Field Aligned Flow in 2-dimensional Magnetofluids and the Genetic Algorithmic Solution of Poisson's Equation

Jack Ireland B.Sc.

Thesis
submitted to the
University of Glasgow
for the degree
of Ph.D.

Department of Physics and Astronomy,
Glasgow University,
Glasgow, G12 8QQ.

June 15, 1995

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

To M+D, A+I

Stop me, oh-ho stop me,

Stop me if you think you've

heard this one before.

Stephen Morrissey (1987), from 'Strangeways, here we come'.

O all of you whose intellects are sound,

look now and see the meaning that is hidden

beneath the veil that covers my strange verses.

Dante (translated by Mark Musa, 1985), Inferno, *Canto IX*, *ll.* 61-63.

Abstract

This thesis splits naturally into two topics, field aligned flow in a two dimensional plasma, and the application of genetic algorithms to the solution of Poisson equations. Genetic algorithmic techniques were developed as a new method of numerical solution to a problem arising in field aligned flow. The relation between plasma physics and computing (particularly novel computing methods) is introduced in chapter 1.

In chapter 2, we begin with Maxwell's equations and a fluid description of a plasma, and derive under various assumptions, equations governing the structure of a field aligned two dimensional plasma. The appearance of field aligned flow in the Earth's magnetotail is discussed along with some treatments in the literature.

Chapter 3 examines the fields arising from having the fluid flow along the field lines time independent. It is shown that only very special fields support exact field aligned flow. These fields can be classed by their corresponding flow function. An equation is derived that describes fields where the flow function is a constant everywhere, which provides the spur for genetic algorithm application. Some magnetotail relevant solutions of this equation are presented.

Chapter 4 investigates time dependent field aligned flow. It is shown that this situation is somewhat more complicated than the time independent case, and that a singularity in the flow may appear, indicating the presence of a large fluid acceleration and the breakdown of the present model.

In chapter 5, the basic concepts of genetic algorithms are introduced. An algorithm is developed to test the efficacy of this method for application to the solution of a class of ordinary differential equations. This work is built on in chapter 6, where a Poisson equation solver is constructed. Comparisons are made between this and other more traditional methods.

Finally, chapter 7 describes some possible extensions to the work presented. Suggestions for both genetic algorithms and field aligned flow are discussed. Appendices A and B contain a listing of the Poisson solver POISGEN and sample input files respectively.

Acknowledgements

As I write, I do not know if I will have to prepare yet another draft of this thing, or if this is *really* The End. I wanted to leave writing this bit until the very end, but now is as good a time as any.

Firstly, I would like to thank Dr. Declan Diver and Professor Ernest Laing for their considerable help and encouragement over the last three years. I would also like to thank the University of Glasgow for awarding me a Postgraduate Scholarship, without which I would not have been here. Thanks must also go to the Astronomy and Astrophysics Group (incorporating what was known as the Plasma Theory Group) for putting up with numerous technical questions and even more bad puns and jokes.

Thanks especially to my family - Mum, Dad, Andrew and Patricia and Ingaret - for putting up with all the moaning and worry that comes free with every Ph.D.

I don't want this to read like the cover notes of a bad funk album, but there are a large number of people that I want to mention without whom, the last three years would not have been as much fun.

'The Office': Saad (for constantly asking awkward questions like 'Yes, but, what does it *mean*?'), Elizabeth, DavidK, DavidR, DavidA, Graeme, Luke, Brian, Gerry, Gordon, Stephen, Susan, Douglas and Sharon, Andrew, Commander Gary 'Kilgore' Benson, Caroline, Christine, Chris, Paul, Rebecca and Anne-Marie.

'Hamilton': John, Euan, Big Healy and Elinor.

'Aldol Rovers', for away fixtures and elbow training: Ian and Sandra, Ally and Jenny, Neil, Gubs, Peter, Alaistar.

'Everyone else who knows me...': Andy Ballantyne, (for the use of his flat), Sophie, (for postcards), The Dutch Class 1992-1994, (Heb *je* een leeuw in jouw tuin?), Paul French (for the occasional light refreshment here and there) and the Friday afternoon 1st year lab (Margaret, Peter, Kevin, Jenny et. al.) section 1993-1994 for a much needed weekly rest from work.

Contents

1 Plasma Physics and Computational Methods **1**

1.1 Controlled Thermonuclear Fusion 1

1.2 Astrophysical Plasmas 2

1.3 Industrial Uses 2

1.4 Computational Methods in Physics 3

2 The Fluid Description of a Field Aligned Plasma **6**

2.1 The Magnetohydrodynamic Equations 6

2.2 The Model Equations 8

2.3 Parallel Flow 11

2.3.1 The Geomagnetic Tail 11

2.3.2 Theoretical Studies of the Geomagnetic Tail 12

2.3.3 Negative Inertia 15

2.4 The Field Aligned Flow Equations 18

3 Time Independent Field Aligned Flow **20**

3.1 The Flow Function 20

3.2 A Bernoulli Equation 22

3.3 Solutions for special cases of the flow function 23

3.3.1 f=constant 23

3.3.2 $f = f(r)$ 25

3.3.3 $f = f(\theta)$ 27

3.4 Current free solutions 27

3.5 Application to a Problem in Field Aligned Flow 30

3.5.1 Magnetotail geometry and simple models 30

3.5.2 Perturbed magnetotail models 33

3.6 Summary and Conclusions 38

4 Time Dependent Field Aligned Flow **46**

4.1 The Flow Function and the Governing Equation 46

4.2 Solutions to the Full Flow Equation 47

4.2.1 $f = f(r,t)$ 47

4.2.2 $f = f(\theta,t)$ 48

4.2.3 $f = f(t)$ 49

4.2.4 A Linear Analysis of the Full Equation 49

4.3 A Nonlinear Flow Equation 52

4.3.1 Linear Analyses 54

4.3.2 Solution by separation of variables 54

4.3.3 $\alpha = constant$ and $\beta = constant$ 55

4.3.4 Assumed form for $\alpha(x), \beta(x)$ 57

4.3.5 Nonlinear Analysis 61

4.4 Summary and Conclusions 66

5 An Application of Genetic Algorithms to Differential Equations **70**

5.1 Finite Difference Solutions 70

5.2 The Genetic Algorithm Concept. 71

5.2.1 An example application of Genetic Algorithms 73

5.3 A Genetic Boundary Value Ordinary Differential Equation solver 74

5.3.1 Representation 75

5.3.2 Weighting Operator 77

5.3.3 Breeding, Mutation and Combination Operators 79

5.3.4 Convergence strategies 81

5.3.5 Experimental Results and Algorithm Behaviour 83

6 A Genetic Poisson Equation solver **95**

6.1 Poisson's Equation 95

6.1.1 A Matrix Method Routine: NAG D03EBF 95

6.1.2 Example application 97

6.2 Representation 99

6.3 Weighting Operator 99

6.4 Breeding, Mutation, Combination and Transcription Operators 101

6.5 Convergence strategies 103

6.6 Experimental Results and Algorithm Behaviour 104

6.7 Further comments on the genetic solution of Equation (6.25) 110

6.8 Summary and Conclusions 112

7 Future Work **122**

7.1 Field Aligned Flow 122

7.1.1 Computational Solutions 122

7.1.2 Analytical Work 123

7.1.3 Modelling 124

7.2 Genetic Algorithms 125

7.2.1 Extensions to POISGEN 125

7.2.2 Alternate Solution Representations 126

7.2.3 Other Differential Equations 128

A POISGEN: listing and documentation **130**

A.1 Outline of the code 130

A.1.1 Developmental History 130

A.1.2 Important variables 131

A.2 Description of POISGEN functions and subroutines 131

A.3 POISGEN listing 136

B POISGEN: sample input files and comments **162**

B.1 Description of input variables 163

B.2 Sample input files 167

List of Tables

4.1 Singularity behaviour for equation(4.62) 64
4.2 Singularity behaviour for equation(4.67) 67
4.3 Singularity behaviour for equation(4.65) 69

5.1 ODE variable values for Fig. (5.3),(5.4) 87
5.2 ODE variable values for Fig. (5.5),(5.6) 89
5.3 ODE variable values for Fig. (5.7),(5.8) 89
5.4 ODE variable values for Fig. (5.9),(5.10) 92
5.5 ODE variable values for Fig. (5.11),(5.12) 92
5.6 ODE variable values for Fig. (5.13),(5.14) 94

A.1 Some POISGEN variables 131

List of Figures

2.1 Plasma regions near the Earth 13
2.2 Viscous wake flow 17

3.1 Field (3.24) with $m = 5, p = 1$. 24
3.2 Field (3.38) with $c_1 = c_2 = 1$. 28
3.3 Field (3.38) with $c_1 = c_2 = 1$ and flow function $f = \sin[r\cos(c_1 - \theta)]$. 29
3.4 Idealised field structure for the magnetotail 31
3.5 Magnetotail model (3.54) with perturbed λ, ϵ negative 39
3.6 Magnetotail model (3.54) with perturbed λ, ϵ positive 39
3.7 Magnetotail model (3.54) with perturbed boundary conditions, δ positive 40
3.8 Magnetotail model (3.55) with perturbed λ, ϵ negative 40
3.9 Magnetotail model (3.55) with perturbed λ, ϵ positive 41
3.10 Magnetotail model (3.55) with perturbed λ, ϵ extreme and negative 41
3.11 Magnetotail model (3.55) with perturbed λ, ϵ extreme and positive 42
3.12 Magnetotail model (3.55) with perturbed boundary conditions, δ positive 42
3.13 Magnetotail model (3.55) with perturbed boundary conditions, δ negative 43
3.14 Magnetotail model (3.56) with perturbed λ, ϵ negative and within approximation . 43
3.15 Magnetotail model (3.56) with perturbed λ, ϵ positive and within approximation. . 44
3.16 Magnetotail model (3.56) with perturbed boundary conditions, δ positive 44
3.17 Magnetotail model (3.56) with perturbed boundary conditions, δ negative 45

4.1 $F(x)$, $\cos[Q_2(x)]$ for $\alpha_0 = 1$, $\alpha_1 = 1$, $\beta_0 = 10$, $\beta_1 = 1$, $\omega_R = 30.1$ and $f_0 = 1$ 60
4.2 Relationship between analyses in section 4.3 68

5.1 Scheme for general genetic algorithm 72
5.2 One point crossover breeding scheme 80
5.3 Genetic versus true solution for eqn.(5.17) 87
5.4 Best candidate weight at each iteration for eqn.(5.17) 88
5.5 Genetic versus true solution for eqn.(5.17) with large boundary penalties 88

5.6 Best candidate weight at each iteration for eqn.(5.17) with large boundary penalties 89
5.7 Genetic versus true solution for eqn.(5.17) with point doubling 90
5.8 Best candidate weight at each iteration for eqn.(5.17) with point doubling. 90
5.9 Genetic versus true solution for eqn.(5.17) with and without point doubling 91
5.10 Best candidate weight at each iteration for eqn.(5.17) for point doubling run in figure (5.9) 91
5.11 Genetic versus true solution for eqn.(5.19) 92
5.12 Best candidate weight at each iteration for eqn.(5.19) 93
5.13 Genetic versus true solution for eqn.(5.21) 93
5.14 Best candidate weight at each iteration for eqn.(5.21) 94

6.1 NAG solution to equation (6.9) when set up as $\nabla^2 A = k_2(x,y)A$ 98
6.2 Region of dependency for equations (6.21) to (6.25) 105
6.3 Genetic solution to equation (6.21) 106
6.4 NAG routine solution to (6.21) 106
6.5 Difference between (6.3) and (6.4) 107
6.6 Evolutionary history of fig.(6.3) 113
6.7 Genetic solution to eqn.(6.22) 113
6.8 NAG routine solution to eqn.(6.22) 114
6.9 Difference between fig.(6.7) and fig.(6.8) 114
6.10 Genetic solution to eqn.(6.23) 115
6.11 NAG routine solution to eqn.(6.23) 115
6.12 Difference between fig.(6.10) and (6.11) 116
6.13 Genetic solution to eqn.(6.24) 116
6.14 NAG routine solution to eqn.(6.24) 117
6.15 Difference between fig.(6.13) and fig.(6.14) 117
6.16 Genetic solution to eqn.(6.25) 118
6.17 NAG routine solution to eqn.(6.25) 118
6.18 Difference between fig.(6.16) and fig.(6.17) 119
6.19 Evolutionary history of figure (6.16) 119
6.20 Genetic solution to eqn.(6.25) with different range of gene values $lowbdry = -highbdry = -5$ 120
6.21 NAG routine solution to eqn.(6.25) with different range of gene values $lowbdry = -highbdry = -5$ 120
6.22 Difference between fig.(6.20) and fig.(6.21) 121
6.23 Evolutionary history of fig.(6.20) 121

A.1 Key to function and subroutine descriptions 132

B.1 Key to variable descriptions 163

Chapter 1

Plasma Physics and Computational Methods

Plasma physics as a distinct discipline arose in the 20th century, yet much of the basic physics - fluid mechanics, Maxwell's equations and the statistical description of a collection of particles, was developed in the 19th century. We have something of an anomaly - a modern discipline using largely 19th century physics. The emergence of easily available computing power making many previously incalculable problems tractable has helped to maintain interest in both plasma and computational physics. Careful calculation of plasmas of interest can act as valuable tool aiding theory and experiment [1]. In addition, the nature of genetic algorithmic techniques (first suggested in the 1950s) requires that we perform a large number of operations: such calculations and their application to relevant problems become possible only with fast enough computers.

Below is a brief summary of the larger fields of interest in plasma physics, and an introduction to some of the computational techniques that are currently being used to advance the subject.

1.1 Controlled Thermonuclear Fusion

Plasma physics is essential in understanding the processes involved in controlling thermonuclear fusion. The biggest experiment, JET at Culham, Oxfordshire, has recently run with a deuterium/tritium plasma, the candidate fuel mixture for a commercial fusion reactor. Many other experiments are underway around the world exploring alternative methods e.g., START (Small Tight Aspect Ratio Torus, also at Culham), JT-60U in Japan, WS-A7, a stellarator device in Germany.

All the above are examples of magnetic confinement, where magnetic fields are used to contain the (fusing) plasma. Inertial confinement seeks controlled thermonuclear fusion by the ablation of

small pellets of fuel mixture by extremely high energy laser beams. The plasmas produced here are much denser than those seen in magnetic confinement devices and hence the plasma physics can be very different.

Computer modelling plays a fundamental role in the design and costing of new machines or improvements to old ones. Recently, Boucher [2] reported on modelling work done on ITER (International Thermonuclear Experimental Reactor) concerning ignition, L and H modes, and divertor coupling.

1.2 Astrophysical Plasmas

Astrophysical plasmas provide many areas in which the subject may be applied e.g., the Sun. The Sun is an example of a successful thermonuclear reactor, and exhibits many phenomena of interest - the solar wind, sunspots, prominences, solar flares all are being actively studied.

Away from the Sun planetary and cometary magnetospheres, coupled with their solar wind interactions are coming under greater study, both theoretically, and observationally. Experiments have largely delineated the Earth's magnetosphere [3] and numerous theories exist to explain the phenomena in regions such as the magnetotail, auroral zones and the bow shock.

Pulsar magnetospheres provide an example of an *equal mass* plasma: in such a plasma, the positive and negative charged particles carry the same charge and have the same mass. A pulsar's magnetosphere is thought to consist largely of electrons and positrons. Using an equal mass plasma model Stewart [4] found that the Faraday rotation of the magnetic field from such a plasma is zero, which is observed experimentally from pulsars. The study of magnetised accretion discs represent important problems in black hole physics and planetary formation.

1.3 Industrial Uses

A large number of industrial processes use a plasma in one form or another, and their academic study, as opposed to their empirical use, is becoming more widespread [5]. Their application may be divided into three broad bands: information technology, materials science and environmental processing.

The information technology industry uses plasma processing to etch and deposit different layers of material on the integrated circuit substrate. This is closely linked with material science, which deals with engineering or modifying a surface with a plethora of techniques, for example, plasma vapour deposition, which seeks to deposit ionised material on a surface in a controllable manner. More crudely arc welding and smelting/refining are complicated, dusty plasma systems. Plasmas are also being used to treat noxious emissions e.g., nitrous (NOx) and sulphurous (SOx) oxides from

power stations, and also waste material; contaminated soil, tyres, chemical weapons and household refuse.

All these systems require better understanding and are of immediate practical use. In particular, the study of dusty plasmas, common in many applications technology, has many overlaps with astrophysical situations, such as interstellar gas and cometary outgassings.

