mu ,
double vd ,
double alpha ,
double nu ,
double alphabar ,
double n),

Outputs the plasma transport parameters to b17 res

void **wppart**(FILE *file ,
double ef ,
double es ,
double eel ,
double einel ,
double eden),

Outputs the power partition in the plasma taken by both elastic and inelastic collisions

void **wcparam**(FILE *file ,
int *ngrid ,
double de ,
double dt ,
int nstep ,
double eb ,
double en ,
double tg ,
double n ,
int qq),

Outputs the grid interval, energy conservation, gas temperature and pressure

```
void    evalalpha(double *alphabar ,
          double *alpha ,
          double *nu ,
          double vd ,
          ELASTIC *xse ,
          INELASTIC *xsi ,
          int nxsi ),
```

Involved with the calculation of how the energy is conserved

```
void    evaldiff(double *diff ,
          double **f ,
          double ne ,
          int *ngrid ,
          double de ,
          ELASTIC *xse ,
          int nxse ,
          int qq ),
```

Calculates the diffusion coefficient

```
void    evalrates(double **f ,
          int *ngrid ,
          INELASTIC *xsi ,
          int nxsi ,
          double ne ,
          int qq ),
```

Calculates the rate constants for each of the different collisional processes

```
double Ebal(  int qq,
              INELASTIC *xsi ,
              ELASTIC *xse ,
              int nxsi ,
              double de ,
              double *ne ,
```

```
        double *ef ,
        double *einel ,
        double *eel ,
        double *eden ,
        double *es ,
        double *ebar ,
        ITERATE *T,
        FILL *F ),
```

Calculates the energy conservation

```
void    Fillc( int q,
        INELASTIC *xsi,
        ELASTIC *xse,
        int nxsi,
        double dt,
        double de,
        FILL *F,
        ITERATE *T,
        int fe ),
```

This calculates the elements of the problem matrix ***A***

```
void    datsort(ELASTIC *xse ,
         int nxse ,
         double *density ,
         char **label ,
         int ns ,
         double n ),
```

Sorts out the data, giving the cross-sections at equal distances Calculates the densities

```
void    Evalab( int q,
                ELASTIC *xse,
                int nxse,
                double n,
                double en,
```

```
                double de,
                double tg
                FILL *F,
                ITERATE *T ),
```

Calculates the tridiagonal elements used for the problem matrix A

```
void    Evalab_fe( int q,
                ELASTIC *xse,
                int nxse,
                double n,
                double en,
                double de,
                double tg,
                FILL *F,
                ITERATE *T),
```

*Calculates the tridiagonal elements used for the problem matrix **A**, only it gives the **f**(ε) normalization problem instead of the **n**(ε)*

```
void    wrate(FILE *outfile ,
        INELASTIC *xsi ,
        int nxsi ,
        ELASTIC *xse ,
        int nxse ),
```

Outputs the collisional rates between electrons and neutral ions for each of the different excitations

```
void    rxsfil(FILE *xsfile ,
        INELASTIC *xsi ,
        int *nxsi ,
        ELASTIC *xse ,
        int *nxse ,
        int *ngrid ,
        double de ,
        int itype ,
        int qq ),
```

Reads in the collisional cross sections for the different excitations from the data file xs dat, and places them into the structures ***xse*** *and* ***xsi***

```
void    xserror(char error_text[] ,
          char cross_section_name[] ,
          char species_name[] ),
```

This function is called in the event of an error, should one arise in the management of the cross section data It prints an error message to the screen

```
void    rxs(FILE *file ,
        double e[] ,
        double xs[] ,
        int npnt ,
        char label[] ,
        char name[] ) /* Read a number of paired data points */,
```

This function is employed by rxsfil, to read in a pair of data points from the cross section file

```
void    rdatfil(FILE *infile ,
          char xsfilnam[] ,
          double *de ,
          double *dt ,
          double *en ,
          double *n ,
          double *ne ,
          int *ns ,
          double density[] ,
          char **label ,
          double *tg ,
          int *itype ,
          double *source ,
          int *qq ),
```

Reads in plasma settings from the data file b17 dat

ITERATE *allocitrn(int nl ,
int nh),

*Allocates memory space for a pointer to the structure **T***

MINIMUM *allocmr(int nl ,
int nh),

*Allocates memory space for a pointer to the structure **mr***

FILL *allocF(int nl ,
int nh),

*Allocates memory space for a pointer to the structure **F***

char *fgetlin(char *str ,
int len ,
FILE *file),

*Employed by the function **rxs()** to read in a line from a data file, of a given length*

