COMPUTATIONAL METHODS

FOR THE

NEUTRON DIFFUSION EQUATION.

By

Peter Lindsay

for the Degree of M.Sc. by research.

September 1995.

Declaration:

I, Peter Lindsay, hereby certify that this thesis, which is approximately 7500 words in length, has been written by me, that it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree.

date: 7.9.95 signature of candidate:

CONTENTS

ABSTRACT

In this thesis,

we consider the numerical solution of the neutron diffusion equation in a square geometry. The emphasis of the thesis is on the estimation of the steady state solutions. An investigation of the nature of the error in a variety of situations is undertaken and the results interpreted in terms of simple physical models of a reactor cross-section. The efficiency of the Multigrid method is demonstrated and the effect of each stage of the process is illustrated graphically. The Multigrid method is then applied to a fuel rod / control rod assembly and the results are discussed.

Chapter 1

The Reactor model and its reduction to the simple first model:

Introduction

Marconi Simulation were interested in developing an analytical method of predicting the transient behaviour of the neutron diffusion equation

$$\nabla^2\phi + \alpha\phi + S = \frac{\partial\phi}{\partial t}$$

with which to compare their own numerical methods. A full discussion of the equation can be found in ref. 1 (Hitchcock, 1953).

Ideally the α term should be spatially dependent, but problems arise if a Fourier Series approach is attempted because the α term would give rise to non-linearity. Initially it was thought that a possible fruitful approach would be to model α in terms of eigenfunctions to exploit orthogonality properties but this generates non-linearity in the equations and therefore precludes the use of the superposition principle in Fourier modelling.

This initial avenue of investigation turned out to be unrealistic, and instead a variety of standard methods of solving the steady state and transient versions of the neutron diffusion equation are studied and are shown to agree closely with one another.

A particular area of investigation is to simulate the effects of control/fuel rods and a variety of different situations are investigated.

The physical model invoked increases in sophistication as the thesis progresses; starting with simple boundary condition manipulation, moving on to correct use of the absorption coefficient and the source term, and ending, in chapter 5, with a simple but plausible model of control rod and fuel rod effects. It should be made clear that this is an extremely simple model compared to that developed by Marconi Simulation, however it does provide a useful comparitor with which to gauge a 'cut-down' version of the Marconi model. It should be mentioned here that access to the Marconi model became problematic due to pressure of work at Marconi, and direct comparison using the Marconi model became impossible.

Particular attention is given to studies of an implementation of the Multi-Grid method of computation for the solution of the steady-state problem; which appears to demonstrate convergence rates approximately fifty-times better than standard Gauss-Seidel methods.

Results are displayed in graphical format using oblique views to give a '3-D' representation. All computations are performed on a 66Mhz '486 PC and the various methods are implemented in Turbo Pascal for WINDOWS vs. 1.5 as specified in ref. 4. Code listings given in Appendix B.

A Simple Model

Reactor dynamics is a very much more complex subject that the details of this thesis would indicate. The equations which actually describe the dynamics are described in ref. 1 (Hitchcock, 1953) and may be summarised as follows

$$l\frac{\partial P}{\partial t} = M^2\nabla^2 P + (k_\infty - 1)P + \sum l_i C_i / \sum_a$$

and

$$\frac{\partial C_i}{\partial t} = \mu_i \sum_a P - \lambda_i C_i$$

and

$$k_\infty = k_{\infty 0} + a_u(T_u - T_{u0}) + a_m(T_m - T_{m0})$$

where the following definitions apply:

l = prompt neutron lifetime

P = Reactor Power density

M = migration length

k_∞ = multiplication factor

$k_{\infty 0}$ = steady state multiplication factor

$\sum_a$ = macroscopic absoption cross - section

λ_i = decay constant

C_i = delayed neutron precursor concentration

μ_i = partial delayed neutron fraction

a_u = uranium coefficient of reactivity

a_m = moderator coefficient of reactivity

T_u = temperature of uranium

T_{u0} = steady state temperature of uranium

T_m = temperature of moderator

T_{m0} = steady state temperature of moderator

There are also equations governing the build up of Xenon and Iodine which each have an effect on the neutron flux distribution stability.

This thesis deals with a much simplified version of the above, where we consider a reactor of square cross-section governed by the equation:

$$\nabla^2\phi + \alpha\phi + S = \frac{\partial\phi}{\partial t},$$

where we are solving the equation to determine the behaviour of the flux, ϕ, directly analogous to the Power distribution across the section.
The term ϕ represents the number of neutron per square meter per second and should be thought of as ' neutron flux density '. The α term, when positive, can be thought of as a neutron flux enhancer; if negative it can be thought of as an absorber. The product of α and ϕ gives the neutron flux due to α enhancement. The term S allows for an independent source of neutrons. The Laplacian term is the diffusion of neutrons due to the rate of change of gradient of flux with respect to space. A derivation of the equation is included as appendix A.

This thesis discusses the implementation of different models of neutron diffusion in a reactor, and of different numerical methods of solving the models. The first stage is to consider a very simple situation:

As a preliminary investigation consider the determination of the steady neutron flux ϕ in a reactor of square cross--section in which there is neither flux 'enhancement' nor source processes. This will be dealt with later. The effect of a single isolated fuel rod is simulated by a very short 'hot-line' on one boundary wall. Assuming the introduction of suitable non-dimensional co-ordinates, the dimensionless neutron flux ϕ (x,y) satisfied Laplace's equation in a unit square. The neutron flux is zero everywhere along the sides of the square except for a small strip representing a uniform neutron source, (fuel rod). The geometric configuration is depicted in fig 1a. An important aspect is that it is assumed that the fuel rod is small in comparison to the area of the reactor.

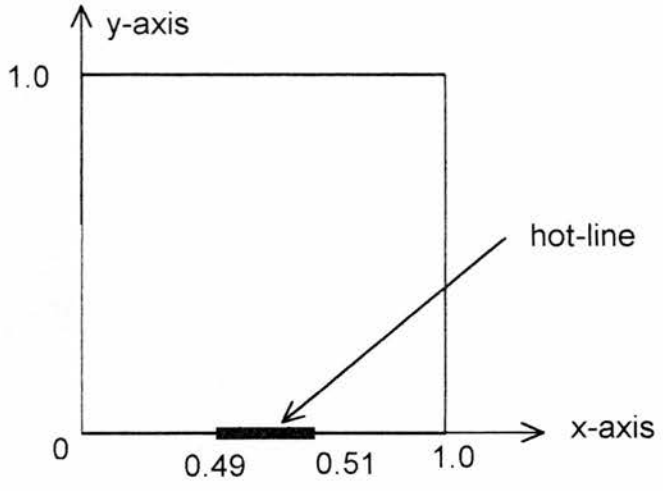


Figure 1a: A Simple Model

A mathematical description of the simple model is that we seek ϕ (x,y) such that

$\nabla^2\phi = 0 \qquad 0 < (x,y) < 1$

$\phi(x,0) = 0 \quad 0 \le x \le 0.49$

$\phi(x,0) = 1 \quad 0.49 \le x \le 0.51$

$\phi(x,0) = 0 \quad 0.51 < x \le 0.1$

$\varphi(0,y) = 0, \quad 0 \le y \le 1$

$\varphi(1,y) = 0, \quad 0 \le y \le 1$

$\varphi(x,1) = 0, \quad 0 \le x \le 1$

The value of such a simple problem is that it allows the nature of the errors associated with standard numerical techniques to be assessed. The standard 5 point finite difference approximation of Laplaces equation is

$$\phi_{i.j+1} + \phi_{i.j-1} + \phi_{i+1.j} + \phi_{i-1.j} - 4\phi_{i.j} = 0$$

where $\phi_{i.j}$ is an estimate of $\varphi(x_i, y_j)$ and (x_i, y_j) is a typical point on a finite difference grid.

In the discussion that follows the unit square in figure 1b is represented by a grid of points (x_i, y_j) where

x_i=ih, i=1,...,99

y_j=jh, j=1,...,99

and h=1/100. The values of φ are known along i=0, i=100, j=0 and j=100 (the sides of the square).

The short ' hot-line' extends from i=49 to i=51 on the boundary line j=0. The distance between the grid lines is 0.01 and the length of the hot-line is 0.02. The rest of the reactor surface is assumed to be 'cold'.

A numerical solution can be generated using the Gauss-Seidel technique ref. 3 (Smith, 1969) which is to iterate with the equation

$$\varphi_{i,j} = \tfrac{1}{4}(\varphi_{i+1,j} + \varphi_{i-1,j} + \varphi_{i,j+1} + \varphi_{i,j-1})$$

at each point of the grid repeatedly.

Mathematical Solution

To aid analysis of the results the boundary condition along j=0, (y=0), which is illustrated in figure 1b, can be expressed as a Fourier Series. That is, taking

$$\varphi(x,0) = \sum_{m=1}^{\infty} A_m \sin(m\pi x),$$

so that $\varphi(0,0) = \varphi(1,0) = 0$, it is merely a matter of integration to determine the values for the coefficients A_m. Omitting algebraic details, the Fourier (Sine) series for the graph in figure 1b is

$$\phi(x,0) = \sum_{m=1}^{\infty} \tfrac{2}{m\pi}(\cos(0.49m\pi) - \cos(0.51m\pi))\sin(m\pi x)$$

Note that the even terms are identically zero. This representation

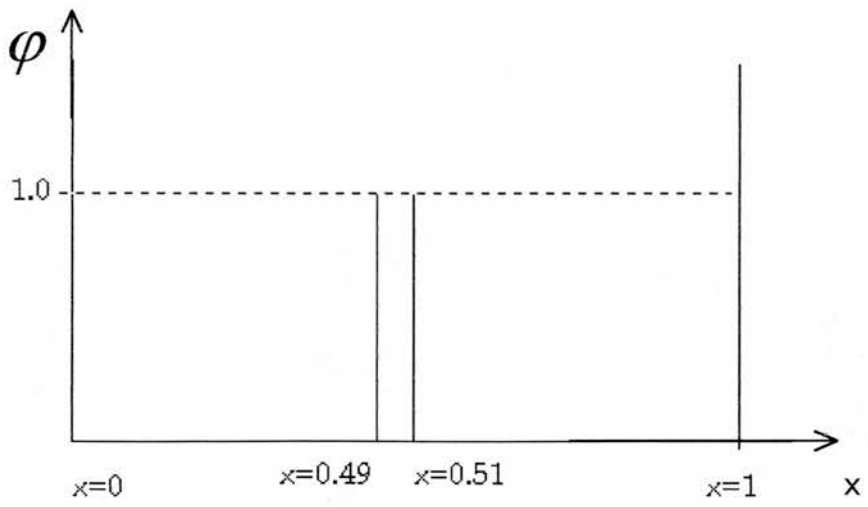


figure 1b

value of φ along y=0

is illustrated in figure 2a when the series is summed to 100001. The difference between figures 1b and 2a are shown in figure 2b. The discrepancy illustrated in figure 2b is due to the fact that a Fourier series converges to its average value at a point of discontinuity which occurs at x=0.49 and 0.51. The importance of this in relation to a numerical solution will be considered shortly.

Firstly, let us exploit the Fourier representation for the boundary condition. Observe that the function

$\sin(m\pi x)\sinh(m\pi(1-y)),$ m integer,

is a solution of Laplaces equation

$\nabla^2\varphi = 0$

such that the function is zero on x=0, x=1 and y=1.

Hence, the form

$$\varphi(x,y) = \sum_{m=1}^{\infty} B_m \sin(m\pi x)\sinh(m\pi(1-y))$$

satisfies the conditions specified in the simple model except possibly the condition on y=0. When y=0

$$\varphi(x,0) = \sum_{m=1}^{\infty} B_m \sin(m\pi x)\sinh(m\pi) \text{ and hence } B_m = \frac{A_m}{\sinh(m\pi)} = A_m \operatorname{cosech}(m\pi)$$

Thus by representing the boundary condition in a Fourier Series, we have derived a separation of variables solution to the problem, as described by ref. 5 (Carslaw & Jaeger, 1959) . The mathematical solution for $\varphi(x,y)$ is therefore

$$\phi(x,y) = \sum_{m=1}^{\infty} \frac{2}{m\pi}(\cos(0.49m\pi) - \cos(0.51m\pi))\sin(m\pi x)\sinh((1-y)m\pi)\operatorname{cosech}(m\pi)$$

Fig 2a: Flux on edge of plate

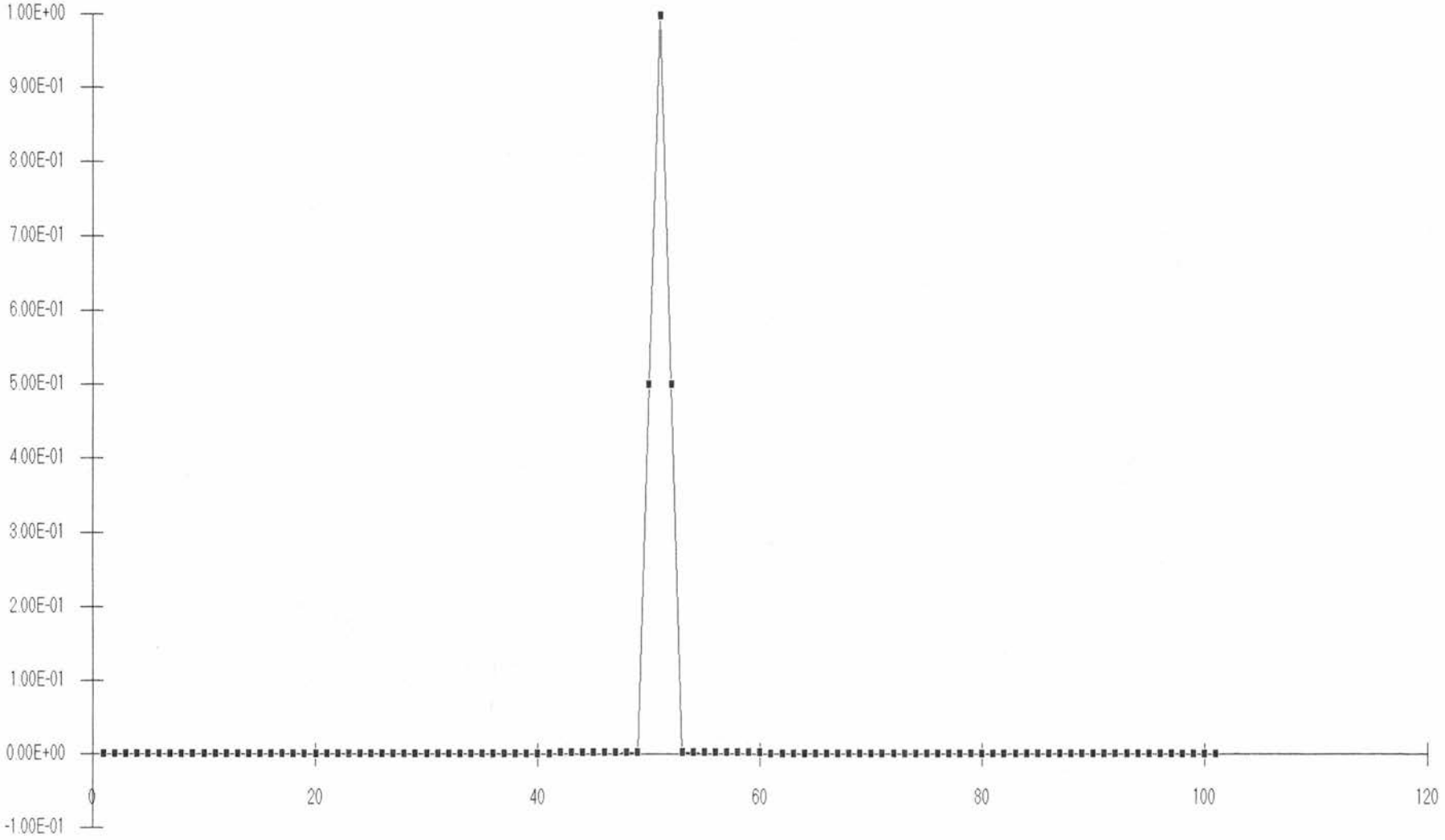
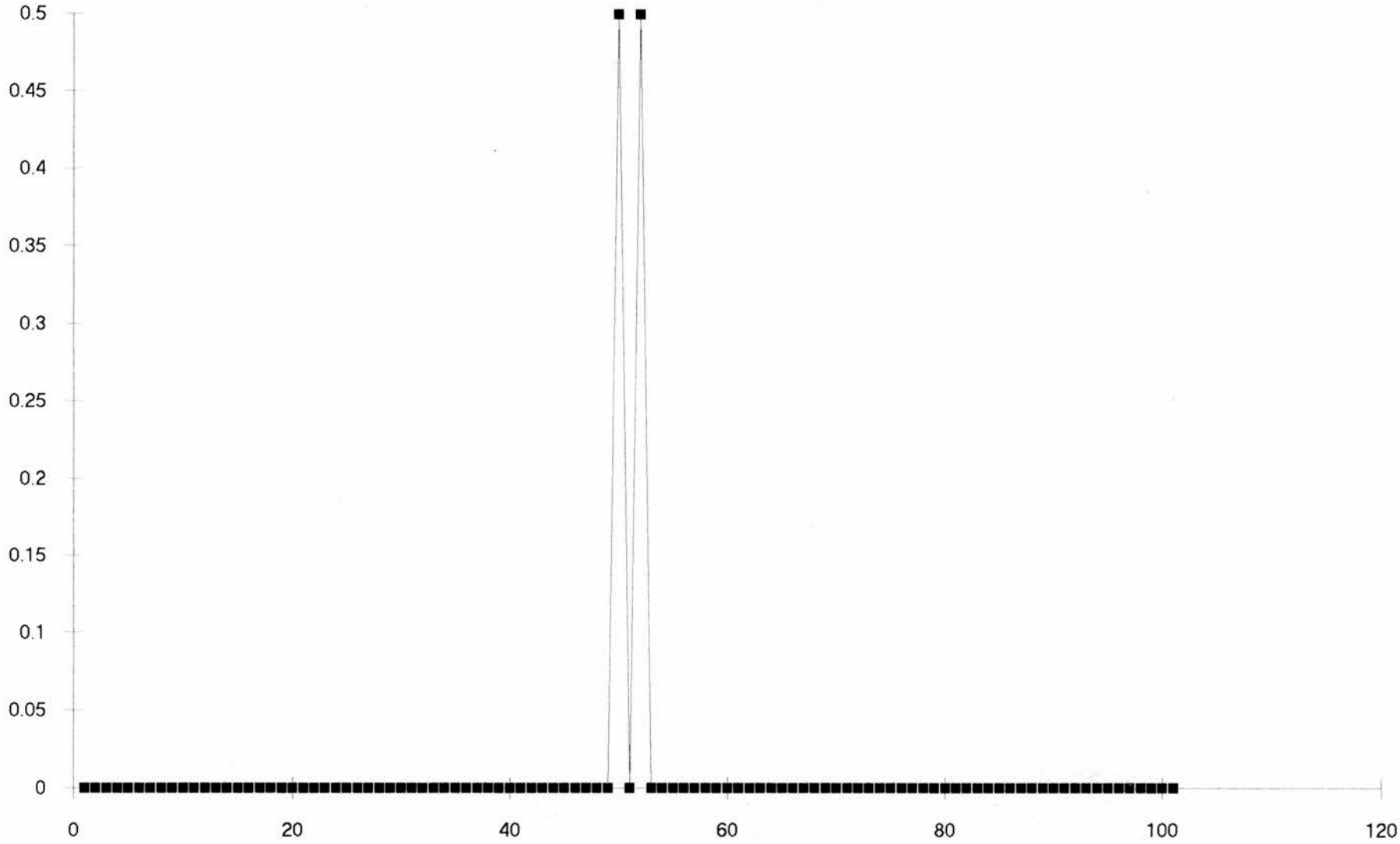


Fig 2b: difference between ideal profile and Fourier profile



This solution for $\phi(x,y)$ with m taken up to 100,001 is plotted on the graphs Fig3a and fig 3b.

The graphs are plotted using a MS EXCEL (ref. 6) spreadsheet application.

The format of the graphs is as follows:

The first graph is the whole 100 x 100 grid, while the second graph is a 'magnification' around the area of interest from grid 40-60 on the x-axis and 0 to 8 on the y-axis. The generating code for the solutions is given by code listing ancont.pas . All code listings are in Appendix B.

Numerical Results

The results of solving the model problem by Gauss-Seidel iteration with

$\varphi_{49,0}$ $\varphi_{50,0}$ and $\varphi_{51,0}$

set to unity are presented in figures 4a and 4b, where as before, figure 4b is a magnification of the area of interest. In order to be confident that convergence had occurred the Gauss-Seidel method used 75,000 iterations from an initial estimate of zero everywhere except at the 'hot-line', because the Fourier technique does not match the prescribed values at the end of the 'hot-line'.

To provide a fairer comparison between the Fourier solution and Gauss-Seidel solution, the G-S method was by modelling the half height values attained by the Fourier method on either side of the central node. The results of the Gauss-Seidel method thus modified are given in fig5a, and the difference between Gauss-Seidel and Fourier are given in the magnified view fig 5b. The flux profiles are seen to match within an absolute error of 0.008, or

Fig 3a: complete flux profile

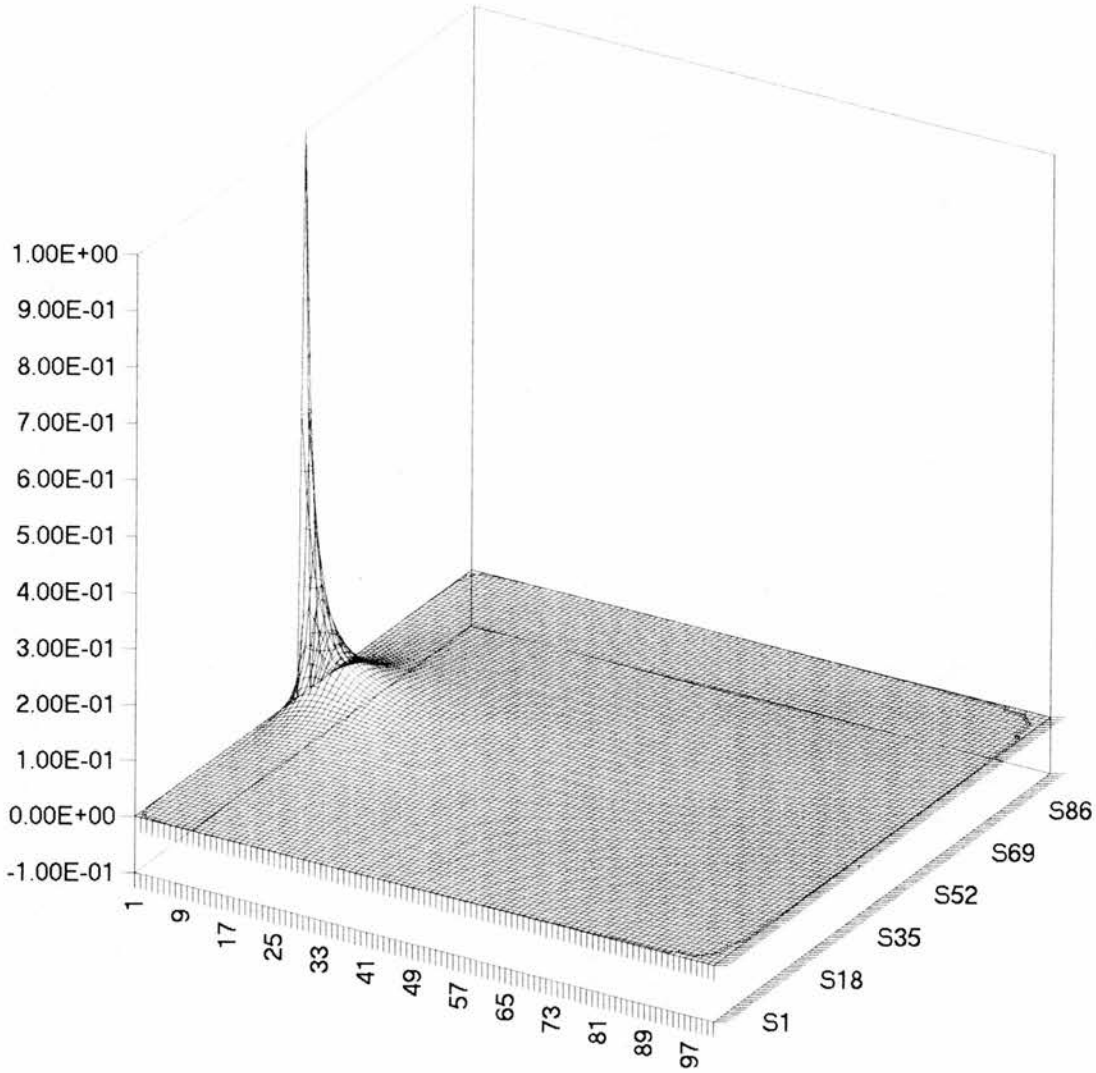


Fig.3b: magnification of fig 3

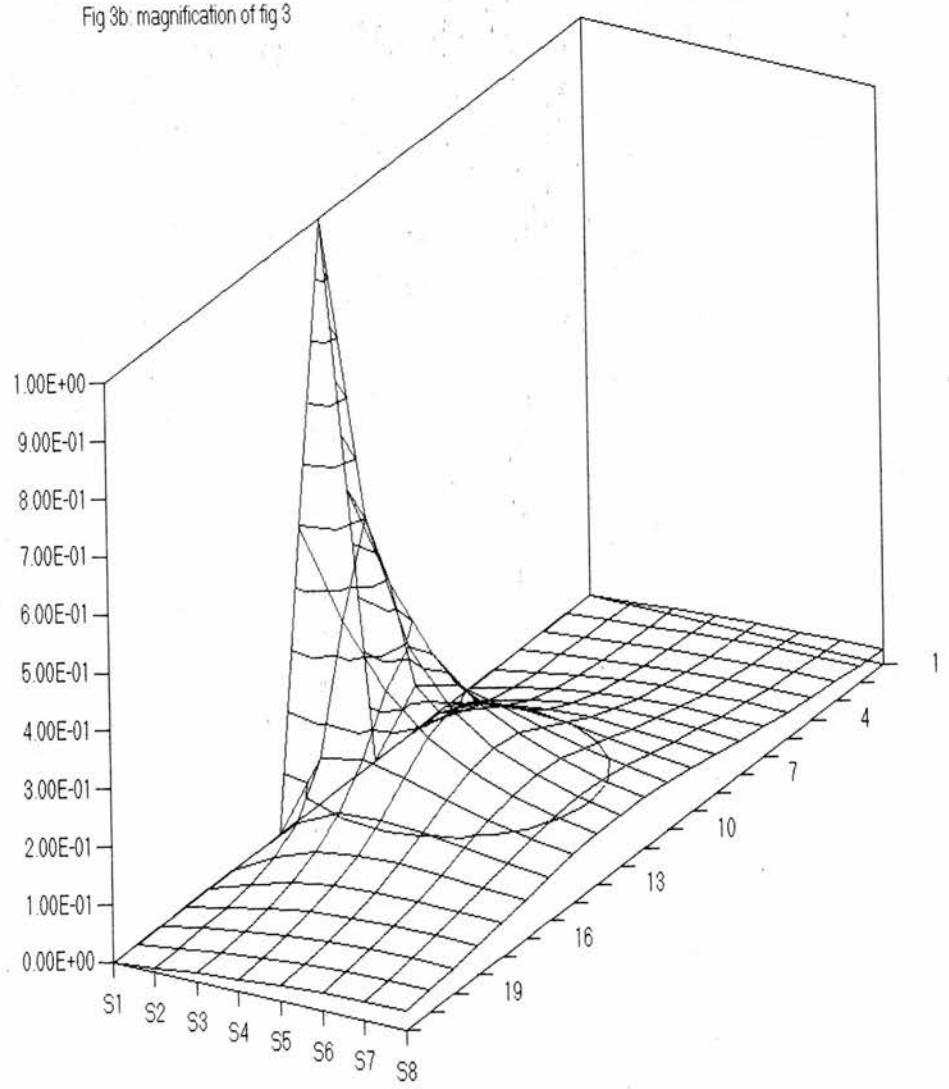


fig 4a Gauss-Seidel

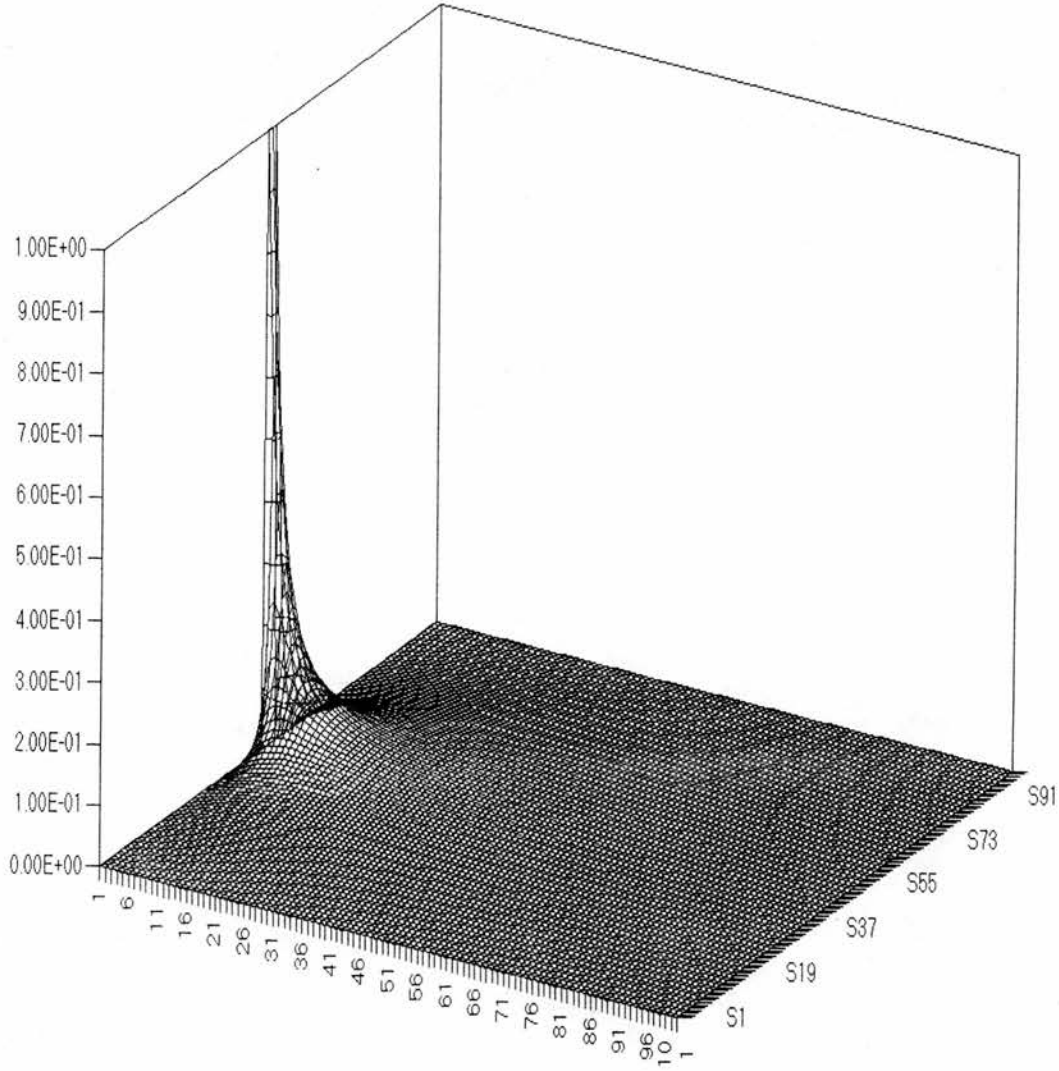


Fig 4b: magnification of fig 4a

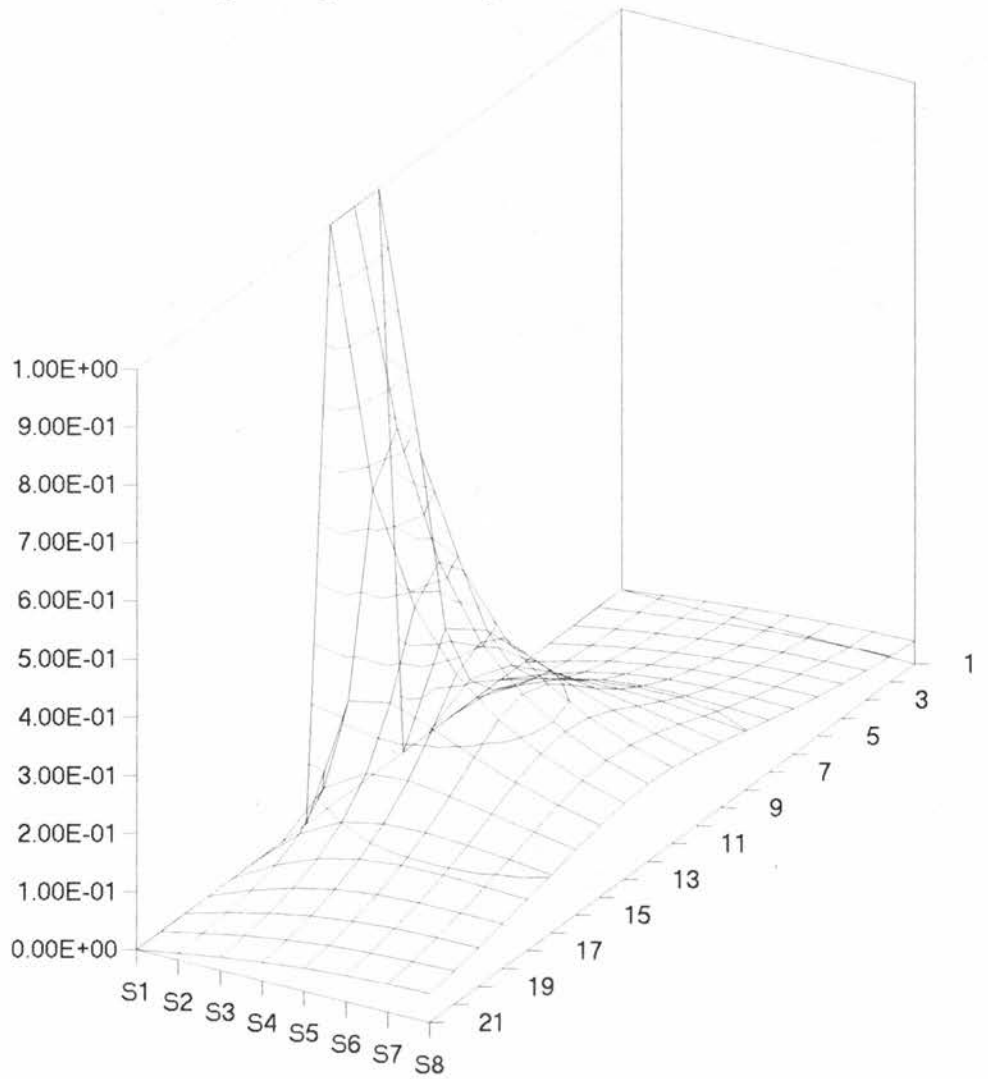


Fig 5a: Modified Gauss-Seidel

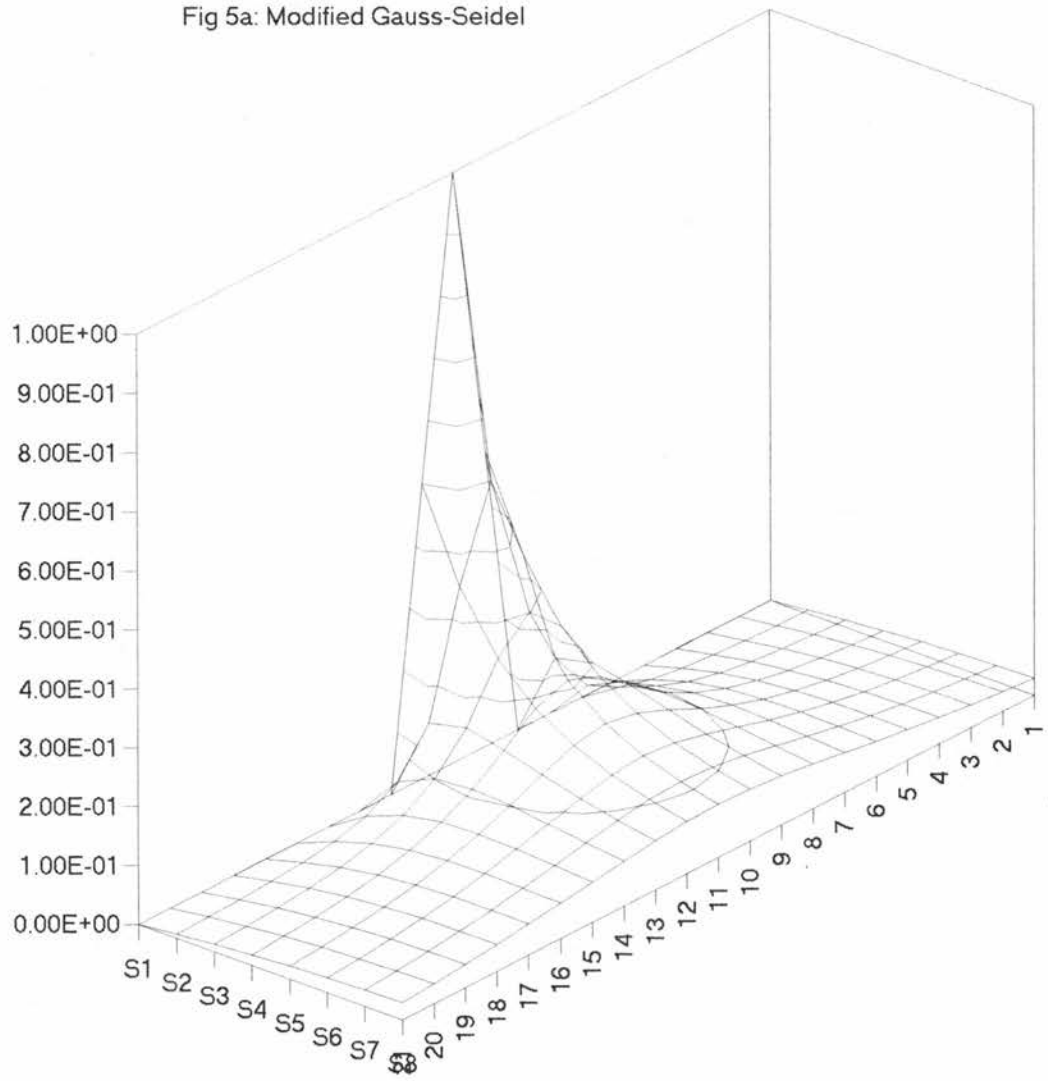
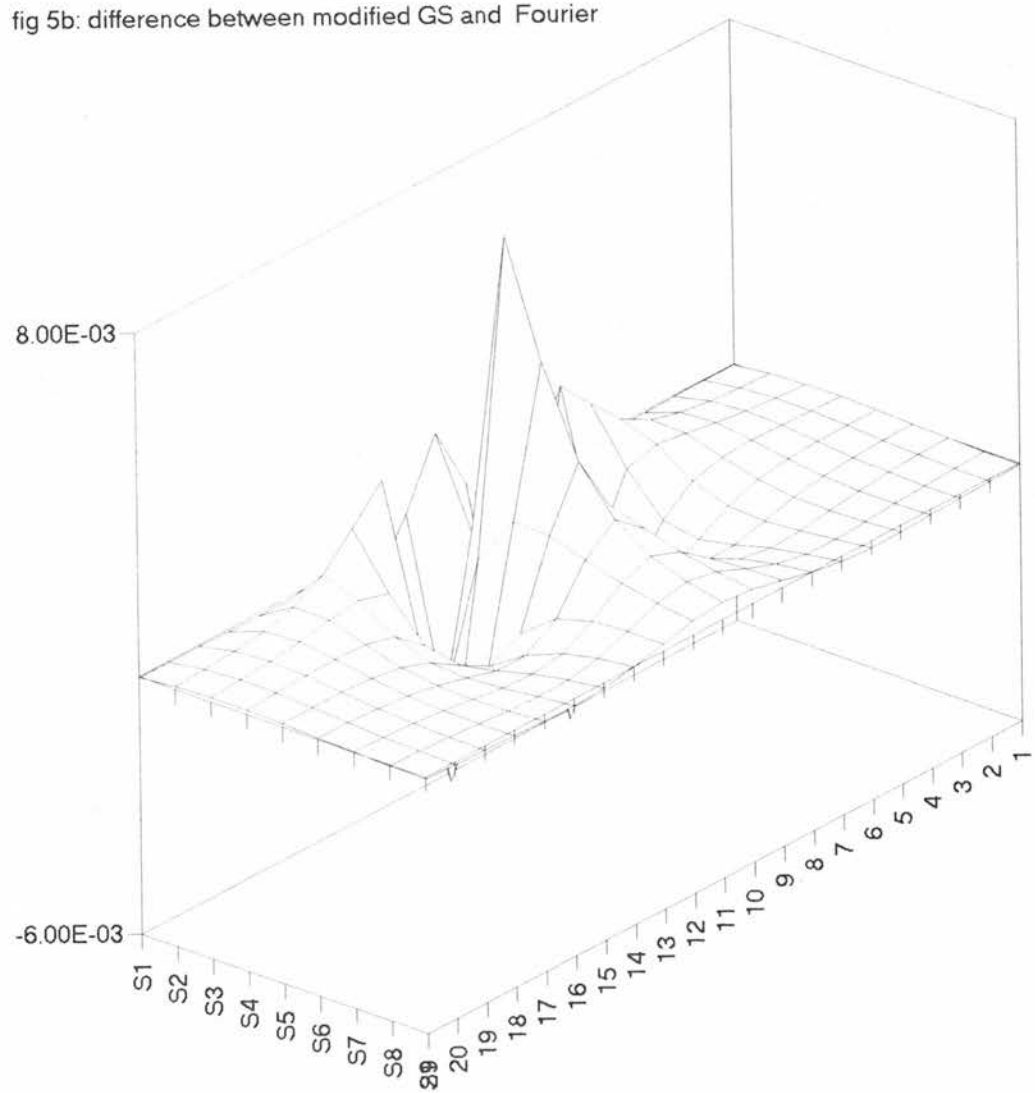


fig 5b: difference between modified GS and Fourier



about 1%. Given the narrowness of the boundary profile such an error seems reasonable.