1.4 Computational Methods in Physics

The computational simulation of many physical systems has become essential to many branches of physics, not least to plasmas. There are numerous techniques that may be used, depending on what is required, giving the computational physicist's job the reputation of being a 'black art'.

One approach is to solve the relevant magnetohydrodynamic equations numerically, using some form of discretisation of the differentials. Finite difference, finite element and finite volume methods are commonly used and find favour in many applications [6].

Finite difference schemes are perhaps the most popular form of numerical discretisation. The method is based upon the use of Taylor series expansions to build a toolkit of equations that describe the (partial) derivatives of a variable as the differences between values of the variable at different points in space and time. Consider the dependent variable U with independent variables x, t. We may consider two points a small distance in space h away from some central point x_0 on the x-axis. We also fix time at t_0. The Taylor series expansions for U at these two points are

$$\begin{aligned} U(x_0+h,t_0) &= U(x_0,t_0) + \frac{h}{1!}\frac{\partial U}{\partial x}(x_0,t_0) + \frac{h^2}{2!}\frac{\partial^2 U}{\partial x^2}(x_0,t_0) + \frac{h^3}{3!}\frac{\partial^3 U}{\partial x^3}(x_0,t_0) + \cdots \\ &= \exp\left[h\frac{\partial}{\partial x}\right] U(x_0,t_0) \end{aligned} \tag{1.1}$$

and

$$\begin{aligned} U(x_0-h,t_0) &= U(x_0,t_0) - \frac{h}{1!}\frac{\partial U}{\partial x}(x_0,t_0) + \frac{h^2}{2!}\frac{\partial^2 U}{\partial x^2}(x_0,t_0) - \frac{h^3}{3!}\frac{\partial^3 U}{\partial x^3}(x_0,t_0) + \cdots \\ &= \exp\left[-h\frac{\partial}{\partial x}\right] U(x_0,t_0) \end{aligned} \tag{1.2}$$

Here we have used the notation of Mitchell and Griffiths [7] to compactly write the Taylor expansion in terms of the exponential of the differential operator. By adding and subtracting these two equations, new equations can be found for the first and second partial space derivative of U at the central point x_0,

$$\frac{\partial^2 U}{\partial x^2} = \frac{U(x_0+h,t_0) - 2U(x_0,t_0) + U(x_0-h,t_0)}{h^2} + O\left(h^2\right) \tag{1.3}$$

and

$$\frac{\partial U}{\partial x} = \frac{U(x_0+h,t_0) - U(x_0-h,t_0)}{2h} + O\left(h^2\right) \tag{1.4}$$

where $O(h^n)$ denotes terms of order h^n and higher. These expressions are known as the centred difference replacements for $\frac{\partial U}{\partial x}$ and $\frac{\partial^2 U}{\partial x^2}$. Other estimates are available from the Taylor expansions; if we truncate (1.1) up to order h then we have

$$\frac{\partial U}{\partial x} = \frac{U(x_0 + h, t_0) - U(x_0, t_0)}{h} + O(h) \tag{1.5}$$

which is accurate up to order h. These difference formulæ allow differential equations to be described at points in the region of interest, but at the expense of a maximum accuracy of $O(h^n)$. Which replacement to use in a particular situation is not immediately clear, which can lead to false answers or even the breakdown of the method. The finite difference replacements are used at the core of the genetic algorithms described in chapters 5 and 6.

Radically new techniques borrowed from computing science are just beginning to make their presence felt in computational physics. Principal among these are *neural networks*, *cellular automata* and *genetic algorithms* [8]. Cellular automata and genetic algorithms are examples of *artificial life*: in both cases the basic ideas are borrowed from the way nature evolves and creates new organisms. Neural networks were born out of *artificial intelligence* research and draw their inspiration from the structure of the human brain. A human brain has a large number of cells called neurons that are thought to be actively involved in information processing. Each of these cells also has a number of connections to other cells. It is thought that this structure enables the brain to process large amounts of information efficiently to provide for example, pattern recognition and sight, tasks that are notoriously difficult implement artificially. A neural network works similarly, by assigning a weighting to each one of the interconnections between nodes. A node is analogous to a neuron, and some active processing of input data may occur here. The weighting values are determined by giving the network some test input data, for which a correct output is known. The output from the network is compared to the known data, and the errors are be used to refine the weightings. This is known as the training phase. This allows the network to 'learn' something about the problem. After training, the network may be used as a problem solver on real input data.

These computing techniques are beginning to find applications in problems that require some form pattern seeking or matching. A neural network has been used to analyse the charge exchange recombination data from DIII-D (a magnetic confinement device at General Atomics, San Diego) to provide information on ion temperatures and rotation velocities [9] . The problem is to separate out and analyse the correct peaks from a large number (up to $\sim 200,000$ a day) of multipeaked spectra. This type of problem is ideal for the pattern recognition qualities of a neural network, and given a fast enough computer, the authors can analyse a day's spectral results in 100 seconds. Another application of neural networks is to the prediction of sunspot number [10].

A promising development is the use of *cellular automata* in fluid and plasma simulations. Cellular automata are based on the behaviour of colonies of cells growing in culture [11] . The creation of a new cell, or equally, the destruction of an existing cell, depends on what is happening locally

to the cell. If there are too many cells in a particular region, then resources are placed under great demand which prohibits the growth of new cells, and may even kill off existing ones. Hence there emerges a complicated overall colony behaviour that has been determined by purely local rules.

This has been successfully recreated computationally. Instead of cells, each unit in the system is termed an automaton. In the early simulations, a set of local rules was defined governing the creation of a new automaton.

The advantage to magnetohydrodynamic simulation is that the local nature of the cellular automata approach makes it very easily parallelisable; each node on a parallel machine could handle a fluid element, the local interaction rules governing the influence of one element on another [12].

Genetic algorithms will be discussed in some detail in chapter 5, but the basic philosophy is simple. Taking a lead from Darwinian evolution, a genetic algorithm solves problems by breeding successively better answers from the best of the previous generation. They have been applied successfully to a number of problems, in particular classifier systems (for machine learning) and combinatorial problems. In classifier system problems [13], a genetic algorithm is used to discern rules of behaviour of some complicated system. Less successful rules are eliminated, and the computer learns an optimised (but not necessarily optimum) set of rules for the problem. Such programs have found application in economics modelling [14] and maze learning [13]. The travelling salesman problem is a famous example of a combinatorial problem (see section 5.2.1). Timetabling is a combinatorial problem that has been successfully attacked with a genetic algorithm: indeed, the Department of Artificial Intelligence at the University of Edinburgh now timetables its M.Sc. exams using a genetic algorithm[13].

Chapter 2

The Fluid Description of a Field Aligned Plasma

In this chapter we introduce a set of magnetohydrodynamic equations and specialise them to the case of 2 dimensional field aligned flow. Section 2.3 discusses the appearance of field aligned flow in the Earth's magnetotail. A short review of some relevant field aligned flow treatments in the literature is also given.

2.1 The Magnetohydrodynamic Equations

A plasma is collection of interacting particles [15] that exhibits a collective behaviour, and as such, we can write down an equation of motion for each of these particles. For a collection of N particles there are $6N$ values to be determined - the three space co-ordinates and three velocity components of each particle. Hence we must have $6N$ equations to fully describe the system. It is obviously impractical to attempt to solve such a set of equations. Therefore we must somehow cut down the number of variables required to adequately describe the plasma. The exact meaning of 'adequately' depends largely on the type of plasma or on the type of problem being examined. In this case we shall use a set of magnetohydrodynamic equations. This is because we wish in principle to examine the overall fluid motion of the plasma in a field aligned flow situation.

Many textbooks detail the necessary manipulations that may be performed to reduce a full particle by particle description down to a magnetohydrodynamic system. The derivation of such a set of equations will not be repeated here, but the interested reader is invited to consult [16], [17].

By resorting to a magnetohydrodynamic description, we are in some sense smearing out the

particulate nature of a plasma. Often the predictions concur with experiment, which is justification enough for a magnetohydrodynamic approach. This is taken as evidence that the system is acting in a magnetohydrodynamic manner.

Since we are dealing with an ionised medium Maxwell equations must hold, i.e.,

$$\nabla.\mathbf{B} = \mathbf{0} \tag{2.1}$$

$$\nabla.\mathbf{D} = \rho_e \tag{2.2}$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{j} \tag{2.3}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.4}$$

In this set $\mathbf{E}$ is the electric field and $\mathbf{D} = \underline{\underline{\epsilon}}.\mathbf{E} + \mathbf{P}$ is the electric displacement. $\mathbf{P}$ is the electric polarization of the medium and $\underline{\underline{\epsilon}}$ is the dielectric tensor. $\rho_e = \rho_f + \rho_b$ is the total electric charge density, where ρ_f is the free charge density and $\rho_b = -\nabla.\mathbf{P}$ is the bound charge density.

We will assume that the material is isotropic and therefore $\underline{\underline{\epsilon}} = \epsilon_0 \mathbf{I}$ where $\mathbf{I}$ is the identity tensor and ϵ_0 is the permittivity of free space. We also take $\mathbf{P} = \mathbf{0}$.

$\mathbf{H}$ is magnetic field strength and $\mathbf{B}$ is the magnetic flux density [18, 19]. Again these are linked via $\mathbf{B} = \mu_0 \mathbf{H} + \mathbf{M}$ where μ_0 is the permeability of free space and $\mathbf{M}$ is the magnetisation of the medium. Already we have made some major assumptions in denying the inherent anisotropy of the medium. Such changes in the dielectric nature of the medium can be handled in different treatments and again show the need for different models (for example, the passage of an electromagnetic wave in a plasma medium [20]). The question we want to answer guides the choice of model. Since we are more concerned about the motion of the plasma, rather than its electromagnetic properties, we shall use the reduced set with no magnetisation or electric polarization and with the permittivity and permeability set to their free space values.

The remaining quantity to be defined is $\mathbf{j}$, the total current density. The expression governing $\mathbf{j}$ in a plasma is usually called Ohm's Law [21] and takes on a form as required by the situation. The plasma we are considering here is one in which the numbers of positively and negatively charged particles are roughly equal. We also assume that the mass of the negative ion m_- is very much less than the mass of the positive ion m_+. The plasma is also assumed to be collisional; i.e., there are a large number of particle-particle interactions of one form or another. Although in general the pressure is a tensor, under the collisional assumption, the pressure may be taken to be isotropic. Under these assumptions it may be shown that

$$\frac{m_+ m_-}{\rho e^2}\frac{\partial \mathbf{j}}{\partial t} = \mathbf{E} + \mathbf{u} \times \mathbf{B} + \frac{m_+}{\rho e}\mathbf{j} \times \mathbf{B} + \frac{m_+}{2\rho e}\nabla p - \frac{1}{\sigma}\mathbf{j} \tag{2.5}$$

This is known as the generalised Ohm's Law, and will be reduced to a more appropriate form later. The quantity σ is the electrical conductivity of the plasma. Charge is conserved in this plasma;

therefore on defining $q = n_- e^- + n_+ e^+$ as the charge density in the plasma, one can show that

$$\frac{\partial q}{\partial t} + \nabla.\mathbf{j} = 0 \tag{2.6}$$

(In relativistic plasmas, charge need not be conserved, as pair production can change charge density).

Having smeared out the particles into a fluid, we must now describe it. The equation of mass conservation states that

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho \mathbf{u}) = 0 \tag{2.7}$$

The equation of charge conservation follows from the Maxwell Equations. The fluid may also undergo acceleration, which is described by

$$\rho \frac{D\mathbf{u}}{Dt} = -\nabla p + \mathbf{j} \times \mathbf{B} \tag{2.8}$$

where $\frac{D}{Dt} = \frac{\partial}{\partial t} + (\mathbf{u}.\nabla)$. Note that this is the isotropic pressure version of the momentum equation, and that in general we should replace ∇p with $\nabla.\mathbf{p}$, where $\mathbf{p}$ is the pressure tensor. Assuming adiabaticity in the gaseous plasma we can write

$$\frac{D}{Dt}\left(p\rho^{-\gamma}\right) = 0 \tag{2.9}$$

where $\gamma = c_p/c_v$, the ratio of the specific heats. Using equation (2.7) we can rewrite this as

$$\frac{\partial p}{\partial t} + \mathbf{u}.\nabla p + p\gamma\nabla.\mathbf{u} = 0 \tag{2.10}$$

This particular form will be important later.

The equations (2.1) to (2.9) form a closed set of fifteen scalar equations in $q, \rho, P, \mathbf{u}, \mathbf{j}, \mathbf{E}$ and $\mathbf{B}$.

2.2 The Model Equations

We now wish to reduce the magnetohydrodynamic set above to a more manageable form. Firstly, a dimensional analysis of (2.4) reveals that

$$\frac{E}{B} \sim \frac{\omega L}{c}$$

where L and ω^{-1} are typical length and time scales over which the fields change appreciably. Now in this fluid description we expect that the fluid flow and the electromagnetic fields to interact: to change $\mathbf{E}, \mathbf{B}$ significantly we would expect that

$$U \sim \omega L$$

where U is a typical flow speed. If we assume that $\frac{U}{c} \ll 1$ then we are restricted to non-relativistic regimes. This is a reasonable assumption as (2.8) is not relativistically invariant, and in any case we do not wish to look at relativistic effects. If we now examine (2.3) dimensionally we find that

$$\frac{\left|\frac{\partial \mathbf{E}}{\partial t}\right|}{|\nabla \times \mathbf{B}|} \sim \left(\frac{\omega L}{c}\right)^2 \ll 1$$

Hence in this limit of low velocity we may write

$$\mu_0 \mathbf{j} = \nabla \times \mathbf{B} \tag{2.11}$$

Consider now the terms in (2.5). In the order they appear they may be written as (the left hand side of the equation is the first term in the list below)

$$\left(\frac{\omega}{\omega_p}\right)^2 \left(\frac{c}{U}\right)^2 : 1 : 1 : \left(\frac{\omega}{\omega_p}\right)\left(\frac{\Omega_e}{\omega_p}\right)\left(\frac{c}{U}\right)^2 : \left(\frac{\omega}{\Omega_i}\right)\left(\frac{c_s}{U}\right)^2 : \left(\frac{\omega}{\omega_p}\right)\left(\frac{\nu_c}{\omega_p}\right)\left(\frac{c}{U}\right)^2 \tag{2.12}$$

where

$$\Omega_e = \frac{e^- B}{m^-}, \quad \Omega_i = \frac{e^+ B}{m^+}, \quad c_s \sim \sqrt{\frac{P}{\rho}} \tag{2.13}$$

are the cyclotron frequencies of the positive and negative species, the sound speed and $\omega_p^2 = \frac{n_- e^2}{m_- \epsilon_0}$ is the electron plasma frequency. The quantity ν_c is the negative-positive collision frequency.

The first term may be neglected if

$$\frac{\omega}{\omega_p} \ll \frac{U}{c}$$

The $\mathbf{j} \times \mathbf{B}$ term in (2.5) may be ignored if

$$\frac{\omega \Omega_e}{\omega_p^2} \ll \left(\frac{U}{c}\right)^2$$

The pressure dependent part may also be removed when

$$\frac{\omega}{\Omega_i} \ll \left(\frac{U}{c_s}\right)^2$$

This leaves (2.5) as

$$\mu_0 \mathbf{j} = \sigma \left(\mathbf{E} + \mathbf{u} \times \mathbf{B}\right) \tag{2.14}$$

The quantity σ is the electrical conductivity of the plasma, which we will take to be constant. If we take the curl of (2.11) and substitute in $\mathbf{E}$ as defined from (2.14) we have

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{u} \times \mathbf{B}) + \eta \nabla^2 \mathbf{B} \tag{2.15}$$

where $\eta = (\mu_0 \sigma)^{-1}$, the magnetic diffusitivity. Equation (2.15) is known as the magnetic induction equation for constant η. This can be seen as an evolution equation for $\mathbf{B}$ in much the same way that (2.8) can be seen as an evolution equation for $\mathbf{u}$. The equivalence is more easily seen on rewriting (2.15) as

$$\frac{D\mathbf{B}}{Dt} = (\mathbf{B}.\nabla)\mathbf{u} + \eta \nabla^2 \mathbf{B} \tag{2.16}$$

This is basically a diffusion equation with an extra 'dynamo' term. (This term is very important in studies of the Earth's core).