INELASTIC *allocxsi(int nl ,
int nh),

*Allocates memory space for a pointer to the structure **xsi***

ELASTIC *allocxse(int nl ,
int nh),

*Allocates memory space for a pointer to the structure **xse***

char *cvector(int n),

Numerical Recipe [10] function, which allocates memory for an array of characters of size n

char **sarray(int nl ,
int nh ,

int length),

Numerical Recipe [10] function, which allocates memory for an matrix of characters of size nh by length

void **free_sarray**(char **s ,
int nl ,
int nh ,
int length),

Numerical Recipe [10] function, which frees memory space allocated by the function sarray

void **spline**(double x[] ,
double y[] ,
int n ,
double yp1 ,
double ypn ,
double y2[]),

Numerical Recipe [10] function, which for a given mathematical table function y(x), calculates the second order derivative

void **xsinterp**(double xa[] ,
double ya[] ,
double y2a[] ,
int n ,
double x ,
double *y ,
int itype),

Interpolates the cross section data so that there is equal distance between the resulting cross sections Interpolation can be linear or cubic

void **ludcmp**(double **a ,
int n ,
int *indx ,
double *d),

Numerical Recipe [10] function, given an n.n matrix a, it replaces a by the LU decomposition of a rowwise permutation of itself This must be used in conjunction with ***lubksb()***

```
void    lubksb(double **a ,
               int n ,
               int *indx ,
               double b[] ),
```

Numerical Recipe [10] function, used after the above function, which implements backward substitution, and outputs the solution into vector b, which is the solution to the matrix problem **Ab = d**

```
void    thres(int n ,
              int *yh ,
              int *yl ,
              int qq ),
```

This function is used to calculate the energy thresholds for the lower level, when the direct matrix redefinition method is employed, as opposed to the Galerkin method

```
void    Matrix(   int q ,
                  int qq ,
                  int *ngrid    ,
                  double h ,
                  double ***a ),
```

Fills the problem matrix A for the Poisson case

```
void    Minres(double wu,
                    int i,
                    int totalwu,
                    double res,
                    int mew,
                    int level,
                    int ah2h,
                    char TypeName[],
```

MINIMUM *mr),

Tests for the best multigrid performance, from all cycles and parameters used, with increasing WU The cycle number, type, v, and level is recorded for each WU

void **Solve**(int q, ITERATE *T, int lud),

Obtains the LUD solution to the matrix problem, putting the solution into f_{sol}, inside structure T

void **Set_Modes**(int q, ITERATE *T),

Sets the initial guess for the multigrid approximation Can only set modes in the case of the Poisson problem as the mathematical expressions for the modes of the Boltzmann problem are unknown

void **DoCycle**(int q, char TypeName[], ITERATE *T, MINIMUM *mr),

Places a string TypeName[] containing the macro name into the structure T

void **AllCycles**(int q, double wusum, ITERATE *T, MINIMUM *mr),

Sets up the initial residual and guess for the problem, and performs the cycles dictated by the macros until a given number of WU cycles has been performed, or the convergence criteria has been reached

double **Cal_WU**(int q, ITERATE *T),

Calculates roughly the number of WU in each of the cycles, given the coarsest grid the cycle proceeds down to

The prototypes in the file Cycles h are given as follows

double **Norm**(int q, ITERATE *T, int NOrM),

Normalizes the approximation, to that of the LUD solution

double **Emax**(int q, ITERATE *T, double emax_0),

Calculates the maximum error element of the error vector

double **Enorm**(int q, ITERATE *T, double enorm_0),

Used to find the norm of the error norm between the normalized approximation and the LUD solution

double **Residual**(int q, ITERATE *T, double res_0),

Used to find the residual norm of residual equation **r = Av-f**

double **Resmax**(int q, ITERATE *T, double resmax_0),

Calculates the maximum residual element of the residual vector

void **Iterate_RB_R**(int start,
int finish,
ITERATE *itrn,
int q),

Execute a red-black Gauss Seidel iteration, which can start or finish at the specified points on the grid

void **Iterate_RB_L**(int start,
int finish,
ITERATE *itrn,
int q),

Execute a red-black Gauss Seidel backward iteration, which can start or finish at the specified points on the grid

void **Iterate_GS_R**(int start,
int finish,
ITERATE *itrn,
int q),

Execute a basic Gauss Seidel forward iteration, which can start or finish at the specified points on the grid

void **Iterate_GS_L**(int start,
int finish,
ITERATE *itrn,
int q),

Execute a basic Gauss Seidel backward iteration, which can start or finish at the specified points on the grid

void **IterateSort**(int rbr,
int rbl,
int gr,
int gl,
ITERATE *itrn,
int adaptive,
int q),