An alternative method of comparison.

A striking feature of the solution illustrated in the previous figures is the localised nature of the solution - ϕ is effectively zero in much of the region. Thus an alternative approach is to assume the region is semi-infinite which allows the construction of a solution using conformal mappings. Using the techniques described in ref. 7 (Kreyszig, 1993) the solution for the flux in a semi-infinite region with φ=1 on y=0, $0.49 \le x \le 0.51$, is

$\varphi(x,y) = \frac{1}{\pi}(\arctan(\theta_1) - \arctan(\theta_2))$ where

θ_1 and θ_2 are derived from Fig 6a in which h is the grid spacing and 2h is the length of the 'hot line'.

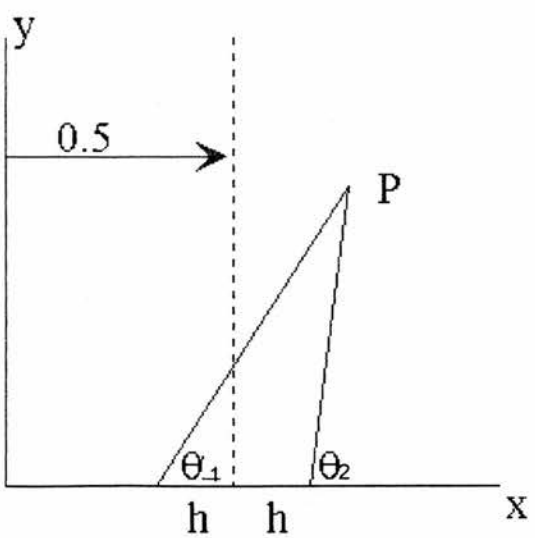


Fig 6a

Thus $\tan(\theta_1) = \frac{y}{x-(0.5+h)}$ and $\tan(\theta_2) = \frac{y}{x-(0.5-h)}$, where care should be taken to correctly account for angles in the second quadrant. The code for this method is given in confcont.pas. and the results are displayed in the fig6b.

Fig 6c shows the difference between the semi-infinite region solution and the modified Gauss-Seidel with agreement to within 1% .

Our simple model has revealed good agreement (1% difference) between the three methods of solution; Gauss-Seidel, Fourier and Mapping. This is reasonable agreement given the localised nature of the of the 'hot-line' and the problems associated with the discontinuity at x=0.49 and x=0.51.
The Fourier method averages the discontinuity and the Gauss-Seidel difference equations cannot correctly match the infinite gradients of the 'hot-line' because of the discrete nature of the finite-element grid. Nevertheless the agreement is good.

Fig 6b: conformal mapping result

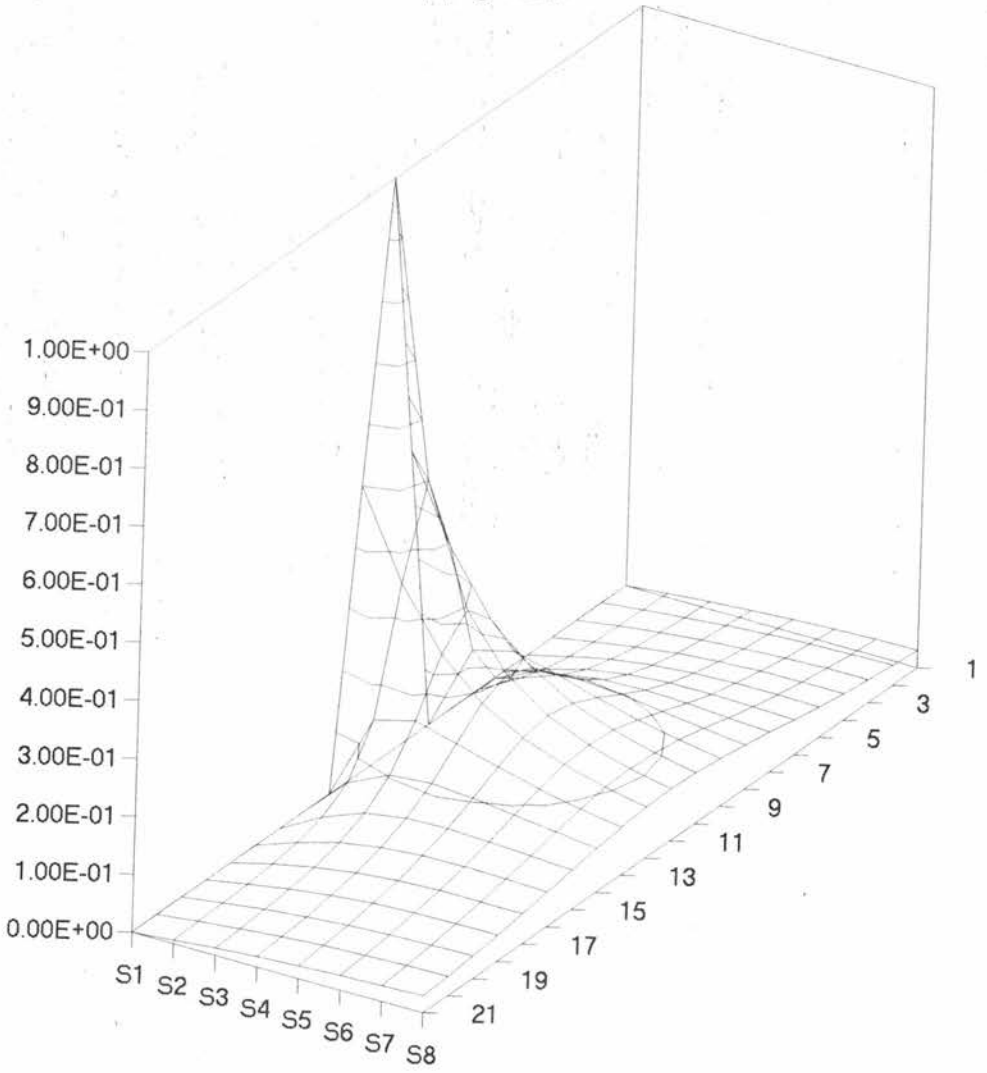
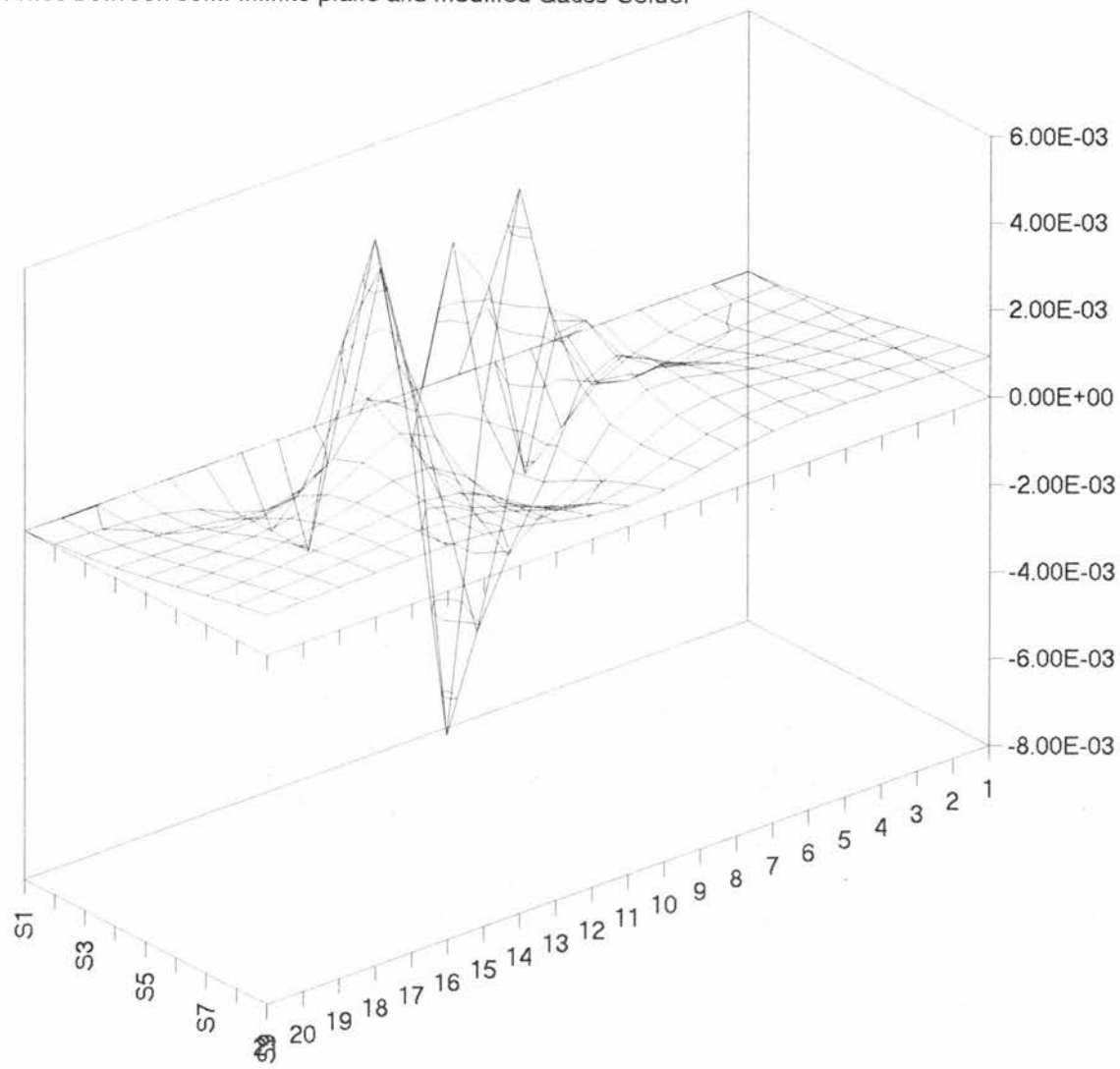


Fig 6c: Difference between semi-infinite plane and modified Gauss-Seidel



Chapter 2 A more realistic physical model.

Introduction

The next stage is to look at a more physically realistic model of the reactor cross-section. We create a source term S within the neutron diffusion equation at the centre of the square cross-section of the reactor. The purpose of the term S is to simulate the neutron flux generation from a fuel-rod. Again we are interested in the steady-state solution.

We will use analytical and numerical methods to solve the equation, and we begin by considering an analytical solution. The steady-state neutron diffusion equation is $\nabla^2\varphi + \alpha\varphi + S = 0$, and is applied within a unit square with boundary condition $\varphi = 0$ around the edges. The value of α is chosen to be 2.0 to give a visible effect on the surface plots, and S is a source term which is zero everywhere except in the region $0.47 \le (x, y) \le 0.53$ where S takes the value 1. The range 0.47-0.53 was chosen because it retains the 'isolation' effect of the fuel rod but is big enough to reduce errors caused by modelling too geometrically narrow a source term using Fourier series.

The source term can be represented by a double Fourier series:

$$S(x,y) = \sum_{m}^{\infty}\sum_{n}^{\infty} A_{m,n}\left\{\sin(m\pi x)\sin(n\pi y)\right\}$$

where $A_{m,n} = \left(\frac{4}{mn\pi^2}\left(\cos(0.47m\pi) - \cos(0.53m\pi)\right)\left(\cos(0.47n\pi) - \cos(0.53n\pi)\right)\right)$

which matches the requirement described above.

Fig 7a gives the Fourier representation of the source term S. For the purposes of computation, the infinite summation was truncated so that the index rises to 501 for each of m and n.

Fig 7b is a magnification of Fig 7a, illustrating the profile of the source term S in more detail in the vicinity of the centre of the region. Given the above Fourier series for the source term, we can derive a mathematical expression for the neutron flux φ as follows.

First observe that $\sin(m\pi x)\sin(n\pi y)$ is an eigenfunction of the equation $\nabla^2\psi + \alpha\psi = 0$ and that $\psi = 0$ on the sides of a unit square.

Writing, $\varphi = \sum_{m}^{\infty}\sum_{n}^{\infty} B_{m,n}\sin(m\pi x)\sin(n\pi y)$

then $\nabla^2\varphi = -\pi^2\sum_{m}^{\infty}\sum_{n}^{\infty} B_{m,n}\sin(m\pi x)\sin(n\pi y)(m^2+n^2)$.

Substituting this back into the neutron diffusion equation and collecting like terms gives:

$$-\pi^2 B_{m,n}\sin(m\pi x)\sin(n\pi y)(m^2+n^2) + \alpha B_{m,n}\sin(m\pi x)\sin(n\pi y) + A_{m,n}\sin(m\pi x)\sin(n\pi y) = 0$$

so that $B_{m,n} = \dfrac{A_{m,n}}{-\alpha + (m^2+n^2)\pi^2}$

and $\varphi(x,y) = \sum_{m}^{\infty}\sum_{n}^{\infty} \dfrac{A_{m,n}}{-\alpha + (m^2+n^2)\pi^2}\sin(m\pi x)\sin(n\pi y)$ as a solution to the equation.

The above derivation assumes that α is not an eigenvalue of the system.

The flux profile computed from the above series is given in fig 8, and the generating code is given in fourier.pas.

Fig. 7a: Source term S calculated from Fourier series, m,n=501.

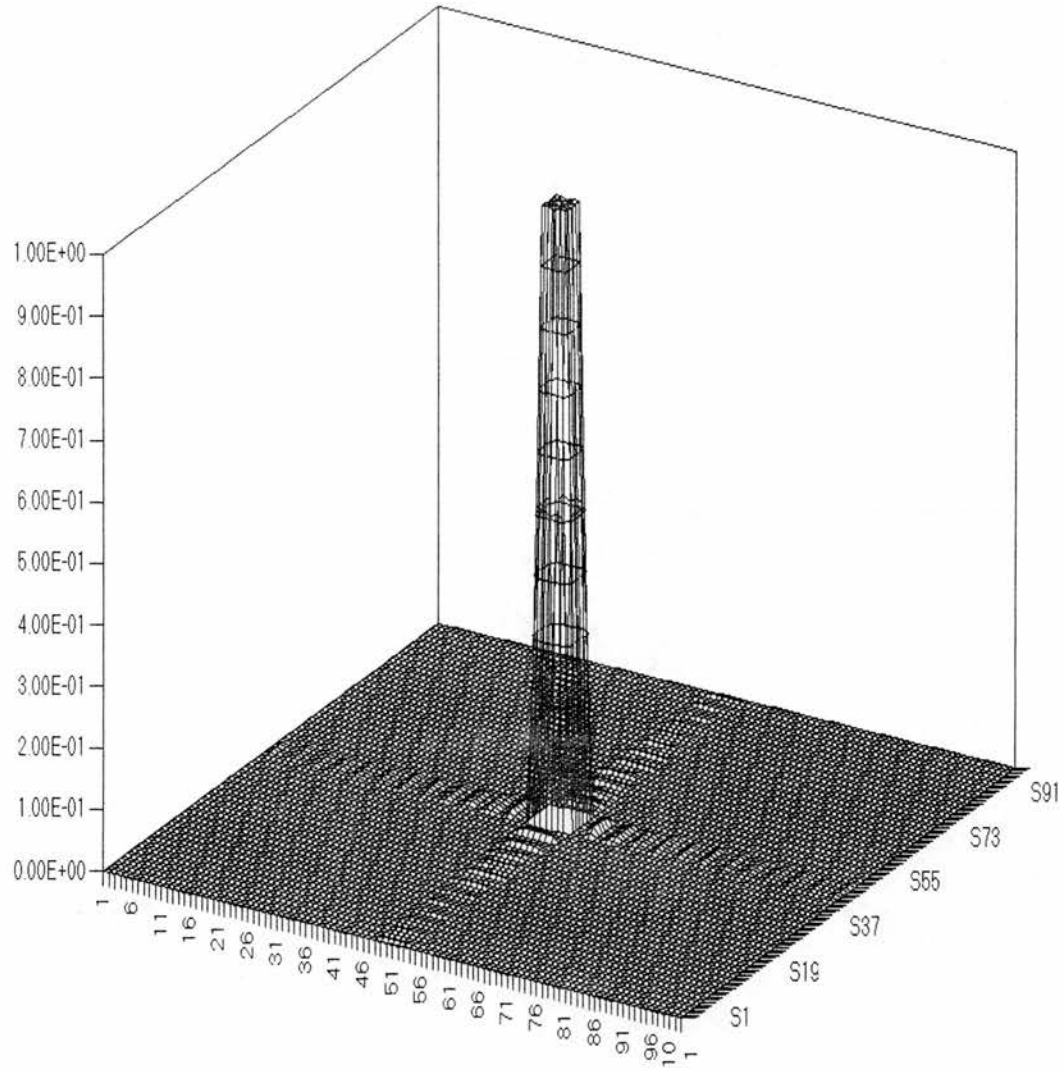


Fig. 7b: Magnified view of Fig. 7

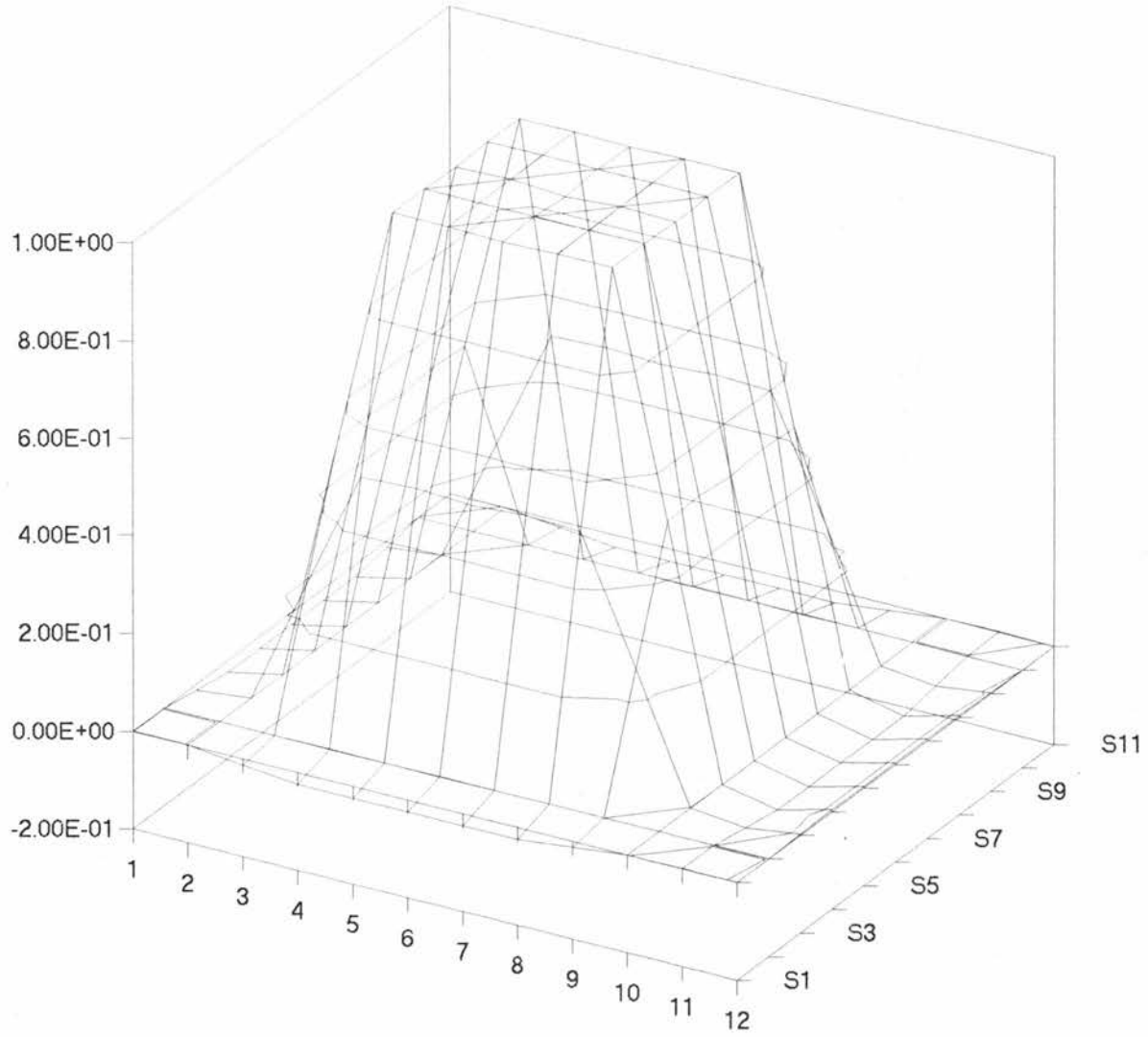
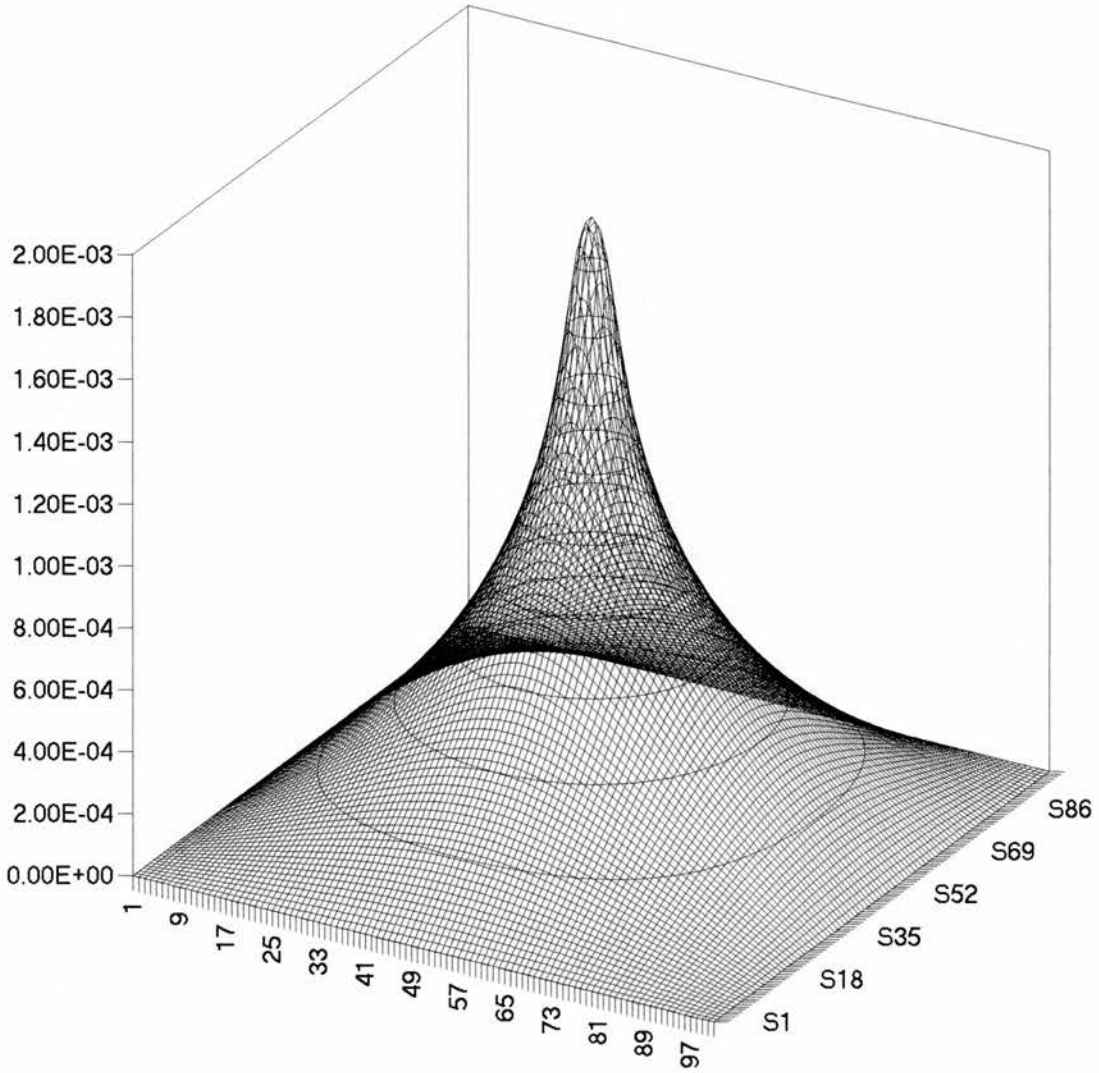


Fig. 8: Standard Fourier solution to flux profile



A Gauss-Seidel solution can be generated in the same way as before by iterating with the formula

$$\varphi_{i,j} = \frac{1}{-\alpha + \frac{4}{h^2}}\left(S_{i,j} + \frac{1}{h^2}(\varphi_{i+1,j} + \varphi_{i-1,j} + \varphi_{i,j+1} + \varphi_{i,j-1})\right),$$

from the initial state

$\varphi_{i,j} = 0.$

In order to most fairly compare the Fourier solution and the results of the Gauss-Seidel method, the Fourier source term is read as data into the Gauss-Seidel algorithm. The Gauss-Seidel method therefore uses exactly the same source term as the Fourier method so we can compare the methods without any obfuscation due to the Fourier method being unable to precisely model a narrow square-wave source term.

The code for the Gauss-Seidel method is given in gs.pas and the resultant flux profile is given in fig9. The Gauss-Seidel results are obtained after typically 50000 iterations.

Comparison of Results

The comparison of the Fourier method and Gauss-Seidel method are given in Fig10a where the Fourier results are subtracted from the Gauss-Seidel results. Fig10b shows a magnified view of the 'error' from grid points 41-59 on both axes. The error is also examined in fig10c, where it is shown more simply by plotting the difference between Gauss-Seidel and Fourier along the line x=0.50, y=0.01,..,1.00. The errors are small , at worst about 10^{-6} - which is less

Fig. 9: Standard Gauss-Seidel solution to flux profile

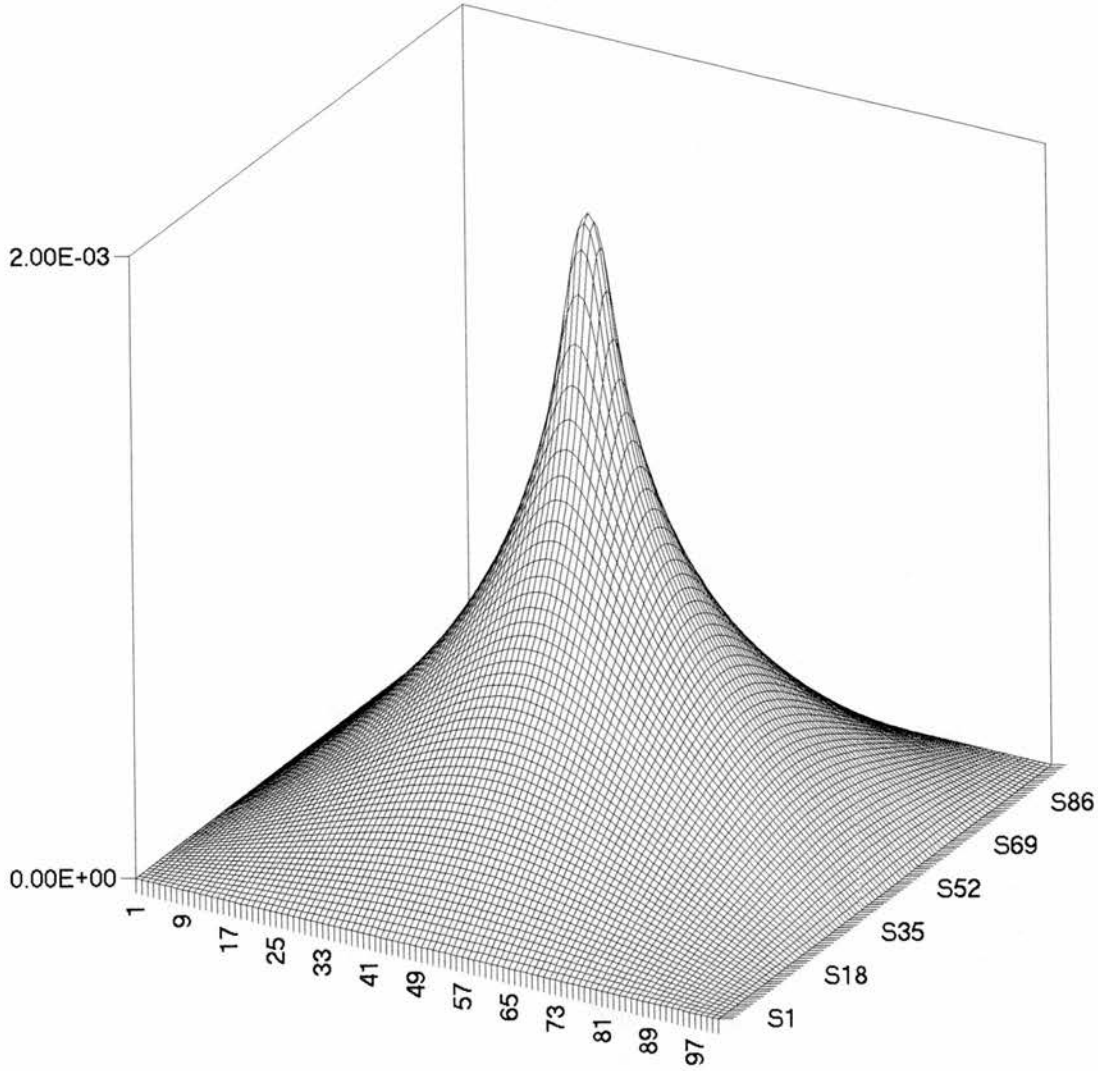


Fig. 10a: Gauss-Seidel minus Fourier solution.

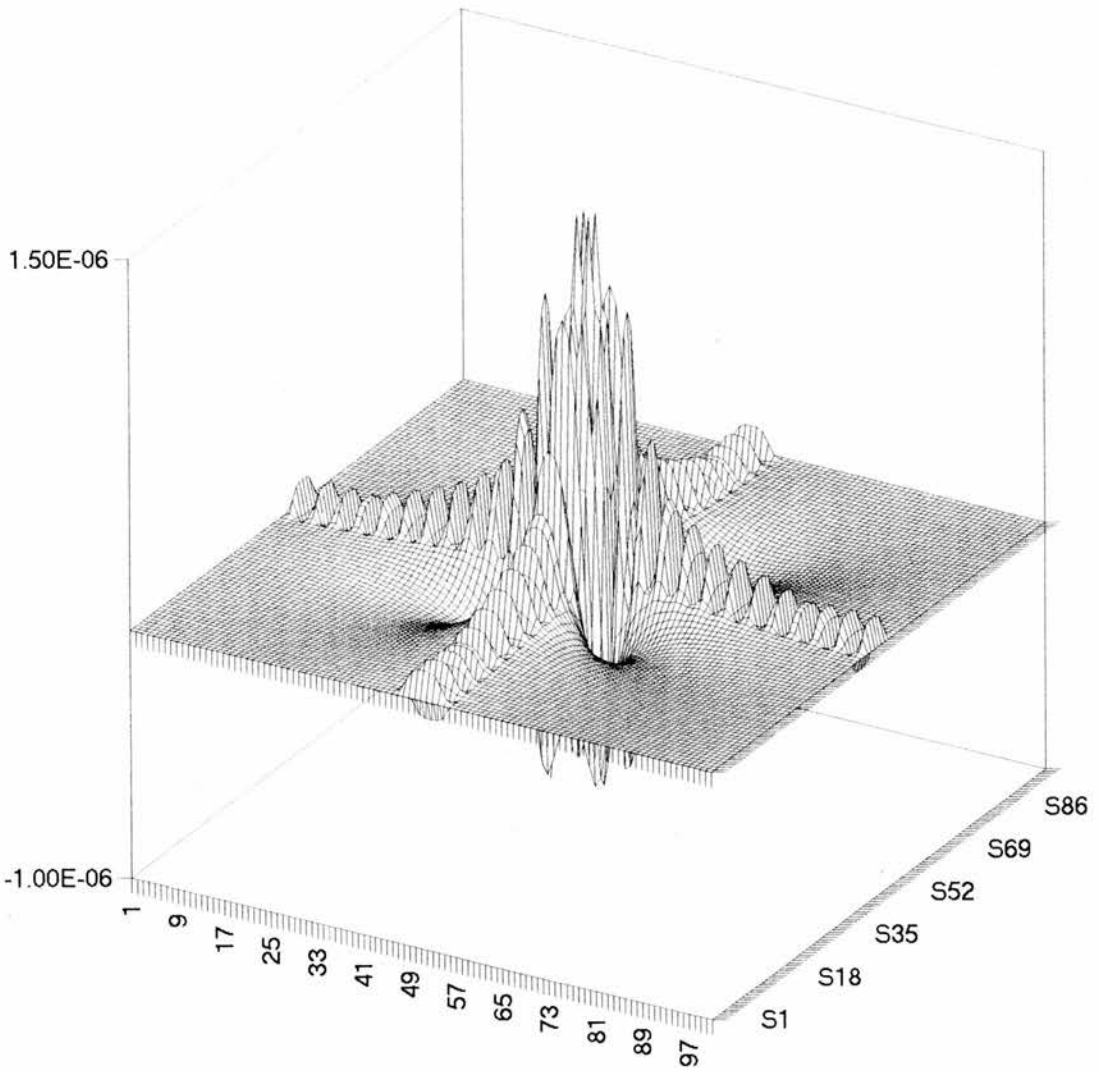


fig 10b: magnified view of fig 10

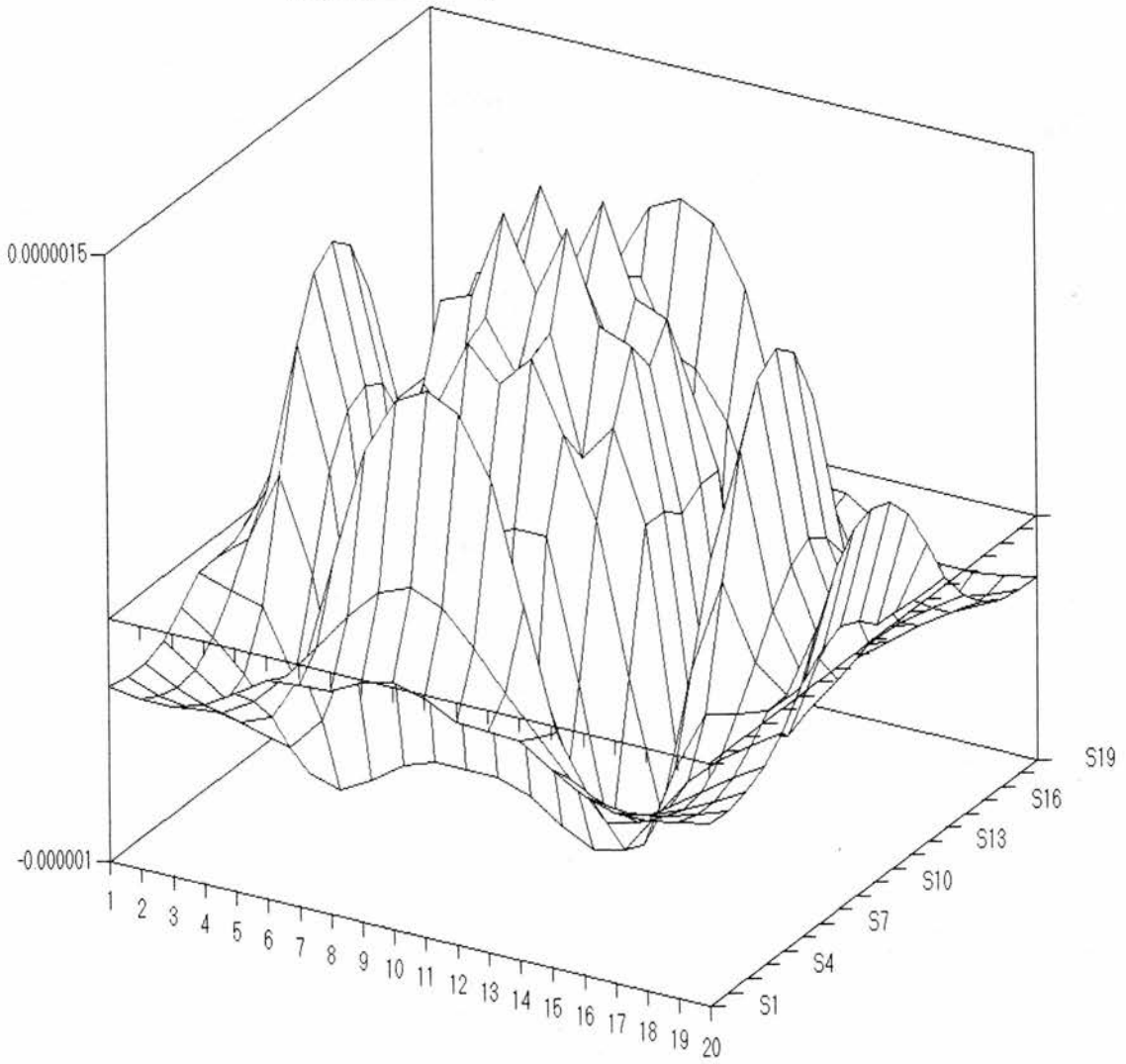


Fig. 10c: Gauss-Seidel minus Fourier solution at x=0.5.

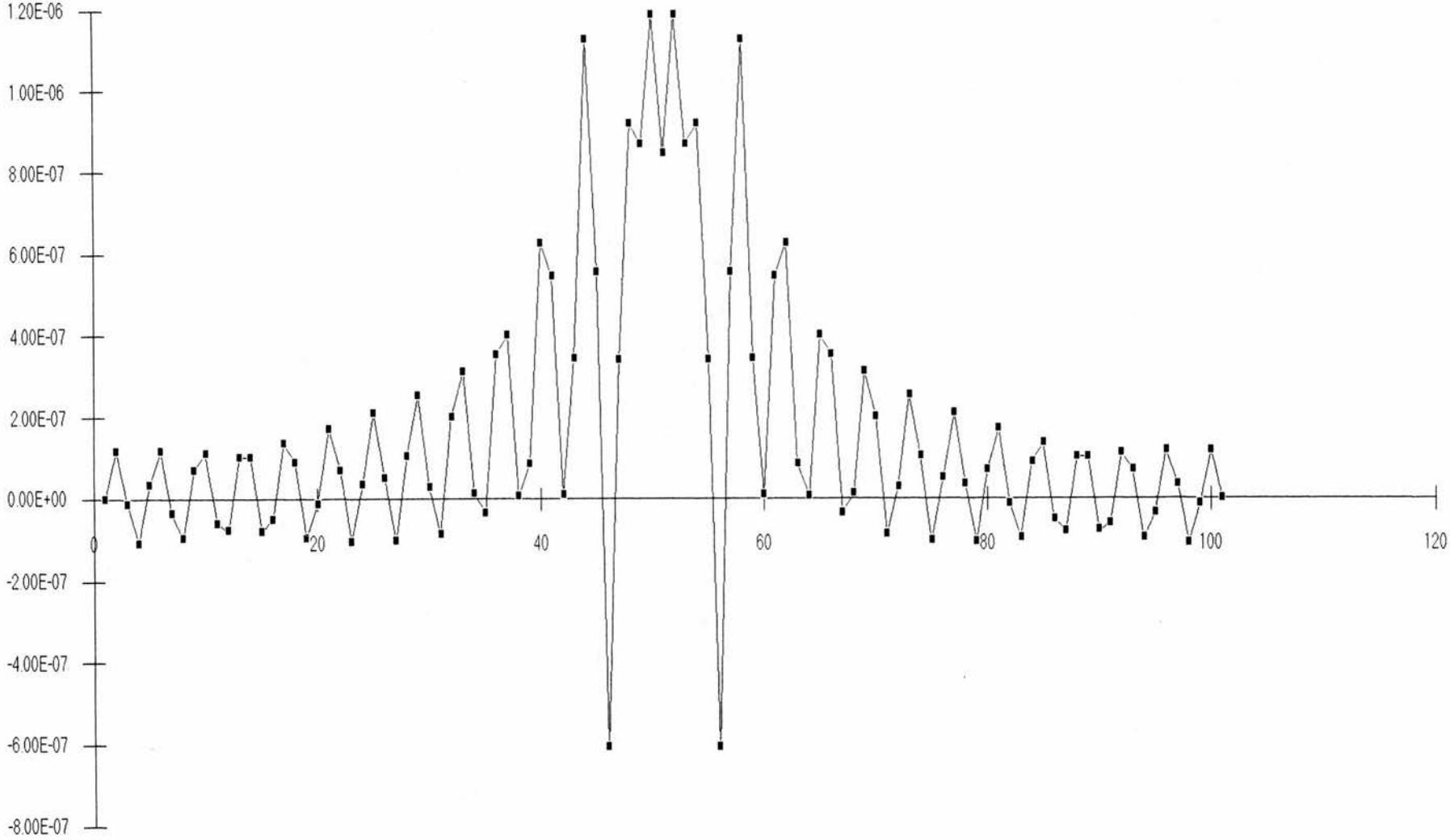
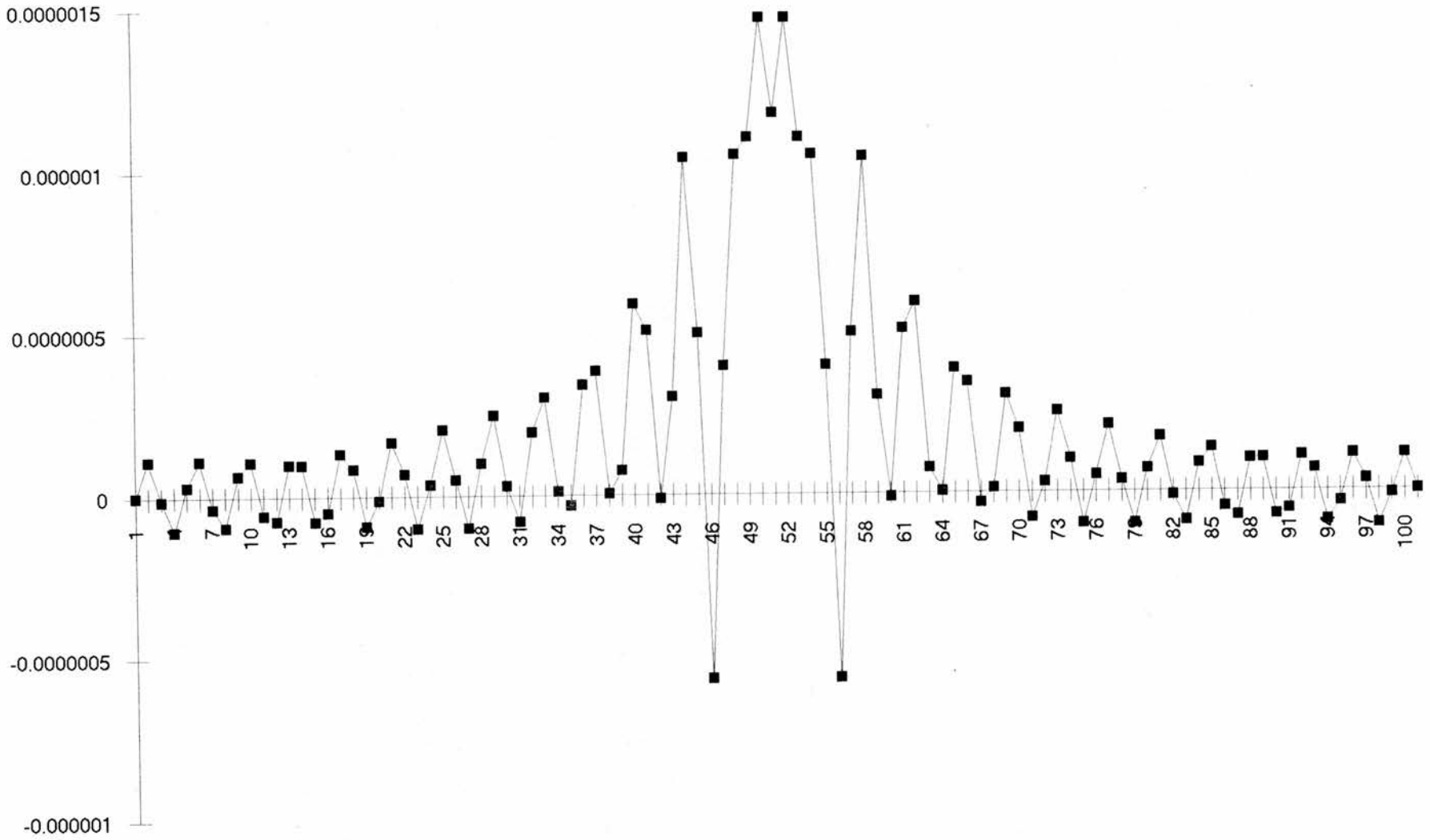


Fig. 10d: as fig 10c but with n,m=901 rather than n,m=501



than 0.1%, and the maximum errors occur in the region of neutron source - which is to be expected. However, one striking feature of fig. 10a is the pronounced oscillating pattern along the lines x=0.5 and y=0.5. The regularity and localised nature of this oscillating pattern seems unusual.