If we assume further that the plasma is a perfect conductor i.e., infinite conductivity then (2.14) and (2.16) become

$$\mathbf{E} + \mathbf{u} \times \mathbf{B} = \mathbf{0} \tag{2.17}$$

$$\frac{D\mathbf{B}}{Dt} = (\mathbf{B}.\nabla)\mathbf{u} \tag{2.18}$$

The plasma is now said to be *ideal*. A perfectly conducting plasma implies that

$$\frac{\omega \nu_c}{\omega_p^2} \ll \left(\frac{U}{c}\right)^2$$

in (2.12); in some sense, the collision frequency is small, which is consistent with the idea of a very good conductor.

Equations (2.7), (2.8), (2.11), (2.18) and (2.9) are the equations of ideal hydromagnetics. These equations form the basis of the papers described in section 2.3. We will work with a still further reduced set.

Setting $\rho = constant$ simplifies the set drastically. Firstly, the fluid is now incompressible since its density is everywhere fixed. This reduces (2.7) to

$$\nabla.\mathbf{u} = 0 \tag{2.19}$$

the familiar incompressibility condition. This arises as a result of constant density. In (2.9), it appears that the pressure must now be convected along by equation (2.10). However, this is not the case. Incompressibility also corresponds to $\gamma \rightarrow \infty$, meaning that equation (2.10) can no longer govern the pressure. The pressure is no longer an independent variable and is completely specified by equation (2.8). The pressure at any point in the fluid is now completely specified by the magnetic and velocity fields, and has no life of its own.

Collecting the relevant equations together we shall use

$$\nabla.\mathbf{B} = 0 \tag{2.20}$$

$$\rho \frac{D}{Dt}\mathbf{u} = -\nabla p + \mathbf{j} \times \mathbf{B} \tag{2.21}$$

$$\frac{D}{Dt}\mathbf{B} = (\mathbf{B}.\nabla)\mathbf{u} \tag{2.22}$$

$$\mu_0 \mathbf{j} = \nabla \times \mathbf{B} \tag{2.23}$$

to describe the plasma.

2.3 Parallel Flow

In a field aligned plasma, we assume that the velocity $\mathbf{u}$ is everywhere parallel (or anti-parallel) to $\mathbf{B}$, the magnetic field direction at that point, i.e.,

$$\mathbf{u} \| \mathbf{B}$$

or, introducing a function of proportionality,

$$\mathbf{u} = F\mathbf{B} \tag{2.24}$$

The function F may be related to the magnetic Alfvén Mach number M_A, defined as

$$M_A^2 = \frac{\mathbf{u}.\mathbf{u}}{v_A^2} = \frac{\mu_0 \rho \mathbf{u}.\mathbf{u}}{\mathbf{B}.\mathbf{B}} \tag{2.25}$$

The Alfvén velocity is given by $v_A^2 = \frac{\mathbf{B}.\mathbf{B}}{\mu_0 \rho}$. With the above field aligned relation between $\mathbf{u}$ and $\mathbf{B}$ it can be seen that $M_A^2 = \mu_0 \rho F^2$. The function F carries directional information in its sign which is lost in the definition M_A. The sign of F will be seen to be important in chapter 4.

The following sections (2.3.1–2.3.3) review some features of parallel flow already seen in the literature.

2.3.1 The Geomagnetic Tail

Field aligned flow has been observed experimentally, principally in the boundary layer and plasma sheet of the Earth's *magnetotail*, or geomagnetic tail. To properly delineate these regions, we first describe the magnetic structures surrounding the Earth.

It was not realised that the Earth was surrounded by a complex plasma system interacting with the solar wind until the discovery of the Van Allen radiation belts in 1958 [22]. Since then many experimental observations have mapped the various regions of the Earth's magnetic influence (see [23] descriptions of some of the observations). The region of space around the Earth may be separated into three main regions (see figure 2.1). First of these is the region where the solar wind dominates. As the wind impinges on the magnetic field of the Earth, it creates a bow shock, lying upstream of the Earth on the sunward side. This represents the boundary between the solar wind and the second region, the magnetosheath. The magnetosheath contains initially compressed and subsonic, sometimes turbulent plasma that expands to super-Alfvénic speeds as it flows along the boundary between the third region - the magnetosphere - and the magnetosheath. The magnetosheath is characterised by discontinuities in plasma parameters, including the magnetic field $\mathbf{B}$.

The magnetosphere itself may also be subdivided into many regions, as is shown in figure (2.1). It is the region where the Earth's magnetic field dominates. As one moves out from the

Earth, downwind from the Sun, the field elongates to form the geomagnetic tail, or the Earth's magnetotail.

The magnetotail is formed by the elongated remainder of the Earth's magnetic field, stretching out on the Earth's nightside. Structures have been observed on the scale $(10 - 20)R_{earth}$ at a distance $\sim 30R_{earth}$ from the Earth, and it is clearly perceptible up to $\sim 80R_{earth}$, although tail phenomena have been observed up to $\sim 1000R_{earth}$. The neutral sheet in the geomagnetic tail is about $(0.1 - 1)R_{earth}$ wide and separates regions in which the magnetic field points in opposing directions [22]. The neutral sheet is also known as the current sheet, due to the large current associated with it. The boundary layer is differentiated from the plasma sheet by the presence of streaming ion beams, travelling towards and away from the Sun above and below the neutral sheet [24]. The central plasma sheet is characterised by a more isotropic electron and proton distribution than is seen in the boundary layer.

The magnetotail may also influence events closer to the Earth. Electron -proton islands in the magnetotail have been observed with energy spectra very close to those seen in auroræ[22]. This suggests that the magnetotail may be important in determining auroral flux behaviour. There is also evidence for the existence of topologically isolated regions of magnetic field, distinct from the background magnetotail. Therefore, there are X-point type configurations present, which suggest reconnection may play a role in the system [25].

The boundary layer and plasma sheet are of principal interest, as it is here that field aligned flow is observed experimentally. Eastman *et al.* [26] describe the presence of high speed field aligned plasma flow in the boundary layer of the magnetotail. DeCoster and Frank [27] observed protons moving faster than $400kms^{-1}$ in the boundary layer, in a manner consistent with parallel flow. Eastman *et al* [24] states that the transport properties of the plasma sheet and its boundary layer make it a very important region for the overall modelling of the magnetosphere. With this in mind, we turn now to some modelling studies of the magnetotail, and their use of field aligned flow.

2.3.2 Theoretical Studies of the Geomagnetic Tail

The study of the effect of field aligned flow in a plasma has largely been confined to the use magnetohydrodynamic equations. Gebhardt and Kiessling's 1992 paper [28] studies 3d steady, incompressible ideal MHD model with field aligned flow. By using the Euler potential representation of a magnetic field $\mathbf{B} = \nabla\alpha(x, y, z) \times \nabla\beta(x, y, z)$ they describe a method to find the magnetic and velocity fields given the appropriate boundary and pressure information. From this one can define a function of proportionality designating how much faster the fluid flow is compared to the local Alfvén velocity (for each field line) by using the parallel specification (2.24) above. However, the authors consider flows in which the function of proportionality for each field line is constant

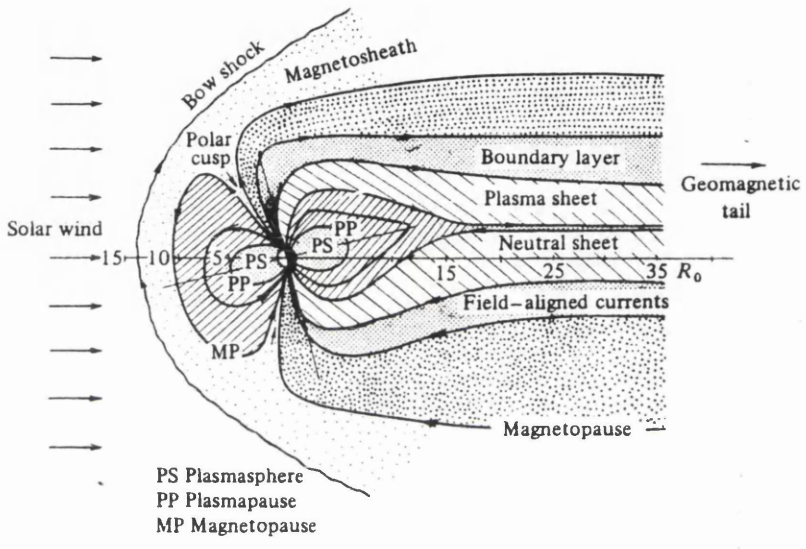

Figure 2.1: Plasma regions near the Earth

on that field line. As they consider steady state equations, the possibility of time dependent flow along the field lines is not admitted.

It is shown that for super-Alfvénic flows with $M_A^2 > 1$ the kinetic energy dominates the system and the principles of stationary hydromechanics (SHM) for an incompressible fluid apply. When $M_A^2 < 1$ the flow is sub-Alfvénic and the system of equations become equivalent to magnetohydrostatics (MHS). Physically, one may say that in those regions where the magnetic energy density dominates the kinetic energy, the magnetohydrodynamic flow *structure* of the fluid is governed by the principles that control MHS. If kinetic energy dominates in a region, then the rules of SHM apply. Transitions from sub- to super-Alfvénic flows mean that we move from one equivalence to the other. The analysis in the paper is generalised and properly includes the transitional $M_A^2 = 1$ case. This shows the special case of material moving at the Alfvén velocity, which will be treated in chapter 3.

Schindler and Birn [29] describe a mechanism to generate field aligned flow at the boundary of the plasma sheet in the Earths' magnetotail. Using a set of 2d steady incompressible magnetohydrodynamic equations they describe the generation of parallel flow from an X-point reconnection diffusion region. Such reconnection events are thought to occur in the magnetotail [3, 25], and may be responsible for the formation of large blobs of material, known as plasmoids, in the magnetotail. They find that parallel flow along the separatrix is the dominant form of transport from such a reconnection region. Flux tubes emerging from regions close to the non-ideal reconnection zone will have weak magnetic fields which increase in strength with distance from the X-point. As the field increases in strength, the frozen in approximation means that the flux tubes decrease in volume, collimating the flow along the field. The analysis relies on a smallness parameter that permits them to drop the inertia term $\rho\,(\mathbf{u}.\nabla)\,\mathbf{u}$ in the momentum balance equation. Since this paper attempts to model an experimentally measured system, it has something less to say about the structure of field aligned magnetohydrodynamics than Gebhardt and Kiessling. However, after modelling the fields for a magnetotail, they calculate a field aligned flow of a magnitude concomitant with observation.

This lends support to the idea of plasmoids forming by reconnection. Two consecutive X-points in the magnetotail would create a closed blob of material. The magnetotail also gives a context to Birn's work on field aligned flow in stretched 3d equilibria. Experimental studies show that the magnetotail is strongly field aligned, a fact that is included in [30] as a generalisation of his paper on magnetotail equilibria, [31]. The author assumes that the magnetic field varies strongly in one space direction only, and weakly in the other two, using a set of 3d ideal, steady magnetohydrodynamic equations. Plasmoids are also formed, travelling at speeds "not inconsistent" with observation. Plasmoids are studied by Young and Hameiri [32] in a 2 dimensional steady field aligned model of the magnetotail.

Lee and Yan [33] go further with field aligned flow in a reconnection context. They examine a 2 dimensional incompressible magnetohydrodynamic simulation, set up to model magnetic separatrices in a reconnecting plasma. They find that the field aligned plasma jets are formed slightly downstream of the magnetic separatrices by a combination of a slow shock wave and a 'compressional' structure: by compressional they mean a structure in which the plasma pressure increases along the streamline. This compression resemble a fast Alfvén mode if the magnetic field decreases with increasing plasma pressure: a slow Alfvén type mode is seen if the magnetic field increases with increasing plasma pressure. The shock accelerates the fluid by converting magnetic energy into kinetic and thermal energy. The 'fast mode' compression structure (located downstream of the shock) decelerates the plasma, converting kinetic energy into thermal and magnetic energy. This collimates the flow with the field so that it is nearly field aligned.

A lot of the work is modelling led which, in the case of the magnetotail, permits one to use the geometry of the magnetic fields to reduce the complexity of the problem. In this thesis we shall examine field aligned flow in a 2 dimensional magnetofluid, but with the minimum number of assumptions on field shape. Also, we shall consider time dependent field aligned flow, and its consequences for fluid acceleration and flow geometries. We will *not* look for any mechanisms to generate field aligned flow, but rather look at the effect parallel flow has on the shape and nature of the plasma.

2.3.3 Negative Inertia

The concept of *negative inertia* is an unusual one, but is a direct result of field aligned flow. Shercliff [34] describes and names this effect which arises in incompressible, ideal and steady magnetohydrodynamics. The argument describing this term will be reproduced here, as it serves to point out that field aligned flow is an unusual situation.

We shall start with an ideal, steady plasma, fluid density ρ constant. Equation (2.21) may be rewritten as

$$\rho\,(\mathbf{u}.\nabla)\,\mathbf{u} = -\nabla p + \mathbf{j}\times\mathbf{B} \tag{2.26}$$

By the argument of section 2.2, the pressure is now defined by equation (2.26) and has no independent life of its own. If we take the curl of this, and further set $\mathbf{B} = \alpha\mathbf{u}$, where α is a constant, then we obtain the equation [35]

$$\left(\rho - \frac{\alpha^2}{\mu_0}\right)\nabla\times(\boldsymbol{\Psi}\times\mathbf{u}) = 0 \tag{2.27}$$

where $\boldsymbol{\Psi} = \nabla\times\mathbf{u}$, the fluid vorticity. If $\alpha^2 = \mu_0\rho$ then equation (2.27) does not restrict the choice of flow, and any velocity field will do. However, if $\alpha^2 \neq \mu_0\rho$ then we have a restricted choice of fields, governed by $\nabla\times(\boldsymbol{\Psi}\times\mathbf{u}) = 0$. (This will be described in more detail in chapter 3.)

Suppose we now introduce a force $\mathbf{F}$ $\nabla \times \mathbf{F} \neq \mathbf{0}$, to the momentum equation (2.26),

$$\rho(\mathbf{\Psi} \times \mathbf{u}) + \nabla\left[p + \frac{\rho u^2}{2}\right] = \frac{\alpha^2}{\mu_0}(\mathbf{\Psi} \times \mathbf{u}) + \mathbf{F} \tag{2.28}$$

On taking the curl and defining $\rho^* = \rho - \frac{\alpha^2}{\mu_0}$ we obtain

$$\rho^* \nabla \times (\mathbf{\Psi} \times \mathbf{u}) = \nabla \times \mathbf{F} \tag{2.29}$$

ρ^* is the effective density, and it can be positive or negative, depending on the size of α. Let v be a typical velocity at a point in the fluid, and $b = \frac{B_0}{\sqrt{\mu_0 \rho}}$, the Alfvén velocity. B_0 is the field strength at that point. If $v/b > 1$ then by the field aligned specification above, $\alpha^2 < \mu_0 \rho$ and ρ^* is positive. This means that by equation (2.29) the vorticity increases in the same direction as the rotationality of the force $\mathbf{F}$ - which is entirely expected. However suppose we have $v/b < 1$. This implies that $\alpha^2 > \mu_0 \rho$, making ρ^* negative. If we consider a fluid element, equation (2.29) now offers the uncomfortable prospect of such an element increasing its spin in the opposite direction from the torques applied to it. The fluid seems to have negative inertia. The origin of this term is most easily understood by considering the simple equation $\Gamma = I\ddot{\theta}$ [36]. For the angular acceleration $\ddot{\theta}$ to have the opposite sign to the torque Γ, the inertia I must be negative.

When ρ^* is negative, $v < b$, and by Gebhardt and Kiessling [28] we are in a magnetohydrostatically dominated regime. Hence, the balance of forces is mainly between $\mathbf{F}$ and $\mathbf{j} \times \mathbf{B}$. Therefore, by the 'frozen in' approximation of ideal magnetohydrodynamics, the fluid must move in such a way as to deform the magnetic field into a configuration that largely balances $\mathbf{F}$ with $\mathbf{j} \times \mathbf{B}$ (since we are taking inertial forces to be unimportant compared to magnetic forces). What the fluid actually does when $\rho^* < 0$ can be seen by writing $\rho^* = -q$, where $q > 0$. Multiplying both sides of equation (2.29) by -1 yields

$$q \nabla \times (\mathbf{\Psi} \times \mathbf{u}) = \nabla \times (-\mathbf{F}) \tag{2.30}$$

This looks like the equation governing the vorticity for a fluid of density q under the action of a force $-\mathbf{F}$. A fluid of negative effective density with forces $\mathbf{F}$ will arrange itself to look like a fluid of density q with forces $-\mathbf{F}$.