This function is used to make iteration calls easier It controls weather or not the iteration is adaptive, and will call one of the above four functions

void **Fix_fr**(double *fr, int qq),

This function is used to set how the adaptive iteration will behave

void **Write_fpcoarse**(int q, ITERATE *T),

Writes the approximation and the LUD solution for the lower grids to the file fpcoarse dat

void **Calculate_R_P**(ITERATE *T),

*Calculates the two matrices **R** and **P** used for restriction and interpolation*

void **Calculate_Galerkin**(ITERATE *T);

*Calculates A on the lower grids, using the Galerkin method, multiplying the matrices **RAP***

void **Restrict_RAP**(int q, ITERATE *T),

*Executes the Galerkin method of restriction, using the matrix **R** for restriction*

void **Restrict**(int q, ITERATE *T),

Executes direct restriction, which can be weighted or linear, and of type AH or A2H

void **Interpolate_RAP**(int q, ITERATE *T),

*Executes Galerkin method of interpolation, using the matrix **P** for interpolation*

void **Interpolate**(int q, ITERATE *T),

Executes a direct interpolation

void **Vcycle**(int q, int i, ITERATE *T),

Executes a Vcycle, descending down to the specified level

void **FMcycle1A**(int q, int i, ITERATE *T),

Executes a FMcycle, of type 1A

void **FMcycle1B**(int q, int i, ITERATE *T),

Executes a FMcycle, of type 1B

void **FMcycle2A**(int q, int i, ITERATE *T),

Executes a FMcycle, of type 2A

void **FMcycle2B**(int q, int i, ITERATE *T),

Executes a FMcycle, of type 2B

void **CCC1cycle**(int q, int i, ITERATE *T),

Executes a CCC1cycle (defined in appendix C)

void **CVCcycle**(int q, int i, ITERATE *T),

Executes a CVCcycle, (defined in appendix C)

Appendix B

Boltzmann Equation

Following from equation -(1 3), an applied external electric field is then treated as a perturbation to the plasma system and the distribution function is then expanded using Legendre Polynomials

$$f(r,v,t) = \Sigma_{k=0} P_k(\cos\theta_l) \; f_k(v_a,r,t) \qquad \text{-(B1)}$$

Substituting this back into the Boltzmann equation results in an equation with an infinite number of terms Using the treatment by Cherrington[8], these can be reduced to an infinite number of equations, the general term is

$$\frac{\partial f_k}{\partial t} + v\left(\frac{k}{2k-1}\frac{\partial f_{k-1}}{\partial z} + \frac{k+1}{2k+3}\frac{\partial f_{k+1}}{\partial z}\right) - \frac{eE_z}{m}\left(\frac{k}{2k-1}v^{k-1}\frac{\partial (f_{k-1}/v_{k-1})}{\partial v}\right.$$

$$\left. + \frac{k+1}{2k+3}\frac{1}{v^{k+2}}\frac{\partial (v^{k+2} f_{k+1})}{\partial v}\right) = \frac{2k+1}{4\pi}\int P_k S \, d\Omega_l$$

-(B2)

The collisional terms on the RHS can also be expanded using Legendre Polynomials, and the first two S_0 and S_1 given by Cherrington[8] correspond to the first two equations from the infinite set S_0 and S_1 are expressed in terms of f_0 and f_1 The higher order terms in f, can be omitted, thus only f_0 and f_1 are required For a steady state case, the time and position gradients present in equation -(B2) disappear, and using this we can substitute for f_1, the anisotropic part, in terms of the first order part f_0 Rewriting the distribution in terms of energy, and Normalising by

$$\int \varepsilon^{1/2} f_0(\varepsilon) d\varepsilon = 1 \qquad \text{-(B3)}$$

we get

$$\frac{d}{d\varepsilon}\left[\frac{e^2E^2\varepsilon}{3N\sigma_m}\frac{df}{d\varepsilon}\right] + \frac{2md}{M}\frac{d}{d\varepsilon}\left[\varepsilon^2 N\sigma_m(f_o + kT\frac{df}{d\varepsilon})\right] = 0 \qquad -(B4)$$

where N is the density of the neutrals in question, and σ_m is the cross section for elastic collisions