An attempt to reduce this error by increasing the number of terms in the Fourier series for S and φ from 250,000 per node (n=501 and m=501) to 800,000 (n=901 and m=901) per node resulted in the graph fig10d., which depicts the errors along x=0.5. (It should be noted that the MS EXCEL spreadsheet ranges from 1 to 101 rather than the Pascal code's nomenclature of 0..100). It is seen that the error is only slightly diminished by this three-fold increase in number of Fourier terms. Changing the specification of the source term does not significantly change the magnitude of the error.

Pascal's REAL declaration can cope with 11 digits per variable. When a number as small as 10^{-11} is written to EXCEL it will appear as 9.9999999999981E-12 demonstrating a very small error in translation between the two applications. The EXCEL spreadsheet can reliably graph differences of this order. Hence the errors shown in figs 10c&d may be due to the truncation error caused by representing an exact derivative by a finite-difference formula. Further evidence for this explanation of the error is given by the following analysis.

The Nature of the Numerical Solution

The source term S is a sum of sine terms:

$$S(x,y) = \sum_{m}^{\infty}\sum_{n}^{\infty} A_{m,n}\{\sin(m\pi x)\sin(n\pi y)\}$$

$$\text{where } A_{m,n} = \left(\frac{4}{mn\pi^2}(\cos(0.47m\pi) - \cos(0.53m\pi)(\cos(0.47n\pi) - \cos(0.53n\pi)\right)$$

and we are solving the equation $\nabla^2\varphi + \alpha\varphi + S = 0$.

The difference approximation of the above is:

$$\varphi_{i,j+1} + \varphi_{i,j-1} + \varphi_{i+1,j} + \varphi_{i-1,j} - 4\varphi_{i,j} + \alpha h^2\varphi_{i,j} + Sh^2 = 0.$$

We can try an exact solution to the difference equation by suggesting

$\varphi_{i,j} = C\sin(i\pi\,\mathrm{nh})\sin(\mathrm{j}\pi\,\mathrm{mh})$ where ih = x and jh = y.

Substituting this formulation into the difference equation yields

$$C(\sin(\mathrm{i}\pi\,\mathrm{nh})(2\sin(\pi\,\mathrm{jmh})\cos(\pi\,\mathrm{mh})) + \sin(\pi\,\mathrm{mjh})(2\sin(\pi\,\mathrm{inh})\cos(\pi\,\mathrm{nh}))$$

$$-4\sin(\pi\,\mathrm{nih})\sin(\pi\,\mathrm{mjh})) + \alpha\,\mathrm{h}^2 C\sin(i\pi\,\mathrm{nh})\sin(\mathrm{j}\pi\,\mathrm{mh}) + \mathrm{Sh}^2 = 0$$

Dividing throughout by $\sin(i\pi\,\mathrm{nh})\sin(\mathrm{j}\pi\,\mathrm{mh})$

gives

$$C(2\cos(\pi\,\mathrm{mh}) + 2\cos(\pi\,\mathrm{nh}) - 4 + \alpha\,\mathrm{h}^2) + Ah^2 = 0$$

and using the identity $\cos(\pi mh) = 1 - 2\sin^2(\frac{\pi mh}{2})$ we obtain

$$C(4\sin^2\left(\frac{\pi mh}{2}\right)+4\sin^2\left(\frac{\pi nh}{2}\right)-\alpha h^2) = Ah^2,$$

hence

$$C = \frac{A}{-\alpha + \frac{4}{h^2}\left(\sin^2\left(m\pi\,\frac{h}{2}\right)+\sin^2\left(n\pi\,\frac{h}{2}\right)\right)}$$

So an infinite series solution to the difference equations is:

$$\varphi_{i,j} = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty}\frac{A_{m,n}}{-\alpha + \frac{4}{h^2}\left(\sin^2\left(m\pi\,\frac{h}{2}\right)+\sin^2\left(n\pi\,\frac{h}{2}\right)\right)}\sin(m\pi x)\sin(n\pi y)$$

where $A_{m,n} = \left(\frac{4}{mn\pi^2}\left(\cos(0.47m\pi)-\cos(0.53m\pi)\right)\left(\cos(0.47n\pi)-\cos(0.53n\pi)\right)\right)$

Evaluating this solution (the code is given in gbfourie.pas) and then comparing the results with the Gauss-Seidel method generates the following graph fig 11a ; fig11b shows the comparison at x=0.5 and demonstrates no high frequency oscillation - unlike the examples in fig 10a and b. Fig 11c shows a comparison of the Fourier solution to the differential equation and the discrete Fourier method along the line at x=0.5.

The 'rippling' effect evident in Fig. 10a requires some explanation.

The number of terms in the Fourier series was taken to be 501. The error, or difference, between the Gauss-Seidel and Fourier methods is given by the equation below:

Fig 11a: comparison of modified fourier coefficients and Gauss-Seidel

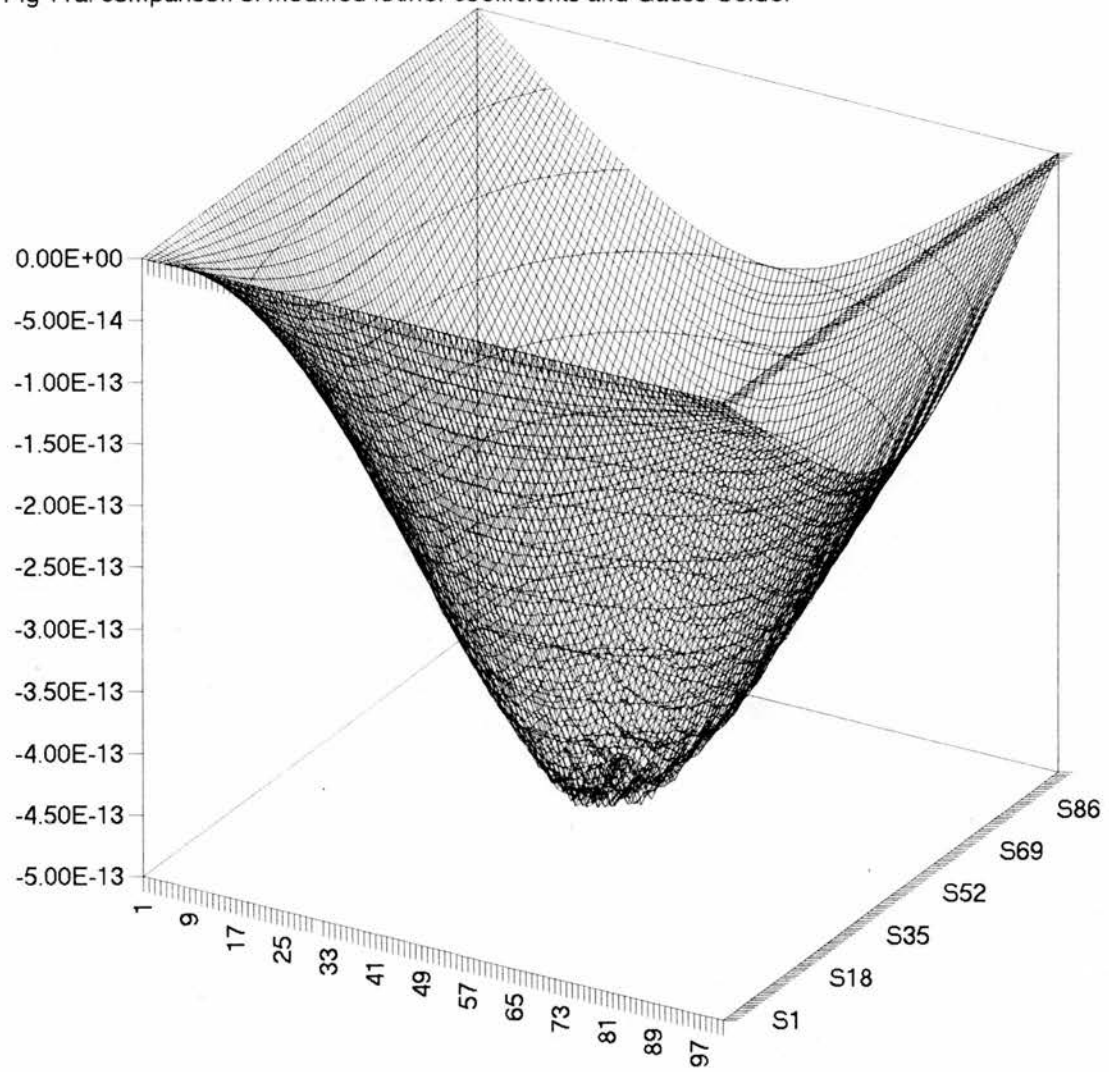


Fig. 11b: As Fig. 11a but with x=0.5

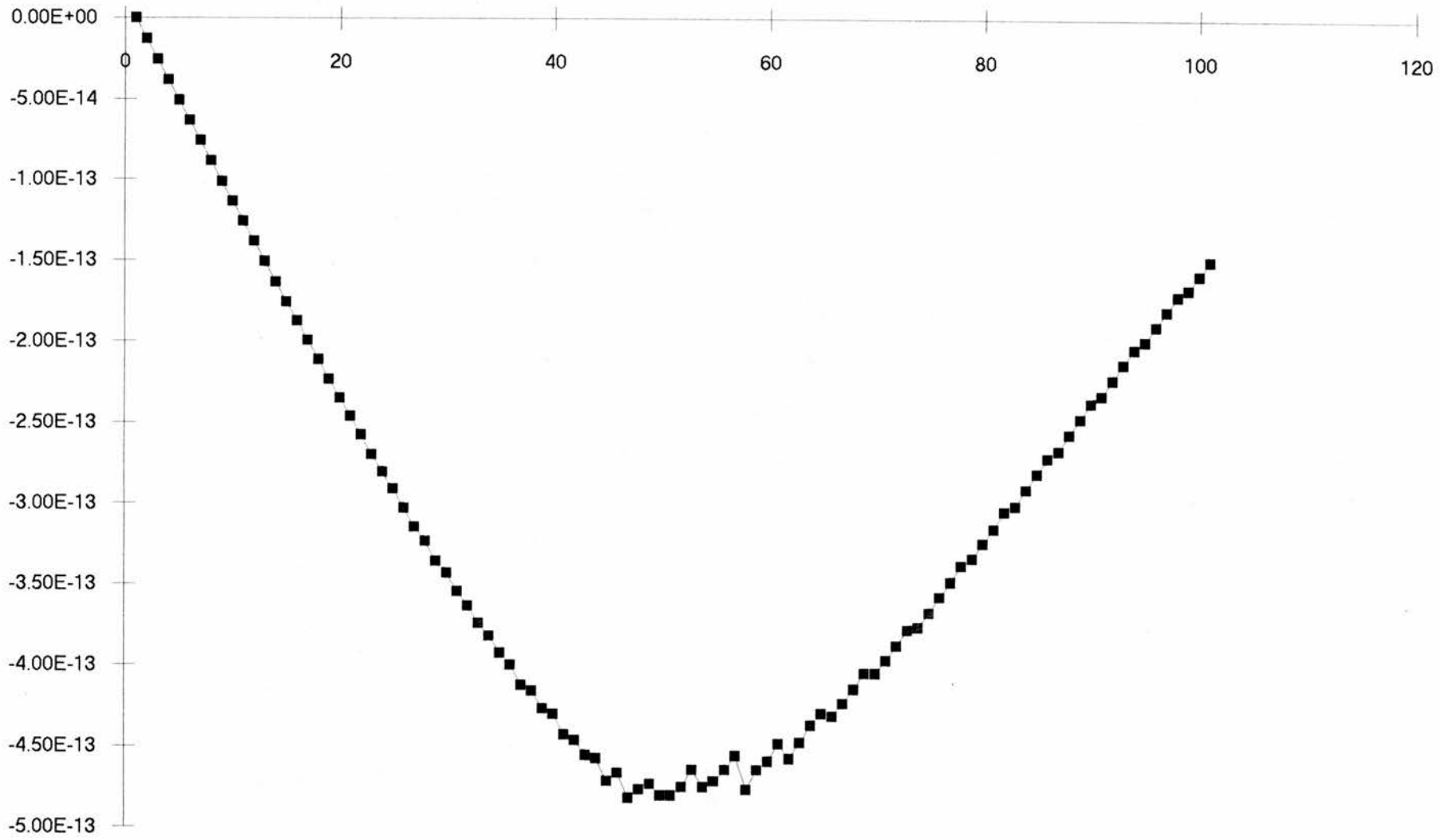
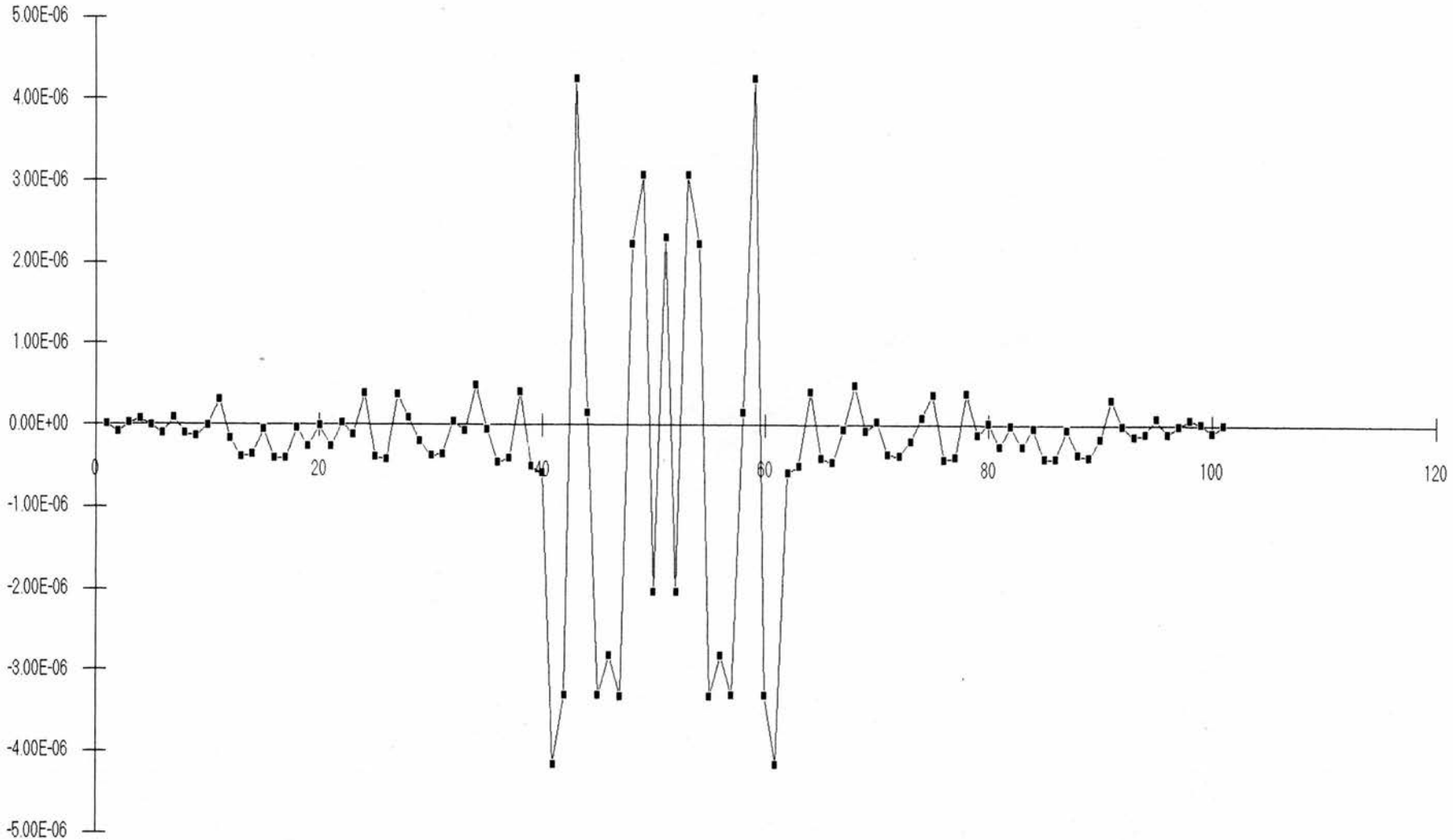


Fig. 11c: Fourier minus exact solution to difference equation at x=0.5.



$$error = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty}\left(\frac{A_{m,n}}{-\alpha + \frac{4}{h^2}\left(\sin^2\left(m\pi\,\frac{h}{2}\right)+\sin^2\left(n\pi\,\frac{h}{2}\right)\right)} - \frac{A_{m,n}}{-\alpha+\left(m^2+n^2\right)\pi^2}\right)\sin(m\pi x)\sin(n\pi y)$$

Where the derivations are given above.

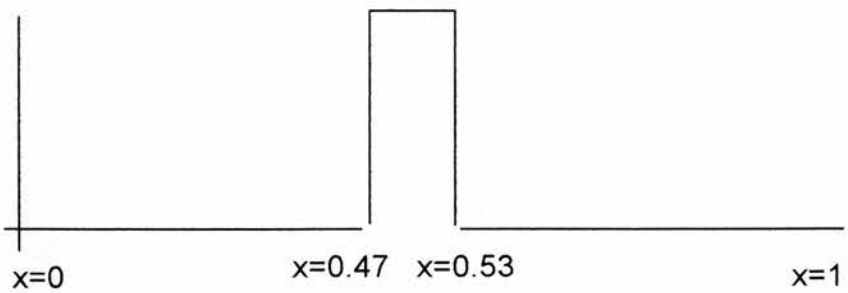
In order to gain some insight into the nature of the discretisation error it is convenient to simplify the problem. Consider a one-dimensional version of the steady state neutron diffusion equation:

suppose φ satisfies

$$\frac{d^2\varphi}{dx^2} + \alpha\varphi + S(x) = 0,$$

where $\varphi = 0$ at x = 0 and x = 1 and

S(x) is the Fourier series representation of a rectangular pulse:



The form of S(x) is

$$S(x) = \sum_{m}^{\infty} A_m \{\sin(m\pi x)\}$$

where $A_m = \left(\frac{2}{m\pi}\left(\cos(0.47m\pi) - \cos(0.53m\pi)\right)\right)$

and in line with the earlier analysis the exact solution to the differential equation is

$$\varphi(x) = \sum_{m}^{\infty} \frac{A_m}{-\alpha + (m^2)\pi^2} \sin(m\pi x)$$

An order $O(h^2)$ approximation to the 1 dimensional equation is:

$$\frac{\varphi_{i+1} + \varphi_{i-1} - 2\varphi_i}{h^2} + \alpha\varphi_i + S(x_i) = 0$$

and the corresponding solution to the set of difference equations is

$$\varphi_i = \sum_{m=1}^{\infty} \frac{A_m}{-\alpha + \frac{4}{h^2}\left(\sin^2\left(m\pi \frac{h}{2}\right)\right)} \sin(m\pi x)$$

Note that the series is infinite as the source term S(x) is an infinite series and the above term is constructed to match S(x) term by term. Thus we can see that, in theory, the error $\varphi_i - \varphi(x_i)$ is the difference between these two infinite series, i.e.

$$error = \sum_{m=1}^{\infty} \left(\frac{1}{-\alpha + \frac{4}{h^2}\left(\sin^2\left(m\pi \frac{h}{2}\right)\right)} - \frac{1}{-\alpha + m^2\pi^2} \right) \left((\cos(0.47m\pi) - \cos(0.53m\pi)) \frac{2}{m\pi} \right) \sin(m\pi x)$$

Each term of the above series can be considered to be an amplitude multiplied by a sine wave, where the amplitude is a function of the frequency of the sine wave. The frequency is $m\pi$ and due to the symmetry of S, m only takes odd values. A plot of the above equation is shown in Figs. 12a and 12b where the

maximum number of terms is actually a variable ranging from 1 to 641 in steps of 2 and is indicated on the axis in the figure. Fig 12a is given from an elevated angle of 35^0 above the horizontal, and Fig 12b is taken from 35^0 below the horizontal. To aid clarity the surface plot utilises a 'hide-line' option effectively making the surface opaque. The other independent variable axis is the x axis which ranges from 0 to 1 in steps of 0.01. The dependent variable (vertical axis) is the error.

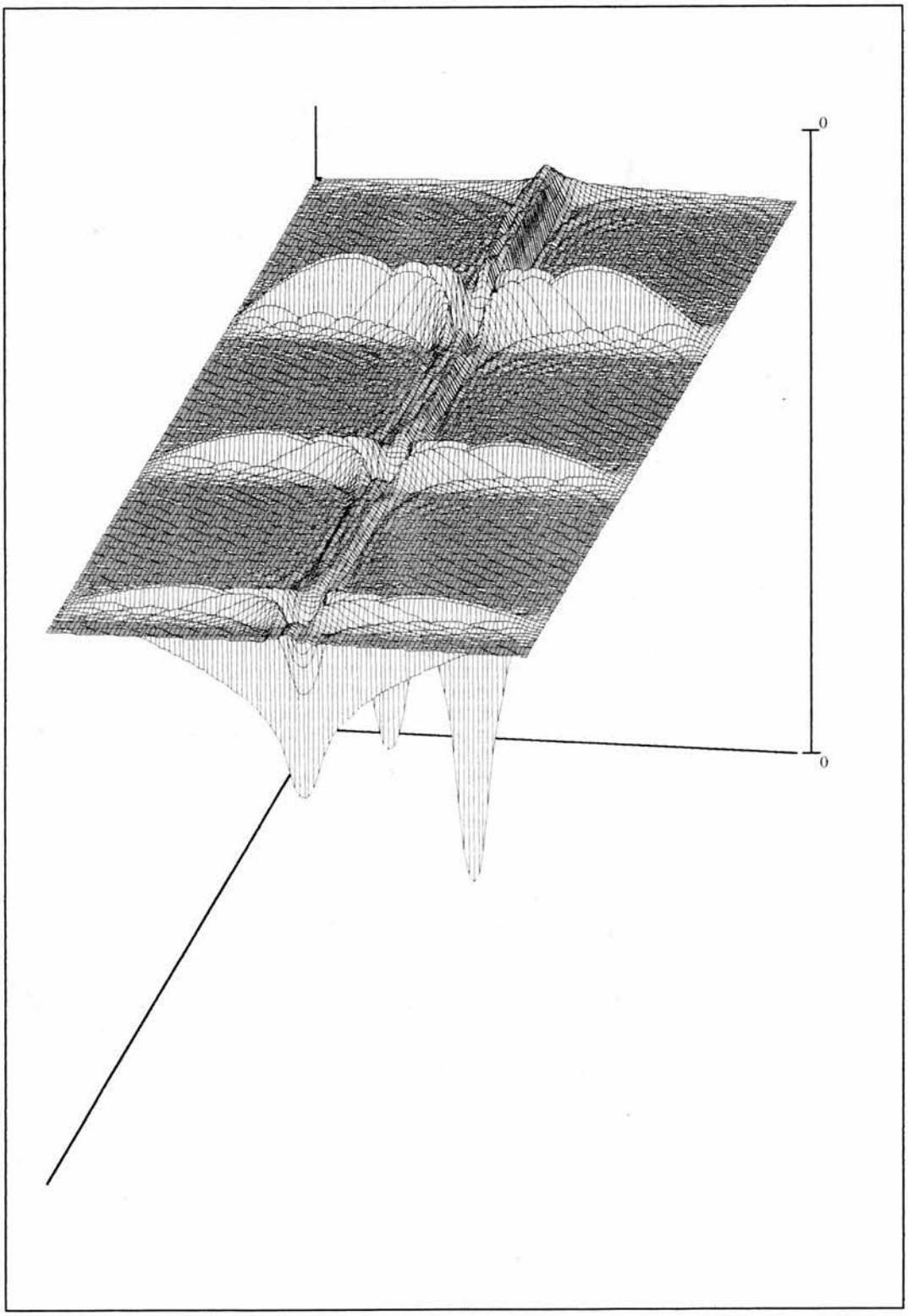
Let the upper limit to which the series is summed be M.

Figure 12a and 12b show an unexpected phenomenon where it is seen that the error becomes comparatively very large when M ≅ even multiple of 100. Closer examination of the figures shows that when the series for S is summed up M=200N+1 the error distribution is very smooth, but the maximum error is large whereas summing the series for S to (2N+1)100+1 terms reduces the error considerably but generates a highly oscillatory pattern. This phenomenon is repeated in the two dimensional case. Figures 13a..13h show magnified details of the error plot with the number of terms ranging from 901 down to 101 in steps of 100. The figures for the one-dimensional and the two-dimensional situations reveal that when $M \cong \frac{2n}{h}$,where n is an integer then a large maximum discretisation error is observed and the spatial distribution is oscillation free. When $M \cong \frac{2n+1}{h}$, the error due to discretisation is much reduced and the spatial error pattern possesses a high frequency oscillation.

$i = 0..320 \quad j = 0..100 \quad h = 0.01 \quad \alpha = 2 \quad x_j = 0.01 \cdot j \quad mm_i = 2 \cdot i + 1$

$$\delta(mm, x) = \sum_{m=1}^{mm} \left[\left(\frac{1}{-\alpha + \frac{4}{h^2} \cdot \sin\left(m \cdot \pi \cdot \frac{h}{2}\right)^2} - \frac{1}{-\alpha + m^2 \cdot \pi^2} \right) \cdot \left[(\cos(0.47 \cdot m \cdot \pi) - \cos(0.53 \cdot m \cdot \pi)) \cdot \frac{2}{m \cdot \pi} \right] \right] \cdot \sin(m \cdot \pi \cdot$$

$M_{i,j} = \delta(mm_i, x_j)$



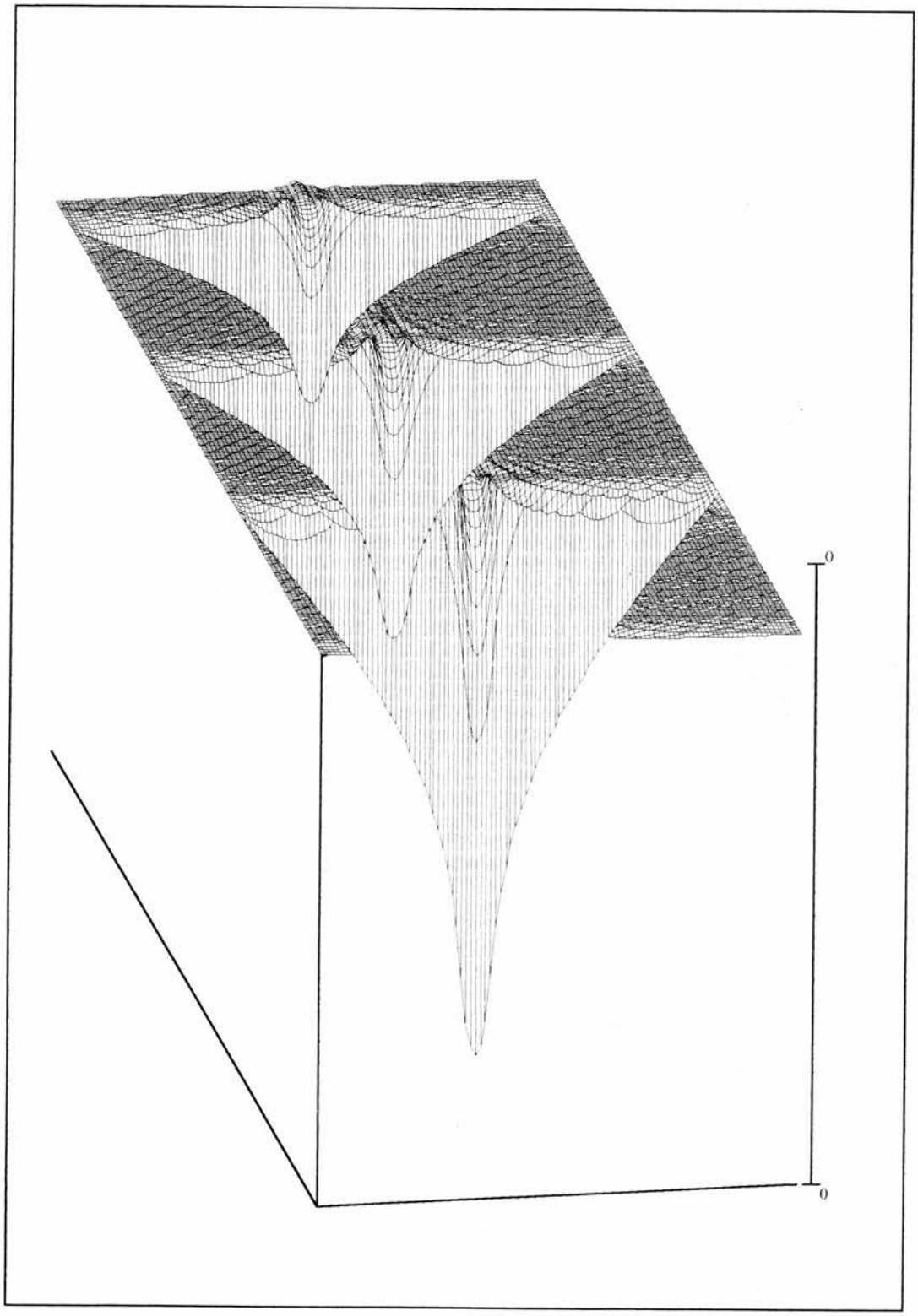
M

Fig. 12a

$i = 0..320$ $j = 0..100$ $h = 0.01$ $\alpha = 2$ $x_j = 0.01 \cdot j$ $mm_i = 2 \cdot i + 1$

$$\delta(mm, x) = \sum_{m=1}^{mm} \left[\left(\frac{1}{-\alpha + \frac{4}{h^2} \cdot \sin\left(m \cdot \pi \cdot \frac{h}{2}\right)^2} - \frac{1}{-\alpha + m^2 \cdot \pi^2} \right) \cdot \left[(\cos(0.47 \cdot m \cdot \pi) - \cos(0.53 \cdot m \cdot \pi)) \cdot \frac{2}{m \cdot \pi} \right] \right] \cdot \sin(m \cdot \pi \cdot$$

$M_{i,j} = \delta(mm_i, x_j)$



M

Fig.12b

fig 13a : GS-Numerical error: 801 terms: magnified view from nodes 45 to 55 on both axes

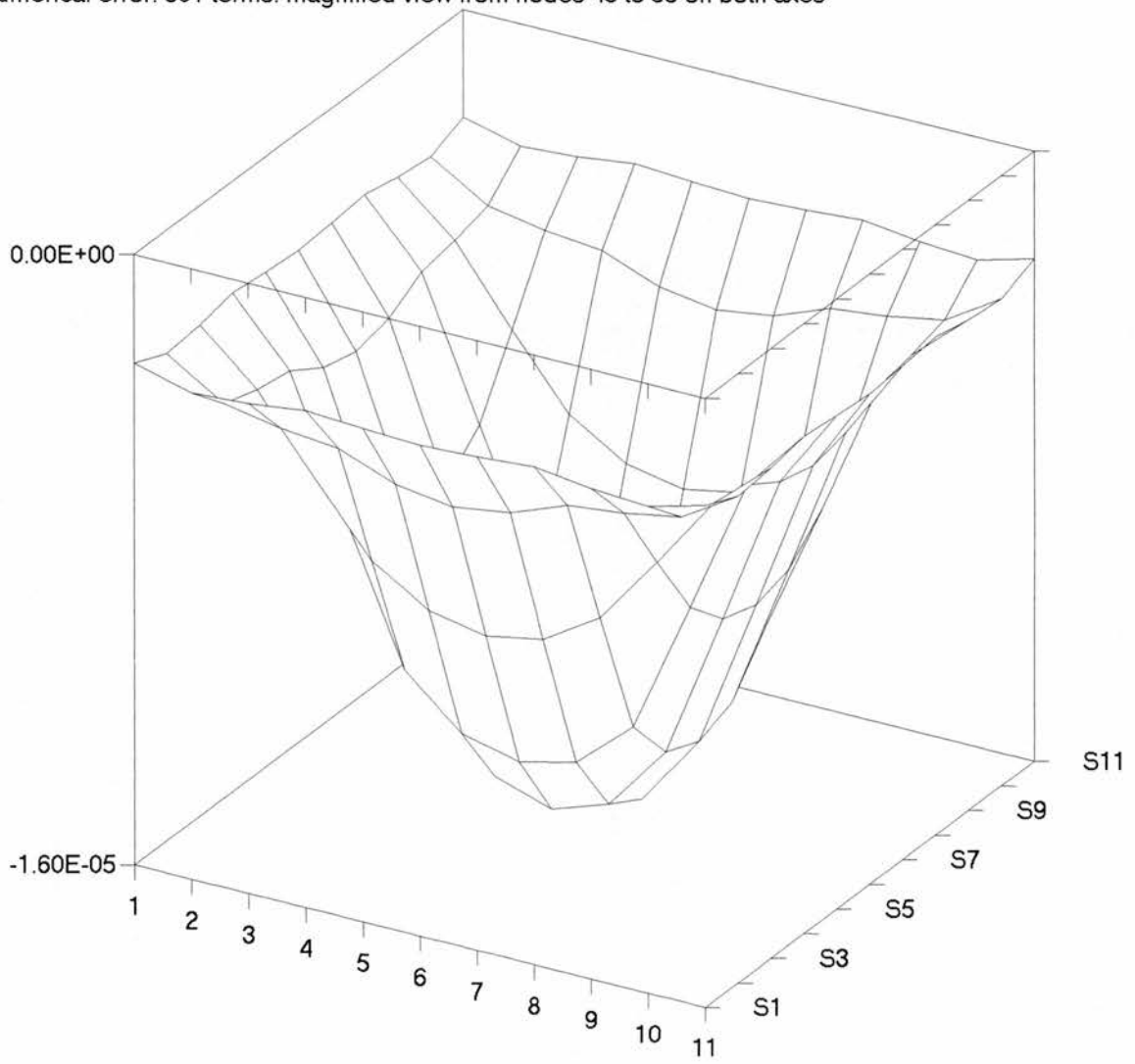


fig 13b: GS-Numerical error: 701 terms: magnified view from nodes 45 to 55 on both axes

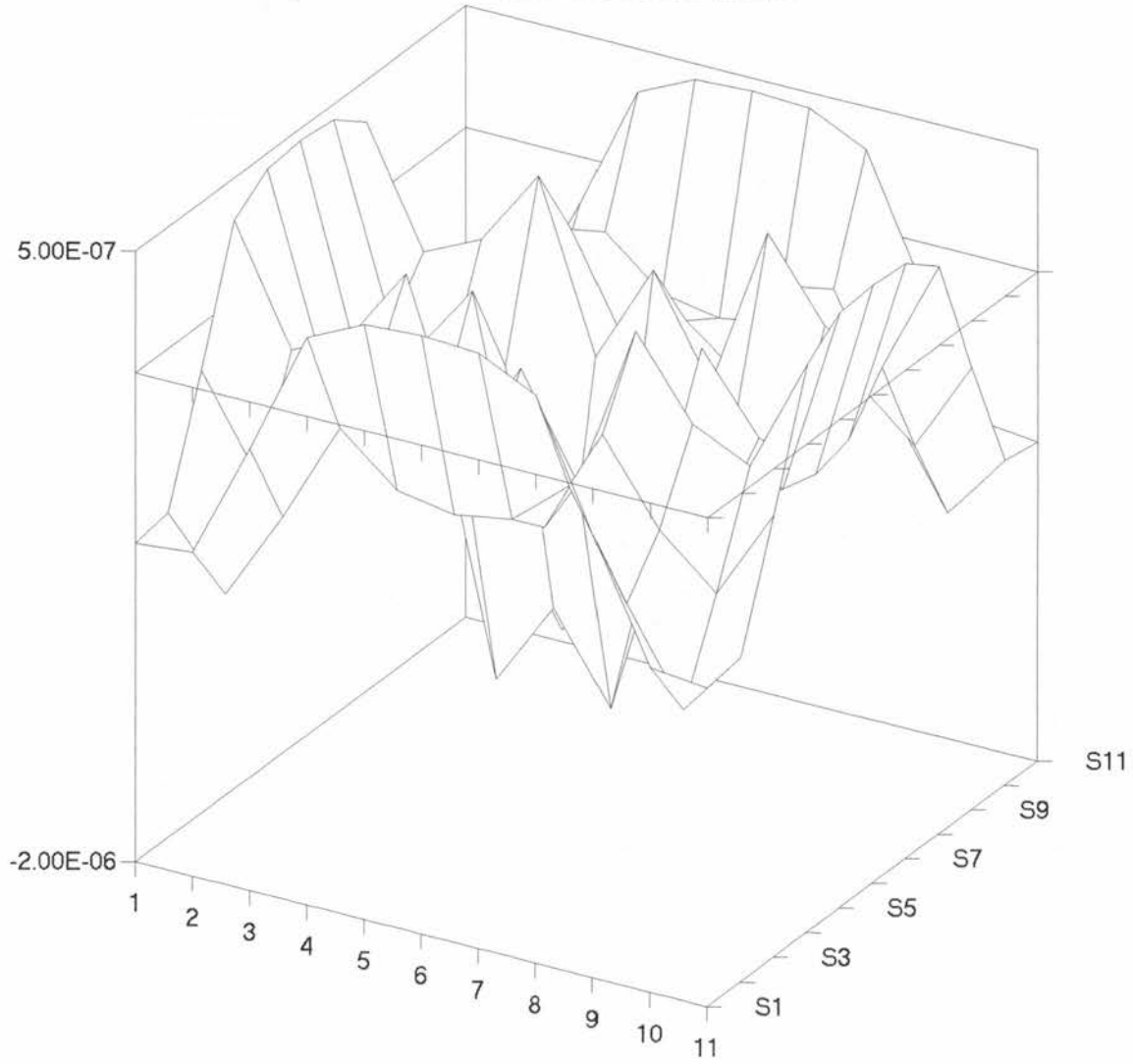


fig 13c: GS-Numerical error:601 terms: magnified view from nodes 45 to 55 on both axes