This is most easily seen when we use a concrete example. Suppose $\mathbf{F} = \eta \nabla^2 \mathbf{u}$ i.e, the fluid is viscous. The momentum balance equation can be written as

$$\rho^* (\mathbf{u}.\nabla)\mathbf{u} = -\nabla p^* + \eta \nabla^2 (\mathbf{u}) \tag{2.31}$$

where $p^* = p + \frac{\rho u^2}{2}$. Then provided ρ^* is positive, the fluid behaves like any other of density ρ^* and viscosity η. The tension in the field lines lowers the effective inertia by helping the viscous forces change the vorticity. In laminar viscous flow close to a plate, the fluid velocity increases from zero as we move away from the plate. The sliding layers of different speed create vorticity in

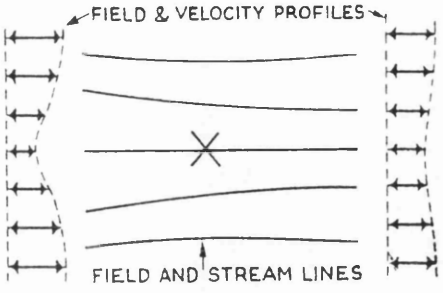

Figure 2.2: Viscous wake flow

the fluid. Since we have a field aligned flow, the tension in the field lines represents a torque and this also creates vorticity. Additionally, the viscous layer close to the plate where this differential sliding of the layers of fluid occurs increases. As ρ^* decreases to zero then the inertial term in (2.30) becomes less important in comparison and the diffusion of vorticity proceeds more quickly. At $\rho^* = 0$, the fluid must move so that the force $\mathbf{F}$ is irrotational; in this case $\mathbf{u}$ is irrotational.

As the magnetic field becomes stronger still, ρ^* becomes negative, and negative inertial effects begin to appear. The momentum balance equation now becomes

$$-\rho^* (-\mathbf{u}.\nabla)(-\mathbf{u}) = -\nabla(-p^*) + \eta\nabla^2(-\mathbf{u}) \tag{2.31}$$

The motion is exactly like that of an ordinary fluid of density $-\rho^*$, viscosity η travelling in the opposite direction. If we consider a viscous wake flow as shown in figure (2.2) then we can see the effect of this flow reversal. In the case $v \gg b$, we are in ordinary hydrodynamics. Figure (2.2) represents a flow from left to right. The velocity profile will flatten out due to viscosity. The case $v \ll b$ demonstrates negative inertia. The fluid must move to convect the field into a form so that the tensions can balance the viscous forces. A fluid element at the point X experiences a viscous force to the left. The field gains a $y-$component, allowing force balance between $\mathbf{j} \times \mathbf{B}$ and viscosity $\eta\nabla^2\mathbf{v}$ (remember that explicit mention of $\mathbf{j} \times \mathbf{B}$ forces will be subsumed by the field aligned specification and vector identity manipulation in equation (2.30).) The field line has now changed the direction in which it is pointing, which means that energy must have been used to do this. Since there are no sources of energy in this system. the energy to do this must have come from the velocity available at X. Therefore, we must have slowed down the fluid along this field line. Hence the streamwise viscous forces decelerate the fluid at X, giving the illusion of negative inertia. We have therefore steepened the velocity profile, where we would expect viscosity to level the profile out.

This is a rather subtle effect and we shall not consider it directly in much of what follows, as

we set $\mathbf{F} = 0$ for the purposes of this work. However, it may be implicated in other results where we change the direction of the fluid flow relative to the background field (see chapter 4).

2.4 The Field Aligned Flow Equations

In section (2.2), we derived a set of magnetohydrodynamic equations describing a plasma, subject to our assumptions. We use cylindrical polar co-ordinates throughout, although Cartesian co-ordinates are entirely equivalent. This is motivated by the geometry of the systems we wish to examine; looking from above, a torus has an annular cross section. Solar flares on the Sun and smaller events have arc shapes, and hence it is natural to consider first cylindrical polar co-ordinates. Cartesian co-ordinates may be employed in magnetotail applications.

We specialise the magnetohydrodynamic equations by dropping all z-dependence in all quantities and setting $\mathbf{u} = (u_r, u_\theta, 0)$ and $\mathbf{B} = (B_r, B_\theta, 0)$. To remove explicit mention of p, the fluid pressure we take the curl of equation (2.21). Note that both $\mathbf{u}$ and $\mathbf{B}$ are in the (r, θ) plane only and since neither have a z-dependency, the curl of each quantity points in the $\hat{\mathbf{z}}$ direction only. Therefore, $(\mathbf{j}.\nabla) \equiv j_z \frac{\partial}{\partial z} \equiv 0$ and $(\mathbf{\Psi}.\nabla) \equiv \Psi_z \frac{\partial}{\partial z} \equiv 0$ where we introduce $\mathbf{\Psi} = \nabla \times \mathbf{u}$ the fluid vorticity. Using the identity[1] $\nabla \times [(\mathbf{u}.\nabla)\mathbf{u}] = (\mathbf{u}.\nabla)\mathbf{\Psi}$ we obtain a convective form of equation (2.21),

$$\rho \frac{D}{Dt} \mathbf{\Psi} = (\mathbf{B}.\nabla)\mathbf{j} \tag{2.33}$$

An expression for p may be regained by taking the divergence of (2.21),

$$\nabla^2 p = \nabla.[\mathbf{j} \times \mathbf{B}] - \rho \nabla.[(\mathbf{u}.\nabla)\mathbf{u}] \tag{2.34}$$

Equation (2.33) relates the vorticities of the two vector fields $\mathbf{u}$, $\mathbf{B}$ and equation (2.22) the fields themselves. Equations (2.19) and (2.20) are treated as initial conditions.

As we want to look at both time dependent and time independent field aligned flow in this we set $F = F(r, \theta, t)$. F is not dimensionless but carries dimension $\sqrt{\mu_0 \rho}$. If we define a dimensionless function $f = f(r, \theta, t)$ by setting $F = \frac{f(r,\theta,t)}{\sqrt{\mu_0 \rho}}$ then this allows us to state that $M_A^2 = f^2$. We call f the flow function. Hence we have

$$\mathbf{u} = f(r, \theta, t) \frac{\mathbf{B}}{\sqrt{\mu_0 \rho}} \tag{2.35}$$

An important consequence of (2.35) is that by (2.22), $\mathbf{B}$ can only be time independent. But by specifying a time dependency in f this allows us to examine the possibility of time dependent flow along time independent field lines. Substituting (2.35) into (2.19) yields

$$\mathbf{B}.\nabla f = 0 \tag{2.36}$$

[1] Valid only in this specialised geometry.

This says that f may at most be a function of time only on a field line. To express the field aligned form of equation (2.33), we first note that $\mu_0(\mathbf{B}.\nabla)\mathbf{j} = \mathbf{B} \times \nabla^2\mathbf{B}$, and analogously, $(\mathbf{u}.\nabla)\mathbf{\Psi} = \mathbf{u} \times \nabla^2\mathbf{u}$ identity holds for $\mathbf{u}$. Equation (2.33) becomes

$$\mu_0\rho\frac{\partial}{\partial t}\mathbf{\Psi} = \mathbf{B} \times \nabla^2\mathbf{B} - \mu_0\rho\mathbf{u} \times \nabla^2\mathbf{u} \tag{2.37}$$

If we now put $\mu_0 = 1, \rho = 1$, then on inserting the field aligned specification and using the identity $\nabla^2(f\mathbf{B}) = f\nabla^2\mathbf{B} + \mathbf{B}\nabla^2 f + 2f(\nabla f.\nabla)\mathbf{B}$ we obtain

$$\frac{\partial}{\partial t}[\nabla \times (f\mathbf{B})] = (1 - f^2)\mathbf{B} \times \nabla^2\mathbf{B} + \mathbf{B} \times [\nabla(1 - f^2).\nabla]\mathbf{B} \tag{2.38}$$

Equation (2.38) may be seen as an evolution equation for f with (2.36) a constraint on f. Physically, any solution we obtain amounts to a fluid moving along the field lines in such a way so that the momentum equation (2.21) is balanced, and that fluid incompressibility (2.19) is maintained. Note that for arbitrary choice of field $\mathbf{B}$ there are always at least two values that f may assume which trivially solve the equations, namely $f = \pm 1$. In this situation the fluid is moving at the Alfvén speed either parallel ($f = +1$) or anti-parallel ($f = -1$) to $\mathbf{B}$. In particular, equation (2.21) reduces to $\nabla\left[p + \frac{\mathbf{B}.\mathbf{B}}{2}\right] = 0$, i.e., the fluid pressure is simply $p + \frac{\mathbf{B}.\mathbf{B}}{2} = constant$. However, if we want to move the fluid with (more interesting) sub/super-Alfvénic velocities by asking for f non-trivial, then we must solve equations (2.36) and (2.38). There are three equations in three unknowns here: (2.36), (2.38) and (2.20) in the variables B_r, B_θ and the flow function f. The pressure is a dependent function, and is not part of the problem for the equation set (2.20)-(2.23).

We can show that solutions to equation (2.38) define the pressure p in such a way that equation (2.21) is balanced. Equation (2.38) is simply the curl of equation (2.21), with the field aligned flow specification. Hence (2.38) may be written as

$$\nabla \times \left[-\frac{D}{Dt}(f\mathbf{B}) + \mathbf{j} \times \mathbf{B}\right] = 0 \tag{2.39}$$

Therefore, if we find a suitable f, $\mathbf{B}$, that does solve (2.39) then by using the vector identity $\nabla \times \nabla q = 0$ (for any q) then we know that there must exist a function q so that we can write

$$\nabla q = -\frac{D}{Dt}(f\mathbf{B}) + \mathbf{j} \times \mathbf{B} \tag{2.40}$$

Hence by (2.40) we can identify q as the fluid pressure p, balancing the momentum equation. Consequently, solutions to (2.38) permit field aligned flow, provided (2.36) is true also. This naturally introduces the idea that only certain choices of f, $\mathbf{B}$ permit field aligned flow, as arbitrary choices will not solve (2.38) or (2.36). The effort has been concentrated on finding suitable f, $\mathbf{B}$, as pressure calculations are relatively straightforward once these have been found. We first consider f time independent.

Chapter 3

Time Independent Field Aligned Flow

We apply the field aligned magnetohydrodynamic equations derived in the previous chapter to the case of time independent flow. A general solution of these equations is presented for nontrivial flow functions. In particular, the case $f = constant$ is described in some detail, as this forms the basis of the magnetotail models presented and motivates the application of genetic algorithms to differential equation's (chapters 5 and 6).

3.1 The Flow Function

In the previous chapter we derived two equations governing field aligned flow for a general flow function $f = f(r, \theta, t)$. By dropping the time dependency in the flow function we examine the case of time independent flow. Equation (2.38) may now be written as

$$\nu \mathbf{B} \times \nabla^2 \mathbf{B} + \mathbf{B} \times [\nabla \nu . \nabla] \mathbf{B} = \mathbf{0} \tag{3.1}$$

where $\nu = 1 - f^2$. Although in vector form (3.1) has only a z-component, and, after some manipulation it may be re-expressed as a scalar equation,

$$a \frac{\partial \nu}{\partial r} + b \frac{\partial \nu}{\partial \theta} + c\nu = 0 \tag{3.2}$$

where

$$a = \frac{\phi_2}{r} \tag{3.3}$$

$$b = \frac{\phi_1}{r^2} \tag{3.4}$$

$$c = \frac{1}{r^2} \frac{\partial \phi_1}{\partial \theta} + \frac{1}{r} \frac{\partial \phi_2}{\partial r} = \hat{\mathbf{z}} . (\mathbf{B} \times \nabla^2 \mathbf{B}) = \hat{\mathbf{z}} . (\mathbf{B} . \nabla) \mathbf{j} \tag{3.5}$$

$$\phi_1 = B_r^2 + B_\theta^2 + B_r \frac{\partial B_\theta}{\partial \theta} - B_\theta \frac{\partial B_r}{\partial \theta} \tag{3.6}$$

$$\phi_2 = r\left(B_r \frac{\partial B_\theta}{\partial r} - B_\theta \frac{\partial B_r}{\partial r}\right) \tag{3.7}$$

We wish to look for magnetic field topologies which support non-trivial values of f; by nontrivial we mean $f \neq \pm 1, 0$. Expanding equation (2.36) we get

$$\frac{\partial \nu}{\partial r} + \vartheta \frac{\partial \nu}{\partial \theta} = 0 \tag{3.8}$$

for $rB_r \neq 0$,where

$$\vartheta = \frac{B_\theta}{rB_r} \tag{3.9}$$

Equations (3.2), (3.8) can be seen as a system of linear equations in two variables, $\frac{\partial \nu}{\partial \theta}, \frac{\partial \nu}{\partial r}$. Hence one can manipulate them and obtain expressions in terms of ν that formally solve (3.2), (3.8), i.e.,

$$\frac{\partial \nu}{\partial \theta} = -\left[\frac{c}{b - a\vartheta}\right]\nu, \quad \frac{\partial \nu}{\partial r} = \left[\frac{c\vartheta}{b - a\vartheta}\right]\nu \tag{3.10}$$

On integration of (3.10), both forms of ν must be identical, and for an arbitrary choice of $\mathbf{B}$, this may not be true. As an example, consider the following simple field

$$B_\theta = h, \quad B_r = \frac{g}{r}$$

where $h = constant \neq 0$ and $g = constant \neq 0$. With this choice of field (3.10) becomes

$$\frac{\partial \nu}{\partial \theta} = \frac{h}{g} r\nu, \qquad \frac{\partial \nu}{\partial r} = -\frac{h^2}{g^2} r\nu,$$

Integrating yields

$$\log(\nu) = c_1(r) + \frac{h}{g} r\theta, \qquad \log(\nu) = c_2(\theta) - \frac{h^2}{2g^2} r^2$$

where c_1, c_2 are arbitrary functions. Since the two expressions for ν must be identical we must have

$$c_1(r) - c_2(\theta) = -\frac{h}{g} r\theta - \frac{h^2}{2g^2} r^2$$

Clearly, no such functions c_1, c_2 exist. Hence (3.10) cannot be solutions to (3.2), (3.8) for an arbitrary choice of magnetic field. This introduces the recurring idea that for nontrivial flow functions, we must be in a precisely defined geometry. Alternatively, one can say that a given magnetic field topology can support only very particular flow functions. We find a set of magnetic fields that do solve (3.2),(3.8) by making an assumption on ν, that is,

$$\frac{\partial}{\partial r}\left[\frac{\partial \nu}{\partial \theta}\right] = \frac{\partial}{\partial \theta}\left[\frac{\partial \nu}{\partial r}\right] \tag{3.11}$$

ν is twice continuously differentiable, or more loosely, is 'smooth'. This prescription chooses fields which obviate the arbitrary function problem above. By applying (3.11) to (3.10) we obtain the statement

$$\nu\left[\frac{\partial}{\partial r}(\chi) + \frac{\partial}{\partial \theta}(\chi\vartheta)\right] = 0 \tag{3.12}$$

where $\chi = \frac{c}{b-a\vartheta}$. Trivially we may have $\nu = 0$ which is simply $f = \pm 1$. However, if

$$\frac{\partial}{\partial r}(\chi) + \frac{\partial}{\partial \theta}(\chi\vartheta) = 0 \tag{3.13}$$

then ν may be nonzero, and f nontrivial. This depends critically on the magnetic field arrangement. If (3.13) does not hold then by (3.12), $f = \pm 1$ and the flow velocity follows the magnetic field variable dependency at all points. If (3.13) does hold then ν is given by integration of (3.10) yielding

$$\begin{aligned} \nu &= C_1(r) \exp\left[-\int \chi d\theta\right] \\ \nu &= C_2(\theta) \exp\left[\int \chi\vartheta dr\right] \end{aligned} \tag{3.14}$$

Both functions $C_1(\theta), C_2(r)$ are specified (guaranteed by (3.11)). Any arbitrary functions or constants left over are specified by boundary conditions for the flow. Hence equation (3.13) can be viewed as a geometrical condition that the magnetic field must satisfy to permit nontrivial field aligned flows.