Appendix C

Multigrid Parameters or Settings

The following gives a list of all the different possible settings, the macros in the program that correspond to these, and also the different cycles

1) Whether the function Restrict() was weighted This is the averaging of a given point to be restricted with its surrounding points, as possible from the C function Restrict() In this case it is either full weighted as given by equation -(2 40)

$$e^{2h}_i = 1/4(v^h_{2i-1} + 2v^h_{2i} + v^h_{2i+1}) \qquad 0 < i < N/2$$

or linear restriction involving no weighting or averaging of surrounding points

$$e^{2h}_i = v^h_{2i} \qquad 0 < i < N/2$$

2) Weather Restrict(AH) or (2AH) was used, the difference between these is apparent when calculating the residual r^{2h} from r^h There are two possible approaches
AH is used to refer to the following

$$\mathbf{r}^{2h} = \mathbf{Restrict}(\mathbf{r}^h = \mathbf{f}^h - \mathbf{A}^h\mathbf{v}^h) = \mathbf{I}_h^{2h}(\mathbf{f}^h - \mathbf{A}^h\mathbf{v}^h)$$

where $(\mathbf{f}^h - \mathbf{A}^h\mathbf{v}^h)$ is calculated first and then the result is restricted to the lower grid 2h

A2H is used to refer to when the restrict operator $\mathbf{I}_h^{2h}$ works inside the brackets as follows, and operates on the equation before $\mathbf{r}^h$ is calculated

$$\mathbf{r}^{2h} = \mathbf{Restrict}(\mathbf{r}^h = \mathbf{f}^h - \mathbf{A}^h\mathbf{v}^h) = \mathbf{I}_h^{2h}(\mathbf{f}^h - \mathbf{A}^h\mathbf{v}^h) = \mathbf{I}_h^{2h}\mathbf{f}^h - \mathbf{I}_h^{2h}\mathbf{A}^h\mathbf{v}^h$$
$$= \mathbf{f}^{2h} - \mathbf{A}^{2h}\mathbf{v}^{2h}$$

3) How low the cycle would progress to or what level the particular cycle would stop at before proceeding back up to finer grids This is referred to as the 'level'

4) The number of iterations ν performed on each grid

5) The definition for the problem matrix **A**, can be given by either the direct approach, or the Galerkin method The direct approach involves redefining the matrix **A** at each level for that given number of grid points, but with a different energy interval The Galerkin method calculates **A** on the lower grids by the expression **RAP** given in chapter 2 by the equations -(2 43,44,45) The program uses the macro RAP to decide which of these two methods it will use If the macro is true, then the Galerkin method is used, if false, the direct approach

6) The type of multigrid cycle used, e g Vcycle or FMcycle The program uses a choice of eight different cycles, and one of which is just monogrid Gauss-Seidel iterations, to which the others can be compared to The program tests the C macro for each cycle to see whether it is to proceed with that option of cycle The C functions for the different cycles are given in the appendix A There are eight different cycles, A-H The cycles and their corresponding C macros are given

A) The Vcycle, which is executed by the C function Vcycle(), has the macro VCYCLE This cycle is the most basic fundamental of all the multigrid cycles

B) A full mulitgrid cycle of type 1A, given by the C function FMcycle1A(), is tested by the macro FMCYCLE1A This is first of four different possible ways to define a full multigrid cycle As suggested initially by Brandt [3,4,5,6], this cycle could start by restricting down to the lowest grid, without iterating on the way down It uses the fact that a better guess can more easily found on the coarsest grid, and thus the multigrid frequency removal mechanism need only start on this grid It was defined that for type 1A, when the cycle reaches the finest grid for the first time, no more Vcycles are performed This is the cheapest of all four full multigrid types

C) A full multigrid cycle of type 1B, given by FMcycle1B(), and tested by the macro FMCYCLE1B This is the same as type 1A only that iterations are performed initially on descending down from the fine grid to the coarsest, at the start of the cycle The rest of the cycle remains the same