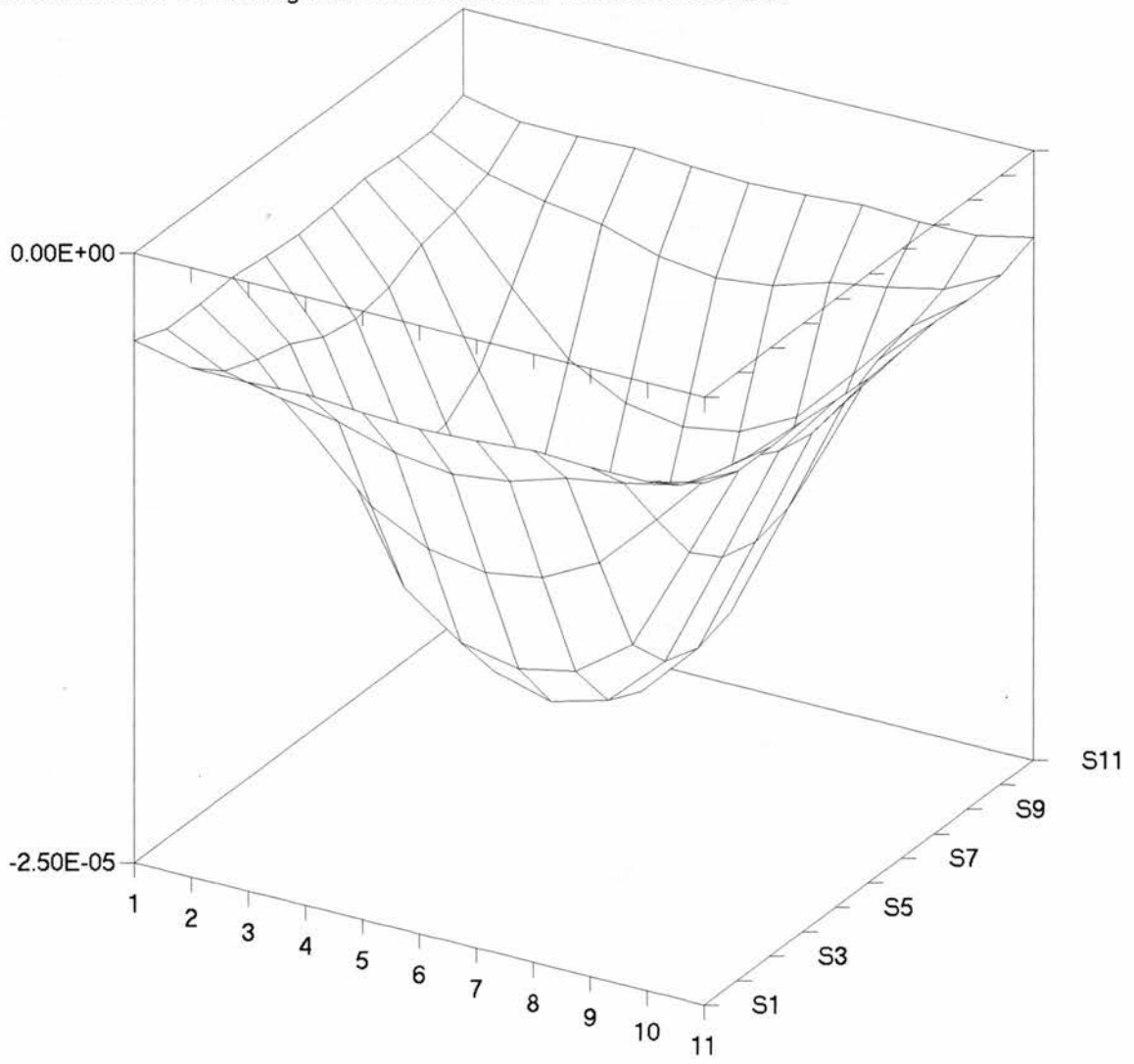


fig 13d: GS-Numerical error: 501 terms: magnified view from nodes 45 to 55 on both axes

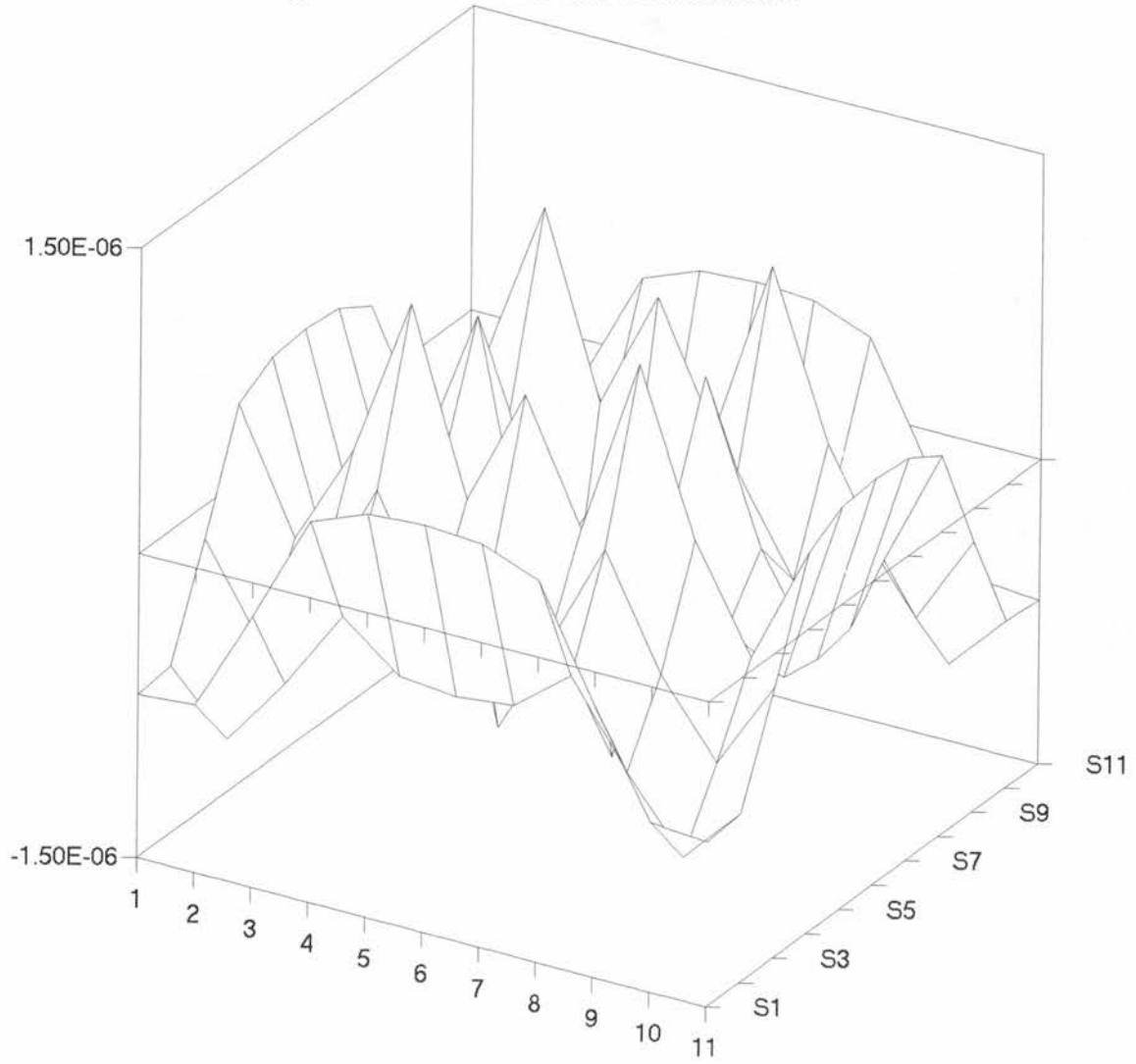


fig 13e: GS-Numerical error: 401 terms: magnified view from nodes 45 to 55 on both axes

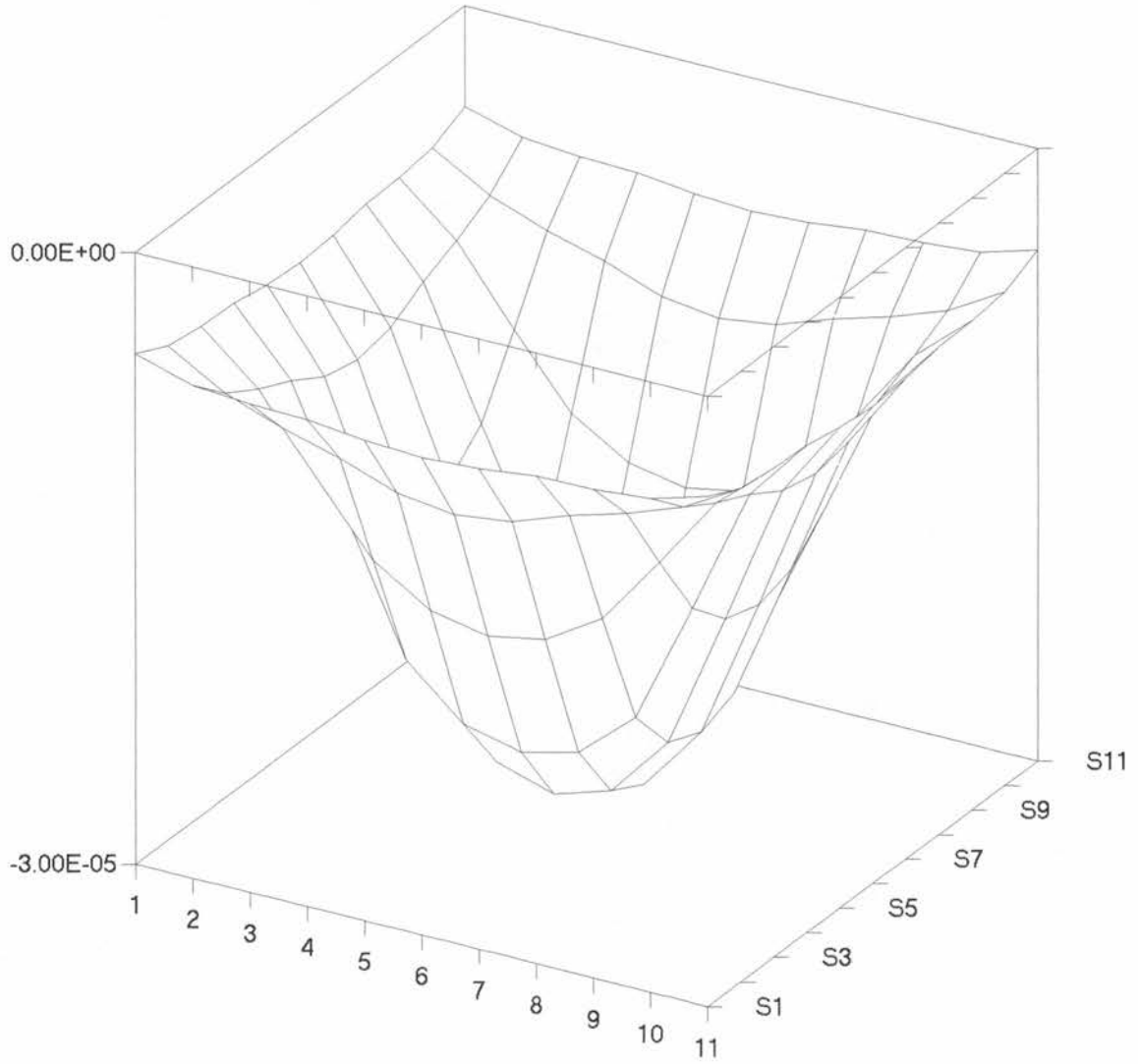


fig 13f: GS-Numerical error: 301 terms: magnified view from nodes 45 to 55 on both axes

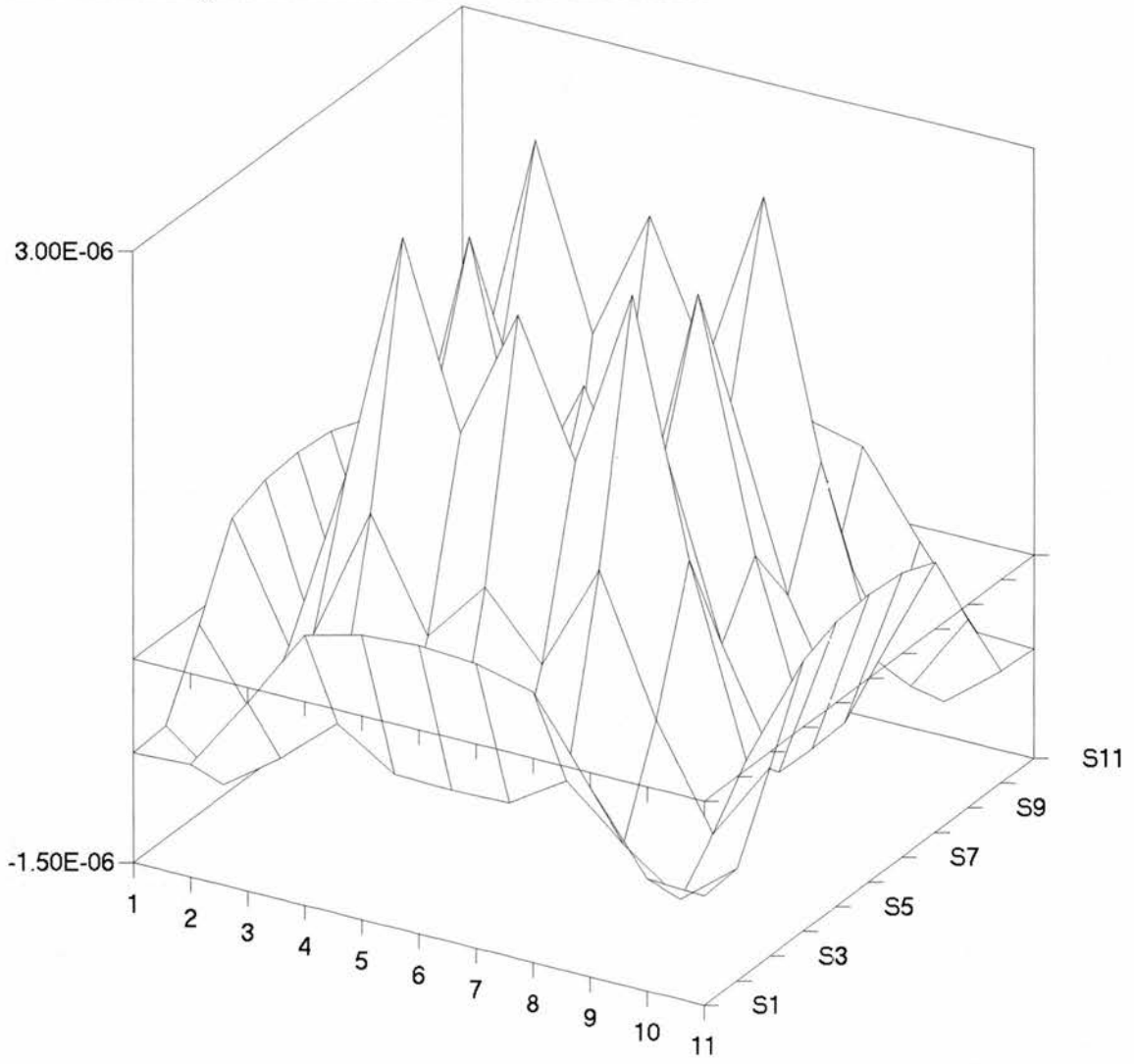
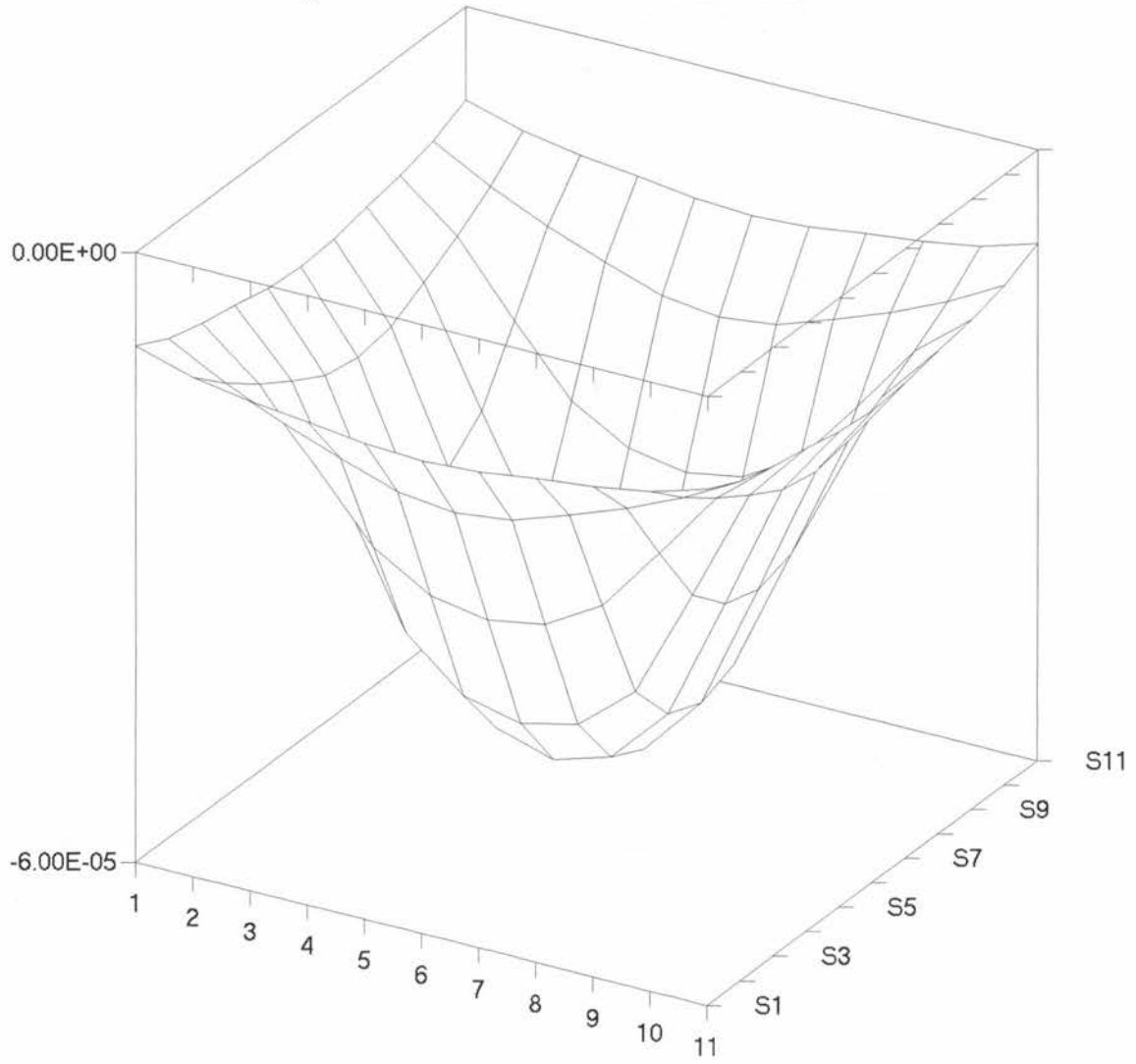


fig 13g: GS-Numerical error: 201 terms: magnified view from nodes 45 to 55 on both axes



Increasing the number of terms in the series for S improves the representation of the rectangular pulse but since the series is used as an input to the Gauss-Seidel routine this error behaviour is not associated with the limitations of the Fourier representation. The nature of the error can be seen by examining the mathematical expression for the error. Writing the error as

$$\sum_{m=1}^{M} A(m)\sin(m\pi x)$$

where $A(m) = \frac{2D(m)}{m\pi}(\cos(0.47m\pi) - \cos(0.51m\pi))$

and

$$D(m) = \frac{1}{-\alpha + \frac{4}{h^2}\sin^2\left(\frac{m\pi h}{2}\right)} - \frac{1}{-\alpha + m^2\pi^2}$$

the source of the error is the approximation of $m^2\pi^2$ by $\frac{4}{h^2}\sin^2\left(\frac{m\pi h}{2}\right)$.

When mh is small $\frac{4}{h^2}\sin^2\left(\frac{m\pi h}{2}\right) \cong m^2\pi^2 - \pi^4\frac{m^4h^2}{12} + O(h^4)$ and the discrepancy

$D(m) \cong \frac{h^2}{12}$

When m is such that $\sin\left(\frac{m\pi h}{2}\right) \cong 1$, say $\frac{m\pi h}{2} \cong \frac{\pi}{2}$ or $m \cong \frac{1}{h}$,

$$D\left(\frac{1}{h}\right)=\frac{1}{-\alpha+\frac{4}{h^2}}-\frac{1}{-\alpha+m^2\pi^2}\cong\frac{\left(\pi^2-4\right)}{h^2}\frac{h^4}{4\pi^2}$$

since $\alpha << \frac{1}{h^2}$

and the size of the discrepancy is similar to D(m) when m is modest.

However, should $\sin\left(\frac{m\pi h}{2}\right)=0$, which occurs when $m=\frac{2}{h},\ \frac{4}{h},\ \frac{6}{h},\ \ldots,$

$$D(m)=\frac{1}{-\alpha}-\frac{1}{-\alpha+m^2\pi^2}\cong\frac{-m^2\pi^2}{\alpha m\pi^2}=-\frac{1}{\alpha}$$

since $m^2\pi^2 >> \alpha$.

This extreme case never happens as m is always odd. Note that the expression for

$$A(2k+1)=\frac{4D(2k+1)}{\pi(2k+1)}(-1)^k\sin(0.03(2k+1)\pi),\quad k=0,12,\ldots$$

and $A(2k)\equiv 0$, $k=1,2,\ldots$

Thus there is never a value of m such that $\sin\left(\frac{m\pi h}{2}\right)\equiv 0$. However, there are values adjacent to these. Consider the terms where $m=\frac{2}{h}\pm 1$, or $\frac{4}{h}\pm 1$, etc,

$$\sin^2\left(\frac{m\pi h}{2}\right)=\left[\sin\left(\left(\frac{2n}{h}\pm 1\right)\frac{\pi h}{2}\right)\right]^2=\left[\sin\left(n\pi\pm\frac{\pi h}{2}\right)\right]^2=\sin^2\left(\frac{\pi h}{2}\right)\cong\frac{\pi^2h^2}{4}$$

and so $D\left(\frac{2m}{h}\pm 1\right)\cong\frac{1}{\pi^2-\alpha}$

which is a substantial discrepancy that will appear throughout the series. Notice also

that $D\left(\frac{2n}{h}+1\right)=D\left(\frac{2n}{h}-1\right)$

$\sin\left(0.03\left(\frac{2n}{h}+1\right)\pi\right) = -\sin\left(0.03\left(\frac{2n}{h}-1\right)\pi\right)$ and

$\sin\left(\left(\frac{2n}{h}+1\right)\pi x\right) = -\sin\left(\left(\frac{2n}{h}-1\right)\pi x\right)$.

Therefore, the term when $m = \left(\frac{2n}{h}+1\right)$ is almost equal and opposite to the term

when $m = \left(\frac{2n}{h}-1\right)$. For example

$A(199)\sin(199\pi x) + A(201)\sin(201\pi x) =$

$-\frac{4}{\pi} D(199)\sin(\pi x)\sin(3\pi x)\left(\frac{1}{199}-\frac{1}{201}\right)$

and considerable cancellation takes place.

Indeed, the sum of these two terms is similar in magnitude to the first term in the series.

Similar remarks can be made about the other terms which are equidistant about the

index $\frac{2}{h}$.

Fig 14 illustrates the relative sizes of the amplitudes A(m) for increasing m

and clearly the major source of error occurs in the localities $\frac{2}{h}$, $\frac{4}{h}$, $\frac{6}{h}$, ...

Hence, the worst error will occur when the series for S is summed to

$\frac{2n}{h} \pm 1$ as little error cancellation of the major contributors takes place.

However, by summing the series to a number of terms which lies between

these locations, such as to $M = \frac{2n+1}{h}$, these large error contributions will, to

an extent, cancel out and the average error discretisation error will be much smaller. Further, when significant cancellation is taking place in the summation it is anticipated that the result is highly oscillatory. This will not happen when the larger contributions dominate the summation.

$$A(m) := \left(\frac{1}{-\alpha + \frac{4}{h^2} \cdot \sin\left(m \cdot \pi \cdot \frac{h}{2}\right)^2} - \frac{1}{-\alpha + m^2 \cdot \pi^2} \right) \cdot \left| (\cos(0.47 \cdot m \cdot \pi) - \cos(0.53 \cdot m \cdot \pi)) \cdot \frac{2}{m \cdot \pi} \right| *$$

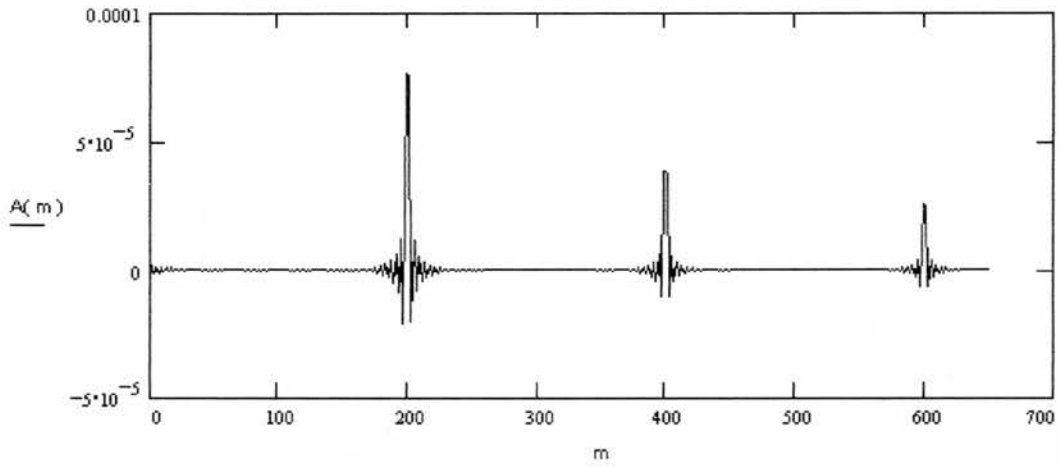


Fig. 14; Term by Term Discrepancy

In conclusion, the error distribution observed in Fig 10a is due to the summation of a series of almost equal terms. Had a different number of terms been selected the oscillation may not have been observed. Although the error pattern itself is of no great significance the role of the number of terms taken in the definition of S is determining the size of the discretisation error was unexpected. One implication of this is that very small changes in the values of S, obtained by adding a few more terms, can change the observed numerical error dramatically as can be seen from figs 13a to 13h where maximum error

when M=601 is -2E-5, wherease the maximum error for 301 and 501 are 4E-6 and 2.5E-6 respectively.

Although the reason for specifying S in this way was to overcome errors associated with discretisation data, the effect is to introduce another possible source of error which can be avoided by carefully selecting the number of terms in the series.

CHAPTER 3 : The transient problem

The complete equation for transient neutron diffusion was given in the introduction in chapter 1: $\nabla^2\varphi + \alpha\varphi + S = \frac{\partial\varphi}{\partial t}$.

The original research proposal intended that the fission enhancing α term be constructed as a spatially dependent function. Where positive, α could then take the form of a fission enhancer; and where negative it could simulate a Neutron absorbing control rod. The proposal envisaged that the model be solved analytically. The function α was to have been constructed from a double Fourier series in x and y, and it was hoped that mutual orthogonality between α and φ would have produced a tractable non-linear problem allowing the principle of superposition to generate a Fourier type solution. Had this proved possible, we would have been able to provide an analytical solution to a particular diffusion problem, and offer this as verification for more standard numerical methods. Unfortunately this strategy proved to be unduly optimistic except for special and artificial choices for α as will be discussed later in this chapter.

Instead it was decided to consider the evolution of the flux from the previously calculated equilibrium with the fuel rod removed. The source

term S simulates the isolated fuel rod and is set to zero everywhere except in the vicinity of the rod. In chapter 2 the solution to $\nabla^2\varphi + \alpha\varphi + S = 0$ with such an S is found. S. Now imagine the situation when S is set to zero everywhere throughout the region. The previous steady state solution becomes the initial condition and evolves to some new equilibrium.

Steady state data obtained from solving the equation $\nabla^2\varphi + \alpha\varphi + S = 0$ using the code in gs.pas is read into the transient problem described by the equation $\nabla^2\varphi + \alpha\varphi = \frac{\partial\varphi}{\partial t}$ where the source term has been removed. The region is a unit square and the boundary conditions remain the same: $\varphi = 0$. Again, an analytical solution to this problem can be found using separation of variables ref. 5 (Carslaw & Jaeger, 1959) and is given by:

$$\varphi = \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} B_{m,n} e^{t\left(\alpha-\left(n^2+m^2\right)\pi^2\right)} \sin(n\pi x)\sin(m\pi y)$$

where $B_{m,n}$ is derived from the results in chapter 2, that is

$$B_{m,n} = \frac{4}{nm\pi^2\left(-\alpha+\left(n^2+m^2\right)\pi^2\right)}\left(\cos(0.47m\pi)-\cos(0.53m\pi)\right)\left(\cos(0.47n\pi)-\cos(0.53n\pi)\right)$$

The code for evaluating this solution is given in antrs.pas , and the results are presented in fig 15.

The solution for comparison is given by the Simple Explicit Method ref. 9 (Mitchell & Griffiths, 1980) using the algorithm below:

$$\varphi_{i,j,k+1} = \frac{k_1}{h^2}\left(\varphi_{i+1,j,k} + \varphi_{i-1,j,k} + \varphi_{i,j+1,k} + \varphi_{i,j-1,k}\right) + \varphi_{i,j,k}\left(1 - k_1\left(\frac{4}{h^2} - \alpha\right)\right)$$

where h is the spacing of a square grid and k is the time measurement. The code for this solution is given by gstrns.pas, and the results are given in fig 16. The advantage of using the Simple Explicit Method is that is simple to implement and reasonably quick on a 66MHz PC. The usual restriction on step size $\frac{k_1}{h^2} \leq \frac{1}{4}$ applies as α is taken to be positive. This guarantees that all terms in the formula are positive and hence errors of cancellation are avoided.The difference between the Simple Explicit and the analytical method is given

in fig. 16 where we see that there is little difference between the solutions. Note that $\varphi \to 0$ as $t \to \infty$ and the numerical values tend to under predict the true solution at all but small values of t. Hence the reactor cools.

Critical behaviour of the Neutron Diffusion Equation.

A reactor is said to be 'critical' when the flux is self sustaining. In terms of the model the 'enhancer' α is sufficient to maintain the flux without need for an explicit source S. This situation is represented in the model by

$\nabla^2\varphi + \alpha\varphi = 0$, subject to $\varphi = 0$ on the boundary of the unit square.

For non-zero solution of the equation $\nabla^2\varphi + \alpha\varphi = 0$, the value of α must be suitably chosen, this α is the eigenvalue corresponding to the eigenfunction. For example,

$\varphi_{m,n} = A\sin(m\pi x)\sin(n\pi y)$

Fig. 15: Analytical transient

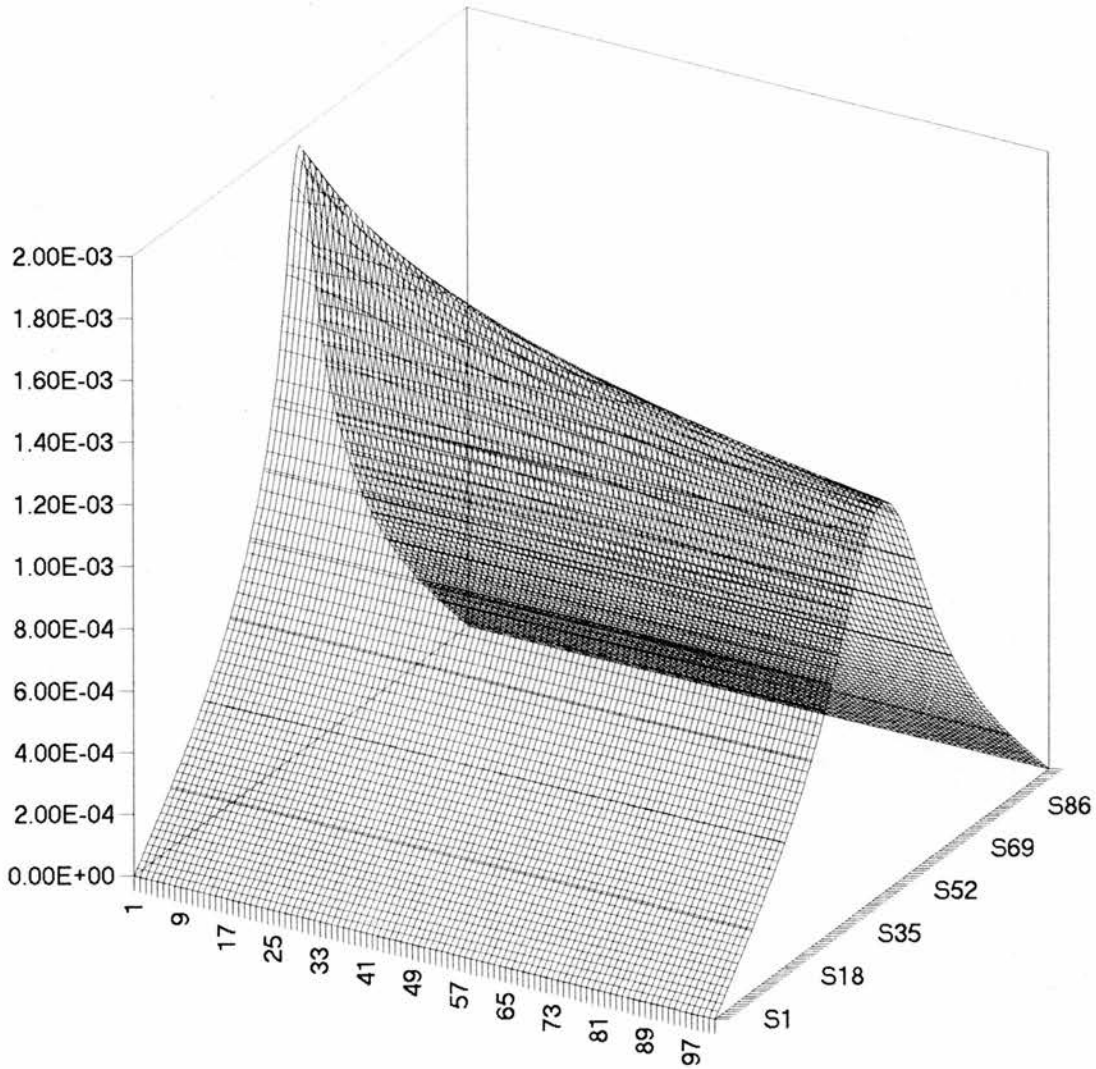
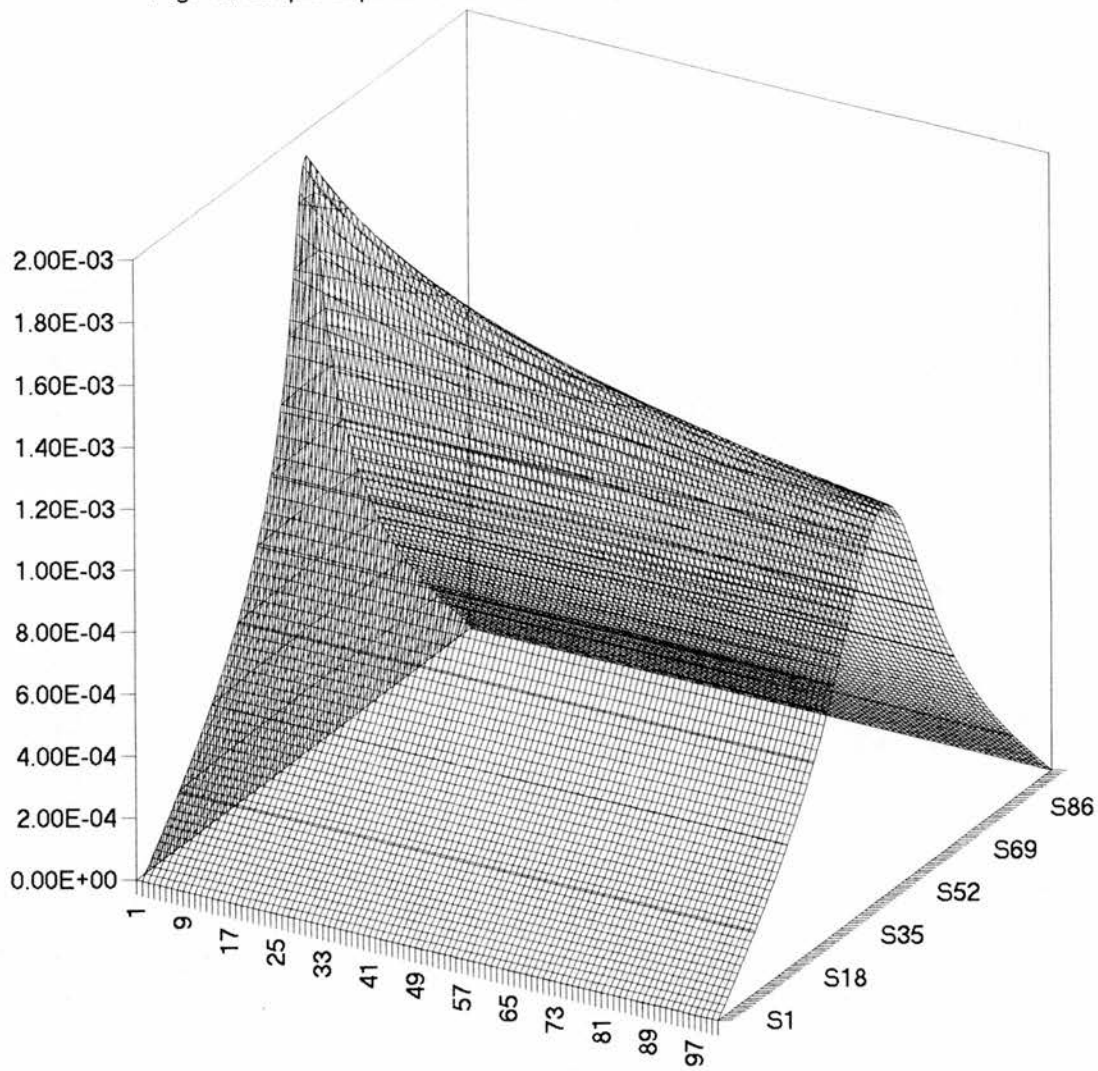


Fig. 16; Simple Explicit Method for transient



satisfies the boundary condition and is a solution of the partial differential equation provided $\alpha = \pi^2(m^2 + n^2)$. In particular, if n=m=1 then $\varphi_{1,1} = A\sin(\pi x)\sin(\pi y)$; representing a self-sustaining flux. 'A' may take any real value but we choose A=1 for simplicity. The analytical solution is given in critanal.pas and the results are shown in fig 18. This result is then fed for comparison into a Gauss- Seidel scheme (where the maximum is seeded to 1.0 - remembering that any value of A would actually provide a solution, so we need to seed the value to the chosen A=1) and allowed to iterate 30,000 times. The results are shown to be stable.

It should be noted that in the previous chapter, our solutions would not have been valid for values of α equal to or greater than the corresponding eigenvalues. Since we chose α =2 and the lowest eigenvalue is over 19, our earlier discussions preclude the case of a critical flux.

A simple analytical solution to the transient diffusion equation where α is a contrived function of x and y.