3.2 A Bernoulli Equation

So far, we have set $\mathbf{u} = F\mathbf{B}$ and examined the model equations (2.19)-(2.23) An equivalent field aligned flow specification is $\mathbf{B} = G\mathbf{u}$, for some function G. Obviously, $G = 1/F, F \neq 0$ but the previous format is easier to handle in the set of equations chosen. Using $\mathbf{B} = G\mathbf{u}$, reduces the induction equation (2.22) to $\frac{\partial}{\partial t}(G\mathbf{u}) = 0$, as well as the equivalents of (2.19) and (2.21). Hence we have three equations to solve instead of two. Previously, $\nabla.\mathbf{u} = 0$ gave rise to $\mathbf{B}.\nabla f = 0$. Here, $\nabla.\mathbf{B} = 0$ yields

$$(\mathbf{u}.\nabla)G = 0 \tag{3.15}$$

on substitution of the field aligned condition. If we consider the special case of $G = constant$ then (3.15) is solved leaving the momentum balance equation

$$(\mathbf{u}.\nabla)\mathbf{u} = -\nabla p + G^2(\nabla \times \mathbf{u}) \times \mathbf{u} \tag{3.16}$$

The induction equation tells us that $\mathbf{u}$ is time independent. If we take the dot product of (3.16) with $\mathbf{u}$ then we have

$$p + \frac{\rho \mathbf{u}.\mathbf{u}}{2} = constant \tag{3.17}$$

along any magnetic, and hence fluid field line in the flow. This is just Bernoulli's equation for an incompressible, inviscid fluid. Hence this magnetohydrodynamical system is effectively equivalent to the steady flow of an inviscid fluid of constant density.

3.3 Solutions for special cases of the flow function

3.3.1 f=constant

When the flow function is globally constant (and $f \neq \pm 1$) then we have a simpler system to solve as equation (3.8) is solved trivially by $f = constant$. This choice of flow function implies that $\frac{\partial \nu}{\partial \theta} = 0$, $\frac{\partial \nu}{\partial r} = 0$ and hence by (3.10) (assuming $\vartheta \neq 0$) we have

$$c = 0 \tag{3.18}$$

where c is defined by (3.5). Solving (3.18) will generate field topologies which support flow functions of $f = constant$. We can rewrite c in the form

$$(\mathbf{B}.\nabla)\mathbf{j} = 0 \tag{3.19}$$

which simply states that the current is constant on a field line. Equation (3.19) is a nonlinear partial differential equation in two variables B_r and B_θ. Since we are only considering $\mathbf{B} = (B_r(r,\theta), B_\theta(r,\theta), 0)$ we may write the magnetic field in terms of the vector potential $\mathbf{A}$

$$\mathbf{B} = \nabla \times \mathbf{A} = \nabla \times [A(r,\theta)\hat{\mathbf{z}}] \tag{3.20}$$

Since we are ultimately interested in the curl of $\mathbf{A}$ its gauge is unimportant. Note that $\nabla.\mathbf{B} = 0$ identically and $\nabla \times \mathbf{B} = \hat{\mathbf{z}}\nabla^2 A$. We may now re-express the problem in terms of A. At first sight, this does not seem very promising, as the resulting equation is highly nonlinear and a degree higher than before: it would appear that we have made the problem more difficult. However, if we make the modelling assumption

$$\nabla^2 A = \lambda(r,\theta) A \tag{3.21}$$

for arbitrary λ, then this makes a considerable difference to the algebra: $c = 0$ becomes

$$\frac{\partial \phi_1}{\partial \theta} + r\frac{\partial \phi_2}{\partial r} = rA\left[\frac{\partial A}{\partial \theta}\frac{\partial \lambda}{\partial r} - \frac{\partial A}{\partial r}\frac{\partial \lambda}{\partial \theta}\right] = 0$$

i.e., we require

$$\frac{\partial A}{\partial \theta}\frac{\partial \lambda}{\partial r} - \frac{\partial A}{\partial r}\frac{\partial \lambda}{\partial \theta} = 0 \tag{3.22}$$

Putting $\lambda = g(A)$ for an arbitrary function g solves (3.22) and hence solves $c = 0$. This is subject to finding an answer to (3.21), in general, a Poisson equation. We also need boundary conditions to determine particular solutions i.e., $A = h(r,\theta)$ on the boundary of the region. For instance, $\lambda = \lambda_0 A^n$ solves (3.22): choosing $n = 0$ yields the Helmholtz equation in (3.21), solutions of which have been extensively covered in the literature. If we put $\lambda = -p^2$ for $p = constant$, then (3.21) reduces to

$$\frac{1}{r}\frac{\partial}{\partial r}\left[r\frac{\partial A}{\partial r}\right] + \frac{1}{r^2}\frac{\partial^2 A}{\partial \theta^2} = -p^2 A \tag{3.23}$$

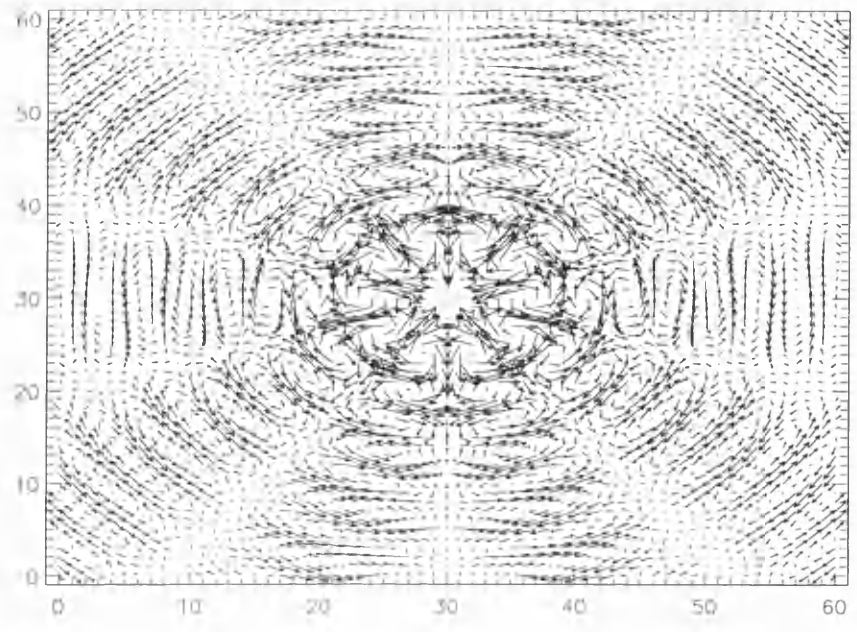

Figure 3.1: Field (3.24) with $m = 5, p = 1$.

Equation (3.23) may be solved by the technique of separation of variables: if we set $A(r,\theta) = R(r)\cos(m\theta)$ ($m \geq 1$,integral) then (3.23) is soluble by $R(r) = J_m(pr)$, where J_m represent the Bessel functions of the first kind. $A = J_m(pr)\cos(m\theta)$ is a solution for (3.23), which corresponds to a magnetic field

$$\mathbf{B} = -\frac{p}{2}\left[\sin(m\theta)\left\{J_{m-1}(pr) + J_{m+1}(pr)\right\}, \cos(m\theta)\left\{J_{m-1}(pr) - J_{m+1}(pr)\right\}, 0\right] \tag{3.24}$$

This is an example of a magnetic field which can support a globally constant value of f (see figure 3.1). Hence by the assumption of (3.21) we have reduced an intricate geometrical condition (3.13) to the solution of Poisson's equation.

We may also reverse the procedure and ask for $A = h(\lambda)$ for some function h. This procedure may give us access to solutions not easily derived by the method above. Equation (3.22) is satisfied identically, leaving (3.21) as

$$h''\left[\frac{1}{r^2}\left(\frac{\partial\lambda}{\partial\theta}\right)^2 + \left(\frac{\partial\lambda}{\partial r}\right)^2\right] + h'\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\lambda}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2\lambda}{\partial\theta^2}\right] - h\lambda = 0 \tag{3.25}$$

where the dash ($'$) on h means derivative with respect to λ. We can make this an ordinary differential equation with dependent variable λ if the coefficients of h'' and h' are functions of λ and they are not both zero, i.e.,

$$q_1(\lambda) = \frac{1}{r^2}\left(\frac{\partial\lambda}{\partial\theta}\right)^2 + \left(\frac{\partial\lambda}{\partial r}\right)^2, \quad q_2(\lambda) = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial\lambda}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2\lambda}{\partial\theta^2} \tag{3.26}$$

To find solutions, we must first find λ such that q_1, q_2 are suitably defined. This fixes the ordinary differential equation (3.25) which must then be solved: any arbitrary functions found on integration are not functions of λ. Resubstitution of a trial solution after this process will further refine the arbitrary functions.

Consider the case for $\lambda = r[\cos(\theta) + \sin(\theta)]$: $q_1 = 2$ and $q_2 = 0$ leaving (3.25) as

$$h'' = \frac{\lambda}{2} h \tag{3.27}$$

known as Airy's equation [37]. Solutions of this equation may be defined in terms of the Airy functions, $Ai(\lambda), Bi(\lambda)$. It can be shown that

$$A(r,\theta) = k_1 Ai\left\{\frac{1}{\sqrt[3]{2}} r[\cos(\theta) + \sin(\theta)]\right\} + k_2 Bi\left\{\frac{1}{\sqrt[3]{2}} r[\cos(\theta) + \sin(\theta)]\right\} \tag{3.28}$$

solves (3.25) for $k_1, k_2 = constants$.

Analytic solutions offer one avenue for exploration in this field aligned system. However, more solutions are available computationally and in section 3.5.1 some magnetotail relevant results are described. For choices other than $\lambda = c_1 + \frac{c_2}{A}$, $c_1, c_2 = constants$, (3.21) is a Poisson equation with a nonlinear source term. Using the alternate specification of $A = h(\lambda)$ presents its own problems, as we have more stages to go through to find a solution. Discussion of the novel numerical techniques that may be used to find solutions to equations (3.21) and (3.25) is deferred until chapters 5 and 6.

Section (3.4) deals with current free solutions to (3.1,3.8) and shows how these solutions can also be extended to flow functions $f = constant$ very easily.

3.3.2 $f = f(r)$

Equation (3.8) demands that $B_r = 0$ with this choice of f, and by (2.20) we must have $\mathbf{B} = b(r)\hat{\theta}$ - i.e., the flow is moving in rings around the origin. Using (3.2), $\frac{\partial \nu}{\partial \theta} = 0$ by assumption and since $\phi_2 = 0$, $\phi_1 = \frac{b^2}{r^2}$ then $a = c = 0$. Therefore the system is solved completely for arbitrary $f = f(r)$. The flow profile is then given by

$$\mathbf{u} = f(r)\, b(r)\, \hat{\theta} \tag{3.29}$$

Although we do not specify at any time what would cause field aligned flow, it is often useful to look at the fluid pressure p in these systems. If we substitute this back into (2.21) then we find that

$$\mu_0 p + \frac{b^2}{2} = constant + \int_{r_0}^{r} \frac{b^2(s)}{s} \left[f^2(s) - 1\right] ds \tag{3.30}$$

The integral contains two terms; one is a fluid term, integrand $\propto f^2$, and the other is a magnetic tension term.

Consider the following example: the magnetic field is given by

$$\mathbf{B}(r) = \frac{Kr}{\alpha + r^2}\hat{\theta}$$

which may be rewritten more usefully as a function with a dimensionless argument if we define $\eta = \frac{r}{\sqrt{\alpha}}$, i.e.,

$$\mathbf{B}(\eta) = \frac{K}{\sqrt{\alpha}}\left[\frac{\eta}{1+\eta^2}\hat{\theta}\right]$$

If the flow function f is

$$f^2(\eta) = g\left[2+\eta^2\right]$$

then the fluid velocity is

$$\mathbf{u}(\eta) = K\sqrt{\frac{g}{\alpha}}\left[\frac{\sqrt{2+\eta^2}\eta}{1+\eta^2}\hat{\theta}\right]$$

This particular flow function is chosen to make the integration easier. The system's equivalence to stationary hydromechanics and magnetohydrostatics (section 2.3.2 and Gebhardt and Kiessling [28]) can also change at a particular radius, depending on the value of g. This is given by

$$\eta_{change}^2 = \frac{1}{g} - 2$$

which defines a real radius when $0 < g < \frac{1}{2}$. At $\eta > \eta_{change}$ the system is equivalent to stationary hydrodynamics. For $\eta < \eta_{change}$, the principles of magnetohydrostatics apply.

Note that the definition of f demands that $g \geq 0$. This does not matter to the system, as the flow function appears as f^2 only in the momentum balance equation (3.1). Also, as $\eta \to \infty$, $\mathbf{u} \to K\sqrt{\frac{g}{\alpha}}\hat{\theta}$, even although $\mathbf{B} \to 0$ and $f \to \infty$ in the same limit. The momentum balance equation becomes

$$\frac{\alpha}{K^2}\frac{dp}{dr} = \frac{\eta\left[2g-2+g\eta^2\right]}{\left(1+\eta^2\right)^2} + \frac{2\eta^3}{\left(1+\eta^2\right)^3}$$

This may be integrated to yield

$$p = C + \frac{K^2}{2\alpha}\left[\frac{1}{1+\eta^2} + g\left\{\log(1+\eta^2) - \frac{1}{1+\eta^2}\right\}\right]$$

where $C = constant$. Note that the pressure is dominated by $\log(1+\eta^2)$ which tends to infinity as $\eta \to \infty$. This is because as η increases, the fluid velocity becomes constant. Consider two fluid elements on different field lines at $r_1 > r_2$. Both field elements have the same velocity in the azimuthal direction and hence the angular velocity of the fluid element on r_2 is greater than the element on r_1. Since both elements are moving in the azimuthal direction, each element must be experiencing an acceleration. Therefore, there is a radially directed force causing this acceleration. If there were no such force, the fluid elements would move off their field lines. Hence the pressure must increase as η increases in order for the pressure gradient to be maintained. This pressure gradient keeps the fluid elements on the field lines.

The factor g allows us to control the size of the fluid flow. For $0 < g < 1$, the pressure has a minimum value at $\eta = \sqrt{\frac{-3}{2} + \frac{1}{2}\sqrt{1+\frac{8}{g}}}$. A minimum in the pressure corresponds to a point in the fluid where $\frac{dp}{dr} = 0$, meaning the net force at this radius is zero. This is caused by a balance between the magnetic and fluid pressures in the system. When $g > 1$, the minimum pressure

occurs at $\eta = 0$ The system in this case is dominated by the fluid flow rather than the magnetic field.

3.3.3 $f = f(\theta)$

The magnetic field for this flow function is chosen by (3.2) again: $\mathbf{B} = \frac{\Theta(\theta)}{r}\hat{\mathbf{r}}$. Equation (3.2) can be integrated to find a functional form for f,

$$f^2 = 1 - \frac{c_1}{\Theta(\theta)} \tag{3.31}$$

On substitution into (3.2) we may define a pressure for the system

$$\mu_0 p = -\frac{\Theta^2}{2r^2} + \frac{c_1}{2r^2} + c_2 \tag{3.32}$$

where $c_1, c_2 = constant$. Although the pressure diverges at the origin, the function for p formally solves (3.2) for the flow function and corresponding magnetic field demanded. Depending on the region of application we may not even need to worry about this.