D) A full multigrid cycle of type 2A, given by FMcycle2A(), and tested by the macro FMCYCLE2A This is the same as type 1A only that when the cycle reaches the fine grid, an extra Vcycle() is executed

E) A full multigrid cycle of type 2B, given by FMcycle2B(), and tested by the macro FMCYCLE2B This is the same as type 1B only in that when the cycle reaches the fine

grid, an extra Vcycle() is executed (as like 2A) This is the most costly type, and as it turns out, any improved performance of this type will not warrant the extra computational cost

F) The Wcycle is given by Wcycle(), and tested by the macro WCYCLE

G) The cycle CCC1 consists of Vcycles, but after any given number of Vcycles, another Vcycle can be executed except the level to which the Vcycle progresses is different to the rest of the Vcycles This cycle is given by the C function CCC1cycle(), and is tested by the macro CCC1

H) The cycle CVC consists of two separate cycles, one after the other In the simplest case, both could be Vcycles, but the levels that the two Vcycles descend down to within the one CVC cycle differs This cycle is given by CVCcycle(), and tested by the macro CVC

Program Details

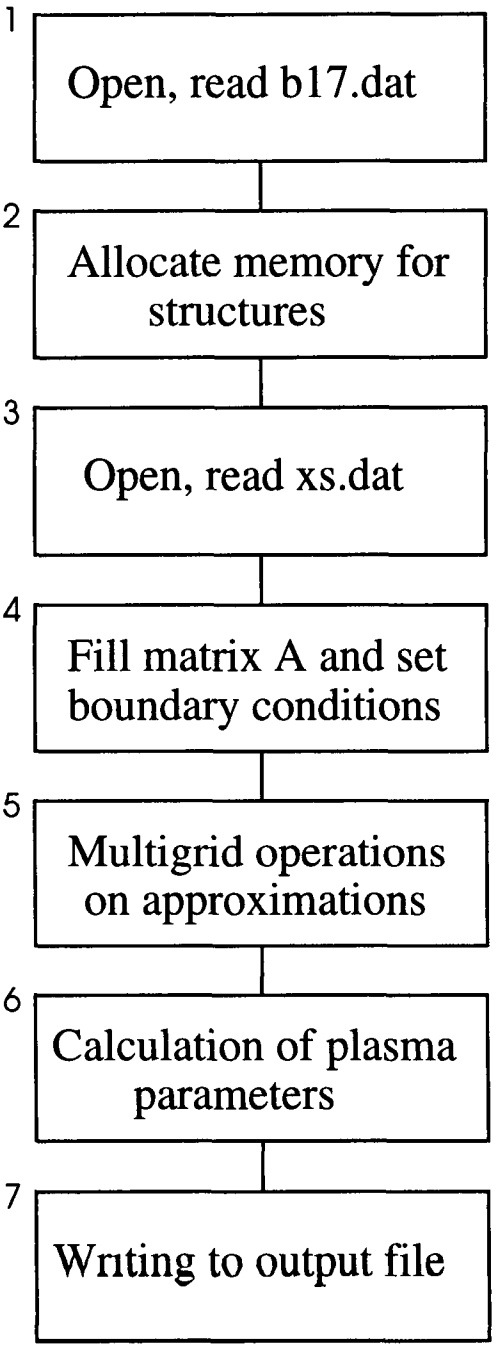
The program has 5 structures, xse defined as type ELASTIC, which provides storage for elastic cross section data, xsi defined as type INELASTIC, which holds inelastic cross section data The structure T holds the multigrid problem itself, $\mathbf{Af} = \mathbf{f}_{rhs}$, the solution $\mathbf{f}_{sol}$, the matrices **R** and **P** for Galerkin Restriction and Interpolation, and multigrid parameters, most of which are set by the Macros at the start of the program The structure mr defined as type MINIMUM, holds the parameters which the function Minres() tests to obtain the best results (lowest fractional residual) The structure F defined as type FILL contains the elements a and b that are produced by the function Evalab() or Evalab_fe These elements are then placed in the problem matrix **A** by Fillc() The parameter list of most functions in the program was reduced by placing as much of the data as possible into structures, which simplified the function calls dramatically Also, pointers to the structures were passed in the function calls which enabled any of the contents in the structure to be altered by the function in question