The equation is

$$\nabla^2\phi + \alpha\phi = \frac{\partial\phi}{\partial t},$$

and we choose a solution of the form

$$\varphi(x,y,t) = xy(1-x)(1-y)e^{-t}$$

Fig. 17: Simple Explicit Method - analytical transient

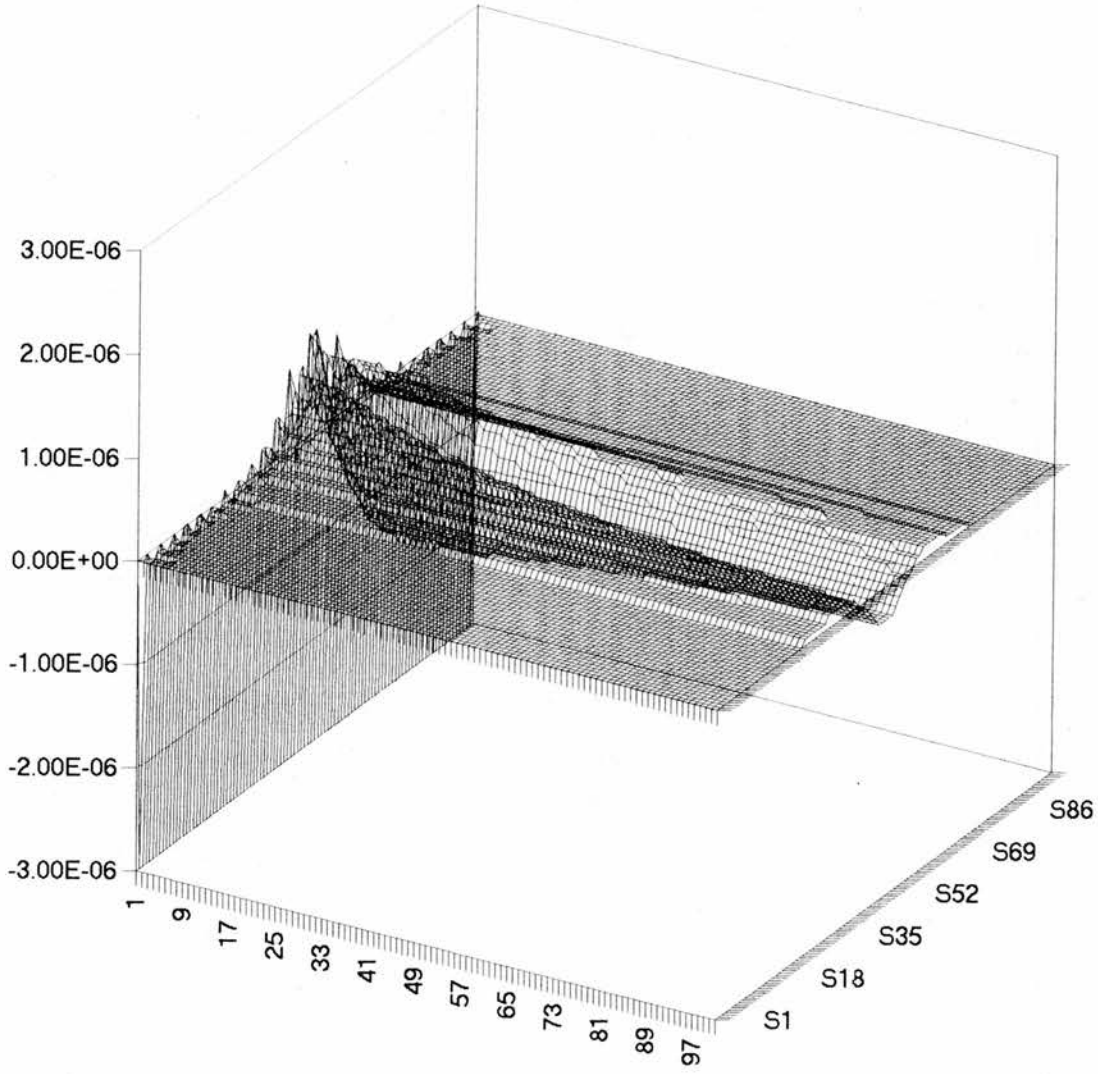
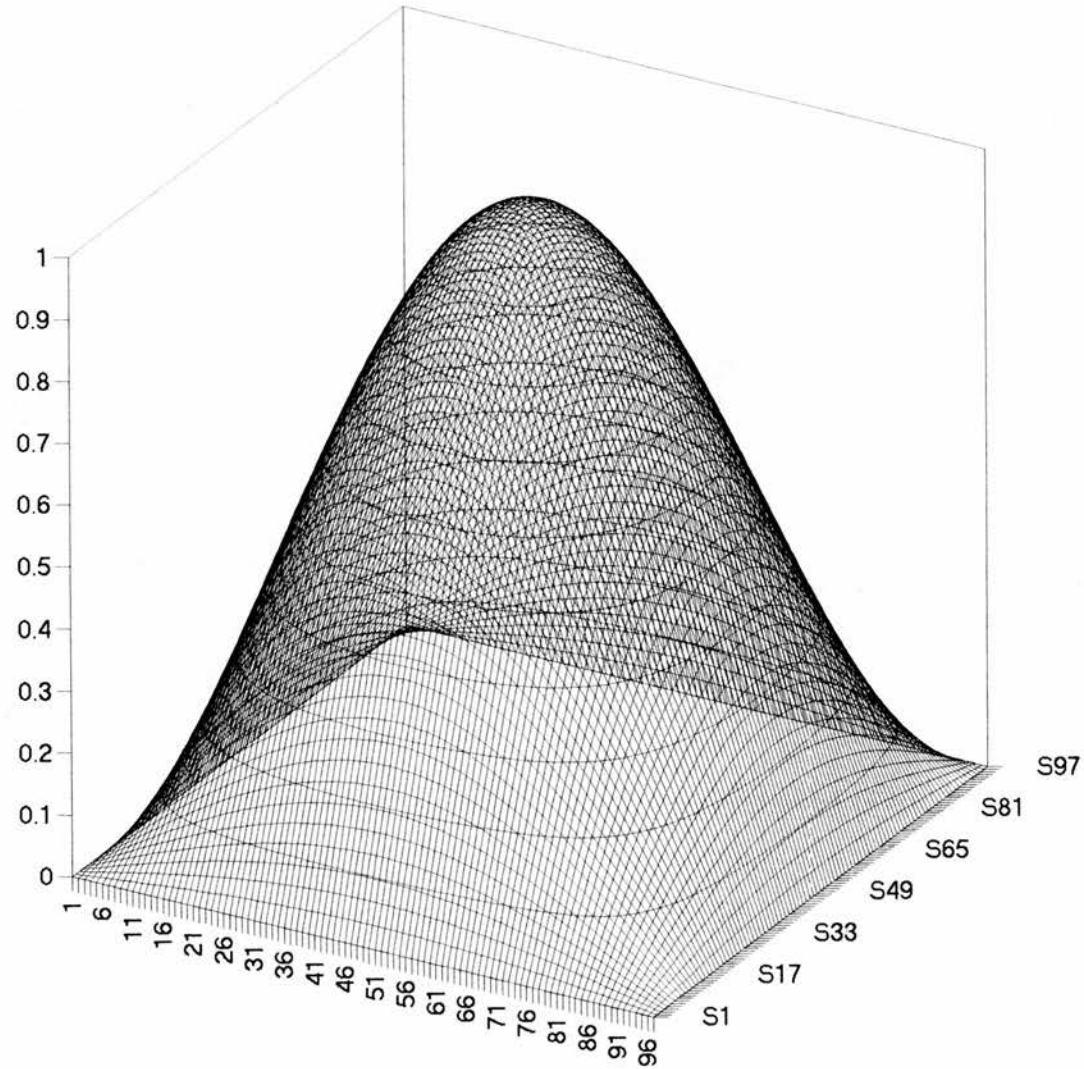


Fig. 18; Analytical critical solution.



where $\alpha(x,y) = -1 + 2\left(\frac{1}{x(1-x)} + \frac{1}{y(1-y)}\right)$ which can easily be verified by substitution back into the differential equation. A graphical representation of the above function $\alpha(x,y)$ is given below in fig 19

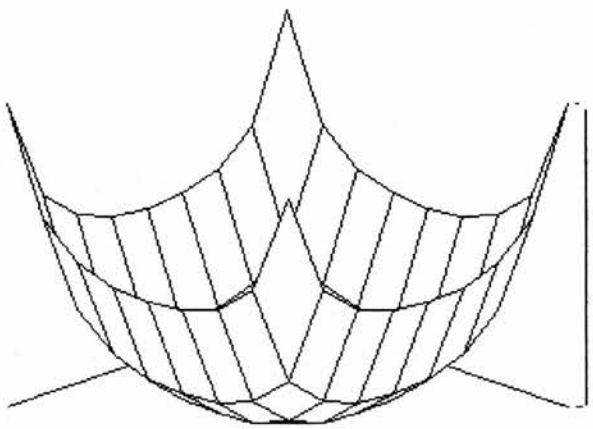
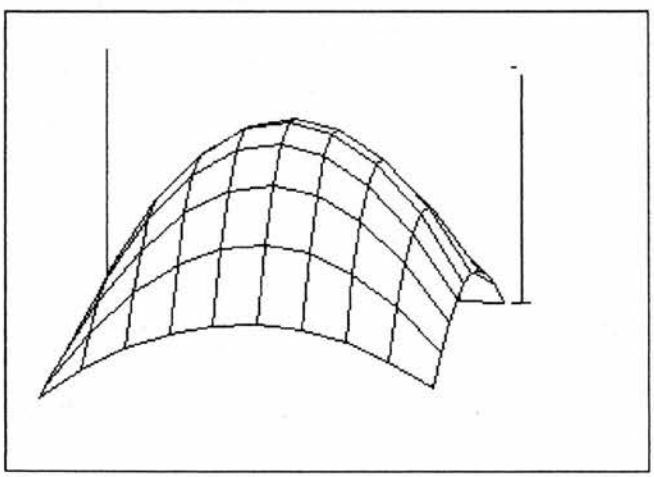


fig 19.

and the flux term $\varphi(x,y)$ at t=0 is given below in fig 20 below:



N

Fig 20

A further simplified version in one-dimension produces transient results which are given graphically in figs 21, 22 and 23.

Fig 21 is the numerical solution , fig 22 is the analytical solution and fig 23 is the error or difference between the solutions. The code for the one-dimensional solution is given in 1dnum.pas and 1dan.pas.

The results show agreement between the two methods to within 0.1% at worst.

It is feasible that even such a simple analysis may be of some value in the verification of a complex numerical method, in that the complex numerical method could be brought to bear on our admittedly artificially constructed α and compared with the analytical method.

fig. 21: 1-D transient: Numerical solution: kay=0.00004, h=0.01, tmax=4

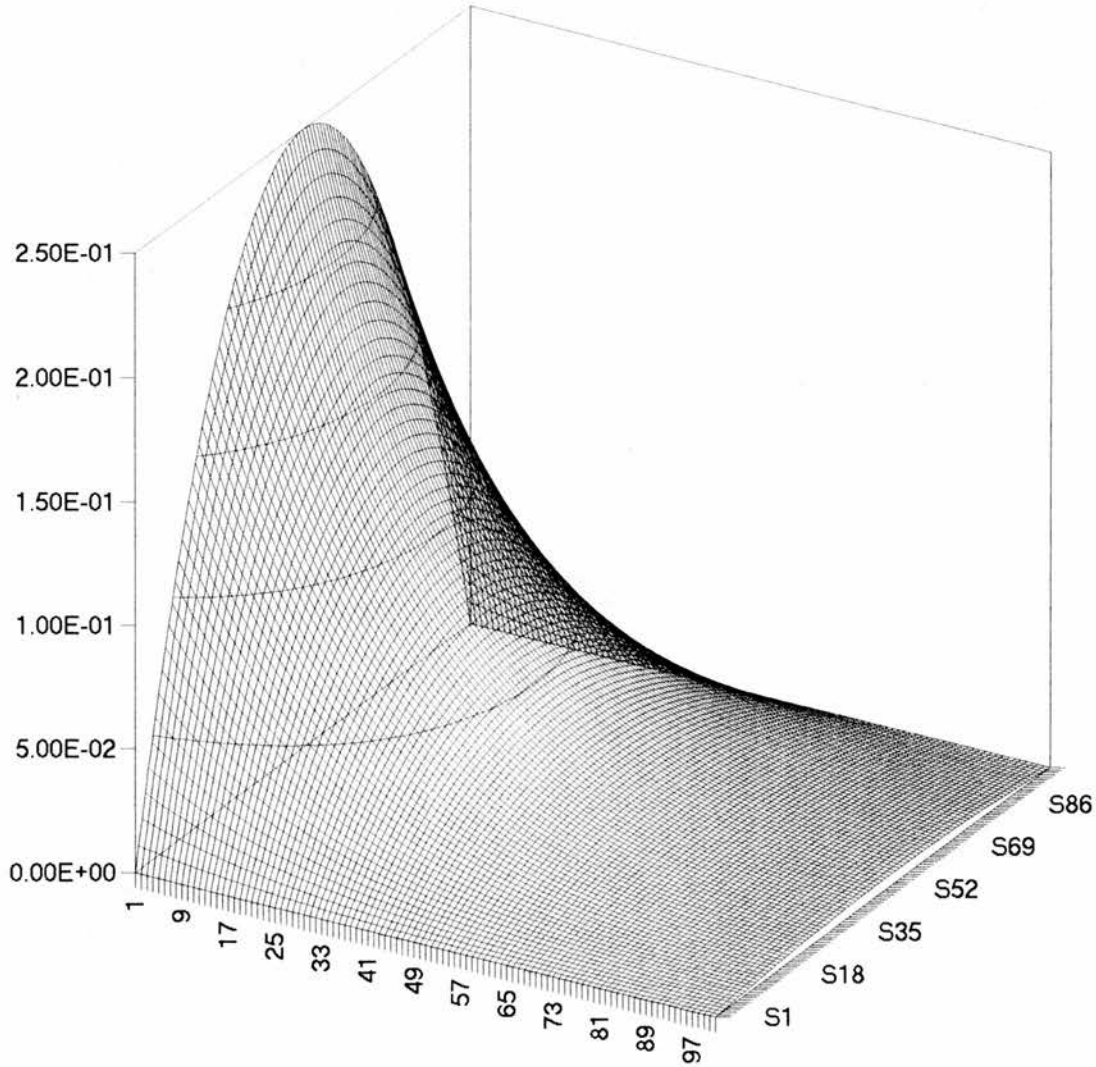


fig 22:1-D transient: Analytical solution: tmax=4

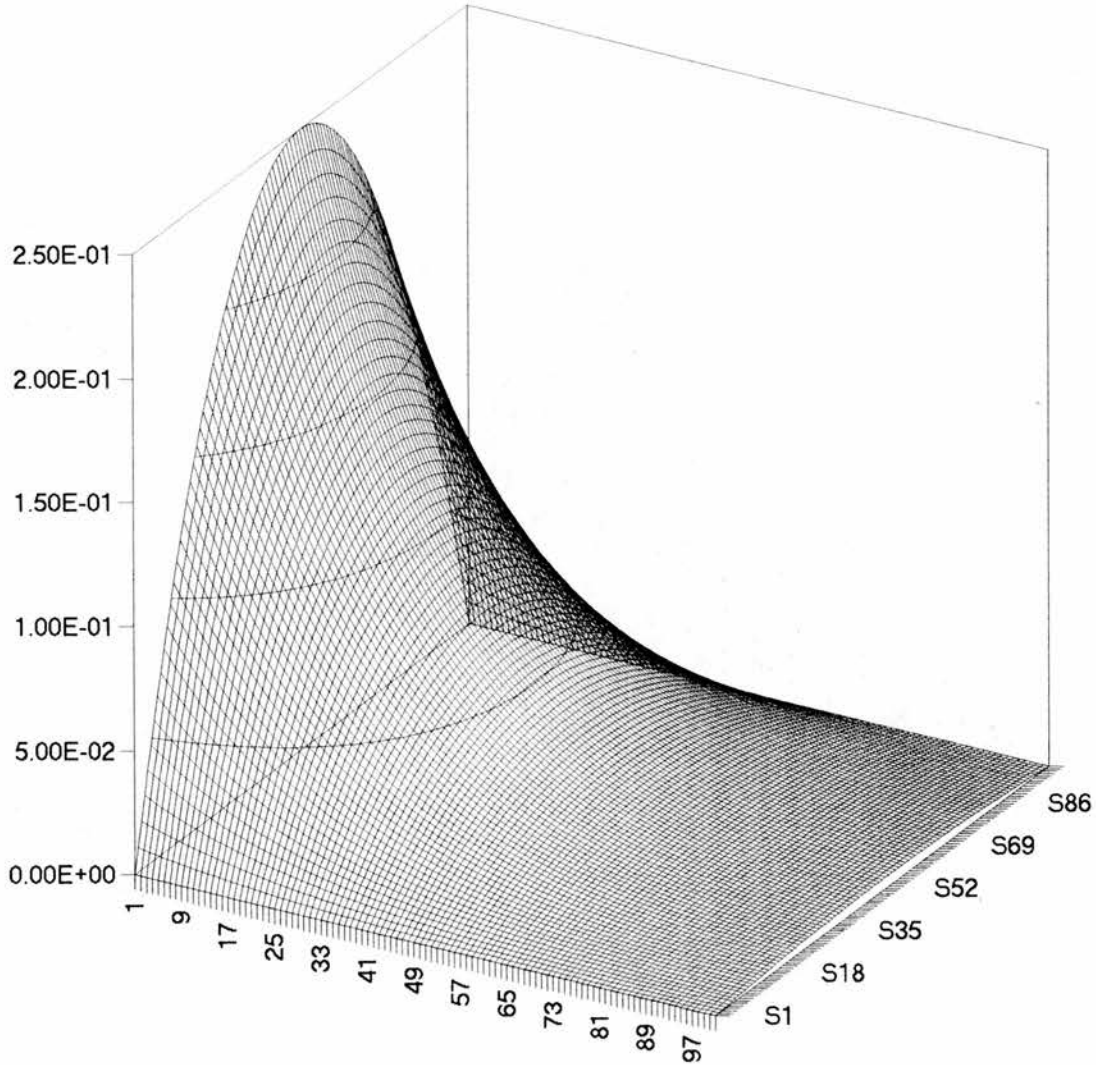
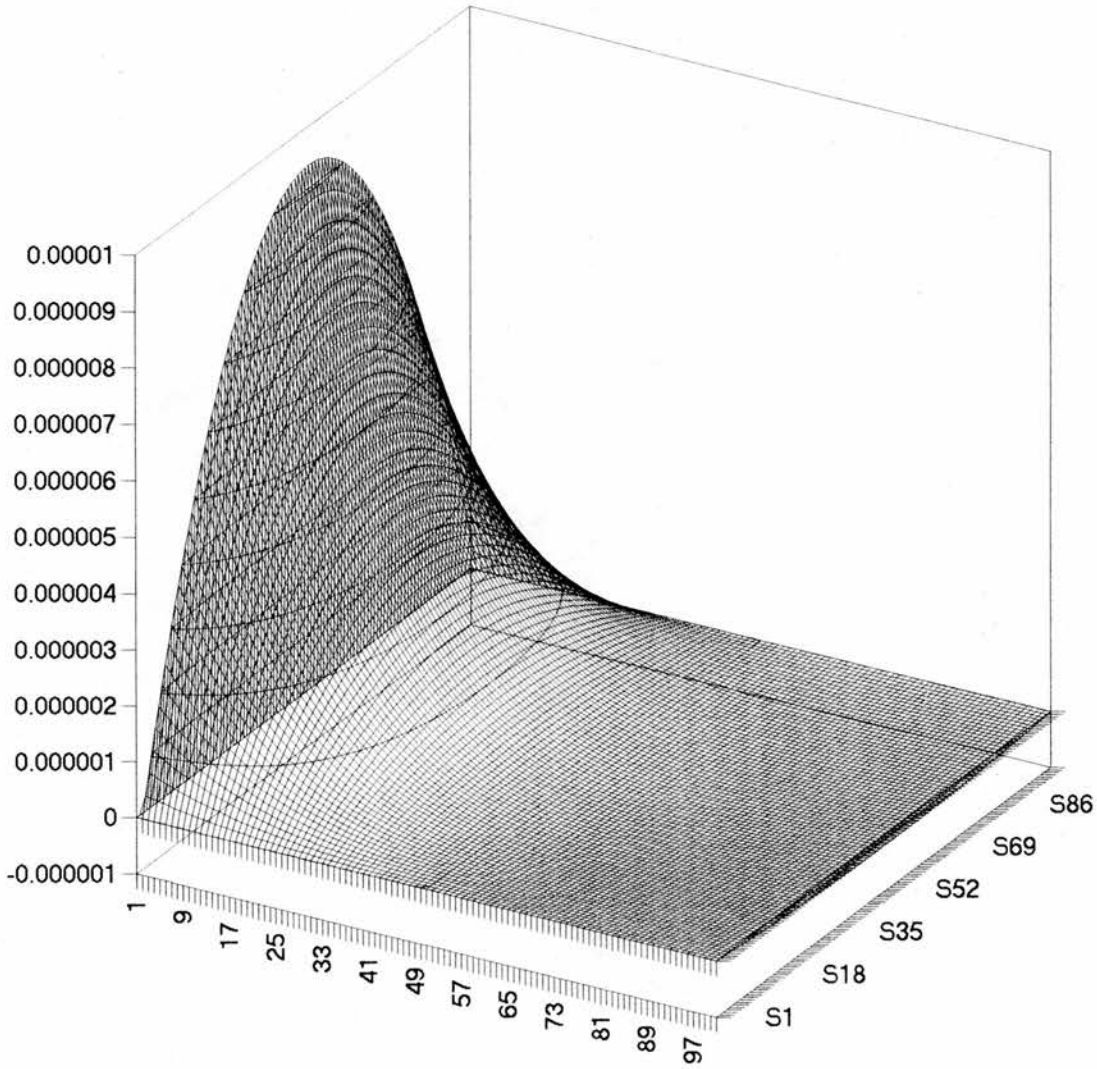


fig 23: 1-D tansient: S.E.M.-Analytical.



Chapter 4: The Multi-Grid Method

The slow rates of convergence of the Gauss-Seidel method (approximately $e^{-\lambda h^2/2}$ where $\lambda = 2\pi^2$) and the fact that doubling grid density will mean a sixteen fold increase in computing time. Even the SOR method (which for Laplace's equation has a convergence rate equal to the square of that of the Gauss-Seidel) has an eight fold computing time increase for a doubling of grid density. (ref. 7). The results in Chapter 1 and 2 indicate just how slow the Gauss-Seidel procedure is when h=0.01.

There is clearly a requirement for a fast numerical method of solving the steady state neutron diffusion equation.

It is a feature of Gauss-Seidel solutions that high frequency error oscillations die out very quickly under iteration. Therefore a high-fidelity solution (on a fine grid) will require relatively few iterations to eliminate the high frequency errors. The lower frequency errors may just as easily be eliminated (smoothed out) on a coarser grid thereby greatly reducing computing time. The idea of using different grid densities to smooth out different frequency components of the error is the basis of the Multi-Grid method.

However the Multi-Grid method is more subtle than simply using a coarse grid to speed things up and then interpolating the results onto a finer grid. The Gauss-Seidel method is basically error driven; i.e. the larger the difference between the 'real' solution and the current numerical approximation then the greater the rate of convergence. The Multi-Grid method also uses this property.

A basic description of the theory behind Multi-Grid (ref. 7: G.E.B. unpublished notes)

Suppose the problem to be solved is

$$\nabla^2 u(x,y) = g(x,y)\,, \qquad (1.0)$$

in some region with u specified on the boundary. Denote an approximation of the equation on a grid of size h by

$$L_h U_h = g_h, \qquad (1.1)$$

After a few iterations using Gauss-Seidel, an estimate to the solution is given by u_h where $U_h=u_h+v_h$ where v_h represents the error in the estimate of U_h. In other words :

u_h =the current computer estimate of the solution

U_h=the ultimate computer estimate of the solution

v_h= a correction term necessary to add to u_h to make up U_h

At this stage only a few iterations will have been performed and consequently the Residual will be significant in size. The residual will be $R_h= L_h u_h - g_h$. Ideally, of course, the Residual should be zero. But because we have been deliberately mean in our use of computer time we find that we have not succeeded in reaching the goal $\nabla^2 u(x,y) - g(x,y) = 0$ and in fact we have a Residual amount R which is not zero but $L_h u_h - g_h$.

From $R_h= L_h u_h - g_h$.

and $L_h u_h+L_h v_h=g_h$, we see that $L_h v_h=g_h- L_h u_h= -R_h$

So the correction term v_h can be found by solving the differential form of the equation

$L_h v_h = -R_h$, (1.2)

where v is zero on the boundary since the boundary conditions are prescribed and therefore error free.

Although R_h and v_h may not be small, they will at least be smooth in that the high frequency errors will have been eliminated. So equation v is solved by solving the differential equation

$$\nabla^2 v^h(x,y) = -R^h(x,y), \quad (1.3)$$

Since the functions v and R are smooth, a good estimate of the correction term v can be obtained by solving the differential equation (1.3) on a grid which is half as dense (spacing 2h), i.e. solve

$$L_{2h} V_{2h}^h = -R_{2h}^h, \quad (1.4)$$

where V_h^{2h} is an estimate of the error v_h obtained on a grid of spacing 2h, and R_{2h}^h are the values of R_h transferred from the fine grid (h) to the coarse grid (2h). The process of estimating the error v_h by solving the differential equation on a grid of size 2h is referred to as making a **coarse grid correction**. The whole process can be extended to several grid densities, in which case one solves for correction terms to the corrections terms. Before and after each coarse grid correction it is necessary to perform a few iterations to smooth out any high frequency errors due to interpolation from coarse to fine grid.

The simplest implementation of the theory is the V-cycle.

Starting on the finest grid G_0, of size h, the procedure is to perform pre-correction smoothing iterations of the grid before making the coarse grid correction by transferring onto a grid G_1 of size 2h. On G_1 further pre-correction smoothing iterations are performed before another coarse grid correction onto grid G_2 of size 4h. This process can be repeated until the grid is so coarse that only one point remains! More realistically, it repeats up to a pre-set value where convergence is so rapid that further coarse grid corrections are not necessary. The corrections are then interpolated back from grid to grid with each grid change accompanied by post correction smoothing iterations on the finer grid. The diagram fig 24 below should help illustrate the process :

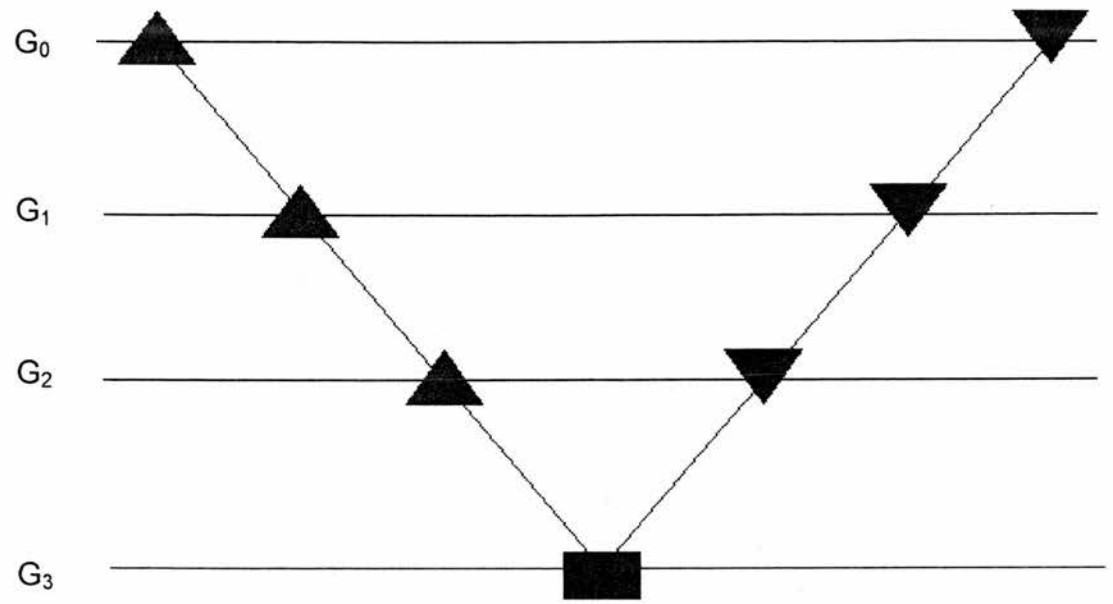


Fig 24 (ref. 7: G.E.B. unpublished notes)

key:

G_0 finest grid

G_1

G_2

G_3 coarsest grid

▲ pre-correction smoothing

▼ post-correction smoothing

Although the Multigrid algorithm may be straightforward in principle, it is not trivial to implement and this can be a major drawback with the method. The implementation given in mg.pas is outlined later in this section. It should be noted that mg.pas was written for diagnostic purposes and is consequently a less elegant implementation than would be the purely functional code. The code could be condensed by increased parameter passing (although this can

incur a cost in computing time) and less extravagant use of memory. In the code mg.pas the use of 20 simultaneous arrays means that over 110000 real numbers are stored simultaneously. This could probably be reduced to perhaps three or four dynamic arrays in a purely functional implementation of the algorithm. The recursive nature of the algorithm would also lend itself to a reduction in the length of the code.

The mg.pas code does have the advantage that it is comparatively easy to follow. Condensed implementations of the algorithm may be difficult to follow, and can incur unexpected computing time penalties. Another point to note is that it is not possible to give a generalised code for any problem. Each problem requires a 'tailor fit' of the Multigrid algorithm. Questions such as

' how may iterations? ' 'how many smoothing sweeps?'

' how best to interpolate?' ' how many grid sizes?' etc are answered by experimentation based on the problem in hand. For example: use of a high number of grid sizes (0..96, 0..48, 0..24, 0..12, 0..6, 0..3) may seem desirable but would be counter productive if the source term S or the α term were to be defined with high fidelity across the finest grid. The fidelity would be lost on the coarser grids. A secondary disadvantage to a high number of grids may be an increase in code complexity.

Implementation of the Multigrid Method

The following is a 'code walk-through' for the program mg.pas listed in Appendix B. The equation for which the code is written is the usual

$$\nabla^2\varphi + \alpha\varphi + S = 0$$

where $\varphi = 0$ on the boundary of a square.

The program implements **one** V cycle and uses three grids:

a course grid $h_1=1/24$

a medium grid $h_2=1/48$

and a fine grid $h_3=1/96$.

To start the iterative process an initial estimate of φ is required on the fine grid and this is generated by exploiting the other two grids. The nomenclature for array names may seem awkward but has the advantage that the name of the array accurately describes the meaning of the array.

Code 'walk-through'

The mg.pas implementation is described below:

Step 1:

All arrays are initialised to zero except the array **source_term** which is defined as the familiar central block as described earlier.

Step 2:

The Gauss-Seidel algorithm is applied to the finite difference approximation of the neutron diffusion equation and is iterated to give u (the current computer solution) on the coarsest grid (0..24). This array is called **u_coarse**.

The Gauss-Seidel algorithm is applied to the finite difference approximation of the neutron diffusion equation and is iterated to give u (the current computer solution) on the coarsest grid (0..24). This array is called **u_coarse**. The resulting profile is not unexpected, having the main flux area in the centre where the source term is positioned.

Step 3:

The array **u_coarse** is interpolated to a medium grid (0..48) and is called **u_medium**. This array is like **u_coarse** but with twice the grid density.

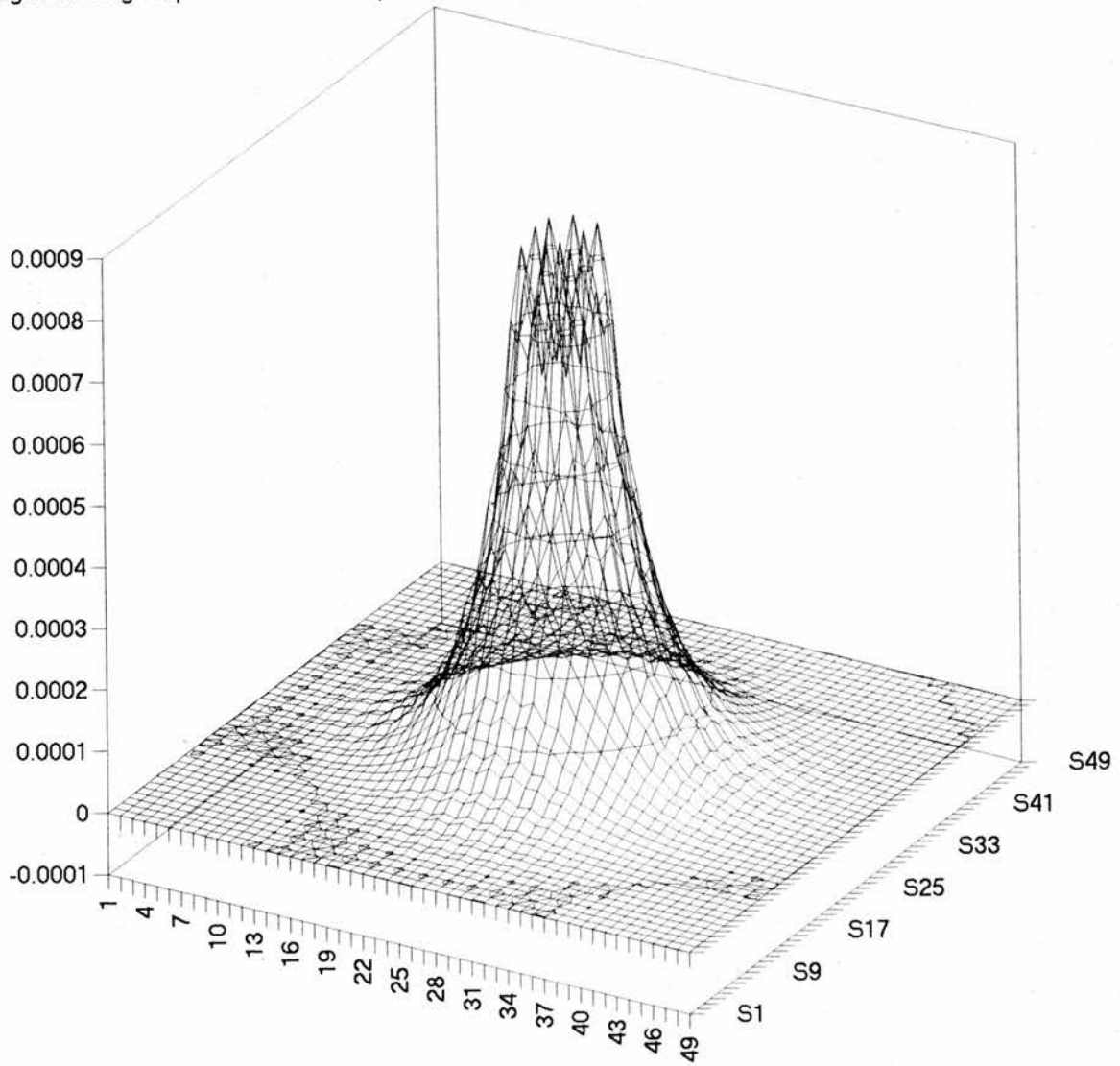
Step 4:

A few Gauss-Seidel iterations are performed on **u_medium** to 'smooth' the data. This eliminates 'noise' due to the interpolation process in step 3. Figure 25 shows the difference between pre-smoothing and post-smoothing. It is not productive to take too many smoothing sweeps since this process will be effectively iterating the algorithm on too fine a grid. The 'error' is due to the fact that the solution is far from convergence and the pre-smoothing solution has over-represented the source term due to the too high grid spacing on the coarse grid.

Step 5:

The array u_medium is interpolated to a fine grid (0..96) and is called **u_medium_to_fine** to emphasise that the array derived from a medium grid.

fig 25: Multigrid; pre-smooth minus post smooth (50 itns) on medium grid



Step 6:

Further Gauss-Seidel iterations are performed on **u_medium_to_fine** is to 'smooth' the data. This eliminates any 'noise' introduced in the interpolation process in step 5. The difference due to smoothing is illustrated in fig 26, and similar comments to those in step 4 apply here. The above processes provide a reasonably good starting solution on which to apply the V-cycle. Fig 27 shows the starting solution.

Step 7: the beginning of theV-cycle

Remembering the initial equation, we have:

$$\nabla^2 U + \alpha U + S = 0$$

where

$U = u + v_1$,

where u is the estimate and v_1 is the required correction.

hence

$$\nabla^2 u + \nabla^2 v + \alpha(u+v) + S = 0$$

$$\nabla^2 u + \alpha u + \nabla^2 v + \alpha v + S = 0$$

giving $\nabla^2 v_1 + \alpha\ v_1 = -R_1$

where

$R_1 \equiv \nabla^2 u + \alpha u + S \neq 0$

The residual R_1 is evaluated from **u_medium_to_fine** and the **source_term** on the fine grid and the resulting array is called **r1**. This non- iterative calculation is described in the routine **get_R1** within the code mg.pas. The residual R_1 on this fine grid is illustrated in fig 28. It is seen that the graph dips sharply in the middle of the grid. This is at first perhaps an unexpected result since it might be thought that if the solution still had some way to go then the residual

fig 26: Multigrid; pre-smooth minus post smooth (200 itns) on fine grid.

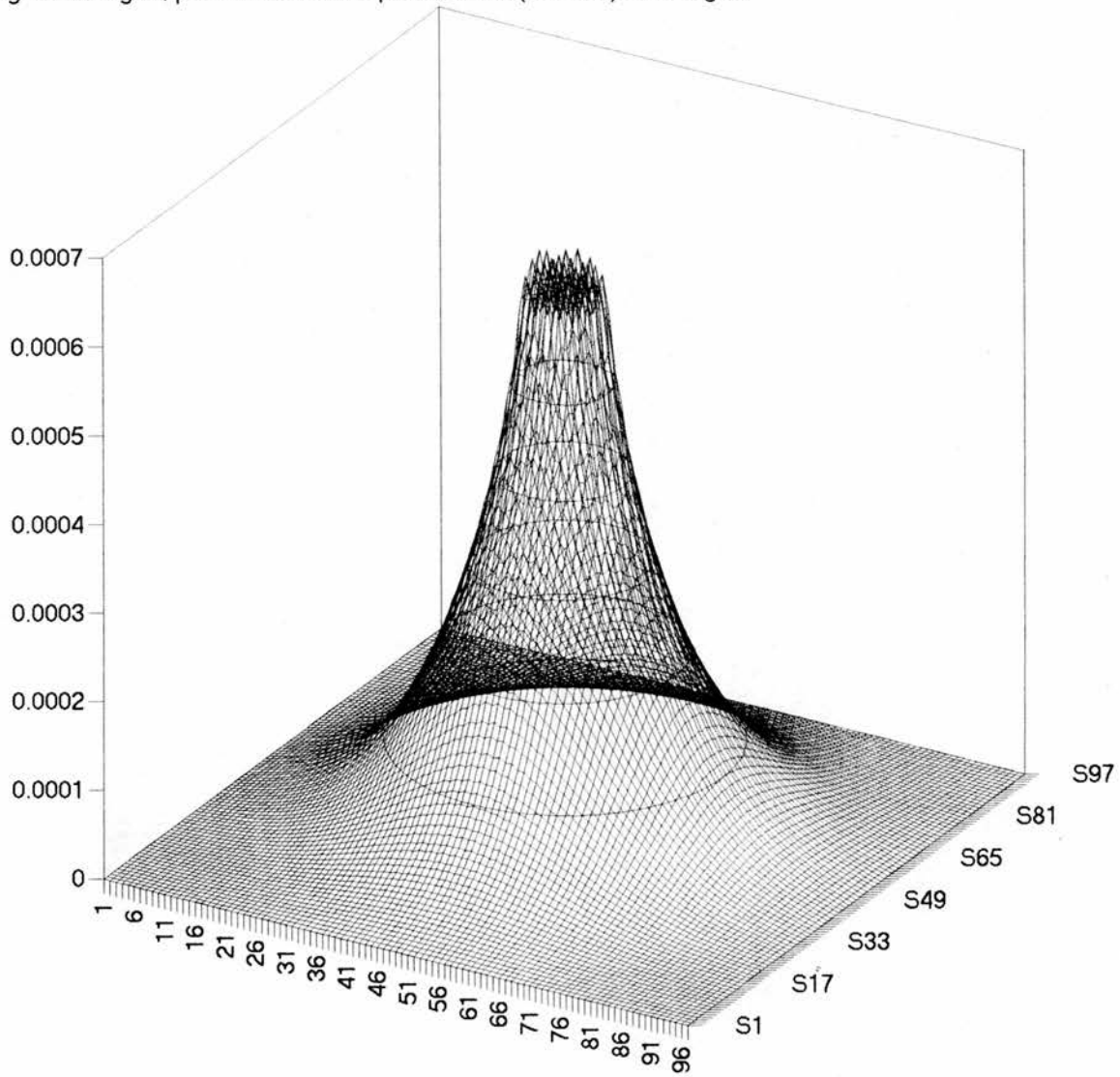


fig 27: array u_medium_to_fine

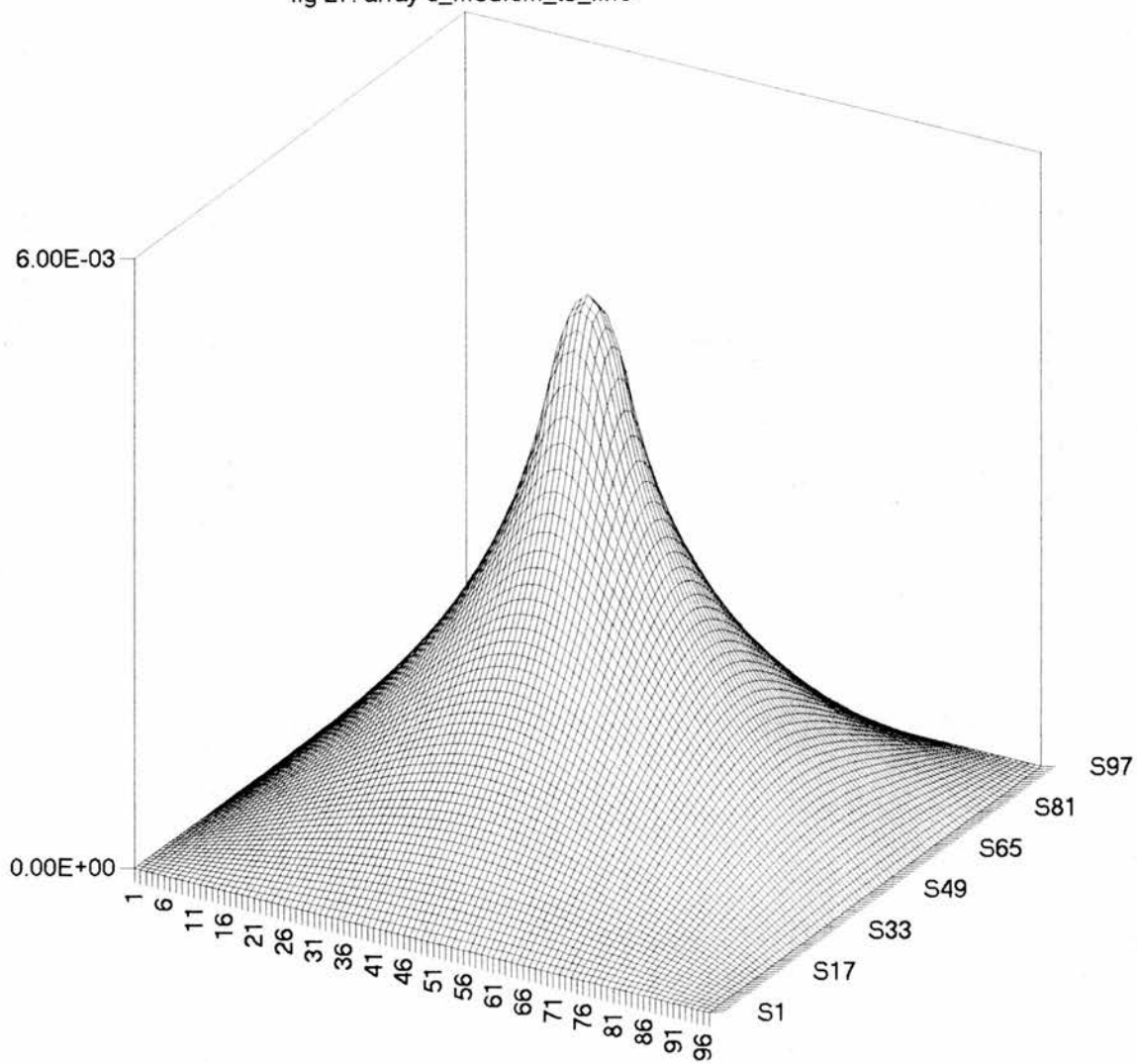
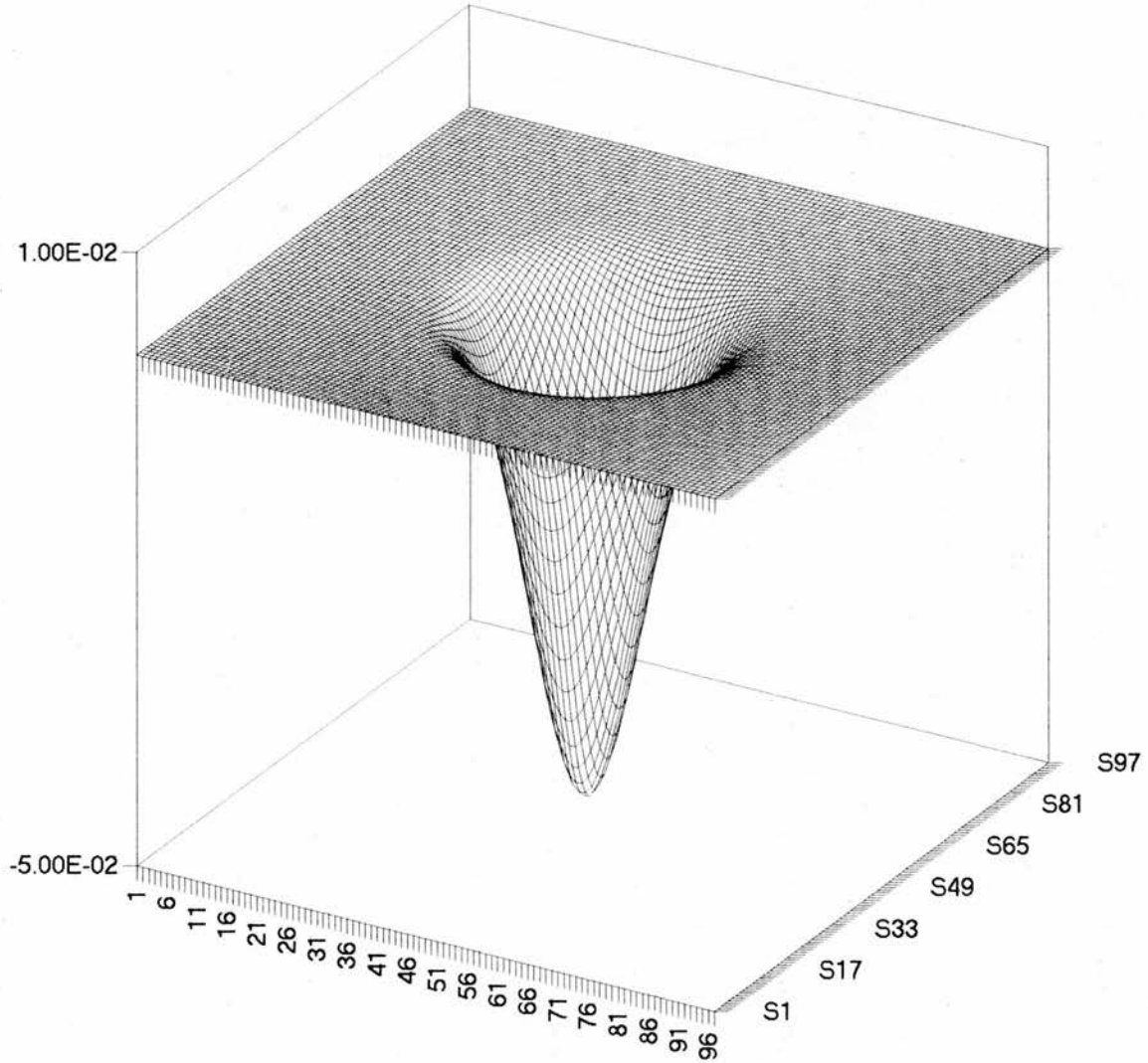


fig 28: array r1



would be positive. However because of the particular construction of our differential equation's source term, we find that the course grid solution has necessarily overset the 'geographic' extent of the effect of the source term.