3.4 Current free solutions

In this section we consider (3.1) with $\mu_0\mathbf{j} = \nabla \times \mathbf{B} = 0$; we are assuming that the magnetic field is of potential form, that is $\mathbf{B} = \nabla\phi$ for some potential ϕ. We must still have (3.8)

$$\frac{\partial f}{\partial r} + \vartheta\frac{\partial f}{\partial \theta} = 0$$

In vector form the equation (3.1) reduces to

$$\mathbf{B} \times [(\nabla f.\nabla)\,\mathbf{B}] = \mathbf{0} \tag{3.33}$$

In scalar form,

$$\frac{\phi_2}{r}\frac{\partial f}{\partial r} + \frac{\phi_1}{r^2}\frac{\partial f}{\partial \theta} = 0 \tag{3.34}$$

We can treat these equations as a system: two linear equations in two unknowns, $\frac{\partial f}{\partial r}, \frac{\partial f}{\partial \theta}$. One can obtain conditions (by simple elimination of one of the unknowns) in order that both (3.8) and (3.34) are in fact the same equations. On manipulation,

$$B_r\phi_1 = B_\theta\phi_2 \tag{3.35}$$

is the condition $\mathbf{B}$ must satisfy in order to solve (3.33) and (3.8). The corresponding flow function is given by integrating (3.34) (or (3.8), their equivalence is guaranteed by (3.35)), a first order linear partial differential equation in f. We can solve this by the method of characteristics: the equation for the characteristic curve is given by integrating

$$\frac{d\theta}{dr} = \vartheta$$

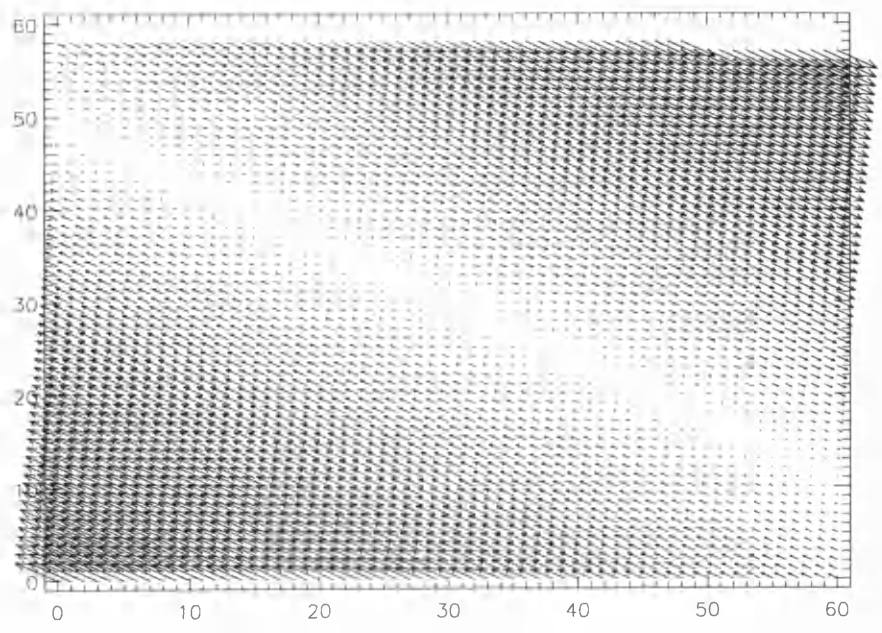

Figure 3.2: Field (3.38) with $c_1 = c_2 = 1$.

i.e.,

$$constant = q(r, \theta)$$

for some function q. the result of the integration. Fixing a constant will choose a particular characteristic This means f has the general formal solution

$$f = F[q] \tag{3.36}$$

for some function F. $\mathbf{B}$ is most easily obtained by using the vector potential $\mathbf{A} = (r, \theta)\hat{\mathbf{z}}$ with $\mathbf{B} = \nabla \times \mathbf{A}$ to reduce the number of unknowns. Assuming a solution of the form $A(r, \theta) = r^n \Theta(\theta)$, $n \geq 1$, integral, we find that

$$A(r, \theta) = c_2 r \cos(c_1 - \theta) \tag{3.37}$$

with $c_1, c_2 = constant$ solves (3.35). This represents the field

$$\mathbf{B} = c_2 [\sin(c_1 - \theta), -\cos(c_1 - \theta), 0] \tag{3.38}$$

For this field the characteristic equation is

$$\frac{d\theta}{dr} = -\frac{\cos(c_1 - \theta)}{\sin(c_1 - \theta)}$$

The characteristic curves of (3.34) for this choice of field are given by

$$constant = r\cos(c_1 - \theta)$$

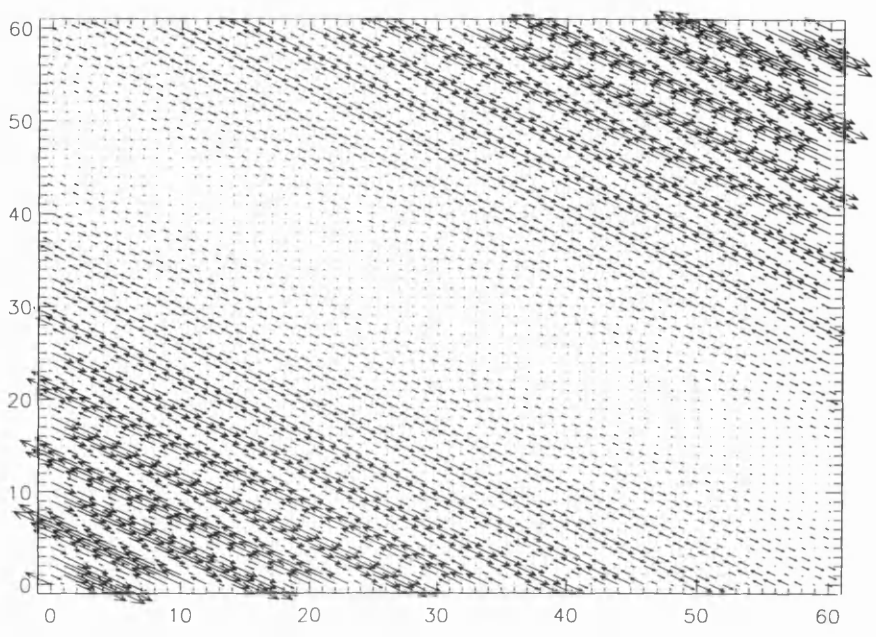

Figure 3.3: Field (3.38) with $c_1 = c_2 = 1$ and flow function $f = \sin[r\cos(c_1 - \theta)]$.

Hence the field (3.38) supports a flow function $f = F[r\cos(c_1 - \theta)]$, where F is an arbitrary function of its argument. Two examples of this field aligned system are given: the first, figure 3.2 has flow function $f = 1$ and hence shows both the field and flow lines. Figure 3.3 shows the flow lines with $f = \sin[r\cos(c_1 - \theta)]$. Note in this case the background magnetic field is still that given by figure 3.2.

It is a simple matter to extend these current free fields into fields with constant current and $f = constant$. Suppose we have a magnetic field $\mathbf{B}$ such that $\nabla \times \mathbf{B} = k\hat{\mathbf{z}} = \mu_0 \mathbf{j}$, with $k = constant$. If we express the magnetic field in terms of its vector potential then

$$\nabla^2 A = -k \tag{3.39}$$

Equation (3.39) is linear: therefore if we write $A = A_0 + A_1$ where $\nabla^2 A_0 = 0$ along with the suitable boundary conditions, then we are left to solve

$$\nabla^2 A_1 = -k \tag{3.40}$$

subject, of course to the boundary conditions. This is a linear inhomogeneous equation – a Poisson equation. But if we can solve it then we have increased the number of solutions available to the $f = constant$ analysis. Substituting the constant current field described above it can be seen that the first term in (3.1) is

$$\nu \mathbf{B} \times \nabla^2 \mathbf{B} = (\mathbf{B}.\nabla)\,\mathbf{j} = (\mathbf{B}.\nabla)\,k\hat{\mathbf{z}} = 0$$

leaving (3.8), which can be solved easily by putting $f = constant$. This also solves (3.8). The final field is then given by

$$\mathbf{B} = \nabla \times (A_0 \hat{\mathbf{z}}) + \nabla \times (A_1 \hat{\mathbf{z}}) \tag{3.41}$$

3.5 Application to a Problem in Field Aligned Flow

The magnetotail is one region where field aligned flow, or at least nearly field aligned flow, is found naturally. Other features are also seen in the magnetotail, principal of which are X-points and plasmoids. Section 2.3.2 describes some of the studies that have been undertaken to mimic these features theoretically in a field aligned context.

We aim to show here that the field aligned flow equations (3.21), (3.22), developed in section 3.3.1 can also reproduce the O and X-point regions seen experimentally and in other theoretical treatments.

3.5.1 Magnetotail geometry and simple models

As has been noted previously, field aligned flow has been observed in the magnetotail, concurrent with large flow velocities (see section 2.3.1). We will model the magnetotail using a highly simplified description and idealised magnetotail conditions.

The magnetotail geometry is, overall, very simple. It consists of two roughly parallel but oppositely directed fields sandwiching a current/quasi-neutral sheet. Here neutral refers to the near zero magnetic field measured here. We shall describe this with reference to a square region of space viewing the magnetotail 'side on'. In this system, $-\hat{\mathbf{x}}$ is directed towards the Sun, $+\hat{\mathbf{x}}$ away from the Sun. $+\hat{\mathbf{y}}$ is directed perpendicularly out of the Earth-Sun plane. The square region R is centred at the origin of the x, y co-ordinate system and has side length of $2r_0$ The remaining $+\hat{\mathbf{z}}$ (in the Earth-Sun plane) is not considered: this is equivalent to assuming that the magnetotail does not vary in this direction, and that x, y planar sections are equivalent at different values of the z coordinate. Hence we assume that no quantity in the system depends on z. The flow and fields are therefore 2 dimensional. These are the same geometrical assumptions that were made in chapter 2, and allow us to apply field aligned flow to this situation. The basic picture we require is shown in figure 3.4. This diagram shows the magnetic field and fluid lines.

The magnetic field is zero (or near zero) at the neutral sheet and increases in magnitude as we move in the $\pm\hat{\mathbf{y}}$ directions. We can approximate the field by assuming that it has only one component, B_X, parallel to the x-axis. By Maxwell's equations and the assumptions above $B_X = B_X(y)$. Such a field must also satisfy one of the field aligned flow equations we have developed. We choose to model the flow with Cartesian versions of the equations developed in

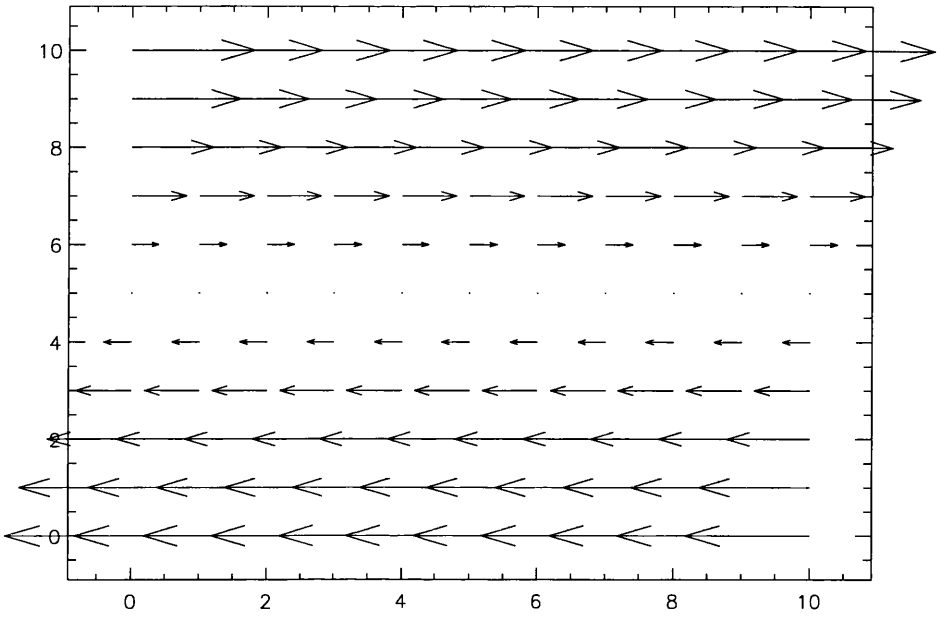

Figure 3.4: Idealised field structure for the magnetotail

section 3.3.1, namely

$$\nabla^2 A = \lambda(x, y)\, A \tag{3.42}$$

and

$$\frac{\partial A}{\partial x}\frac{\partial \lambda}{\partial y} - \frac{\partial A}{\partial y}\frac{\partial \lambda}{\partial x} = 0 \tag{3.43}$$

where $\mathbf{B} = \nabla \times [A(x, y)\hat{\mathbf{z}}]$. These may be derived in an exactly analogous way to equations (3.21) and (3.22). Again, (3.43) may be solved by putting $\lambda = \lambda(A)$, and from here on, λ will take this form. There are a number of reasons why we might start with this equation. Firstly, it is time independent which means that we are looking at steady flows. Steady flows are observed experimentally over long time scales, with reconnection events happening infrequently. Solutions to (3.42) represent fields that have flow function $f = constant \neq \pm 1$ everywhere. Therefore we can fix the Alfvén Mach number to an average derived experimentally, ignoring local changes. In any case, it does not matter as f does not affect the field structure. Thirdly, it is relatively easy to calculate when compared to the other equations on offer.

If we just have one component of $\mathbf{B}$ in the x-direction then (3.42) becomes an ordinary differential equation,

$$\frac{d^2 A}{dy^2} = A\lambda(A) \tag{3.44}$$

Using (3.42) also allows us to influence the current, since in this model $-j\hat{\mathbf{z}} = \nabla^2 A\hat{\mathbf{z}} = A\lambda(A)\hat{\mathbf{z}}$. Therefore, changing λ changes the current distribution over the surface. For choices of λ other than $\lambda = c_1 + \frac{c_2}{A}$, the equation is nonlinear. (A special nonlinear case is $\lambda = 6A$: the solution is

$A = P(C_2 + y; 0, C_1)$, C_1, C_2 constants where P is the Weierstrassian elliptic function.) Equation (3.44) can be formally integrated to give

$$\int \frac{dA}{\sqrt{C_1 + 2\left[\int A\lambda(A)dA\right]}} = y + C_2 \tag{3.45}$$

Analytic solutions are rather special, but numerical methods could be used e.g., NAG D02HAF. Additionally, the genetic algorithm ODE could be used to provide initial approximations. However, we will only consider three analytic solutions as a basis for further modelling.

1. $\lambda(A) = \frac{k}{A}$, $k = constant$.

   With this choice, equation (3.42) becomes $A'' = k$, and on integration $A(y) = ky^2 + a_0 y + a_1$. The curl of $A\hat{\mathbf{z}}$ is the magnetic field, and we can use this to determine the values of the constants k, a_0, a_1. At $y = \pm r_0$, $\mathbf{B} = \pm B_0\hat{\mathbf{x}}$ and $\mathbf{B} = 0$ at $y = 0$, fixing $a_0 = 0$ and $k = \frac{B_0}{2r_0}$. Hence $A = \frac{B_0}{2r_0}y^2 + a_1$ and $\mathbf{B} = \frac{B_0}{r_0}y\hat{\mathbf{x}}$.

   Note that this model cannot describe the gross structure of the magnetotail as the current is constant everywhere. This description is more readily applicable nearer the neutral sheet where the current is approximately constant and the field is zero. We can imagine this model representing field aligned flow near neutral sheets in regions of constant current.

2. $\lambda(A) = k$, $k = constant$.

   This is the next simplest case. There are two types of solution to the equation

   $$\frac{d^2A}{dy^2} = kA \tag{3.46}$$

   depending on the sign of k. Note that the current is now proportional to A, and therefore we have an opportunity to model the current as well as the magnetic field. Taking the lead from observations, we would like to have the current profile peaked towards $y = 0$, (the centre of the square region) then tailing off to lower values towards $y = \pm r_0$, while maintaining the same magnetic field profile as before, i.e., $B_X(0) = 0$ and $B_X(\pm r_0) = \pm B_0$. When $k = c^2 > 0$, c real , $A = \alpha\exp(-cy) + \beta\exp(cy)$ is an acceptable solution to equation (3.46) by itself. However, the boundary conditions forbid this solution, as we cannot generate the required current brightening towards $y = 0$. With $k = -c^2$, c real, the general solution is $A = \alpha\sin(-cy) + \beta\cos(cy)$. This solution can be fitted to the magnetotail boundary conditions, yielding $A = M\cos(cy)$ and $\mathbf{B} = cM\sin(cy)\hat{\mathbf{x}}$ where we define $B_0 = cM$. The constants c and r_0 should be chosen so that range of y values pass through one half period of the cosine function only, limiting us to only one current peak in the region. This allows us to model the magnetotail on a bigger scale as we have now reproduced the current brightening at the neutral sheet.

3. $\lambda(A) = k + \frac{p}{A}$, $k, p = constants$.

This choice of λ combines 1 and 2 above into

$$\frac{d^2A}{dy^2} = kA + p \tag{3.47}$$

an inhomogeneous second order ordinary differential equation. Again, we have two solutions depending on the sign of k. When $k = -c^2$, c real, then on substitution of the magnetotail boundary conditions, the solution is

$$A(y) = \left[M - \frac{p}{c^2}\right]\cos(cy) + \frac{p}{c^2} \tag{3.48}$$

where $A(0) = M$ and $c\left[M - \frac{p}{c^2}\right] = B_0$. The magnetic field is given by $\mathbf{B} = \frac{B_0}{\sin(cr_0)}\sin(cy)\hat{\mathbf{x}}$, $\sin(cr_0) \neq 0$, with current $\mathbf{j} = \frac{cB_0}{\sin(cr_0)}\cos(cy)\hat{\mathbf{x}}$. When $k = c^2$, c real, then the analytic solution $A = \alpha\exp(-cx) + \beta\exp(cx) - \frac{p}{c^2}$ cannot reproduce the magnetotail conditions required. It appears that there is not that much to be gained from this model: the magnetic field and currents are the same as in 2 above. However, this λ function when applied to a perturbed system (see section (3.5.2)) may generate different fields.