All functions beginning with a capital letter are either new or have been completely altered from the original code

All file pointers apart from infile, and outfile, begin with fp, and are identical in name to the output files only that the output files contain the extension dat A list of output files and their purpose is given in appendix A

Program Flow

Diagram -(C1) shows a simple flow, where the main program is split up into 7 blocks, each of which is numbered



-(C1)

The source code for the main program is given in Multigrid_Study c The source file Cycles c contains the algorithms for all the cycles The prototypes for these are both given in the appendix A The flow of the main program Multigrid_Study c is now discussed

Once the variable types and the five structures in main have been declared, main then opens the first data file (block 1), called b17 dat The function rdatfil() then reads some plasma parameters, the most important being the size q, giving 2^q-1 internal grid points

Memory is then allocated for the structures (block 2) xsi, xse, T, mr and F This is done by allocating a one dimensional array for each of the structures For the structures T, mr and F, an array of size one is allocated The reason an array of structures was allocated instead of just a structure is to enable the contents of all these structures to be manipulated more easily by any functions which use them Allocating an array of structures offers easier notation in the function call than the other option of using a pointer to each structure in the function call, although this is a matter of preference

All functions beginning with Alloc, are associated with memory allocation The functions Alloc_T(), Alloc_mr(), and Alloc_F actually allocate the memory for the contents inside each individual structure The contents of the xsi and xse structures is allocated when needed in the different functions as required later The function Set_T_F_mr() then fills or sets the arrays and values of the contents of the three structures

Input from the second file xs dat is then read by block 3 The function rxsfil() among other things reads the cross-sections from the data file xs dat into the two structures xse (elastic cross-sections) and xsi (inelastic) The function datsort() then alters the number and spacing of the cross-sections in energy such that there are cross-sections given for each energy point on the grid The function also performs manipulations on the densities with a view to obtaining the specific mass ratios for each of the different species and states present in the plasma, which are required for equation -(1 9)

Block 4 then proceeds to fill the contents of the problem matrix **A**, and also sets the boundary conditions, by assigning a value for the first element on the right hand side, as equation -(1 9) There is a simple 'if' switch, testing as to what type of lower grid matrix redefinition is used, e g the Galerkin method (using full the operator matrices **R**

and **P**) tested by the macro RAP. If RAP is not true (has a value of zero) then the second option of direct matrix redefinition for each level is then used. For every level q down to 1, the problem matrix **A** is defined. If the RAP is true, then **A** is found only for the fine grid, and the function Calculate_R_P() proceeds to find the matrices **P** and **R** for each level, and when calculated calls the function Calculate_Galerkin() which then calculates **RAP** for each level. The matrices for **R** and **P** are found in accordance to Briggs [7]. **R** has $N/2$ -1 rows and N-1 columns, given N-1 internal grid points in the system. **P** has N-1 rows and $N/2$ -1 columns.

Block 4 tests for the option of either a Boltzmann problem or Poisson, and also gives the option (by testing the macro GAUSS_JORDON) to calculate the eigenvalues of the problem matrix **A** using the function Eigen(), which employs the C Numerical Recipes functions [10] gaussj(), elmhs() and hqr().

If the option of Boltzmann is chosen, the function Evalab() calculates the values for the coefficients a and b in equation -(1.9a). These are stored in the structure F (of type FILL). The function Fillc(), then uses these diagonal coefficients along with the cross-sections and other parameters to calculate and fill all the elements in the problem matrix **A**. As mentioned, there are two possible normalisations, one solving for $\mathbf{n}(\varepsilon)$ and the other $\mathbf{f}(\varepsilon)$. The function Evalab() and Fill(,,,0) correspond to the first, $\mathbf{n}(\varepsilon)$, where the last argument 0 that Fill() takes sets the function to the $\mathbf{n}(\varepsilon)$ option. If the $\mathbf{f}(\varepsilon)$ normalisation is required, the function Evalab() is replaced by Evalab_fe(), and the last argument of Fill() becomes 1 to give Fill(,,,1).

If the Poisson option is taken, the function Matrix() calculates and fills the problem matrix **A**.

At the end of block 4, the system is now defined, all input information has been assimilated, and the multigrid aspect takes over.