Step 8:

The array **r1** is transferred to a medium grid (0..48) and is called **R1_medium**. As this is a point to point transfer of data, there are no 'noise' implications. The array **R1_medium** is illustrated in fig 29 and is like fig 28 but now with half the grid density.

Step 9:

The first correction term v_1 is calculated by iteration on the medium grid and the resulting array is called **v1_medium**. This array is illustrated in fig 30 and shows a negative correction for the reasons discussed in step 7.

Following from the mathematics in step 7 we have:

$$\nabla^2 v_1 + \alpha v_1 = -R_1$$

and the algorithm to calculate the first correction term v_1 is

$$v_{m,n} = \frac{\left[v_{m+1,n} + v_{m-1,n} + v_{m,n+1} + v_{m,n-1}\right] + Rh^2}{4 - \alpha h^2}.$$

Step 10:

From step 9 we have an estimate of v_1 called v_1_medium in the program. Let

$v_1 = v_{1m} + v_2$

fig 29: array r1_medium

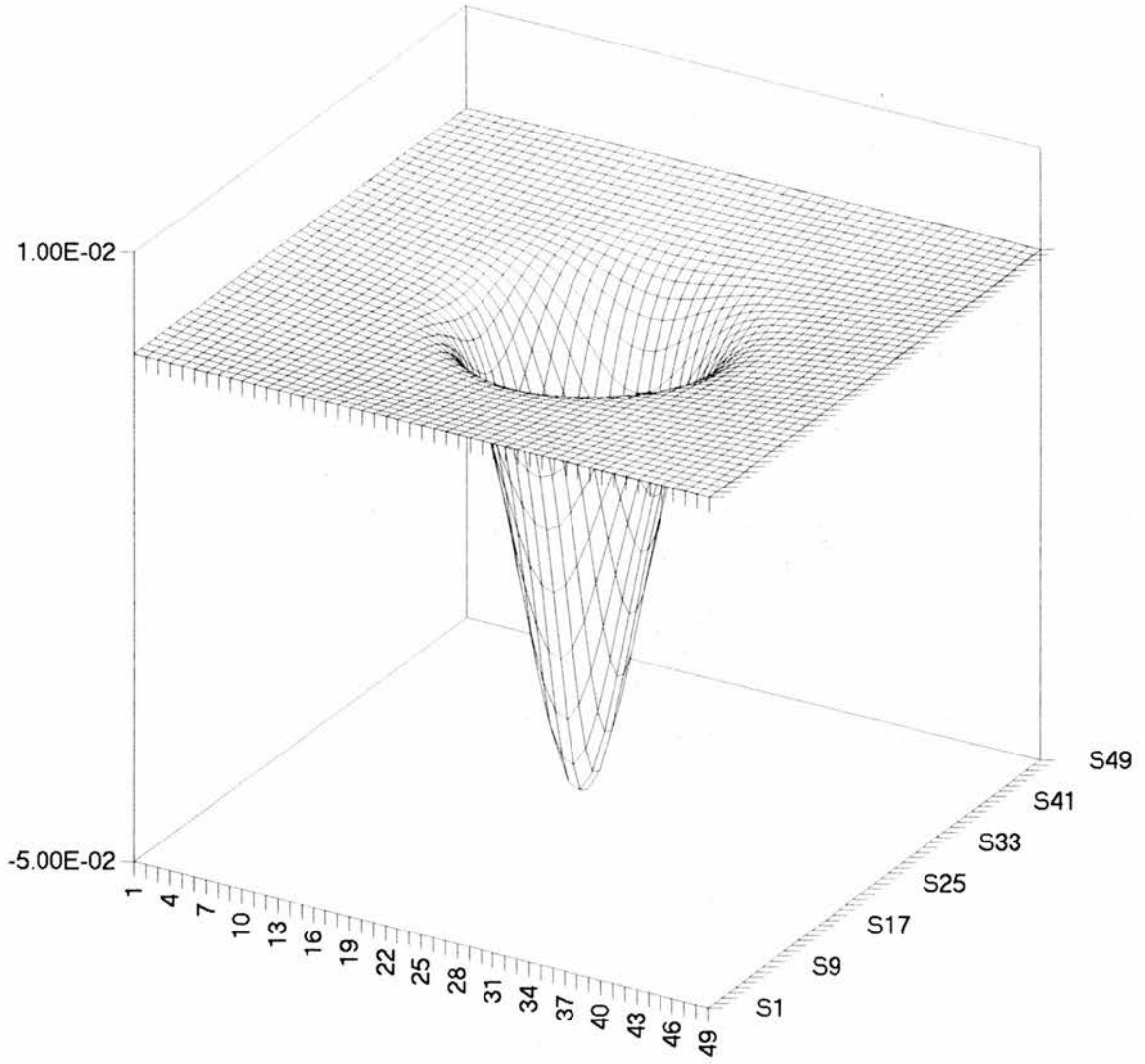
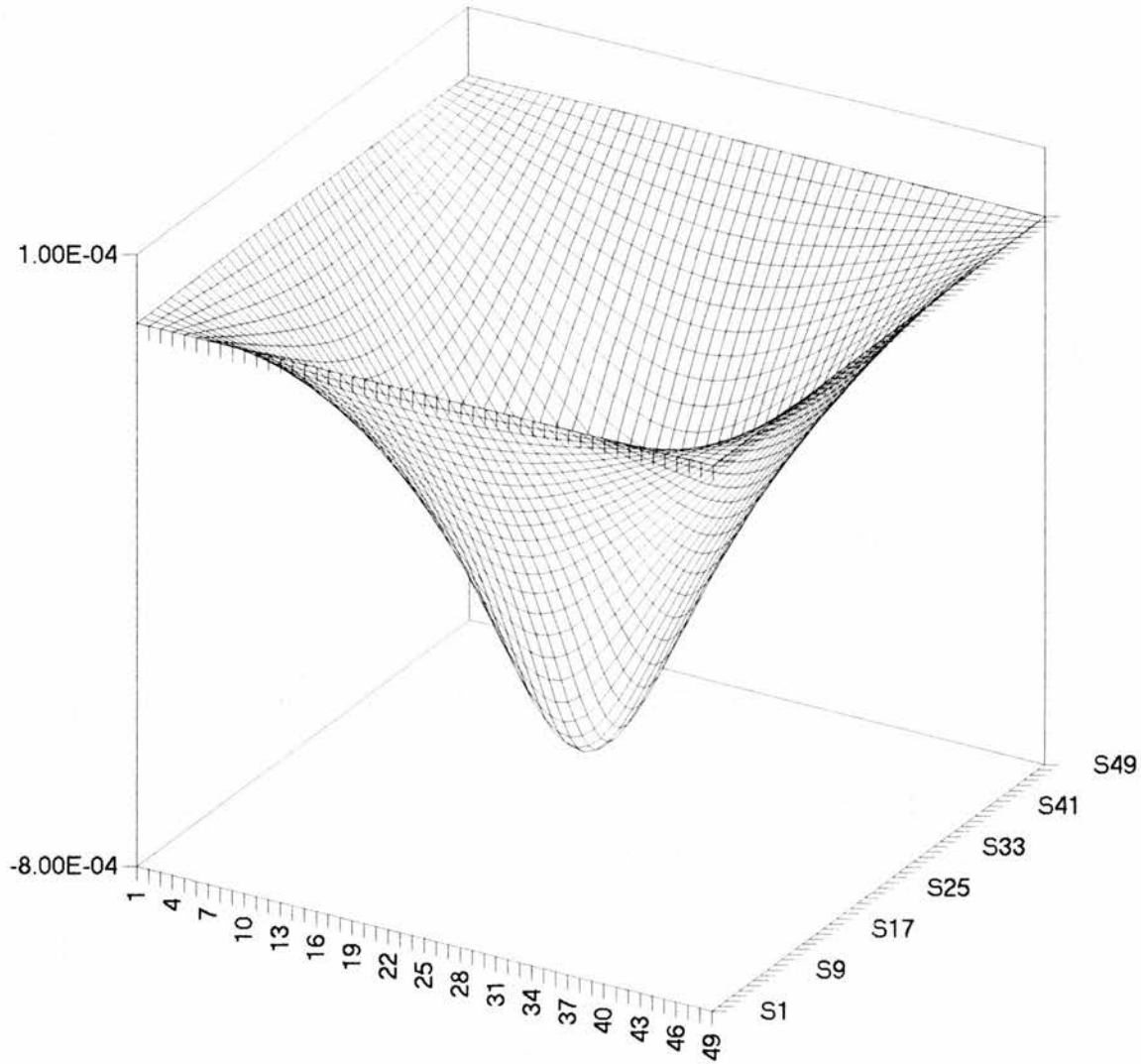


fig 30: array v1_medium



where v_{1m} denotes the estimate of v_1 and v_2 is the new correction required to generate the correct value of v_1. The second residual R_2 is obtained as follows.

$$\nabla^2 v_1 + \alpha v_1 = -R_1.$$

$$\nabla^2 v_{1m} + \nabla^2 v_2 + \alpha v_{1m} + \alpha v_2 = -R_1$$

Or, $\nabla^2 v_2 + \alpha v_2 = -R_2$

where

$$R_2 \equiv \nabla^2 v_{1m} + \alpha v_{1m} + R_1$$

The residual R_2 is evaluated from **R1_medium** and **v1_medium** on the medium grid and the resulting array is called **R2_medium**. The non- iterative calculation is described in the routine **get_R2**. This second residual **R2_medium** has its array illustrated in fig 31 and is still negative for reasons already described.

Step 11:

The array **R2_medium** is transferred to a coarse grid (0..24) and is called **R2_coarse**. Again there are no 'noise' implications. **R2_coarse** is illustrated in fig 32.

Step 12:

The second correction term v_2 is calculated on the coarse grid by approximating the equation

$$\nabla^2 v_2 + \alpha v_2 = -R_2$$

and iterating with Gauss-Seidel. The resulting array is called **v2_coarse**.

fig 31: array r2_medium

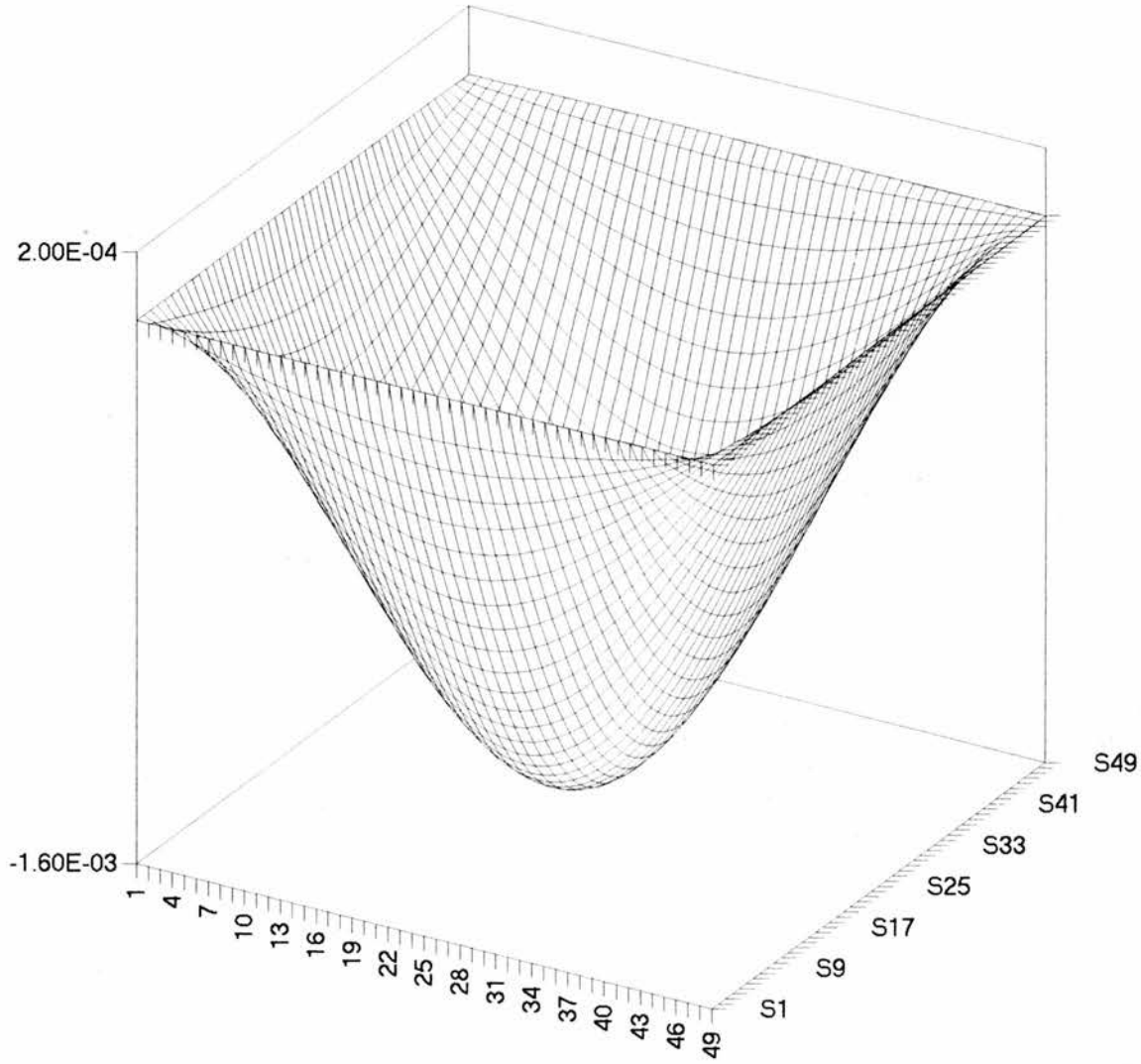
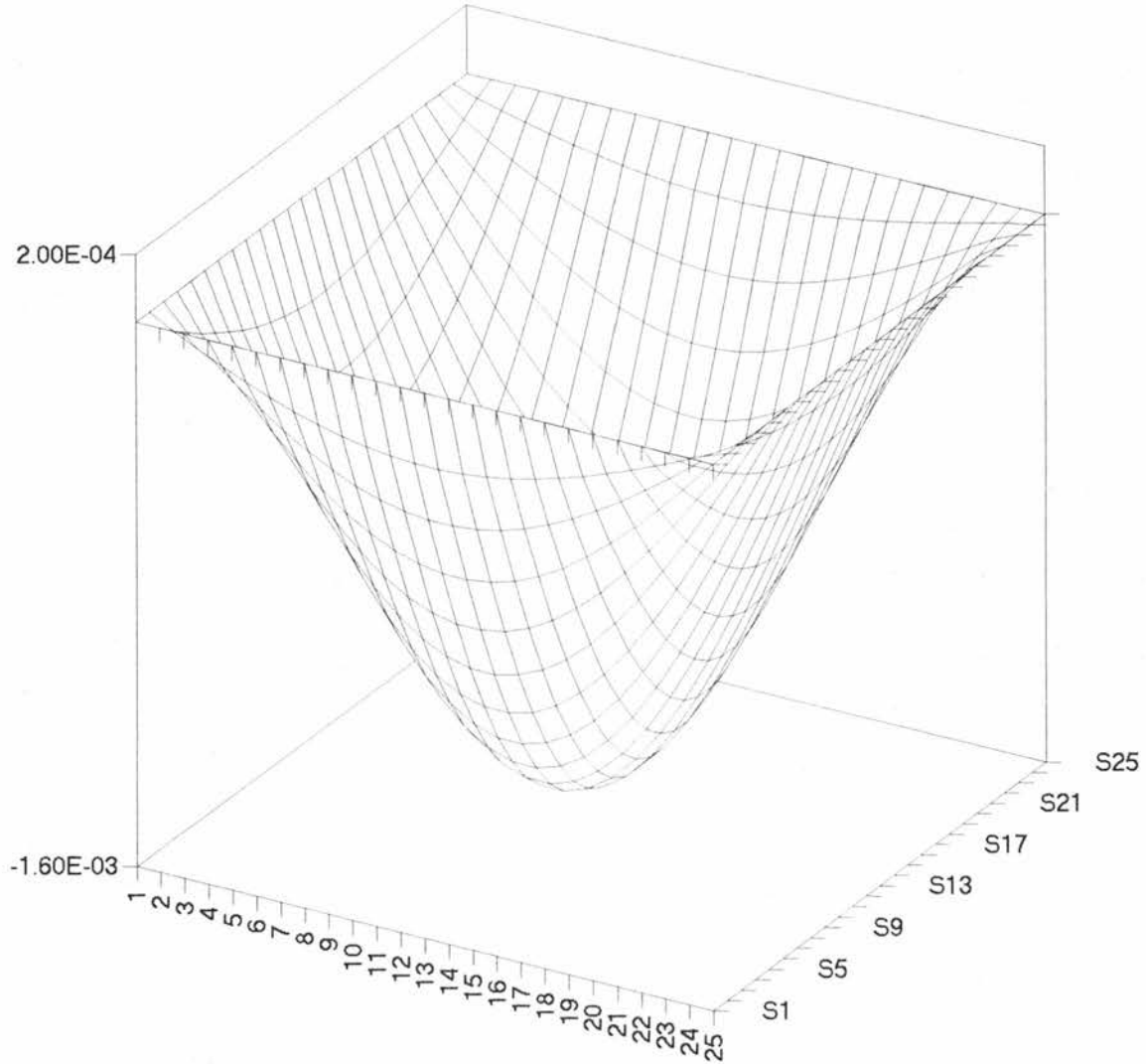


fig 32: array r2_coarse



This array is illustrated in fig 33 where we see the second correction term also negative.

Step 13:

v2_coarse is interpolated to a medium grid and is called **v2_coarse_to_medium** and is illustrated in fig 34.

Step 14:

The two correction terms **v2_coarse_to_medium** and **v1_medium** are added to give a corrected correction term now called **v1_medium_corrected** illustrated in fig 35.

Step 15:

The noise due to interpolation in **v1_medium_corrected** is 'smoothed' out by a few Gauss-Seidel iterations on the approximation for

$\nabla^2 v_1 + \alpha v_1 = -R_1$.

Figures 36a and 36b demonstrate the effects of smoothing on this correction term. Fig. 35a is **v1_medium_corrected** before smoothing - **v1_medium_corrected** after 10 smoothing iterations. Fig. 36b is **v1_medium_corrected** before smoothing - **v1_medium_corrected** after 50 smoothing iterations. We can see that there is some high frequency error evident in Fig. 36a, but fig. 36b shows the high frequency error much more clearly since now we have 50 iterations and so 'rough-smooth' is likely to much more pronounced. Note also that, as expected, the overall level of the error has dropped in Fig. 36b.

Step 16:

v1_medium_corrected is interpolated to the fine grid and the new array is called **v1_medium_corrected_to_fine**.

Step 17:

The original fine grid computer estimate called **u_medium_to_fine** in the code and denoted by u in step 7 is corrected by adding the **v1_medium_corrected_to_fine** to give current best computer solution on a fine grid.

Step 18:

This new estimate of **u** is smoothed and illustrated in fig 37.

fig 33: array v2_coarse

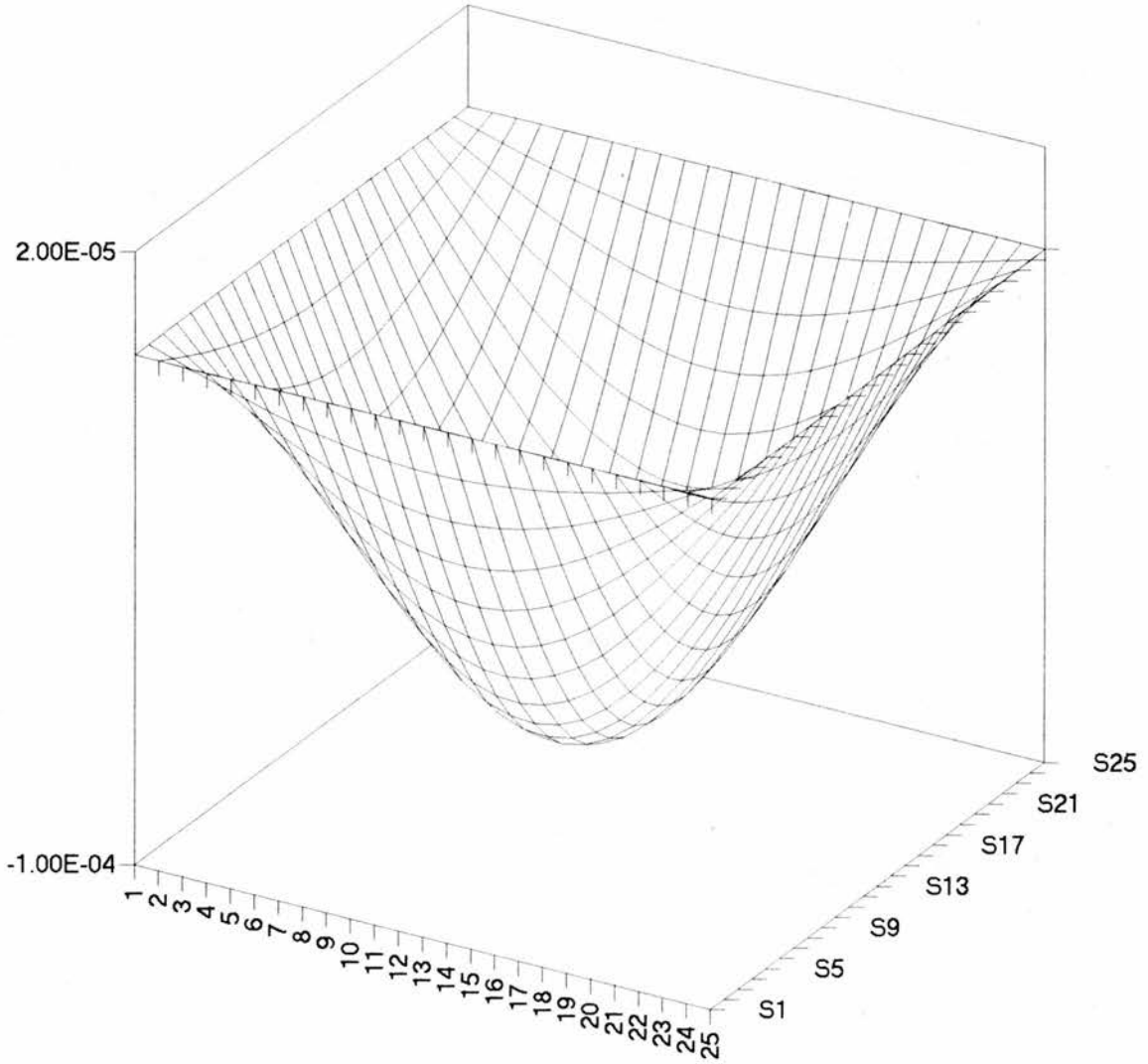


fig 34: array v2_coarse_to_medium

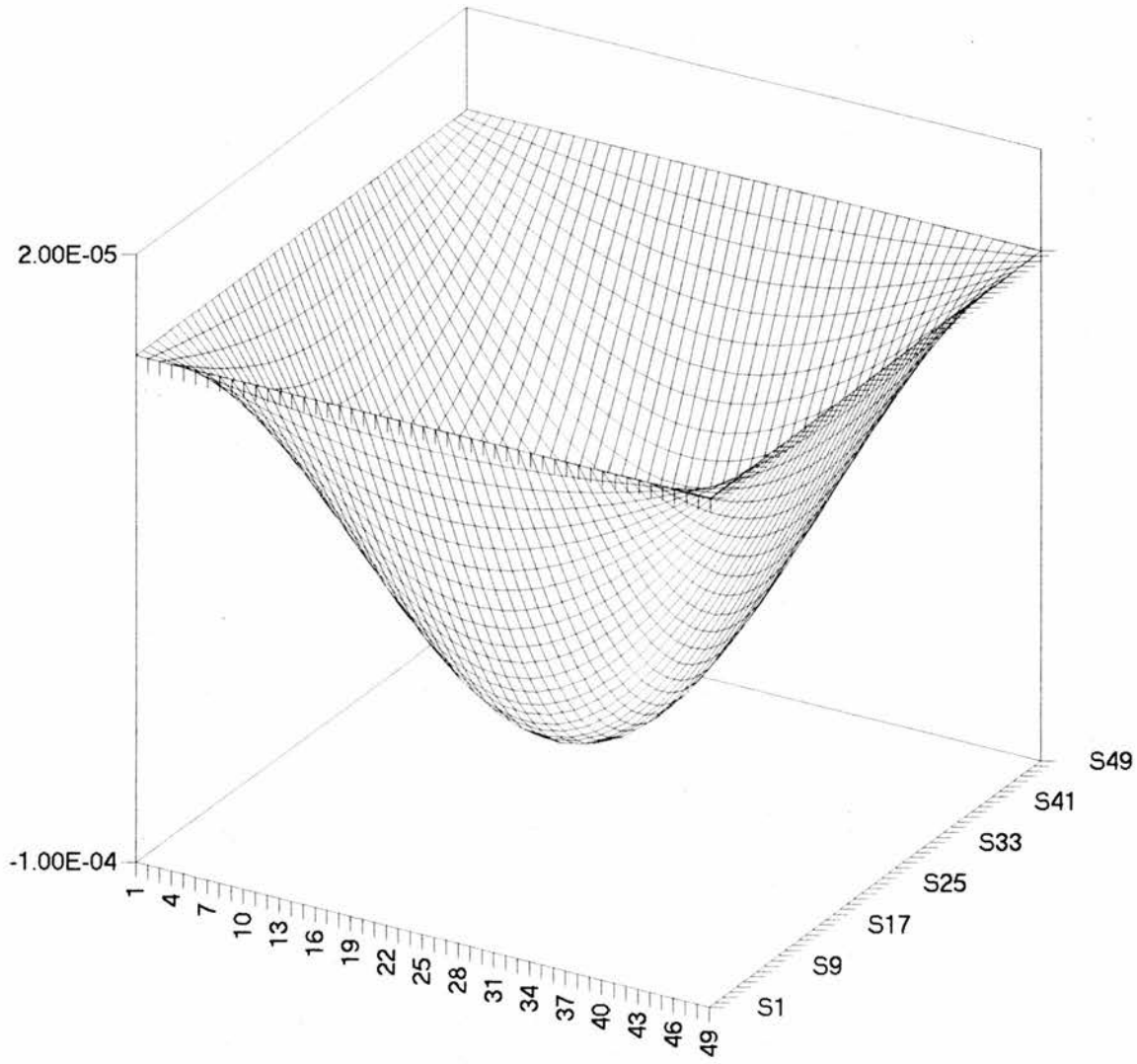


fig 35: array v1_medium_corrected

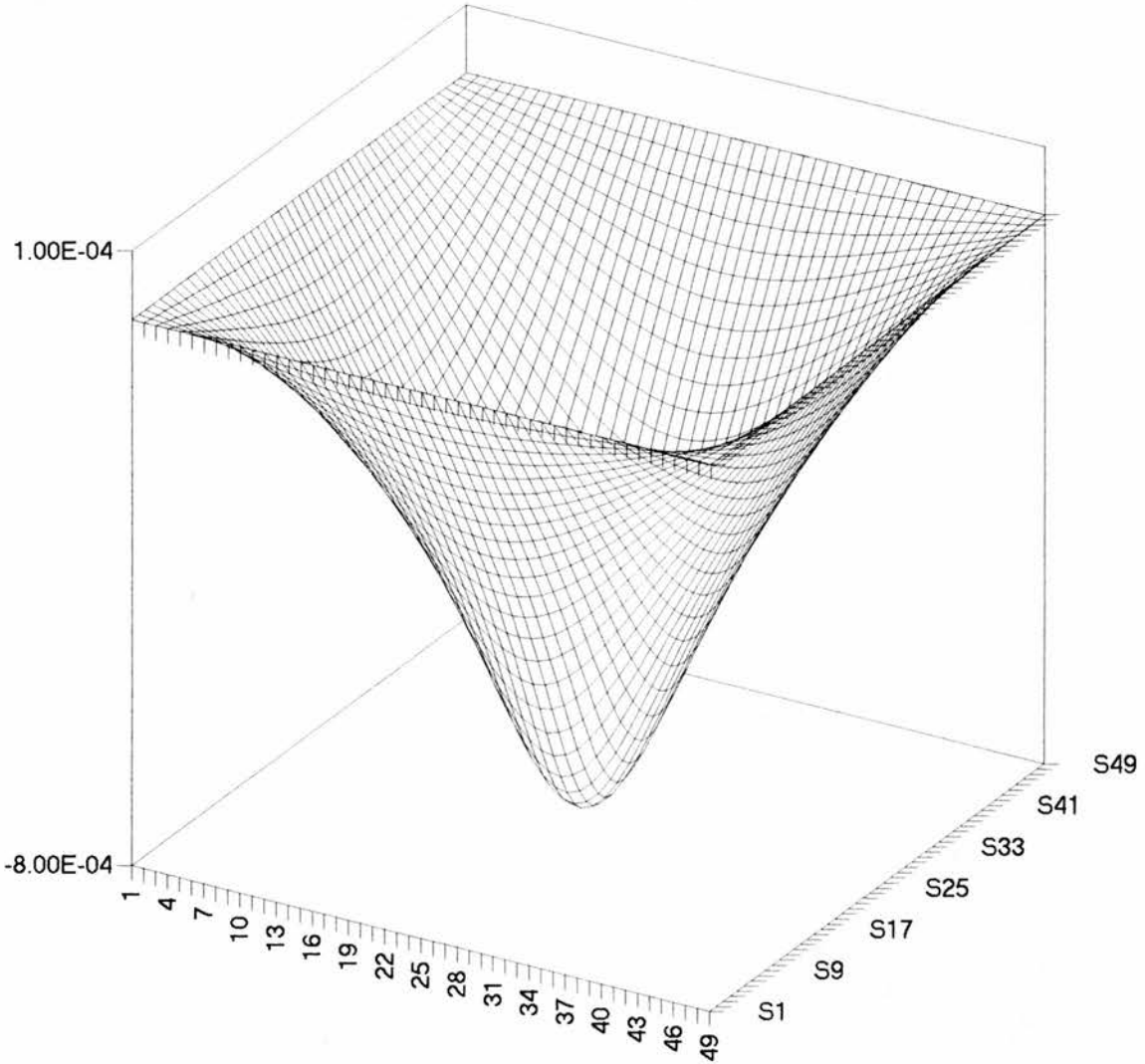


fig. 36a: v1_medium_corrected; before-after smoothing with 10 iterations.

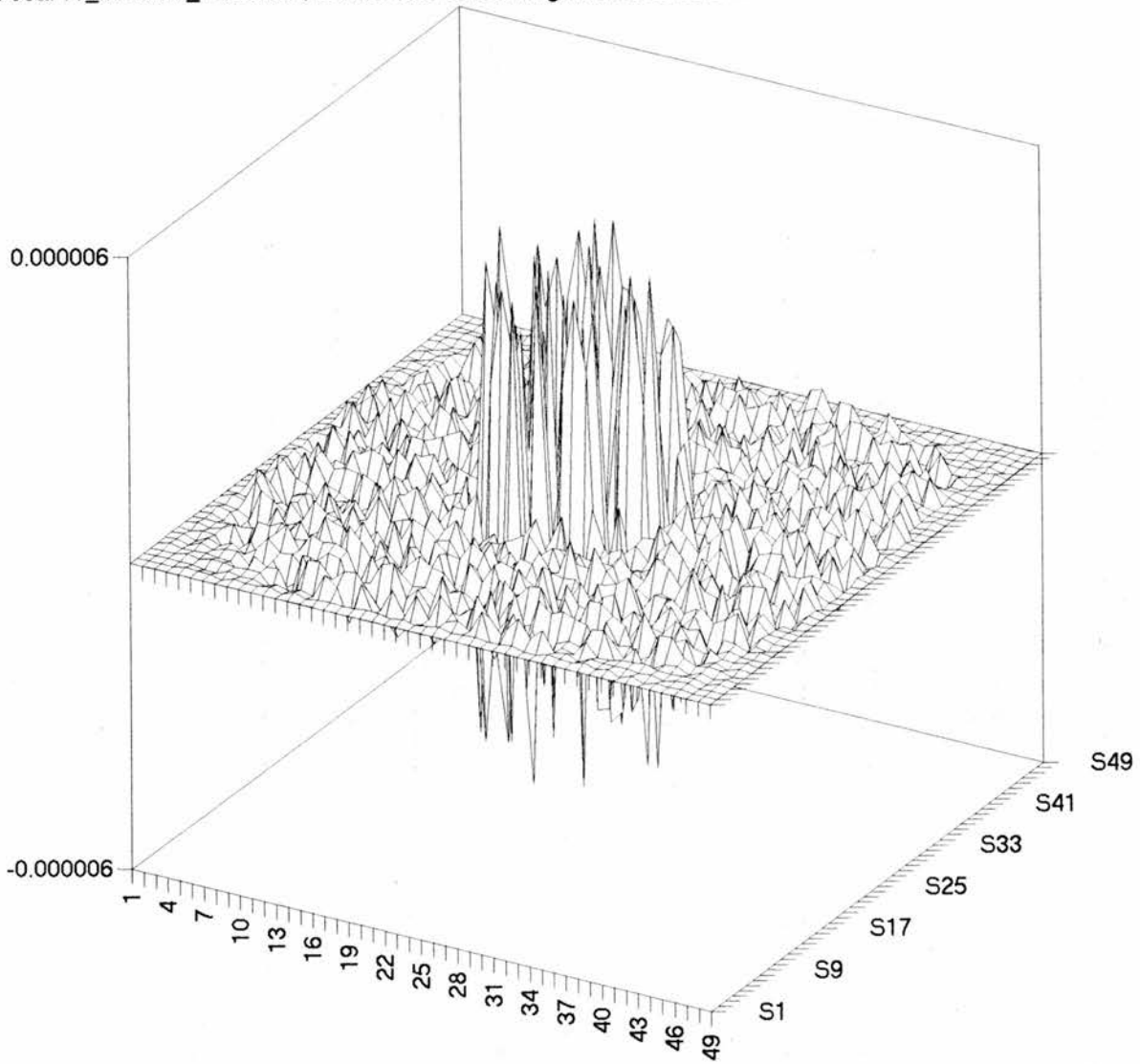


fig. 36b:v1_medium_corrected; before-after smoothing with 50 iterations.