These three models are useful bases to begin describing more complicated geometries. In the next section, we examine the effect of perturbing the above systems.

3.5.2 Perturbed magnetotail models

The above models represent idealisations of the structure of the magnetotail. If we now perturb these models using equations (3.42) and (3.43) very slightly, then we will keep the main features of the system while hopefully introducing some more realistic structure. We want to maintain the parallel but oppositely directed flows of the original idealisations as this is one of the largest scale features of the magnetotail. We also want to keep (or at least introduce current brightening towards the centre of the system, the plane $y = 0$.

There are two features we can perturb easily, the λ function and the boundary conditions. Perturbing λ and keeping the same boundary information is equivalent to changing the current distribution over the surface influencing $\mathbf{B}$. Varying the boundary conditions but keeping λ constant changes the magnetic field without perturbing the current directly.

It is easiest to perturb λ in such a way that the perturbation is also a function of A, as this will satisfy condition (3.43) exactly. This changes the current distribution and hence the magnetic field in the system. Suppose we have a system

$$\nabla^2 A^0 = A^0\lambda^0(A^0) \quad \text{, where } A^0 = f(x,y) \quad \text{, } \forall (x,y) \in \partial R$$

where we denote the boundary of a region R by ∂R. The superscript 0 denotes an unperturbed quantity. Now perturb λ^0 so that $\lambda(A) = \lambda^0(A) + \epsilon\zeta(A)$ and keep the boundary conditions the

same. Then the system becomes

$$\nabla^2 A = A\lambda^0(A) + \epsilon[A\zeta(A)] \tag{3.49}$$

As $\epsilon \rightarrow 0$, we regain the original system. If $|\epsilon|$ is too large, then we will lose the features of the original model. In order to remain 'close' to the original model, the perturbation must be small: the new current $\epsilon A\zeta(A)$ must not change the original current $\lambda^0 A^0$ by very much. If j^0_{max} and j^0_{min} are the largest and smallest values of the unperturbed current, then the current over the entire surface must lie is the range $j^0_{max} - j^0_{min}$. Any perturbation we make must be small compared to this range, otherwise we are varying $\mathbf{j}$ too much. Hence

$$|\epsilon| \ll \frac{|A\zeta(A)|}{j^0_{max} - j^0_{min}} \tag{3.50}$$

$\forall (x, y) \in R$. This ensures that we do not deviate too far from the models described in the previous section.

If ϵ is chosen to break condition (3.50), then the equations still hold, but we can no longer expect the field to be much like the modelled version. This is because the perturbed current significantly changes the total current. It also becomes increasingly difficult to see what the corresponding magnetic field looks like. Increasing ϵ varies the current nonlinearly, as A is unknown until the ϵ-dependent Poisson equation is solved. Therefore, choosing ϵ large only guides the final outcome.

We can also vary the boundary conditions of the above models keeping the λ function constant. Again, we want to stay close to the desirable modelled features, and so we must perturb the boundary conditions only very slightly. Define A^0_{max} and A^0_{min} respectively to be the maximum and minimum values of A^0 on the boundary for the unperturbed system. All the boundary values of A^0 must lie in the range $A^0_{max} - A^0_{min}$. A 'small' perturbation is one that does not change the boundary information greatly. If the perturbed boundary data is

$$A = f(x, y) + \delta g(x, y) \quad , \forall (x, y) \in \partial R \tag{3.51}$$

then for a small perturbation we must have

$$|\delta| \ll \frac{|g(x, y)|}{A^0_{max} - A^0_{min}} \tag{3.52}$$

$\forall (x, y) \in \partial R$. This does not say anything about what happens locally: the value may change drastically locally, but not perturb the system greatly overall by the definition above. The equation we have to solve is

$$\nabla^2 A = A\lambda^0(A) \tag{3.53}$$

with boundary information (3.51) subject to condition (3.52). Condition (3.43) is satisfied since we keep the same functional form for λ^0.

To use equation (3.42) we must extend the ordinary differential equation boundary information into two dimensions. Below we quote the unperturbed λ^0 functions and the corresponding unperturbed boundary information required to generate the models using (3.42). Since the y component of $\mathbf{B}$ is everywhere zero, A is a constant on $y = \pm r_0$

1. $\lambda^0(A^0) = \frac{k}{A^0}$, $k = constant$.

$$\nabla^2 A^0 = k$$

$$A^0(x,y) = \begin{cases} \frac{k}{2}y^2 + a_1 & , x = \pm r_0 \\ \frac{k}{2}r_0^2 + a_1 & , y = \pm r_0 \end{cases} \tag{3.54}$$

   where $a_1 = constant$.

2. $\lambda^0(A^0) = -c^2$, c a real constant.

$$\nabla^2 A^0 = -c^2 A^0$$

$$A^0(x,y) = \begin{cases} M\cos(cy) & , x = \pm r_0 \\ M\cos(cr_0) & , y = \pm r_0 \end{cases} \tag{3.55}$$

3. $\lambda^0(A^0) = -c^2 + \frac{p}{A^0}$, c, p real constants.

$$\nabla^2 A^0 = -c^2 A^0 + p$$

$$A^0(x,y) = \begin{cases} [M - \frac{p}{c^2}]\cos(cy) + \frac{p}{c^2} & , x = \pm r_0 \\ [M - \frac{p}{c^2}]\cos(cr_0) + \frac{p}{c^2} & , y = \pm r_0 \end{cases} \tag{3.56}$$

   where $M, p = constants$.

We are now in a position to calculate A for a perturbed model. We shall give examples of current and boundary perturbations to each of the three models. Figures (3.5)-(3.17) show the magnetic field and fluid lines of the perturbed models.

1. (a) Model (3.54) with perturbed current.

   Since the current profile is flat everywhere, it is very likely that any perturbation to λ will change it. Some brightening of the current towards the centre of the region would be welcome as the beginnings of a more realistic magnetotail, within the confines of this model. But as has been noted, changing λ and ϵ varies A nonlinearly, since A must be determined by solving a Poisson equation.

   In all the perturbation work, we shall calculate the magnetic field on a square region of side length 4. This fixes $r_0 = 2$. The region of interest is shown in figure (6.2). The unperturbed vector potential A^0 has a maximum value of 10 at $y = 0$, fixing $k = 20, a_1 = 0$, and $A = 0$ (excepting model (3.56)) on the upper and lower boundaries. The unperturbed current is $-j = k = 20$ everywhere.

Choosing $\zeta(A) = 1$ requires that $|\epsilon| \ll \frac{A}{20}$ everywhere by condition (3.50). With $\epsilon = 0.05$, we generate what looks like the start of an O-point type field (see figure (3.6)). However, when $\epsilon = -0.05$, the field takes on an X-point type field (figure (3.5)). The direction of the perturbed current relative to the unperturbed current has an effect on the new field shape. It is encouraging that these fields reproduce two features seen in the magnetotail - plasmoids and X-points - in a field aligned flow context. Other choices of $\zeta(A)$ also create this type of behaviour, and it will be seen again in the following models.

(b) Model (3.54) with perturbed boundary.

The models described have no x-dependence, which suppresses the y component of the magnetic fields. Therefore, if we introduce an x-dependence we will force a y component into the magnetic field. Hence we will add a x-dependent function $g(x)$ to the $y = \pm r_0$ boundary. In all the boundary perturbations, $g(x) = \exp\left(-\frac{x^2}{r_0^2}\right)$. Hence the maximum perturbation will be located at $x = 0$: away from this, g drops to $1/e$ at the edge of the region. One can imagine that this represents some small, localised current outside the region of interest creating a field described by this vector potential distribution.

Using the same choice of parameters as above but with $\delta = 1$, we obtain an X-point (figure (3.7). Perturbing the current with ϵ positive produced an O-point. Therefore, these two types of perturbation affect the system in opposing ways

For the O-point, the vector potential has a slight dip along $y = 0$ with the largest dip at the origin, $x = y = 0$. Concomitant with this is a similarly shaped current profile. The vector potential exhibits a slight brightening along $y = 0$ for X-points, the maximum located at the origin: again, the current profile resembles the vector potential shape. Hence X-points appear to brighten the current, while O-points depress the current. Although the effect is small, it is definitely present; the new field components are larger than those generated by approximation errors in the expression of the derivatives required by the vector potential representation of the magnetic field.

2. (a) Model (3.55) with perturbed current.

M is fixed at 10 and we define $c = \frac{\pi}{2r_0}$. This guarantees only one peak in the vector potential over the range. A also takes on the same range of values as the previous model, i.e., $A^0_{max} - A^0_{min} = 10$, but $j^0_{max} - j^0_{min} = \frac{5\pi^2}{8}$. With $\zeta = 1$, we must have $|\epsilon| \ll |c^2|$. $\epsilon = -0.05$ creates a better defined X-point (figure (3.8)). Similarly, on setting $\epsilon = 0.05$, the O-point is again more apparent, even although we are at the same level of perturbation. This is caused by the gradients present in the background profile; we have to perturb the field more to balance the equation.

Increasing ϵ breaks the similarity with the magnetotail conditions, but still solves equation (3.43). With $\epsilon = -0.4$ (figure (3.10)), the field strongly resembles an X-point. The O-point is also much more pronounced (figure (3.11)) with $\epsilon = 0.4$.

(b) Model (3.55) with perturbed boundary.

Changing the boundary conditions in this model gives results very similar to those above (see figure (3.12) for $\delta = +1$, and figure (3.13) for $\delta = -1$.) Again, the shape is more pronounced for the same level of perturbation.

The same features in A and j are seen in this model as were seen in the previous case. X-points generate a current brightening and O-points a slight depression in the same manner as seen previously. These features are naturally much stronger in the extreme ϵ cases.

3. (a) Model (3.56) with perturbed current.

Again, M is set to be 10 and $c = \frac{\pi}{2r_0}$. This fixes A^0_{max} to be the same as previously, with a minimum of $\frac{p}{c^2}$ on $y = \pm r_0$. $\zeta(A) = 1$ also. We also set $p = 1$. With $\epsilon = -0.05$ which is much smaller than the range $j_{max} - j_{min} = c^2 M - p$ we obtain the expected X-point: $\epsilon = 0.05$ yields an O-point. The background λ causes an increased bending in the field required to balance equation (3.56).

(b) Model (3.56) with perturbed boundary.

A on the boundaries ranges from $A^0_{min} = 1$ to $A^0_{max} = M - \frac{p}{c^2} = 10 - \frac{16}{\pi^2}$. In figures (3.17) and (3.16), $\delta = -0.8$ and $\delta = 0.8$ respectively. This is close to the level of perturbation permitted, but demonstrates well the fact that one can create the same features with this model.

The choice of λ here combines the two models above and appears to give the strongest features for approximately the same level of perturbation. In the unperturbed case, the current will have the same shape as model (3.56), but on perturbation the effect of p as a source term of A will be apparent. This will have a knock on effect in the magnetic field.

All three models generate much the same kind of features for most choices of perturbation. Of course, we cannot hope to describe all types of perturbation possible, but it seems that whatever is chosen, field aligned flow can describe at least a qualitative agreement with features seen experimentally. The above shows that the correct shapes - X and O points - can be generated in this simplified model. The X-point configurations are taken to be analogues for reconnection regions in the magnetotail. This should not be taken to imply that reconnection exists in this model. Rather, we simply state that the model can be made to resemble the traditional X-point picture of reconnection.

The O-points are taken to represent plasmoids in the magnetotail. The model is limited, as the plasmoids are seen to move in the magnetotail, whereas the plasma here is taken to be time

independent and does not translate in space.

3.6 Summary and Conclusions

The key conclusion we can make is that for flows with flow function other than $f = \pm 1$ we must be in a very particular magnetic field geometry. This is expressed by equation (3.13). If $\mathbf{B}$ solves this equation, then we can say that the system can now support flow functions other than $f = \pm 1$. If the field does not solve (3.13) then the fluid can only move at the Alfvén velocity, either parallel or anti-parallel to the field direction - i.e., $f = \pm 1$.

Further classifications have been made. In the main, we have fixed the flow function and asked what fields may support such a flow. But we have also adopted the complementary approach of fixing a class of field and asking what form of flow function is permissible.

This process of classification has allowed us to identify and choose a set of equations that have been used to model a field aligned flow system, the magnetotail. We have used three idealised models of the magnetotail in order to generate some features that are observed experimentally, namely X-points and O-points. We can compare this to the system treated by Birn [30]. In this paper he specialises an earlier 3-dimensional treatment [31] to a 2-dimensional, steady incompressible system. The density ρ is not constant in this model. The pressure is given and the equivalent of the flow function are given, along with the relevant boundary conditions. It is found that a plasmoid is formed with a shape not dissimilar from those quoted here. This model is rather different from the one presented here; for instance, the pressure now plays a role in determining the topology of the plasma. In the models presented here, it is merely a bystander. However, both approaches generate O-points which are taken to be plasmoids. This suggests that the common field aligned nature of the plasma is more important than the differences between the models.

Lee and Yan [33] describe the structure of field aligned plasma jets associated with magnetic reconnection. Since reconnection is not a principal concern of this thesis, we merely state that in the models presented here we can generate X-point configurations that are entirely field aligned, as opposed to the field aligned jets seen in [33]. Lee and Yan examine the flip-side of this coin, the creation of field aligned flow by an X-point with reference to the magnetotail. Hence both treatments have a degree of overlap.