The function Solve() at the start of block 5 obtains the LUD solution to the defined problem, and stores this separately from the multigrid solution. A results file is then set by Set_fpcycle(), which will record the performance of each different case scenario. This is then opened and left open for the duration of each case in block 5. A series of nested for loops is used to sweep through all the options. In the outer loop Set_Wexecute() sets the number of times this 'for' loop is repeated, depending on the two macros LINEAR and WEIGHT. These give the option of weighted restriction or linear (or both). Similarly, Set_Aexecute() sets the number of times the next 'for' loop

is repeated, depending on the two macros AH and A2H, which correspond to the type of restriction discussed earlier The function Set_Weights() then sets the multigrid parameters inside the structure T, to either linear, weighted or both Similarly Set_Restrict() sets the type of restriction in the multigrid structure T These functions were written to reduce the size of the function main() in Multigrid_Study c

They enable all the different cycles and multigrid parameters to be attempted on the same set problem, so that the file fpcycle dat can contain all results from each different case This file could be used to compare the results, and plot any case against another This was the approach taken to observe multigrid performance (e g by experimentation)

All functions beginning with Write, are new functions writing to files associated with the multigrid performances Those beginning with w, are original existing functions which have been manipulated, which output the plasma properties to the results file b17 res

Code Testing of the Fourier Function

The Fourier() function which obtains the sine fourier transform of a 1 x N-1 vector, using the Numerical Recipes [10] function sinefft(), has been tested by using the coincidence that the eigen functions or eigen modes of the Poisson problem are in fact sine waves between the two end boundary points Thus by setting up the Poisson problem, solving it, then adding a particular eigen mode, which becomes the error, it is possible to treat this as an initial guess A sine fourier transform of the error should reveal that frequency mode If all modes are present in equal amounts, then the fourier transform of the error is a horizontal line

Appendix D

Boltzmann Normalization

Equation -(1 9) gives the Boltzmann problem for the normalization n(ε), which describes the number of electrons in the plasma at energy ε, and is given by

$$\int n(\varepsilon)\, d\varepsilon = n_e \qquad -(D1)$$

where n_e is the number of electrons in the plasma However, given the relation between f(ε) and n(ε) as

$$n(\varepsilon) = f(\varepsilon)\, \varepsilon^{1/2}\, n_e \qquad -(D2)$$

then

$$\int \varepsilon^{1/2} f(\varepsilon)\, d\varepsilon = 1 \qquad -(D3)$$

Using the above equations we can now rewrite the Boltzmann problem given by equation -(1 9) as,

$$\frac{dn_k}{dt} = c_{k-1} f_{k-1} + g_{k+1} f_{k+1} - (c_k + g_k) f_k + \sum_{s,J} N_s \left[R_{sjk+m} \Delta\varepsilon^{1/2} k^{1/2} f_{k+m_{sj}} + R'_{sjk-m_{sj}} \Delta\varepsilon^{1/2} k^{1/2} f_{k\,m_{sj}} \right] N_s^J / N_s$$
$$+ R^1_{sk+m_{sk}} \Delta\varepsilon^{1/2} k^{1/2} f_{k+m_{sk}} + \delta_{1k} \sum_m R^1_{sm} \Delta\varepsilon^{1/2} k^{1/2} f_m - (R_{sjk} + R'_{sjk} + R^1_{sk}) \Delta\varepsilon^{1/2} k^{1/2} f_k \Big]$$

-(D4)

where a, and b from equation -(1 9) become c, and g respectively, using the above relations, and also equations -(1 9a,b,c) We can define c, and g (which correspond to a and b) as follows

$$c_k = \frac{2Ne^2}{3m}\left(\frac{E}{N}\right)^2 \frac{1}{v_k^+/N}\left(\frac{1}{\Delta\varepsilon}\right)\left(\varepsilon_k^+ + \frac{\Delta\varepsilon}{4}\right)\Delta\varepsilon^{1/2}k^{1/2} + \frac{\bar{v}_k}{2\Delta\varepsilon}\left(\frac{KT}{2} - \varepsilon_k^+ + \frac{2KT}{\Delta\varepsilon}\varepsilon_k^+\right)\Delta\varepsilon^{1/2}k^{1/2}$$

$$g_k = \frac{2Ne^2}{3m}\left(\frac{E}{N}\right)^2 \frac{1}{v_k^+/N}\left(\frac{1}{\Delta\varepsilon}\right)\left(\varepsilon_k^+ k - \frac{\Delta\varepsilon}{4}(k-1)^{1/2}\right)\Delta\varepsilon^{1/2} + \frac{\bar{v}_k}{2\Delta\varepsilon}\left(\varepsilon_k^+ - \frac{KT}{2} + \frac{2KT}{\Delta\varepsilon}\varepsilon_k^+\right)\Delta\varepsilon^{1/2}k^{1/2}$$