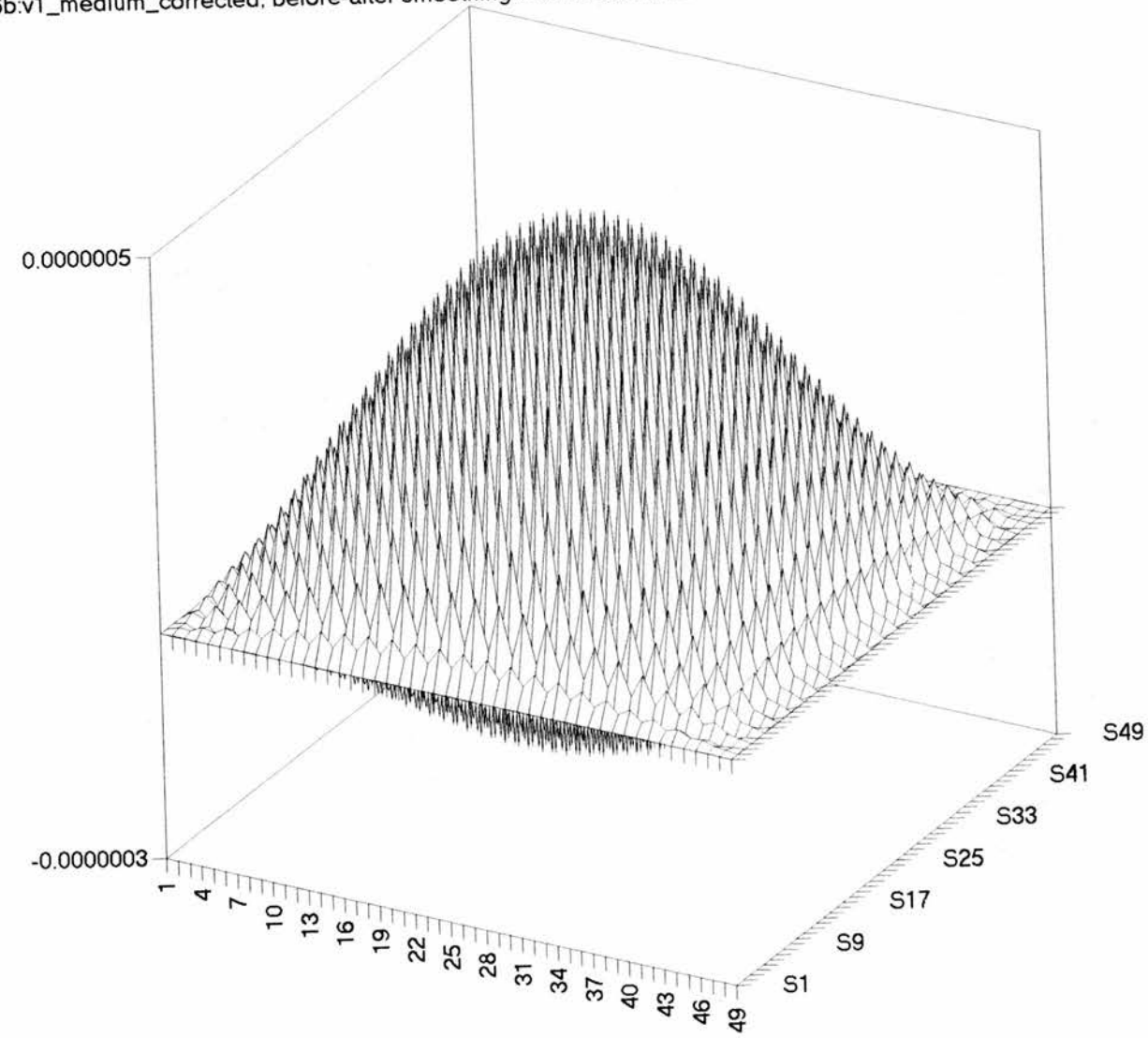
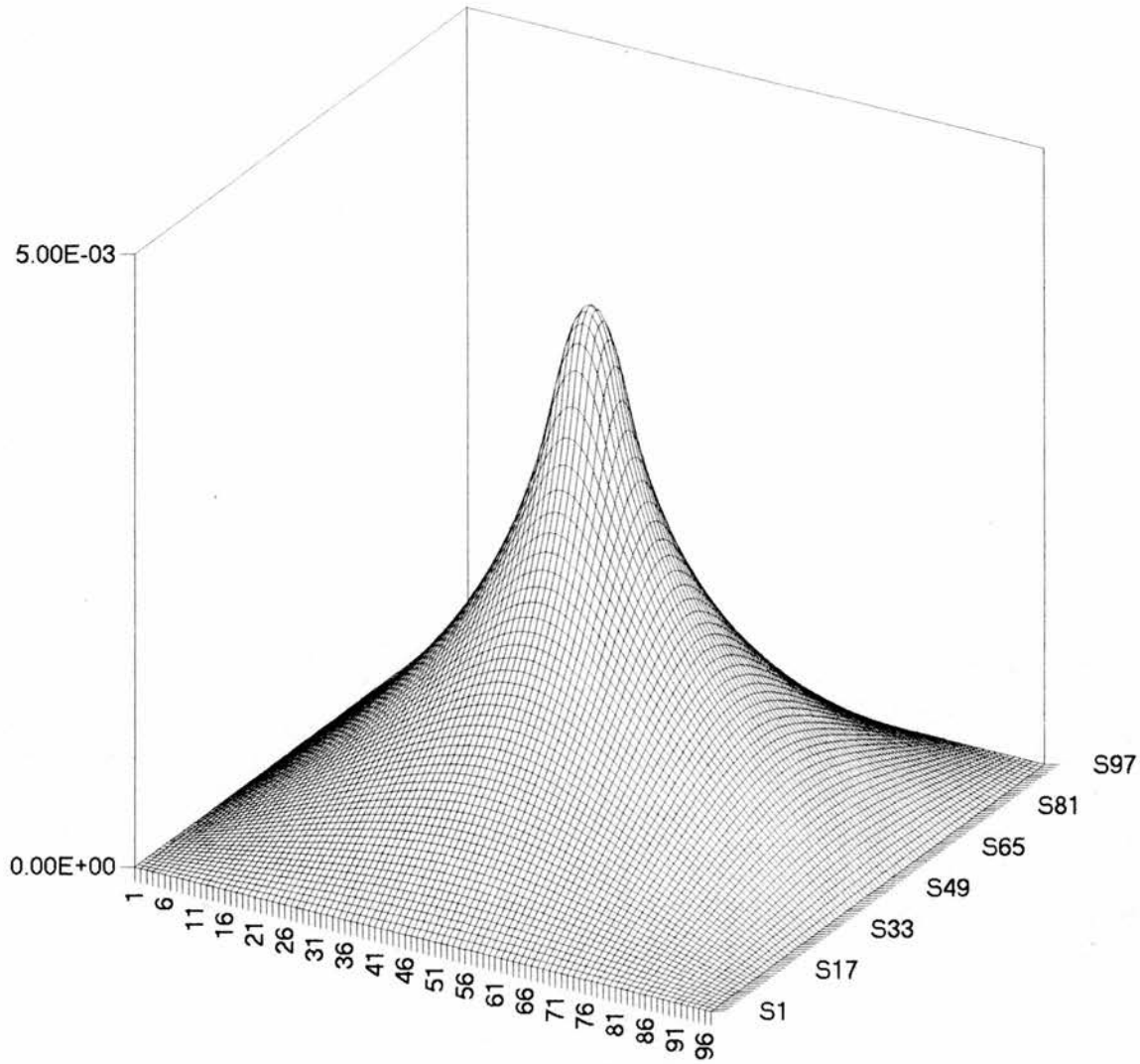


fig 37: array u



Convergence Results

An estimate of the rate of convergence was obtained by comparing the solution to a highly accurate solution of the equation. This 'exact' solution was obtained by using a standard Gauss-Seidel method and allowing it to run for 100,000 iterations; a sufficiently high number to be considered complete convergence, with results given in fig 38. Allowing the Gauss-Seidel and Multigrid methods each to approach 0.1% of the fully converged solution we then compare the Gauss-Seidel and Mulitigrid solutions in fig 39. Our 66Mhz PC running Gauss-Seidel took 720 seconds. The Multigrid method took 40 seconds. Without any fine-tuning, the Multigrid method appears to be 20 times faster than the Gauss-Seidel method. In summary:

	computing time 66mhz PC (seconds)	Maximum error (absolute)	Maximum error (%)
GAUSS-SEIDEL	720	3.58×10^{-7}	0.1%
MULTI-GRID	40	3.58×10^{-7}	0.1%

fig 38: Gauss-Seidel solution

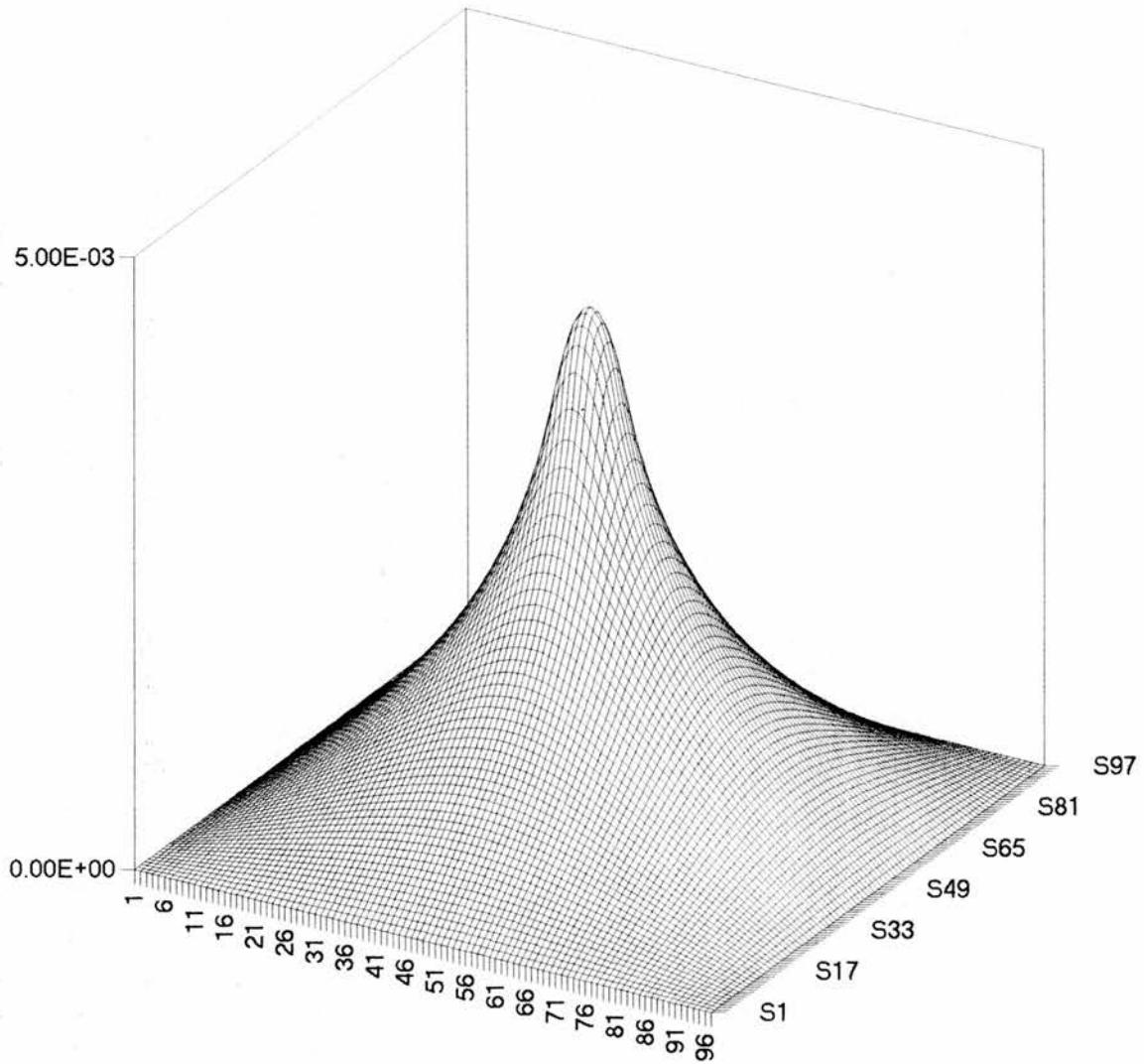
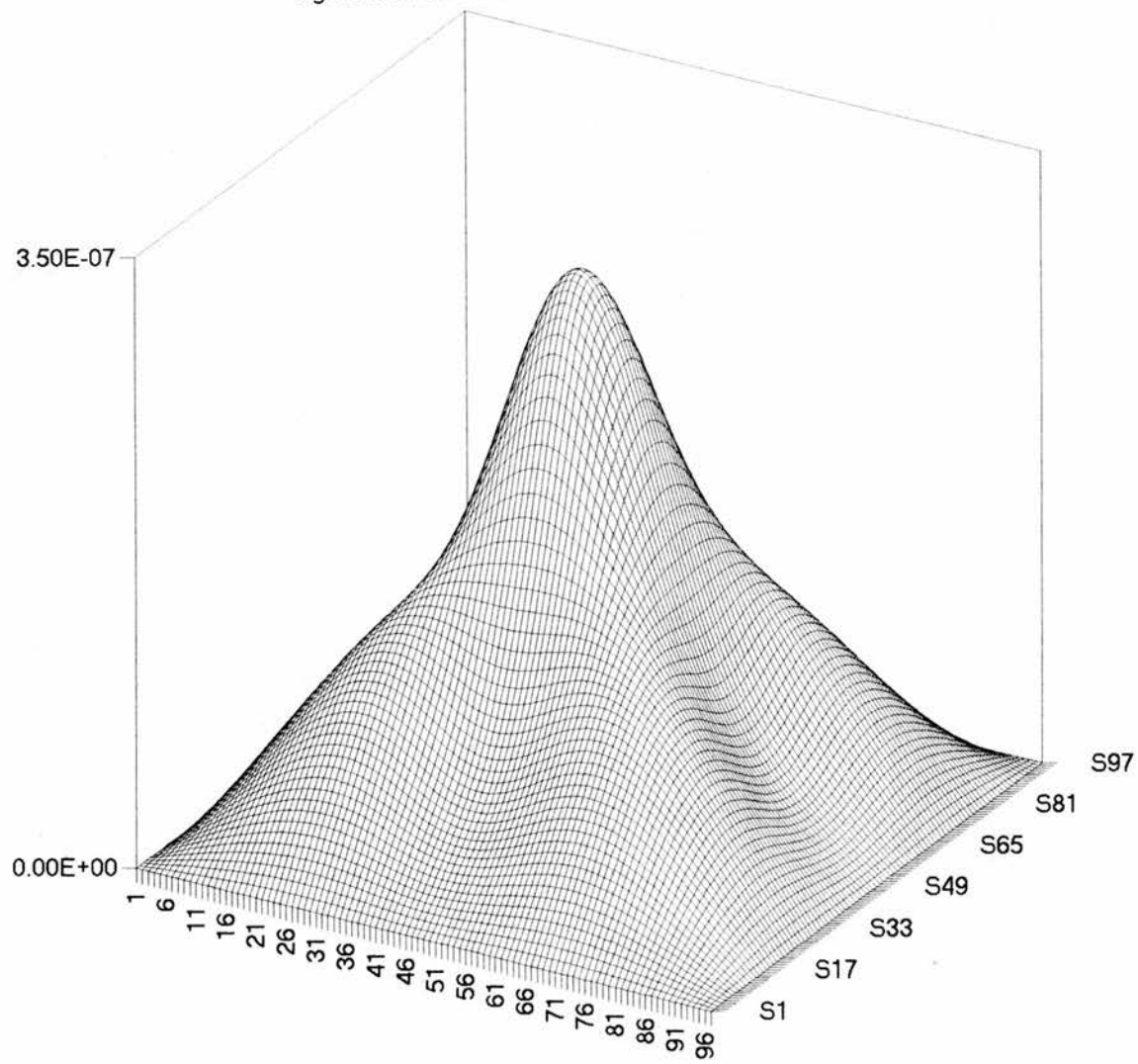


fig39: MultiGrid-GaussSeidel

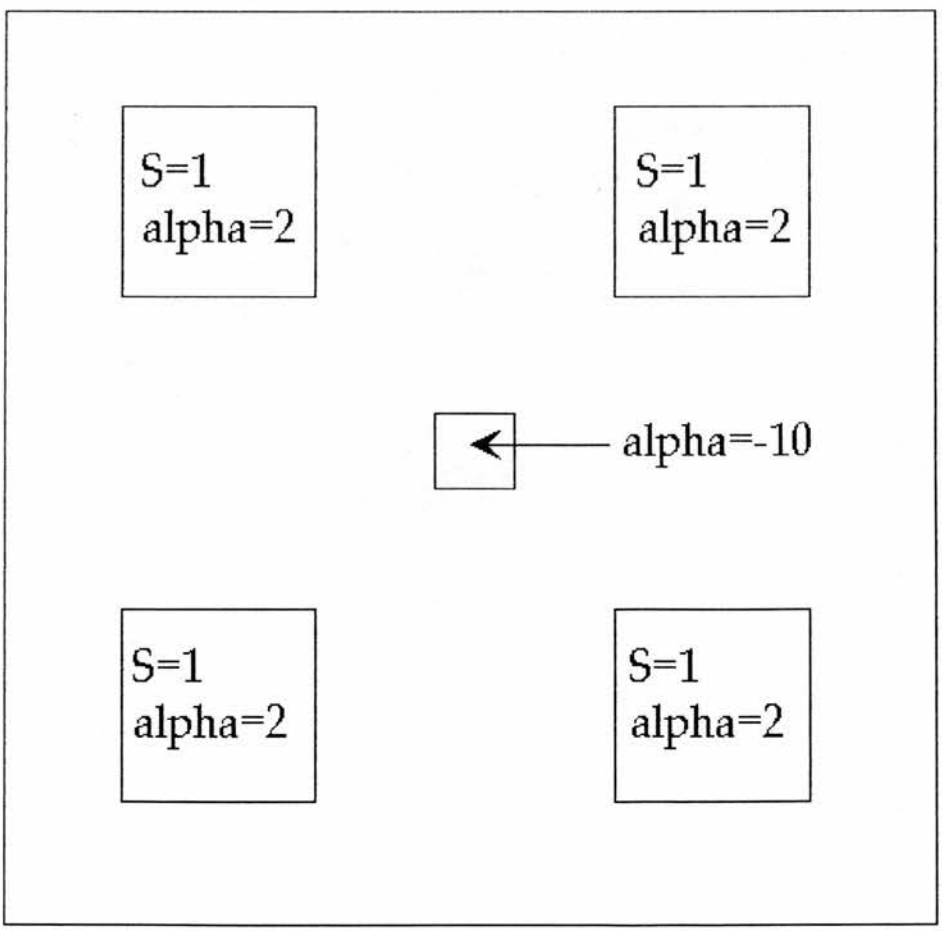


Chapter 5

Implementation of the Multi-Grid method to an improved Physical Model.

Having now developed a code which converges rapidly, it is appropriate to improve the physical model and consider a slightly more realistic situation. The previous model had a single neutron fuel rod which was modelled by a source term S(x,y) which was zero everywhere except near the centre where S(x,y) was unity.

The new model has the characteristics shown below:



Here, we can see that four fuel systems surround an absorbing control rod. The fuel systems have both S and α active since they will provide an independent source of neutrons S and the neutrons from the fissile effect α. The control rod will absorb neutrons at a rate proportional to the neutron flux in its vicinity, but will not have a pure 'sink' term S since the rod cannot absorb neutrons which may not be there. The code described in chapter 4 is easily modified to take account of the new geometry.

The results are given in graph form in fig. 40. A convergence test shows that the standard Gauss-Seidel algorithm reached within about 0.2% of a fully converged solution (obtained by iterating Gauss-Seidel 30,000 times). This took 24 minutes. The Multigrid took 1 minute to achieve this same degree of precision. The error graph is also shown in fig. 41, illustrating the difference between that Multigrid result and the 30,000 iteration Gauss-Seidel result. The maximum error can be seen to be 0.00002 or about 0.2%. In summary:

	computing time 66mhz PC (seconds)	Maximum error (absolute)	Maximum error (%)
GAUSS-SEIDEL	1440	$3x10^{-4}$	0.2
MULTI-GRID	60	$3x10^{-4}$	0.2

Thus, the efficiency of Multigrid is clearly apparent, even for the more complex problem.

The effects of varying the parameters.

Having developed an efficient procedure for calculating $\varphi(x,y)$ in a more complex geometry it is now feasible to consider the effects of varying some of the parameters in the model. For example the extent to which the control rod suppresses the maximum neutron flux is of interest. A series of runs were carried out with different values of α for the control rod. These are illustrated in figures 40 to 44 where we see that the effect of varying the central value of α is not significant (probably because the geometric extent of the central rod is so small) to the calculated value of φ until the central value of α is made a high neutron absorbing value of -100 (fig. 42). We can see that the central hole is becoming much deeper and the overall flux level is down as expected. The central 'hole' remains even when α is zero because the flux production is so active in the regions of the fuel rods.

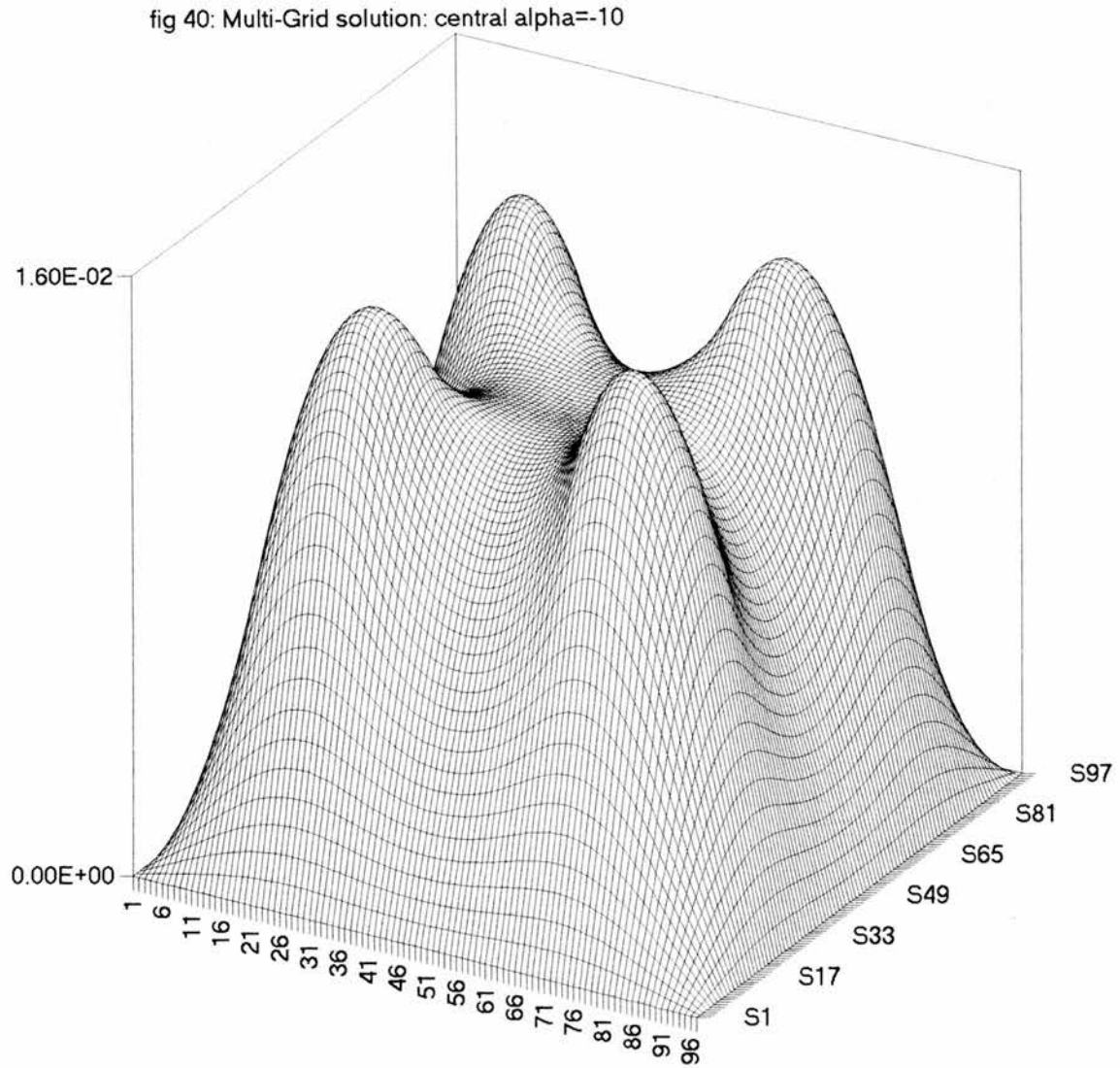
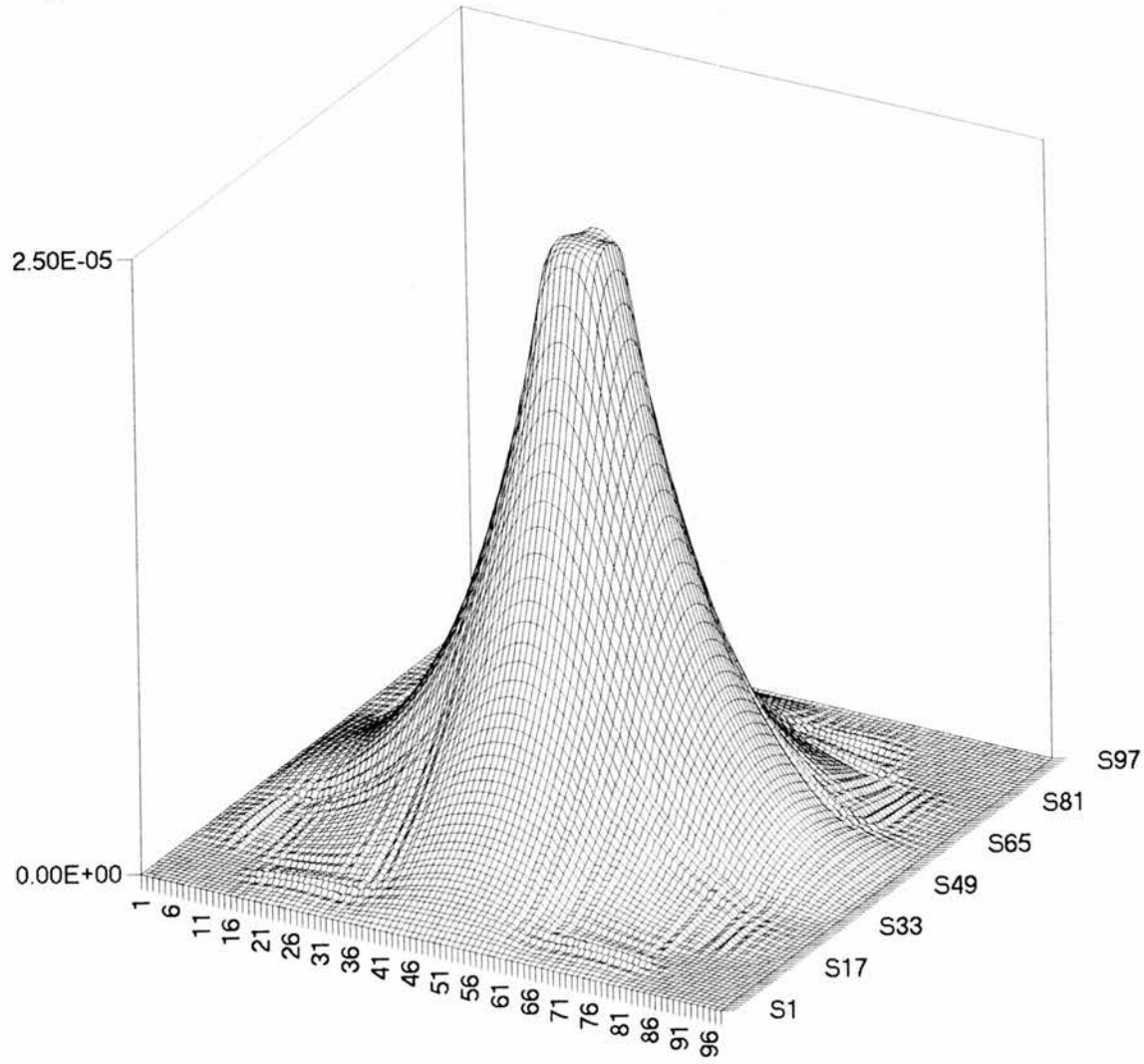


fig 40: Multi-Grid solution: central alpha=-10
1.60E-02
0.00E+00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
S1
S17
S33
S49
S65
S81
S97

fig 41: error; difference between MultiGrid and Gauss-Seidel



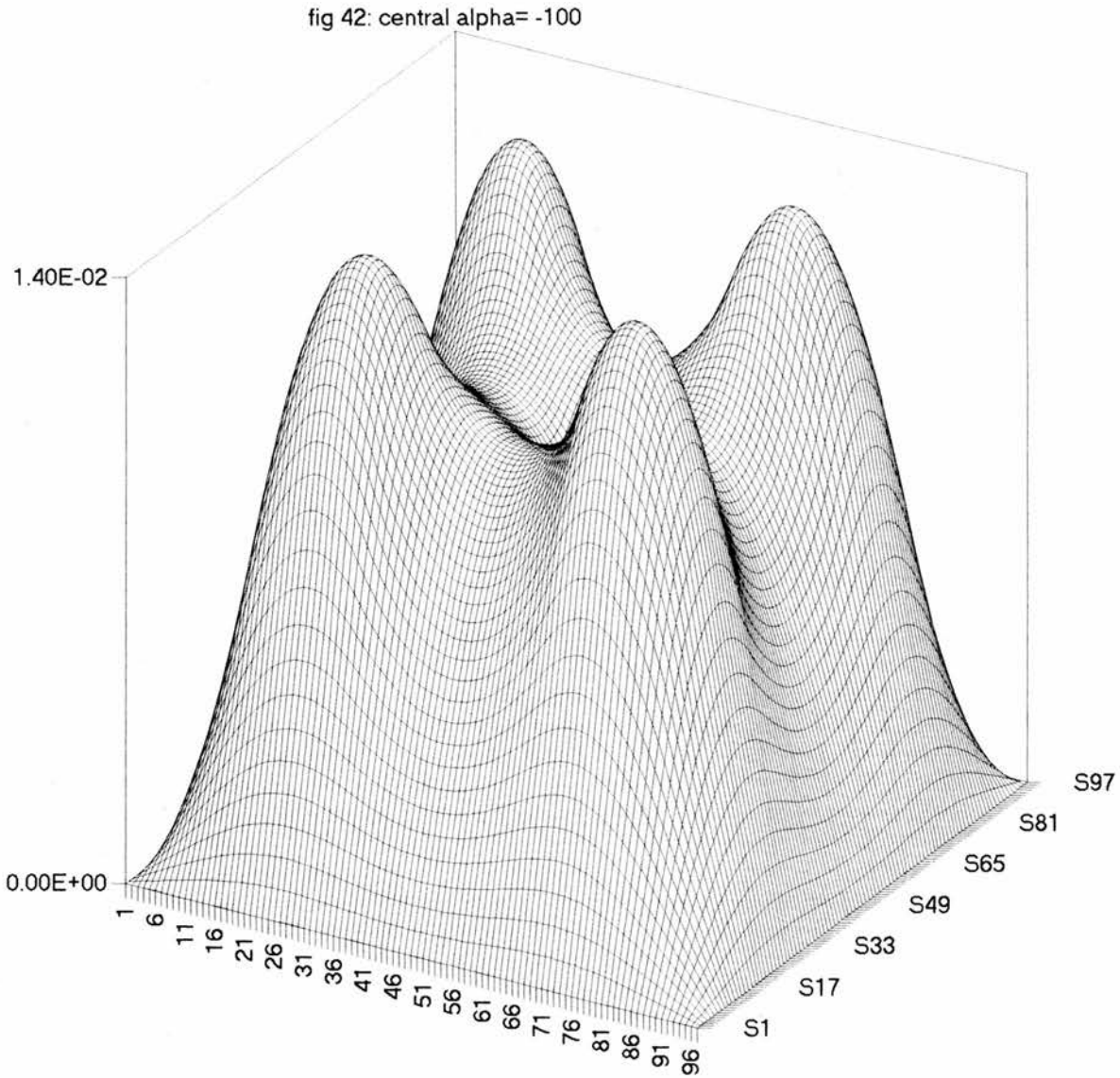
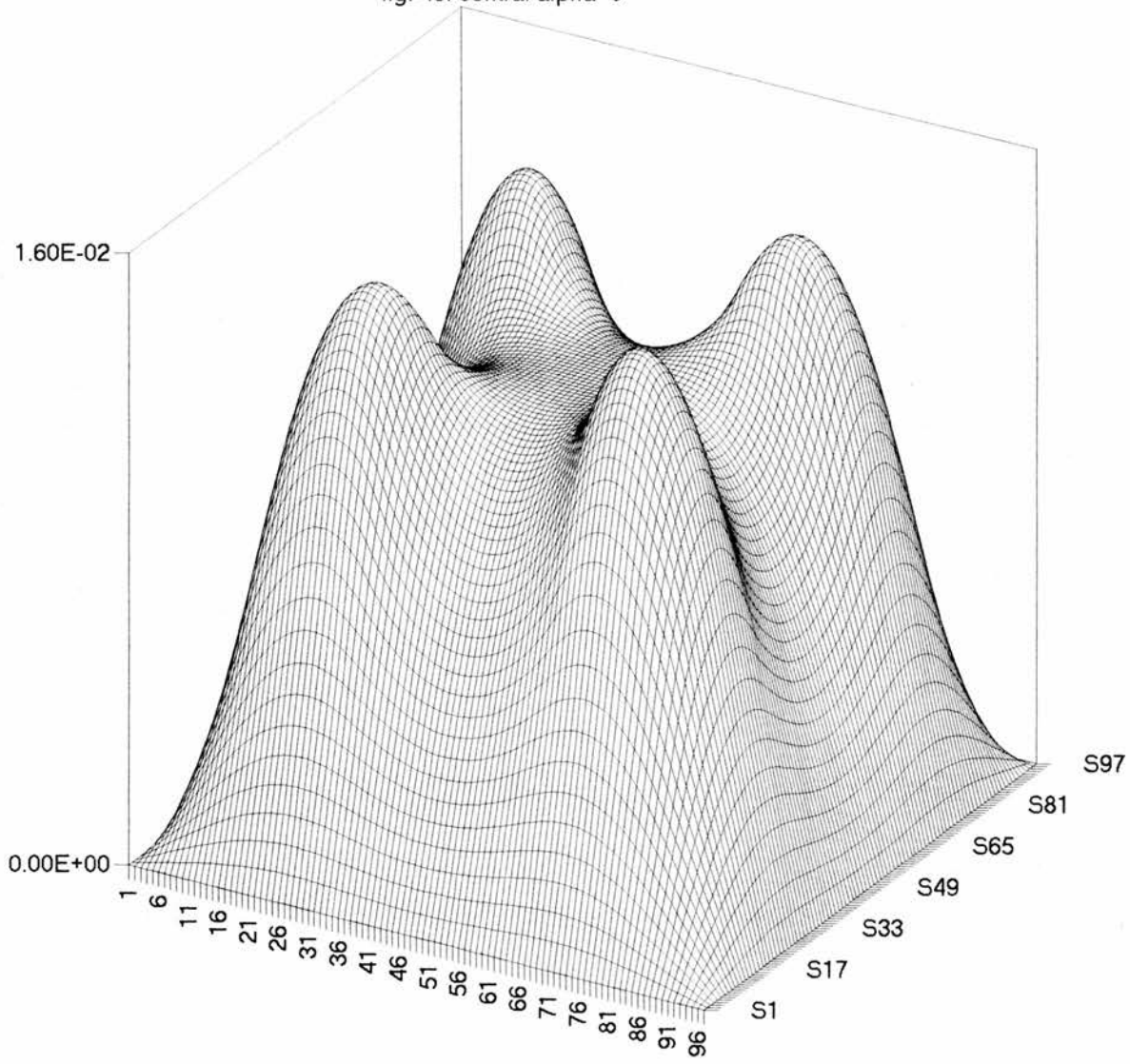


fig 42: central alpha= -100
1.40E-02
0.00E+00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
S1
S17
S33
S49
S65
S81
S97

fig. 43: central alpha=0



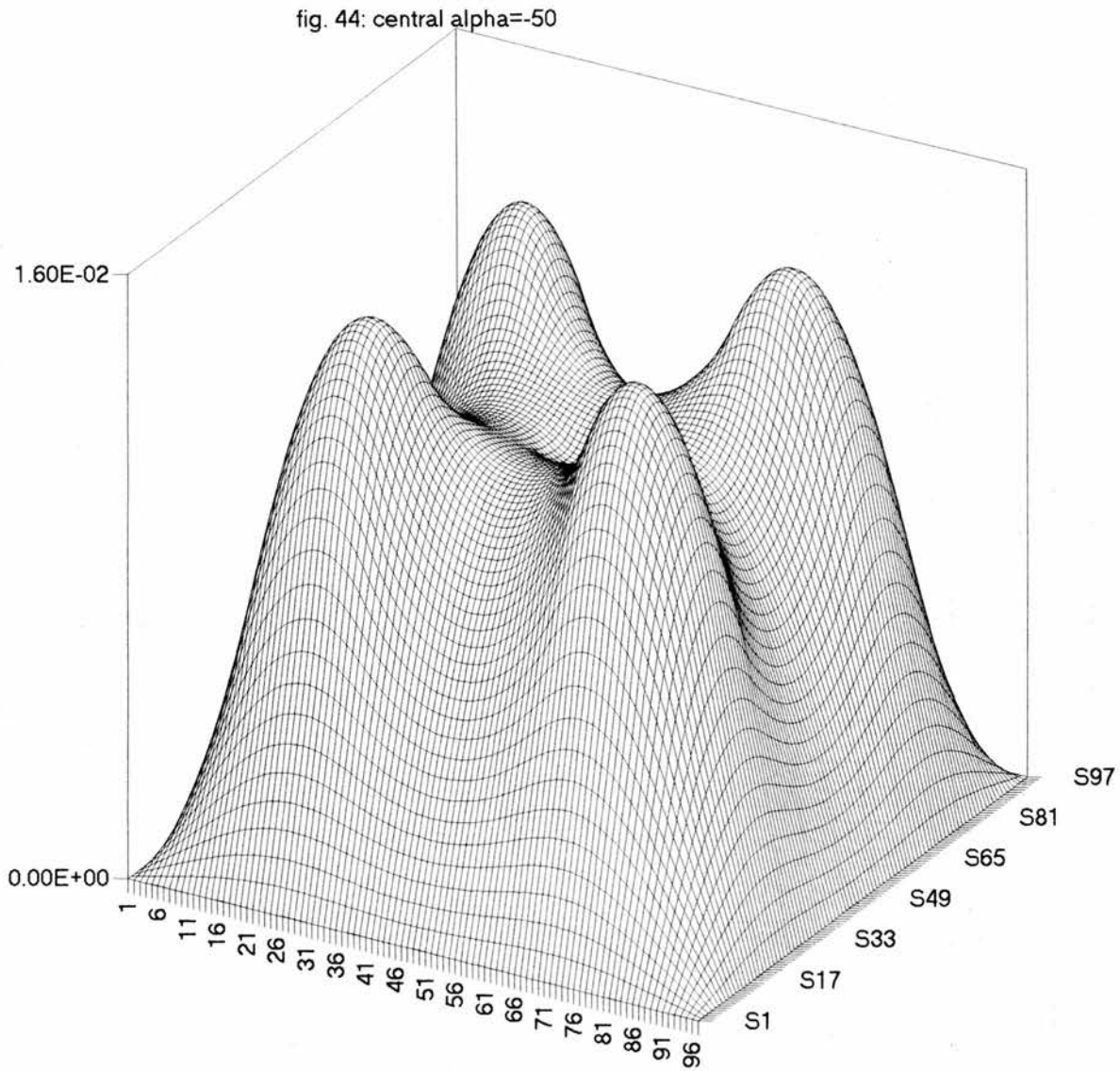


fig. 44: central alpha=-50
1.60E-02
0.00E+00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
S1
S17
S33
S49
S65
S81
S97

Summary of Results

Initially a very simplified model of a neutron source within a square cross-section of a reactor and the effects of modelling discontinuities assessed. Standard solutions to this simple model were explored and presented where good agreement between three different methods of approach was obtained. In Chapter 2 a more sophisticated model was developed where, again, good agreement was found between different methods of solution. In the process of analysing these methods an interesting phenomenon was uncovered where 'errors', or differences between methods manifest themselves as either high frequency with low amplitude or low frequency with high amplitude depending on how the source term was specified. An explanation of this phenomenon is discussed.

An analysis of transient behaviour of the neutron diffusion equation is undertaken in Chapter 3 where two models are discussed. The first model is simple with a constant α allowing comparison of different methods of solution. Good agreement between these methods is evident. The second model contains an artificially constructed α which varies across the computational grid. An analytical solution to this is compared with a numerical solution and good agreement is achieved. The second model gets closest to the original aim of the thesis which was to analytically investigate

the transient behaviour of the neutron diffusion equation with spatially dependent α.

In the latter half of the thesis a more sophisticated computational technique was adopted. The results demonstrating the speed and effectiveness of the Multigrid method. An implementation of the Multigrid method is discussed in detail and the effects of the various stages are illustrated graphically.

In Chapter 5 a more sophisticated physical model of the reactor cross-section is considered. The effects of different 'strengths' of α as a control rod are demonstrated.

Further Work

The models considered are very artificial and an obvious development is to tackle a more realistic model which takes account of the correct geometry and the bundling and distribution of the fuel and control rods. To construct such a model would require collaboration with the Industry to tease out the most important features of an enhanced model and this would represent a substantial piece of future work. In this thesis little attention was paid to the transient case and ideally, more attention to the evolution of the neutron flux would be desirable.

References:

1: Hitchcock, 1953, 'Nuclear Reactor Stability', Temple Press, page 14.

2: Brandt, 1977, 'Mathematics of Computation', Vol 31, Number 138, pages 333-390.

3: Smith, 1969, 'Numerical Solution of Partial Differential Equations', Oxford University Press, page 26.

4: 'Turbo Pascal for WINDOWS vs. 1.5', Borland.

5: Carslaw and Jaeger, 1959, 'Conduction of Heat in Solids', Clarendon Press, page 13.

6: 'EXCEL', Microsoft.

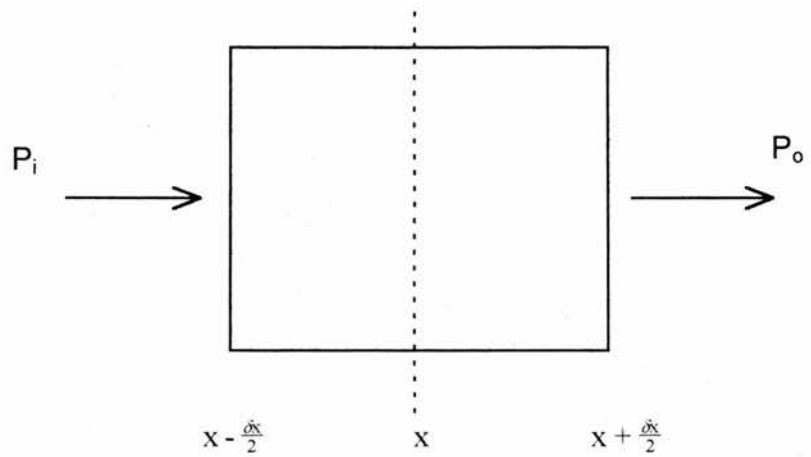
7: Kreyszig, 1993, 'Advanced Engineering Mathematics', Wiley, page 888.

8: 'MathCad', MATHSOFT.

9: Mitchell & Griffiths, 1980, 'Finite Difference Method in Partial Differential Equations', Wiley, page 20.

Appendix A

The Neutron diffusion equation and its derivation is analogous to that of temperature in a conducting medium. The derivation described below is based on an analysis where heat is flowing in the x-direction. Heating power will flow into and out of the control medium. The laws of conductivity are such that input power will be proportional to the temperature gradient in the x direction; see below



Heat entry:

$$P_i = -Ak \frac{\partial T}{\partial x}_{x-\frac{\delta x}{2}}$$ negative gradient as heat travels from high to low temperatures

$$P_o = -Ak \frac{\partial T}{\partial x}_{x+\frac{\delta x}{2}}$$

Let P_G be Heat generated per unit time and P_L be Heat absorbed (or radiated) per unit time. Thus, the increase in Heat energy in time δt is

$\rho s(A\delta x)\delta T$ = Heat entering - Heat leaving + Heat generated - Heat radiated

$$\rho s(A\delta x)\delta T = -Ak\left(\frac{\partial T}{\partial x}\right)_{x-\frac{\delta x}{2}} + -Ak\left(\frac{\partial T}{\partial x}\right)_{x+\frac{\delta x}{2}} + P_G A\delta x.\delta t - P_L A\delta x.\delta t$$

$$\rho s\frac{\delta T}{\delta t} = k\left[\frac{\left(\frac{\partial T}{\partial x}\right)_{x+\frac{\delta x}{2}} - \left(\frac{\partial T}{\partial x}\right)_{x-\frac{\delta x}{2}}}{\delta x}\right] + P_G - P_L$$

As δx becomes small,

$$\frac{\partial T}{\partial t} = k\frac{\partial^2 T}{\partial x^2} + P_G - P_L$$

Now we have heat being absorbed (or radiated) at a rate proportional to the Temperature so we have $-P_L = \alpha T$ and $P_G = S$. We also have the term T directly analogous to the neutron flux term φ.