-(D4a)

where $\varepsilon_k^+ = k\Delta\varepsilon$, $\varepsilon_k^- = \varepsilon_{k-1}^+$ -(D4b)

$$\frac{v_k^+}{N} = \left(\frac{2\varepsilon_k^+}{m}\right)^{1/2}\sum_s q_s\sigma_s(\varepsilon_k^+) \quad , \qquad \bar{v}_k^+ = 2mN\left(\frac{2\varepsilon_k^+}{m}\right)^{1/2}\sum_s \frac{q_s\sigma_s(\varepsilon_k^+)}{M_s} \quad ,$$

-(D4c)

Equation -(D4b and c) remain the same

[1] J H Bramble, *Multigrid Methods*, Pitman Research Notes in Mathematics Series 294, Longman Scientific Technical, New York, (1993)

[2] S Barnett, *Matrices, Methods and Applications*, Clarendon Press, Oxford, (1990)

[3] A Brandt, *Multigrid-level Adaptive Solutions to Boundary Value Problems*, Mathematics of Computation, **31**, 333-390, (1977)

[4] A Brandt, *Multilevel Adaptive Techniques (MLAT) for Partial Differential Equations, Ideas and Software*, Proc Symposium on Mathematical Software, J Rice (ed), Academic, New York, 277-318, (1977)

[5] A Brandt, *Guide to Multigrid Development*, Proceedings of Oberwolfach (1976), Lecture Notes in Mathematics 631, Springer, Berlin (1978)

[6] A Brandt, *Multilevel Adaptive Combinations in Fluid Dynamics*, AIAA, vol 18, 1165-1172, (1980)

[7] W Briggs, *A Multigrid Tutorial*, SIAM, Philadelphia, (1987)

[8] B E Cherrington, *Gaseous Electronics and Gas Lasers*, Pergamon Press, (1979)

[9] J L Davis, *Finite Difference Methods in Dynamics of Continuous Media*, Macmillan, (1988)

[10] B P Flannery, W H Press, S A Teukolsky, W T Wetterling, *Numerical Recipes in C*, Cambridge, (1988)

[11] W Hackbusch and U Trottenberg, *Multigrid Methods*, Springer-Verlag, Berlin, (1982)

[12] W Hackbusch, *Multigrid Methods and Applications*, Springer Series in Computational Mathematics 4, Springer Verlag, Berlin (1985)

[13] L A Hageman, D M Young, *Applied Iterative Methods*, Academic Press, (1986)

[14] P W Hemker, P Wesseling, *Multigrid Methods IV*, ISNM, vol 116, Proceedings of the 4th European Multigrid Conference, Amsterdam, Birkhauser-Verlag, (1993)

[15] T Holstein, *Energy Distribution of Electrons in High Frequency Gas Discharges*, Phys Rev 70, 367-384, (1946)

[16] J J Lowke, A V Phelps, and B W Irwin, *Predicted Electron Transport Coefficients and Operating Characteristics of CO_2-N_2He Laser Mixtures*, J Appl Phys 44, 4664-4671

[17] S McCormick, *Multigrid Methods*, vol 3, SIAM Frontier Series, SIAM, Philadelphia, (1987)

[18] S McCormick, *Multigrid Methods*, Frontiers in Applied Mathematics, SIAM, Philadelphia, (1987)

[19] S McCormick, *Multigrid Methods*, vol 110, Marcel Dekker, inc, (1987)

[20] S McCormick, *Multilevel Adaptive Methods for Partial Differential Equations*, SIAM Frontier in Applied Mathematics 6, Philadelphia, (1989)

[21] S D Rockwood, *Elastic and Inelastic Cross Sections for Electron-Hg Scattering from Hg Transport Data*, Phys Rev A **8**, 2348, (1973)

[22] K A Stroud, *Further Engineering Mathematics*, 2nd ed, Macmillon Education Ltd, (1990)

[23] P Wesseling, *An Introduction to Multigrid Methods*, Wiley Internation Science, (1992)