APPENDIX B Code Listings

antrns.pas

```
alpha:=2.0;
kay:=0.00002;
for tyme:=0 to 100 do
begin
for wyy:=0 to 100 do
begin
ptrglobaldata^[wyy,tyme]:=0.0;
end;
end;
z:=50.0/100.0;
assign(result,'antrsdt');
rewrite(result);
for tyme:=0 to 100 do
begin
for wyy:=0 to 100 do
begin
m:=251;
while (m>=1) do
begin
n:=251;
while (n>=1) do
begin
t:=tyme*kay;
y:=wyy/100.0;
bee:=(4.0/(n*m*pi*pi))*(cos(0.47*m*pi)-cos(0.53*m*pi))
*(cos(0.47*n*pi)-cos(0.53*n*pi))/(-alpha+(m*m+n*n)*pi*pi);
bee:=bee*exp(-(n*n+m*m)*pi*pi*t)*exp(alpha*t);

ptrglobaldata^[wyy,tyme]:=ptrglobaldata^[wyy,tyme]+
bee*sin(n*pi*y)*sin(m*pi*z);
n:=n-2;
 end;
m:=m-2;
end;
writeln(result,ptrglobaldata^[wyy,tyme]);
end;
writeln(tyme);
end;
   close(result);
 end;
  end;
  end;
```

gstrns.pas

```
{***** read these calculated data from data file 'gs' **************}
 assign(result2,'gs');
 reset(result2);
 for Z:=0 to 100 do
 begin
 for Y:=0 to 100 do
 begin
        {writeln(result,P^[Y,Z]);}
    readln(result2,atemp);
    P^[Y,Z]:=atemp;
 end;
 end;
 close(result2);
 writeln('done reading gs');
 {*************************************************************}
  {*******************************}
   { time decay calculation}
 for axis2:=0 to 100 do
 begin
 for axis1:=0 to 100 do
 begin
time^[axis1,axis2]:=0.0;
end;
end;
 for tyme:=0 to 100 do
 begin
 writeln(tyme);
 for Y:=0 to 100 do
 begin
 for Z:=0 to 100 do
 begin
 if (Y=0) or (Y=100) or (Z=0) or (Z=100) then
 begin
 P^[Y,Z]:=0.0;
 P_previous^[Y,Z]:=0.0;
 time^[Y,Z]:=0.0;
 end;
  if (Y>0) and (Y<100) and (Z>0) and (Z<100)  then
  begin
        P_previous^[Y,Z]:=(kay/(hsq))*(P^[Y+1,Z]
         +P^[Y-1,Z]
        +P^[Y,Z+1]+P^[Y,Z-1])
        +P^[Y,Z]
        *(1.0-kay*((4.0/hsq)-alpha));
  end; {Y>0  Y<100  Z>0   Z<100}
  if (Y=0) or (Y=100) or (Z=0) or (Z=100) then
 begin
 P^[Y,Z]:=0.0;
  P_previous^[Y,Z]:=0.0;
 time^[Y,Z]:=0.0;
 end;
 time^[Y,tyme]:=P^[Y,50];
end;  {for Z=0 to 100}
end;    { for Y=0 to 100}
 for Y:=0 to 100 do
begin
for Z:=0 to 100 do
```

```
 begin
 P^[Y,Z]:=P_previous^[Y,Z];
 end;
 end;
  end;  {for tyme =0 to 100}
assign(result3,'gstime');
rewrite(result3);
for Y:=0 to 100 do
begin
for tyme:=0 to 100 do
begin
{writeln(time^[Y,tyme]);}
writeln(result3,time^[tyme,Y]);   {write back to front to line up with gstime}
 end;
 end;
 close(result3);
 writeln('done');
 end;
 end;
 end;
```

ANCONT.PAS

```
for Y:=0 to 100 do
 begin
 for X:=0 to 100 do
 begin
 P^[X,Y]:=0.0;
 for em:=100001 downto 1 do
 begin
    empi:=em*pi;
    bee:=(2/(empi))*(cos(0.49*empi)-cos(0.51*empi));
    p1:=empi*X/100;
    p2:=(1-Y/100)*empi;
    p3:=-p2;
    p5:=-empi;
    p6:=-empi*Y/100;
    p7:=exp(p6*0.1);
    if em<10 then
    begin
    P^[X,Y]:=P^[X,Y]+bee*sin(p1)*(exp(p2)-exp(p3))/(exp(empi)-exp(p5));
    end else
    begin
    P^[X,Y]:=P^[X,Y]+bee*sin(p1)*p7*p7*p7*p7*p7*p7*p7*p7*p7*p7*p7;
    end;
    end;
 end;
 writeln('y=',y);
 end;
```

CONFCONT.PAS

```
for X:=0 to 100 do
  begin
P^[x,y]:=0.0;
end;
  end;
for Y:=0 to 100 do
  begin
  for x:=0 to 100 do
begin
  if x>51 then
  begin
  atan1:=arctan((Y/100.0)/((x-50)/100.0-0.01));
  atan2:=arctan((Y/100.0)/((x-50)/100.0+0.01));
  end;
  if x=51 then
  begin
  atan1:=pi/2.0;
  atan2:=arctan((Y/100.0)/(0.02));
  end;
  if x=50 then
  begin
  atan1:=pi-arctan((Y/100.0)/(0.01));
  atan2:=arctan((Y/100.0)/(0.01));
  end;
  if x=49 then
  begin
  atan1:=pi-arctan((Y/100.0)/(0.02));
  atan2:=pi/2.0;
  end;
  if x<49 then
  begin
  atan1:=pi-arctan((Y/100.0)/(((50-x)/100.0)+0.01));
  atan2:=pi-arctan((Y/100.0)/(((50-x)/100.0)-0.01));
end;
P^[X,Y]:=(1/pi)*(atan1-atan2);
```

GSCONT.PAS

```
for Y:=0 to 100 do
  begin
for X:=0 to 100 do
  begin
P^[x,y]:=0.0;
  P^[0,Y]:=0.0;
  P^[X,0]:=0.0;
  P^[100,Y]:=0.0;
  P^[X,100]:=0.0;
  P^[49,0]:=1.0;
  P^[50,0]:=1.0;
  P^[51,0]:=1.0;
  end;
  end;
for em:=1 to 75000 do
  begin
  for Y:=1 to 99 do
  begin
  for X:=1 to 99 do
  begin
  P^[X,Y]:=0.25*(P^[X+1,Y]+P^[X-1,Y]+P^[X,Y+1]+P^[X,Y-1]);
  end;
  end;
  writeln('em=',em);
  end;
```

GSCONTX.PAS

```
for Y:=0 to 100 do
  begin
for X:=0 to 100 do
  begin
P^[x,y]:=0.0;
  P^[0,Y]:=0.0;
  P^[X,0]:=0.0;
  P^[100,Y]:=0.0;
  P^[X,100]:=0.0;
  P^[49,0]:=0.5;
  P^[50,0]:=1.0;
  P^[51,0]:=0.5;
  end;
  end;
for em:=1 to 75000 do
  begin
  for Y:=1 to 99 do
  begin
  for X:=1 to 99 do
  begin
  P^[X,Y]:=0.25*(P^[X+1,Y]+P^[X-1,Y]+P^[X,Y+1]+P^[X,Y-1]);
  end;
  end;
  writeln('em=',em);
  end;
```

FOURIER.PAS

```
alpha:=2.0;
 vee:=0.0;
 veesource:=0.0;
  assign(result,'flux');
 rewrite(result);
   assign(result2,'sourcetm');
 rewrite(result2);
 for Z:=0 to 100 do
 begin
        for Y:=0 to 100 do
        begin
                m:=501;
          while (m>=1) do
          begin
                n:=501;
                        while (n>=1) do
                        begin
                                zz:=Z/100.0;
                                yy:=Y/100.0;
                             mpi:=m*pi;
                            npi:=n*pi;
 bee:=(4.0/(npi*mpi))*(cos(0.47*mpi)-cos(0.53*mpi))*
                                (cos(0.47*npi)-cos(0.53*npi))/
                                (-alpha+(mpi*mpi+npi*npi));
 veesource:=veesource+bee*(-alpha+(mpi*mpi+npi*npi))*
                                sin(mpi*yy)*sin(npi*zz);
                           vee:=vee+bee*sin(mpi*yy)*sin(npi*zz);
                                n:=n-2;
                        end;
                m:=m-2;
          end;
 writeln(result,vee);
 writeln(result2,veesource);
 vee:=0.0;
 veesource:=0.0;
                writeln('Z=',Z,' y=',y);
        end;  {Y}
 end;  {Z}
   close(result);
   close(result2);
 end;
  end;
  end;
```

GS.PAS

```
{***** read these source term data from data file 'sourcetm' **************}
 assign(result,'sourcetm');
 reset(result);
 for Z:=0 to 100 do
 begin
 for Y:=0 to 100 do
 begin
          {writeln(result,P^[Y,Z]);}
    readln(result,atemp);
    SOURCE_TERM^[Y,Z]:=atemp;
 end;
 end;
          writeln(' got  here ');
  close(result);
 {*************************************************************}
  for Z:=0 to 100 do
 begin
for Y:=0 to 100 do
 begin
 P^[Y,Z]:=0.0;
  end;
 end;
 for em:=1 to 60000 do
 begin
 writeln(em);
   for Z:=0 to 100 do
 begin
 for Y:=0 to 100 do
 begin
 P^[0,Z]:=0.0;
 P^[Y,0]:=0.0;
  P^[100,Z]:=0.0;
 P^[Y,100]:=0.0;
if (Y>0) and (Y<100) and (Z>0) and (Z<100) then
  begin
  P^[Y,Z]:=
  (SOURCE_TERM^[Y,Z]+(1/hitchsq)*(P^[Y+1,Z]+
  P^[Y-1,Z]+P^[Y,Z+1]
  +P^[Y,Z-1]))*(1.0/(-alpha+(4.0/hitchsq)));
 end;
   P^[0,Z]:=0.0;
 P^[Y,0]:=0.0;
  P^[100,Z]:=0.0;
 P^[Y,100]:=0.0;
  end; {Y}
  end; {Z}
  end;
```

GBFOURIE.PAS

```
for z:=0 to 100 do
 begin
        for y:=0 to 100 do
        begin
                m:=501;
          while (m>=1) do
          begin
                n:=501;
                        while (n>=1) do
                        begin
                        zz:=z/100.0;
                        yy:=y/100.0;
                        mpi:=m*pi;
                        npi:=n*pi;
                        sinn:=sin(npi*0.005);
                        sinm:=sin(mpi*0.005);
                        sinn2:=sinn*sinn;
                        sinm2:=sinm*sinm;
                        sinsin:=sin(mpi*yy)*sin(npi*zz);
                        A:=(4.0/(npi*mpi))*(cos(0.47*mpi)-cos(0.53*mpi))*
                        (cos(0.47*npi)-cos(0.53*npi));
                        veesource:=veesource+A*sinsin;
                        temp1:=-alpha+(40000.0)*(sinn2+sinm2);
                        C:=A/temp1;
                        vee:=vee+C*sinsin;
                        n:=n-2;
                        end;
                m:=m-2;
           end;
 writeln(result,vee);
 writeln(result2,veesource);
 vee:=0.0;
 veesource:=0.0;
 writeln('y= ',y,' z= ',z);
        end;  {y}
        writeln('z=',z);
 end;  {z}
   close(result);
   close(result2);
 end;
```

critanal.pas

```
alpha:=2*3.1415927*3.1415927;    {fission coefficient EIGENVALUE}
 H:=1.0/96.0;
 Hsq:=H*H;
 assign(result50,'critanal');
 rewrite(result50);
 for Z:=0 to 96 do
 begin
 for Y:=0 to 96 do
 begin
  P^[Y,Z]:=sin(3.1415927*Y/96.0)*sin(3.1415927*Z/96.0);
        writeln(result50,P^[Y,Z]);
 end;
 end;
  close(result50);
end;
end;
end;
```

1dan.pas;

```
begin
  alpha:=2.0;
kay:=0.00004;
pp:=0;
z:=50.0/100.0;
assign(result,'expt2');
rewrite(result);
for tyme:=0 to 100 do
begin
for wyy:=0 to 100 do
begin
t:=tyme*kay*1000;
y:=wyy/100.0;
pp:=y*(1-y)*exp(-t);
  writeln(result,pp);
end;
writeln(tyme);
end;
    close(result);
 end;
  end;
  end.
```

1dnum.pas;

```
assign(result3,'expt1');
rewrite(result3);
 kay:=0.00004;
 h:=0.01;
 hsq:=h*h;
 tt:=0;
 writeln('got here');
 for Y:=0 to 100 do
 begin
 P^[Y]:=0.00;
 P_previous^[Y]:=0.0;
 end;
 for Y:=1 to 99 do
 begin
    alpha^[Y]:=-1+(2/((Y*0.01)*(1-(Y*0.01))));
 end;
 for Y:=0 to 100 do
 begin
    P^[Y]:=(Y*0.01)*(1-(Y*0.01));
 end;
  {*****************************************************************}
 {*******************************}
  { time decay calculation}
 for tyme:=0 to 100000 do
 begin
{ writeln(tyme);          }
 for Y:=0 to 100 do
 begin
                  if (Y=0) or (Y=100) then
                   begin
                       P^[Y]:=0.0;
                        P_previous^[Y]:=0.0;
                   end else
                   begin
                   P_previous^[Y]:=(kay/(hsq))*(P^[Y+1]
          +P^[Y-1])+P^[Y]*(1.0-kay*((2.0/hsq)-alpha^[Y]));
                   end;
if (tt=999) or (tyme=0) then writeln(result3,P^[Y]);
end;   { for Y=0 to 100}
 for Y:=0 to 100 do
 begin
 P^[Y]:=P_previous^[Y];
 end;
 tt:=tt+1;
 if (tt=1000) then tt:=0;
  end; {for tyme =0 to 100}
close(result3);
 writeln('done');
 end;
 end;
 end.
```

mg.pas

```
type
        pbigarray=^tee;
        tee=array[0..96,0..96] of real;
    pba2=^tee2;
        tee2=array[0..96,0..96] of real;
        pba3=^tee3;
        tee3=array[0..96,0..96] of real;
        pba4=^tee4;
        tee4=array[0..96,0..96] of real;
        pba5=^tee5;
        tee5=array[0..96,0..96] of real;
        pba6=^tee6;
        tee6=array[0..96,0..96] of real;
        pba7=^tee7;
        tee7=array[0..96,0..96] of real;
        pba8=^tee8;
        tee8=array[0..96,0..96] of real;
        pba9=^tee9;
    tee9=array[0..48,0..48] of real;
        pba10=^tee10;
        tee10=array[0..48,0..48] of real;
        pba11=^tee11;
        tee11=array[0..48,0..48] of real;
        pba12=^tee12;
        tee12=array[0..24,0..24] of real;
        pba13=^tee13;
        tee13=array[0..24,0..24] of real;
        pba14=^tee14;
        tee14=array[0..48,0..48] of real;
        pba15=^tee15;
        tee15=array[0..48,0..48] of real;
        pba16=^tee16;
        tee16=array[0..96,0..96] of real;
        pba17=^tee17;
        tee17=array[0..24,0..24] of real;
        pba18=^tee18;
    tee18=array[0..48,0..48] of real;
        pba19=^tee19;
        tee19=array[0..96,0..96] of real;
        pba20=^tee20;
        tee20=array[0..96,0..96] of real;
        pba21=^tee21;
        tee21=array[0..96,0..96] of real;

var
        g1:thandle;
        g2:thandle;
        g3:thandle;
        g4:thandle;
        g5:thandle;
        g6:thandle;
        g7:thandle;
        g8:thandle;
        g9:thandle;
        g10:thandle;
        g11:thandle;
```

```
        g12:thandle;
        g13:thandle;
        g14:thandle;
        g15:thandle;
        g16:thandle;
        g17:thandle;
        g18:thandle;
        g19:thandle;
        g20:thandle;
        g21:thandle;
        hr,hrr,min,minn,sec,secc,hund,hundd:word;
        u:pbigarray;
        result,result2,mg1dat,mg2adat,mg3adat,zdat,result_u,result_R1:text;
result_R2,result_v1,result_v2,result_r1_medium,result_v1_medium:text;
    result_r2_medium,result_r2_coarse,result_v2_coarse,result_v2_coarse_to_medium:text;
result_v1_medium_corrected,result_v1_medium_corrected_to_fine:text;
        result_u_coarse,result_u_medium,result_u_medium_to_fine:text;
    ccc:pba2;
        udatum:pba3;
        R1:pba4;
        v1:pba5;
        v2:pba6;
        R2:pba7;
        dummy_array:pba8;
    v1_medium:pba9;
        R1_medium:pba10;
        R2_medium:pba11;
        R2_coarse:pba12;
        v2_coarse:pba13;
        v2_coarse_to_medium:pba14;
        v1_medium_corrected:pba15;
        v1_medium_corrected_to_fine:pba16;
        u_coarse:pba17;
        u_medium:pba18;
        u_medium_to_fine:pba19;
        source_term:pba20;
        alpha_term:pba21;

var
 TheRect: TRect;
 I, J: Integer;
 CenterX,
 CenterYY: Integer;
 Radius,
 StepAngle: Word;
 Radians: real;
 arrayname:integer;
 t,x,y,z,p1,p2,p3,p4,p5,p6,vee,vee2,H,conv,conv2,atemp,mag:real;
 kay,bee,tze,xt,yt,zt,dt,sx,es,sy,sz,cx,cy,twat,cz,teek,plot_difference:real;
 em,strings_plot,strings_plot_inc,s_p:longint;
 dummy,ZZ,veee,col2,row2,row2a,en,enn,ennn,ennnn,YY,ii,row,row3,
 color,counter,kntr:integer;
 col,loopcount,ahem,tyme,coladj,rowadj,g_or_not,AA,BB:integer;
 ch:char;

 label 1;
 label 2;
 label 3;
```

```
label 4;
label 5;
label 6;


{**********START  GET R1  ROUTINE **********}
procedure get_R1(YYL:integer;ZZL:integer;hitch:real);
begin
ZZ:=ZZL;      { or zz:=zzl ???}
while ZZ<96 do
begin
YY:=YYL;
        while YY<96 do
        begin
        R1^[YY,ZZ]:=(1/(Hitch*Hitch))*(u_medium_to_fine^[YY+YYL,ZZ]-4*
        u_medium_to_fine^[YY,ZZ]+u_medium_to_fine^[YY-YYL,ZZ]+
        u_medium_to_fine^[YY,ZZ+ZZL]+u_medium_to_fine^[YY,ZZ-
ZZL])+alpha_term^[YY,ZZ]*u_medium_to_fine^[YY,ZZ]+source_term^[YY,ZZ];
        inc(YY,YYL);
        end;
inc(ZZ,ZZL);
end;
end;
{**********FINISH  GET R1 ROUTINE **********}
{**********START  GET R2 medium  ROUTINE **********}
procedure get_R2(YYL:integer;ZZL:integer;hitch:real);
begin
ZZ:=ZZL;
while ZZ<48 do
begin
YY:=YYL;
        while YY<48 do
        begin
                R2_medium^[YY,ZZ]:=(1/(Hitch*Hitch))*(v1_medium^[YY+YYL,ZZ]-4*
                v1_medium^[YY,ZZ]+v1_medium^[YY-YYL,ZZ]+
                v1_medium^[YY,ZZ+ZZL]+v1_medium^[YY,ZZ-ZZL])
                +alpha_term^[2*YY,2*ZZ]*v1_medium^[YY,ZZ]+R1_medium^[YY,ZZ];
                inc(YY,YYL);
        end;
inc(ZZ,ZZL);
end;
end;
{**********FINISH  GET R2 medium ROUTINE **********}
{****start INTERPOLATE **************************}
procedure interpolate(YYL:integer;ZZL:integer;arrayname:integer);
begin
ZZ:=0;
        while ZZ<=ZZL           do
        begin
                YY:=0;
                        while YY<=YYL do
                        begin
                        dummy_array^[YY,ZZ]:=0.0;
                         {if (arrayname=0) then dummy_array^[YY,ZZ]:=1.0;
                    if (arrayname=1) then dummy_array^[YY,ZZ]:=1.0;     }

                        inc(YY,1);
                        end; {YY}
```

```
        inc(ZZ,1);
        end; {ZZ}
 ZZ:=0;
        while ZZ<=ZZL/2 do
        begin
                YY:=0;
                        while YY<=YYL/2 do
                        begin
                    if (arrayname=0) then dummy_array^[2*YY,2*ZZ]:=u_coarse^[YY,ZZ];
                    if (arrayname=1) then dummy_array^[2*YY,2*ZZ]:=u_medium^[YY,ZZ];
                    if (arrayname=2) then dummy_array^[2*YY,2*ZZ]:=v2_coarse^[YY,ZZ];
                    if (arrayname=3) then
dummy_array^[2*YY,2*ZZ]:=v1_medium_corrected^[YY,ZZ];
                        inc(YY,1);
                        end; {YY}
        inc(ZZ,1);
        end; {ZZ}
 ZZ:=1;
        while ZZ<ZZL do
        begin
                YY:=0;
                        while YY<YYL do
                        begin

        dummy_array^[YY,ZZ]:=0.5*(dummy_array^[YY,ZZ+1]+dummy_array^[YY,ZZ-1]);
                        inc(YY,2);
                        end;
        inc(ZZ,2);
        end;
 YY:=1;
        while YY<YYL do
        begin
                ZZ:=0;
                        while ZZ<ZZL do
                        begin

        dummy_array^[YY,ZZ]:=0.5*(dummy_array^[YY+1,ZZ]+dummy_array^[YY-1,ZZ]);
                        inc(ZZ,2);
              end;
        inc(YY,2);
    end;
 ZZ:=1;
        while ZZ<ZZL do
        begin
                YY:=1;
                        while YY<YYL do
                        begin


        dummy_array^[YY,ZZ]:=0.25*(dummy_array^[YY+1,ZZ]+dummy_array^[YY-1,ZZ]
                        +dummy_array^[YY,ZZ+1]+dummy_array^[YY,ZZ-1]);
                    inc(YY,2);
                    end;
        inc(ZZ,2);
    end;
        ZZ:=0;
        while ZZ<=ZZL do
        begin
```

```
                YY:=0;
                        while YY<=YYL do
                        begin
                if (arrayname=0) then u_medium^[YY,ZZ]:=dummy_array^[YY,ZZ];

                if (arrayname=1) then
                begin
                        u^[YY,ZZ]:=dummy_array^[YY,ZZ];
                        u_medium_to_fine^[YY,ZZ]:=dummy_array^[YY,ZZ];
          end;
                if (arrayname=2) then
        v2_coarse_to_medium^[YY,ZZ]:=dummy_array^[YY,ZZ];
                if (arrayname=3) then
        v1_medium_corrected_to_fine^[YY,ZZ]:=dummy_array^[YY,ZZ];
                        inc(YY,1);
                        end; {YY}
        inc(ZZ,1);
        end; {ZZ}
 ZZ:=0;
end;
{*******************end interpolate *****}
{*****************START  WRITE ROUTINE**********************}
 procedure write_all;
 begin
 assign(result_u,'u_dat');
 rewrite(result_u);
 ZZ:=0;
 while ZZ<=96 do
 begin
 YY:=0;
 while YY<=96 do
 begin
    writeln(result_u,u^[YY,ZZ]);
 inc(YY,1);
 end;
 inc(ZZ,1);
 end;
close(result_u);
 end;
 { *************END OF  WRITE ROUTINE*****************************}
 begin     {***********beginning of main code block*******************}
  g1:=globalalloc(gmem_moveable,100000);
 g2:=globalalloc(gmem_moveable,100000);
 g3:=globalalloc(gmem_moveable,100000);
 g4:=globalalloc(gmem_moveable,100000);
  g5:=globalalloc(gmem_moveable,100000);
  g6:=globalalloc(gmem_moveable,100000);
 g7:=globalalloc(gmem_moveable,100000);
 g8:=globalalloc(gmem_moveable,100000);
 g9:=globalalloc(gmem_moveable,100000);
  g10:=globalalloc(gmem_moveable,100000);
  g11:=globalalloc(gmem_moveable,100000);
  g12:=globalalloc(gmem_moveable,100000);
  g13:=globalalloc(gmem_moveable,100000);
  g14:=globalalloc(gmem_moveable,100000);
  g15:=globalalloc(gmem_moveable,100000);
  g16:=globalalloc(gmem_moveable,100000);
  g17:=globalalloc(gmem_moveable,100000);
```

```
 g18:=globalalloc(gmem_moveable,100000);
  g19:=globalalloc(gmem_moveable,100000);
  g20:=globalalloc(gmem_moveable,100000);
  g21:=globalalloc(gmem_moveable,100000);
 if (g1<>0) and (g2<>0) and (g4<>0) and (g3<>0)
 and (g5<>0) and (g6<>0) and (g7<>0)
  and (g8<>0) and (g9<>0) and (g10<>0) and (g11<>0) and (g12<>0) and (g13<>0)
  and (g14<>0) and (g15<>0) and (g16<>0) and (g17<>0) and (g18<>0)  and (g19<>0)
   and (g20<>0) and (g21<>0) then
 begin
 u:=globallock(g1);
 ccc:=globallock(g2);
 udatum:=globallock(g3);
R1:=globallock(g4);
 v1:=globallock(g5);
 v2:=globallock(g6);
 R2:=globallock(g7);
 dummy_array:=globallock(g8);
 v1_medium:=globallock(g9);
 R1_medium:=globallock(g10);
 R2_medium:=globallock(g11);
 R2_coarse:=globallock(g12);
 v2_coarse:=globallock(g13);
 v2_coarse_to_medium:=globallock(g14);
 v1_medium_corrected:=globallock(g15);
 v1_medium_corrected_to_fine:=globallock(g16);
 u_coarse:=globallock(g17);
 u_medium:=globallock(g18);
  u_medium_to_fine:=globallock(g19);
  source_term:=globallock(g20);
  alpha_term:=globallock(g21);
 if (u<>nil) and (ccc<>nil) and (udatum<>nil) and (R1<>nil) and (v1<>nil)
  and (v2<>nil) and (R2<>nil)
  and (dummy_array<>nil) and (v1_medium<>nil) and (R1_medium<>nil) and
(R2_medium<>nil)
  and (R2_coarse<>nil) and (v2_coarse<>nil) and (v2_coarse_to_medium<>nil)
   and (v1_medium_corrected<>nil) and (v1_medium_corrected_to_fine<>nil)
   and (u_coarse<>nil) and (u_medium<>nil)  and (u_medium_to_fine<>nil)
    and (source_term<>nil) and (alpha_term<>nil) then
 begin
 H:=0.01;
{**************************START OVERALL LOOP*********************}
gettime(hr,min,sec,hund);
{********************INITIALISE ALL ARRAYS************************}
ZZ:=0;

        while ZZ<=96 do
        begin
                YY:=0;
                while YY<=96 do
                begin
                dummy_array^[YY,ZZ]:=0.0;

            u^[YY,ZZ]:=0.0;
                ccc^[YY,ZZ]:=0.0;
                R1^[YY,ZZ]:=0.0;
                R2^[YY,ZZ]:=0.0;
                v1^[YY,ZZ]:=0.0;
```

```
                v2^[YY,ZZ]:=0.0;
                R1_medium^[YY,ZZ]:=0.0;
                v1_medium^[YY,ZZ]:=0.0;
                R2_medium^[YY,ZZ]:=0.0;
                R2_coarse^[YY,ZZ]:=0.0;
                v2_coarse^[YY,ZZ]:=0.0;
                v2_coarse_to_medium^[YY,ZZ]:=0.0;
                v1_medium_corrected^[YY,ZZ]:=0.0;
                v1_medium_corrected_to_fine^[YY,ZZ]:=0.0;
                u_coarse^[YY,ZZ]:=0.0;
                u_medium^[YY,ZZ]:=0.0;
                 u_medium_to_fine^[YY,ZZ]:=0.0;
                 source_term^[YY,ZZ]:=0.0;
                inc(YY,1);
                end;
        inc(ZZ,1);
        end;
for zz:=0 to 96 do
  begin
for yy:=0 to 96 do
  begin
source_term^[yy,zz]:=0.0;
 alpha_term^[yy,zz]:=0.0;
 end;
 end;
for yy:=44 to 52 do
   begin
   for zz:=44 to 52 do
   begin
    source_term^[yy,zz]:=0.0;
    alpha_term^[yy,zz]:=-10.0;
   end;
   end;
for yy:=12 to 32 do
   begin
   for zz:=12 to 32 do
   begin
    source_term^[yy,zz]:=1.0;
    alpha_term^[yy,zz]:=2.0;
   end;
   end;
for yy:=12 to 32 do
   begin
   for zz:=64 to 84 do
   begin
    source_term^[yy,zz]:=1.0;
    alpha_term^[yy,zz]:=2.0;
   end;
   end;
for yy:=64 to 84 do
   begin
   for zz:=12 to 32 do
   begin
    source_term^[yy,zz]:=1.0;
    alpha_term^[yy,zz]:=2.0;
   end;
   end;
for yy:=64 to 84 do
```

```
  begin
  for zz:=64 to 84 do
  begin
   source_term^[yy,zz]:=1.0;
   alpha_term^[yy,zz]:=2.0;
  end;
  end;
{*******************END INITIALISE ALL ARRAYS**********************}
  {****************START SOLVE u ON COARSE GRID step 7*********************}
  assign(result_u_coarse,'ucrsdat');
rewrite(result_u_coarse);
  for counter:=0 to 1000 do
  begin
  for ZZ:=0 to 24 do
  begin
     u_coarse^[0,ZZ]:=0.0;
     u_coarse^[24,ZZ]:=0.0;
  end;
  for YY:=0 to 24 do
  begin
     u_coarse^[YY,0]:=0.0;
      u_coarse^[YY,24]:=0.0;
  end;
          ZZ:=1;
          while ZZ<24 do
          begin
                  YY:=1;
                          while YY<24 do
                          begin
u_coarse^[YY,ZZ]:=(u_coarse^[YY+1,ZZ]+u_coarse^[YY-1,ZZ]
 +u_coarse^[YY,ZZ-1]+u_coarse^[YY,ZZ+1]+(1/24.0)*(1/24.0)*source_term^[4*YY,4*ZZ])
 /(4.0-alpha_term^[4*YY,4*ZZ]*(1/24.0)*(1/24.0));
                                  inc(YY,1);
                          end; {YY}
          inc(ZZ,1);
          end; {ZZ}
  ZZ:=0;
  for ZZ:=0 to 24 do
  begin
     u_coarse^[0,ZZ]:=0.0;
     u_coarse^[24,ZZ]:=0.0;
  end;
  for YY:=0 to 24 do
  begin
     u_coarse^[YY,0]:=0.0;
      u_coarse^[YY,24]:=0.0;
  end;
  end;
          ZZ:=0;
          while ZZ<=24 do
          begin
                  YY:=0;
                          while YY<=24 do
                          begin
                  writeln(result_u_coarse,u_coarse^[YY,ZZ]);
                          inc(YY,1);
                          end;
          inc(ZZ,1);
```

```
        end;
 ZZ:=0;
 close(result_u_coarse);
 writeln('got past u coarse');
  {***************END SOLVE u ON COARSE GRID*******************}
{****start INTERPOLATE u_coarse down from coarse to medium*step
8*************************}
  interpolate(48,48,0);
   assign(result_u_medium,'umedat');
 rewrite(result_u_medium);
 ZZ:=0;
        while ZZ<=48 do
        begin
                YY:=0;
                        while YY<=48 do
                        begin
                writeln(result_u_medium,u_medium^[YY,ZZ]);
                        inc(YY,1);
                        end;
        inc(ZZ,1);
        end;
 ZZ:=0;
   close(result_u_medium);
    writeln('got past u med');
{*******************end interpolate u down from coarse to medium*****}
{ *********  gauss-seidel SWEEP of grid to iron out high freq errors on u_medium*********}
 for counter:=0 to 500 do
 begin

  ZZ:=1;
        while ZZ<48 do
        begin
                YY:=1;
                        while YY<48 do
                        begin
        u_medium^[YY,ZZ]:=(u_medium^[YY+1,ZZ]+u_medium^[YY-1,ZZ]
        +u_medium^[YY,ZZ-
1]+u_medium^[YY,ZZ+1]+(1/48)*(1/48)*source_term^[2*YY,2*ZZ])
        /(4.0-alpha_term^[2*YY,2*ZZ]*(1/48)*(1/48));

                        inc(YY,1);
                        end;
        inc(ZZ,1);
        end;
 ZZ:=0;
end;
 writeln('got past u sweep');
{ ********* END gauss-seidel swep of grid to iron out high freq errors** on u_medium*******}
{****************start u_medium interpolate down to u fine  down to grid 0    step 12*********}
 interpolate(96,96,1);
  assign(result_u_medium_to_fine,'umedfine');
 rewrite(result_u_medium_to_fine);
 ZZ:=0;
        while ZZ<=96 do
        begin
                YY:=0;
                        while YY<=96 do
                        begin
```

```
        writeln(result_u_medium_to_fine,u_medium_to_fine^[YY,ZZ]);
                        inc(YY,1);
                        end;
        inc(ZZ,1);
        end;
 ZZ:=0;
  close(result_u_medium_to_fine);
   writeln('got past u med intrpl');
 {**********end interp**********************************************}
{**********RUN GAUSS SEIDEL ON FINE GRID****step 3*********}
 for counter:=0 to 50 do
 begin
 for ZZ:=0 to 96 do
 begin
    u_medium_to_fine^[0,ZZ]:=0.0;
    u_medium_to_fine^[96,ZZ]:=0.0;
 end;
 for YY:=0 to 96 do
 begin
    u_medium_to_fine^[YY,0]:=0.0;
     u_medium_to_fine^[YY,96]:=0.0;
 end;
 for ZZ:=1 to 95 do
 begin
 for YY:=1 to 95 do
 begin
 u_medium_to_fine^[YY,ZZ]:=(u_medium_to_fine^[YY+1,ZZ]+u_medium_to_fine^[YY-
1,ZZ]
 +u_medium_to_fine^[YY,ZZ-
1]+u_medium_to_fine^[YY,ZZ+1]+(1/96)*(1/96)*source_term^[YY,ZZ])
 /(4.0-alpha_term^[YY,ZZ]*(1/96)*(1/96));
 end;
 end;
 for ZZ:=0 to 96 do
 begin
    u_medium_to_fine^[0,ZZ]:=0.0;
    u_medium_to_fine^[96,ZZ]:=0.0;
 end;
 for YY:=0 to 96 do
 begin
    u_medium_to_fine^[YY,0]:=0.0;
     u_medium_to_fine^[YY,96]:=0.0;
 end;
 end;
  writeln('got past u fine');
{**********END RUN GAUSS SEIDEL ON FINE GRID**********}
{**********START GET R1 **********step 4**********}
        get_R1(1,1,0.01041667);     { R1 obtained on fine grid}
        assign(result_r1,'r1fine');
     rewrite(result_r1);
 ZZ:=0;
        while ZZ<=96 do
        begin
                YY:=0;
                while YY<=96 do
                begin
                writeln(result_r1,r1^[YY,ZZ]);
                inc(YY,1);
```

```
                end;
        inc(ZZ,1);
        end;
   close(result_r1);
    writeln('got past R1');
 {*****************end GET R1 ************************************}
{**************transfer R1 on fine grid to R1 on medium grid ******************}
  assign(result_r1_medium,'R1meddat');
 rewrite(result_r1_medium);

 ZZ:=0;
        while ZZ<=48 do
        begin
                YY:=0;
                while YY<=48 do
                begin
                R1_medium^[YY,ZZ]:=R1^[2*YY,2*ZZ];
                writeln(result_r1_medium,R1_medium^[YY,ZZ]);
                inc(YY,1);
                end;
        inc(ZZ,1);
        end;
 close(result_r1_medium);
  writeln('got past r1_medium');
 {**********end   transfer R1 on fine grid to R1 on medium grid ******************}
 {*****************START SOLVE V1 ON MEDIUM GRID******step 5********}
assign(result_v1_medium,'v1meddat');
 rewrite(result_v1_medium);

 for counter:=0 to 500 do
 begin
        ZZ:=1;
        while ZZ<48 do
        begin
                YY:=1;
                        while YY<48 do
                        begin
        v1_medium^[YY,ZZ]:=(v1_medium^[YY+1,ZZ]+v1_medium^[YY-1,ZZ]
                        +v1_medium^[YY,ZZ-
1]+v1_medium^[YY,ZZ+1]+(1/48)*(1/48)*R1_medium^[YY,ZZ])
                        /(4.0-alpha_term^[2*YY,2*ZZ]*(1/48)*(1/48));
                        inc(YY,1);
                        end; {YY}
        inc(ZZ,1);
        end; {ZZ}
 ZZ:=0;
end;
ZZ:=0;
        while ZZ<=48 do
        begin
                YY:=0;
                        while YY<=48 do
                        begin
                writeln(result_v1_medium,v1_medium^[YY,ZZ]);
                        inc(YY,1);
                        end;
        inc(ZZ,1);
        end;
```

```
ZZ:=0;
  close(result_v1_medium);
    writeln('got past v1_medium');

{**********END SOLVE V1 ON GRID 1****medium**********}
{**********START GET R2**on medium grid***step 6**********}
get_R2(1,1,0.020833333);
   assign(result_r2_medium,'R2meddat');
rewrite(result_r2_medium);
ZZ:=0;
        while ZZ<=48 do
        begin
                YY:=0;
                while YY<=48 do
                begin
                writeln(result_r2_medium,R2_medium^[YY,ZZ]);
                inc(YY,1);
                end;
        inc(ZZ,1);
        end;
close(result_r2_medium);
  writeln('got past r2_medium');

 {**********END GET R2**********}
{********** transfer R2 on medium grid to R2 on coarse grid **********}
  assign(result_r2_coarse,'R2crsdat');
rewrite(result_r2_coarse);
ZZ:=0;
        while ZZ<=24 do
        begin
                YY:=0;
                while YY<=24 do
                begin
                R2_coarse^[YY,ZZ]:=R2_medium^[2*YY,2*ZZ];
                writeln(result_r2_coarse,R2_coarse^[YY,ZZ]);
                inc(YY,1);
                end;
        inc(ZZ,1);
        end;
 close(result_r2_coarse);
   writeln('got past r2_crs');

{********** transfer R2 on medium grid to R2 on coarse grid **********}
 {**********START SOLVE V2 ON COARSE GRID step 7**********}

assign(result_v2_coarse,'v2crsdat');
rewrite(result_v2_coarse);

for counter:=0 to 5000 do
begin
        ZZ:=1;
        while ZZ<24 do
        begin
                YY:=1;
                        while YY<24 do
                        begin
        v2_coarse^[YY,ZZ]:=(v2_coarse^[YY+1,ZZ]+v2_coarse^[YY-1,ZZ]
```

```
                        +v2_coarse^[YY,ZZ-
1]+v2_coarse^[YY,ZZ+1]+(1/24)*(1/24)*R2_coarse^[YY,ZZ])
                        /(4.0-alpha_term^[4*YY,4*ZZ]*(1/24)*(1/24));
                        inc(YY,1);
                        end; {YY}
        inc(ZZ,1);
        end; {ZZ}
 ZZ:=0;
end;
ZZ:=0;
        while ZZ<=24 do
        begin
                YY:=0;
                        while YY<=24 do
                        begin
                writeln(result_v2_coarse,v2_coarse^[YY,ZZ]);
                        inc(YY,1);
                        end;
        inc(ZZ,1);
        end;
 ZZ:=0;
   close(result_v2_coarse);
     writeln('got past v2_coarse');

 {***************END SOLVE V2 ON COARSE GRID*******************}
{****start INTERPOLATE v2 down from coarse to medium*step 8**************************}
interpolate(48,48,2);
  assign(result_v2_coarse_to_medium,'v2c-mdat');
rewrite(result_v2_coarse_to_medium);
        ZZ:=0;
        while ZZ<=48 do
        begin
                YY:=0;
                        while YY<=48 do
                        begin
                writeln(result_v2_coarse_to_medium,v2_coarse_to_medium^[YY,ZZ]);
                        inc(YY,1);
                        end;
        inc(ZZ,1);
        end;
ZZ:=0;
   close(result_v2_coarse_to_medium);
     writeln('got past v2 crs med');

{*******************end interpolate v2 down from coarse to medium*****}
{ ********** add v1_medium to v2_coarse_to_medium to get revised
v1_medium_corrected****}

 assign(result_v1_medium_corrected,'v1m_corr');
rewrite(result_v1_medium_corrected);
ZZ:=0;
        while ZZ<=48 do
        begin
                YY:=0;
                        while YY<=48 do
                        begin
```

```
        v1_medium_corrected^[YY,ZZ]:=v1_medium^[YY,ZZ]+v2_coarse_to_medium^[YY,
ZZ];
                writeln(result_v1_medium_corrected,v1_medium_corrected^[YY,ZZ]);
                        inc(YY,1);
                        end; {YY}
        inc(ZZ,1);
        end; {ZZ}
 ZZ:=0;
  close(result_v1_medium_corrected);
    writeln('got past v1_medium corr');

 { *****END** add v1_medium to v2_coarse_to_medium to get revised
v1_medium_corrected****}
 { ********* gauss-seidel SWEEP of grid to iron out high freq errors*********}
 for counter:=0 to 50 do
 begin
  ZZ:=1;
        while ZZ<48 do
        begin
                YY:=1;
                        while YY<48 do
                        begin

v1_medium_corrected^[YY,ZZ]:=(v1_medium_corrected^[YY+1,ZZ]+v1_medium_corrected
^[YY-1,ZZ]
  +v1_medium_corrected^[YY,ZZ-
1]+v1_medium_corrected^[YY,ZZ+1]+(1/48)*(1/48)*R1_medium^[YY,ZZ])
                        /(4.0-alpha_term^[2*YY,2*ZZ]*(1/48)*(1/48));
                        inc(YY,1);
                        end;
        inc(ZZ,1);
        end;
 ZZ:=0;
end;
   writeln('got past v1_medium corr sweep');

 { ********* END  double gauss-seidel swep of grid to iron out high freq errors*********}
{*****************start v1_medium_corrected INTERPOLATE  down to grid 0    step 12*********}
   interpolate(96,96,3);
    assign(result_v1_medium_corrected_to_fine,'v1mcrfne');
 rewrite(result_v1_medium_corrected_to_fine);
        ZZ:=0;
        while ZZ<=96 do
        begin
                YY:=0;
                        while YY<=96 do
                        begin

        writeln(result_v1_medium_corrected_to_fine,v1_medium_corrected_to_fine^[YY,ZZ
]);
                        inc(YY,1);
                        end;
        inc(ZZ,1);
        end;
 ZZ:=0;
   close(result_v1_medium_corrected_to_fine);
     writeln('got past v1_medium corr fine');
```

```
{******************end interpolate v1 down to grid 0*******************}
{**************start add correction term v1 to original u   step 14****}
 ZZ:=0;
        while ZZ<=96 do
        begin
                YY:=0;
                while YY<=96 do
                begin

        u^[YY,ZZ]:=u_medium_to_fine^[YY,ZZ]+v1_medium_corrected_to_fine^[YY,ZZ];
                inc(YY,1);
                end; {YY}
        inc(ZZ,1);
        end; {ZZ}
          writeln('got past u+v1');

{**************end add correction term v1 to original u   step 14****}
 {******************run fine grid to smooth ****************************}
 for counter:=0 to 50 do
 begin
 for ZZ:=0 to 96 do
 begin
    u^[0,ZZ]:=0.0;
    u^[96,ZZ]:=0.0;
 end;
 for YY:=0 to 96 do
 begin
    u^[YY,0]:=0.0;
     u^[YY,96]:=0.0;
 end;
 for ZZ:=1 to 95 do
 begin
 for YY:=1 to 95 do
 begin
        u^[YY,ZZ]:=(u^[YY+1,ZZ]+u^[YY-1,ZZ]
        +u^[YY,ZZ-1]+u^[YY,ZZ+1]+(1/96)*(1/96)*source_term^[YY,ZZ])
        /(4.0-alpha_term^[YY,ZZ]*(1/96)*(1/96));

 end;
 end;
 for ZZ:=0 to 96 do
 begin
    u^[0,ZZ]:=0.0;
    u^[96,ZZ]:=0.0;
 end;
 for YY:=0 to 96 do
 begin
    u^[YY,0]:=0.0;
     u^[YY,96]:=0.0;
 end;
 end;


   writeln('got past end');

{************end*run fine grid to smooth ****************************}
```

```
 write_all;
 gettime(hrr,minn,secc,hundd);
writeln('u^[48,48]=',u^[48,48]);

writeln('current time is: ',hr, ':', min, ':', sec, ':', hund);
writeln('current time is: ',hrr, ':', minn, ':', secc, ':', hundd);

{ **********END OVERALL LOOP**********}
 end;
 end;
 end;
```