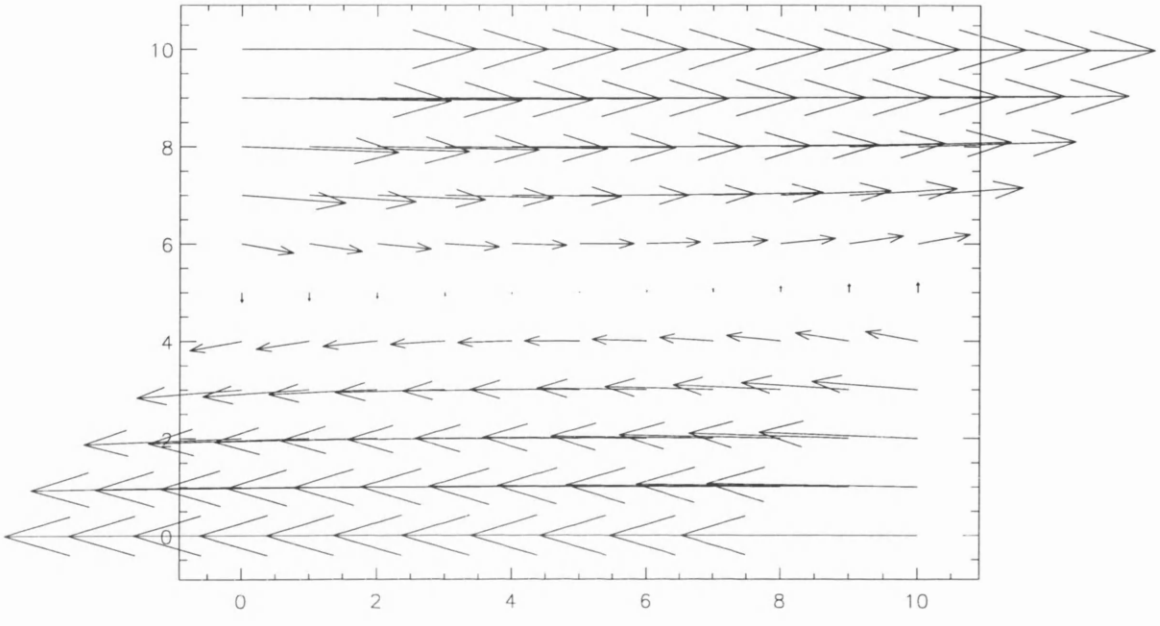

Figure 3.5: Magnetotail model (3.54) with perturbed λ, ϵ negative

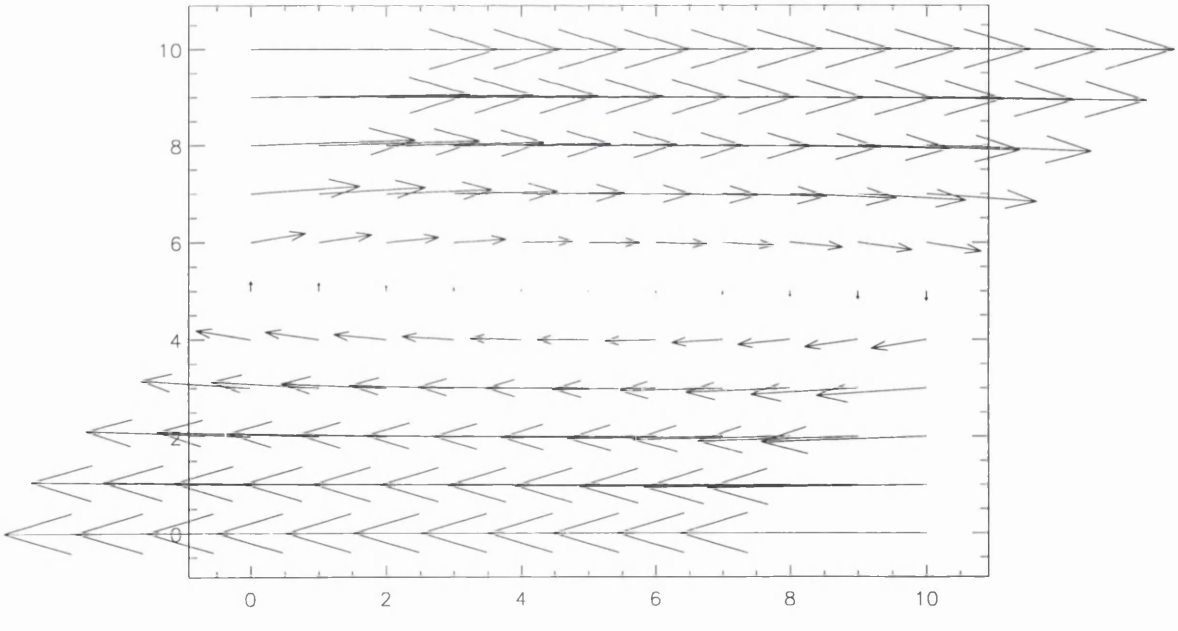

Figure 3.6: Magnetotail model (3.54) with perturbed λ, ϵ positive

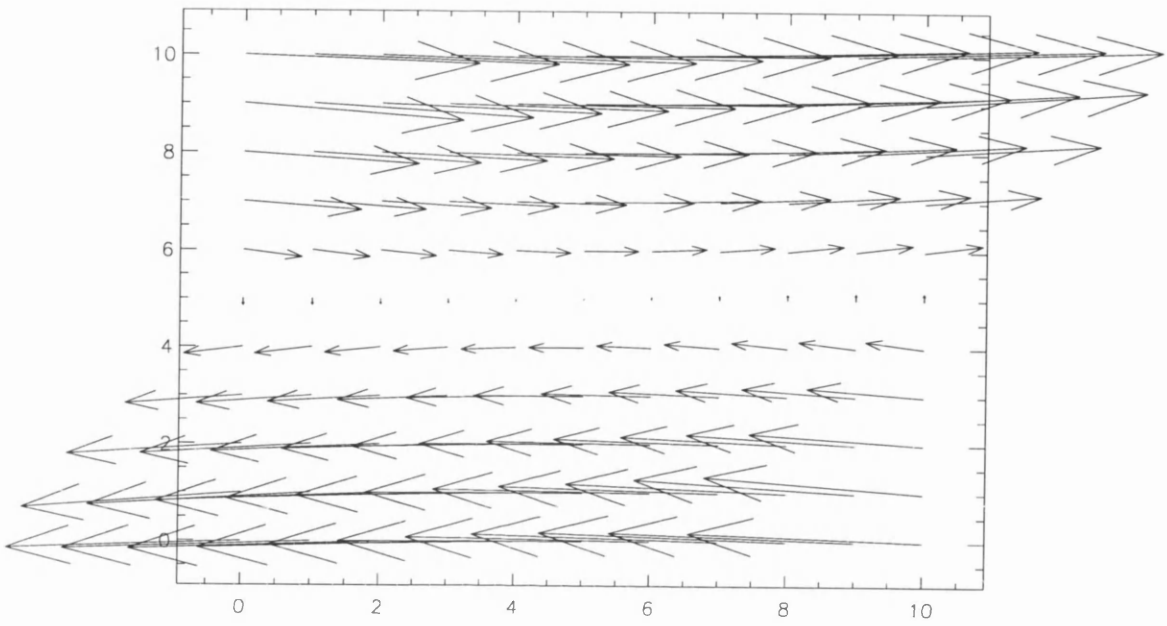

Figure 3.7: Magnetotail model (3.54) with perturbed boundary conditions, δ positive

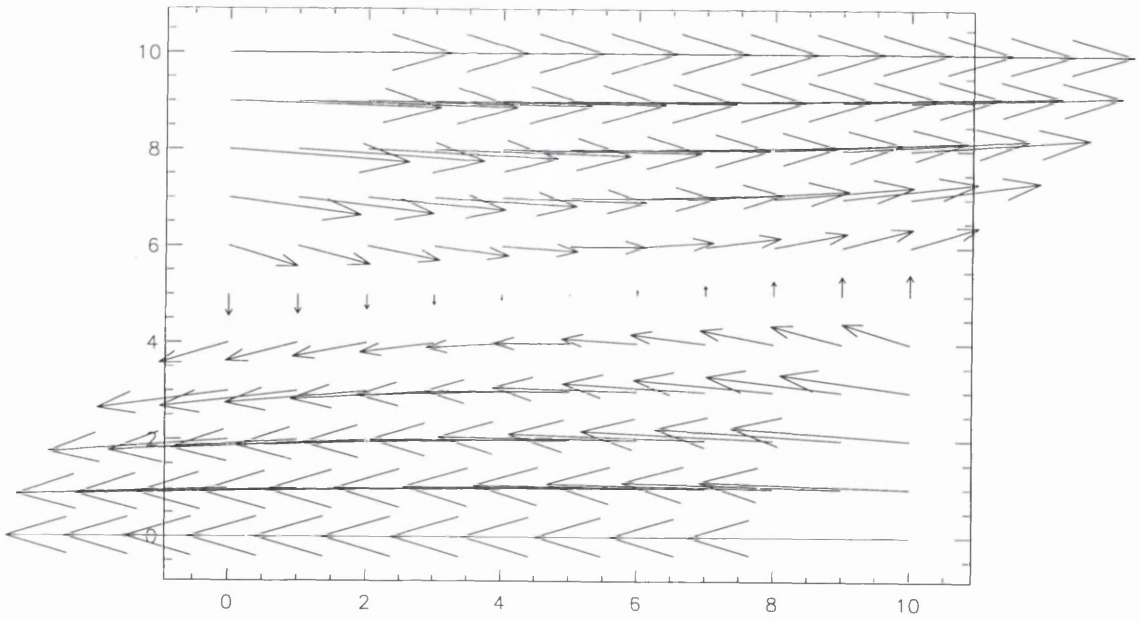

Figure 3.8: Magnetotail model (3.55) with perturbed λ, ϵ negative

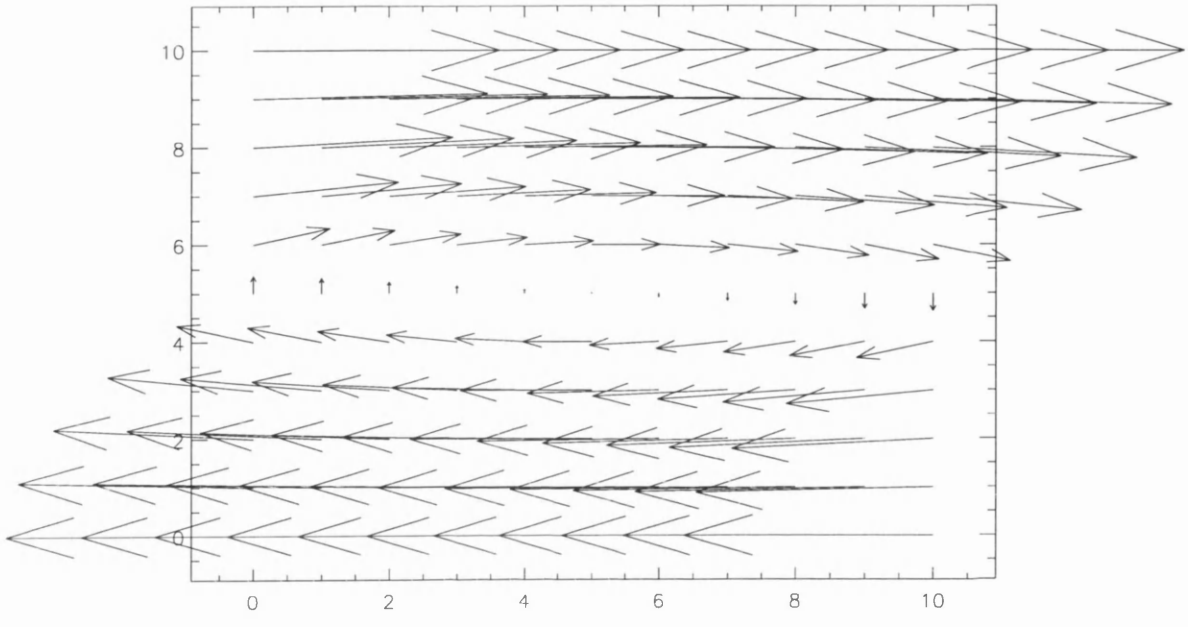

Figure 3.9: Magnetotail model (3.55) with perturbed λ, ϵ positive

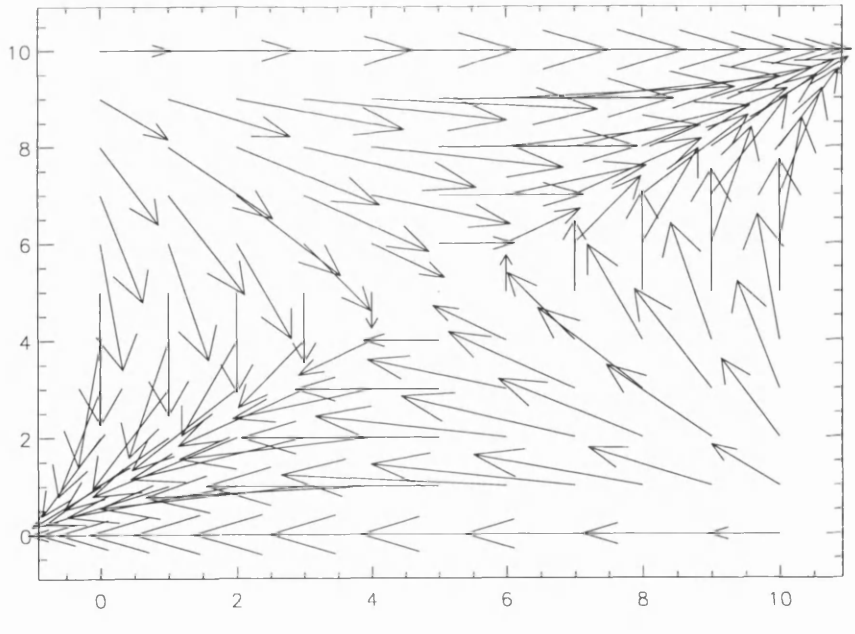

Figure 3.10: Magnetotail model (3.55) with perturbed λ, ϵ extreme and negative

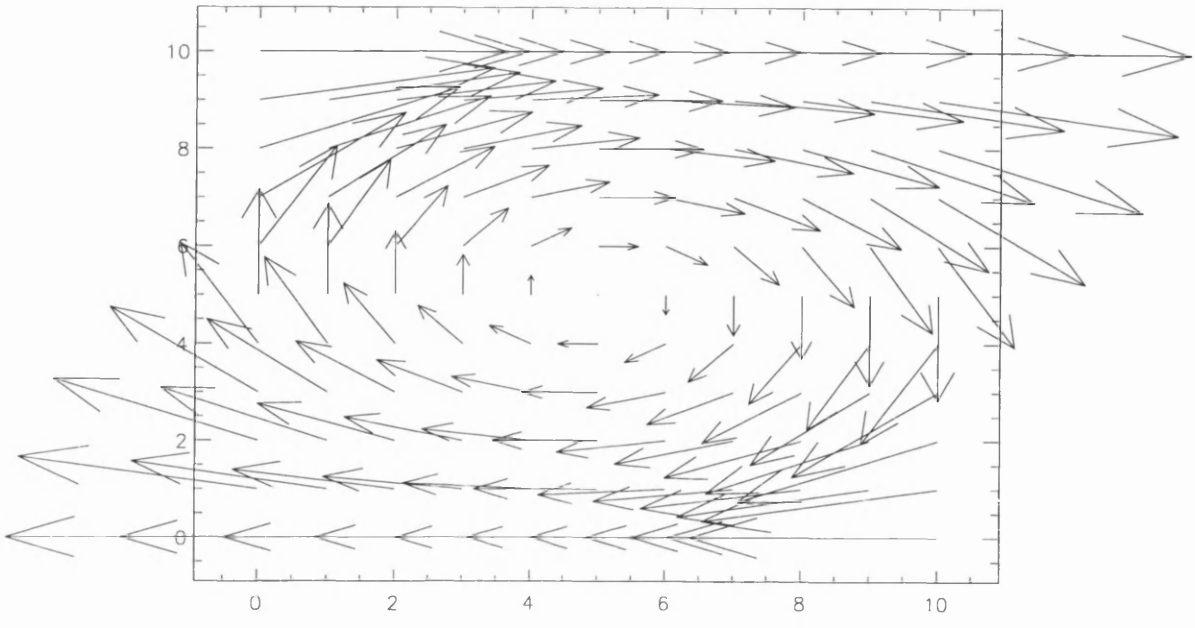

Figure 3.11: Magnetotail model (3.55) with perturbed λ, ϵ extreme and positive

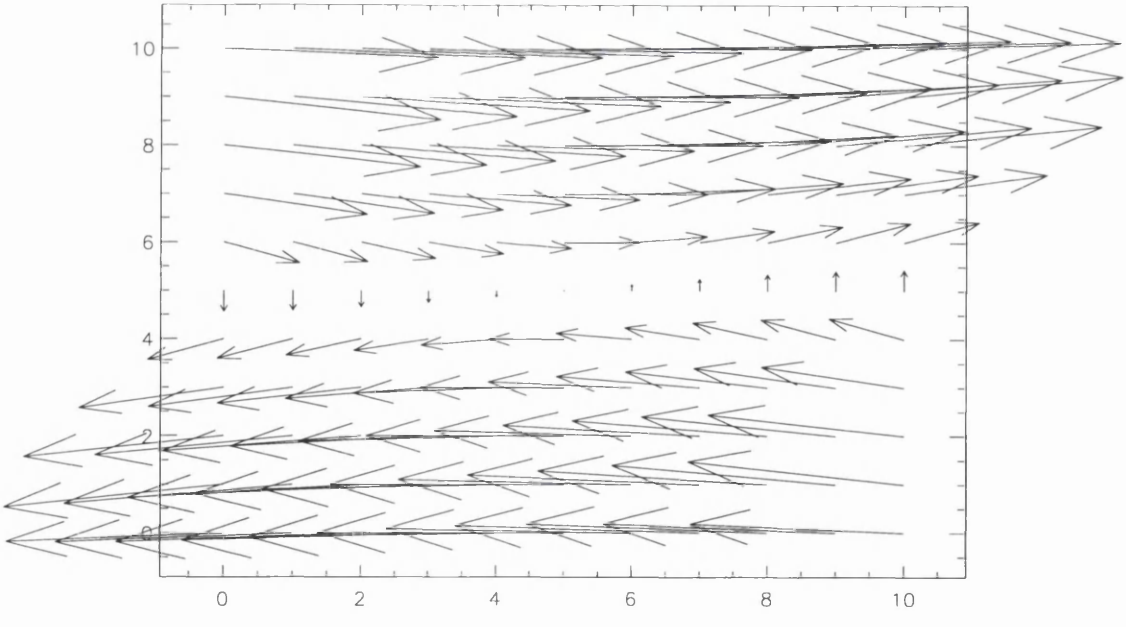

Figure 3.12: Magnetotail model (3.55) with perturbed boundary conditions, δ positive

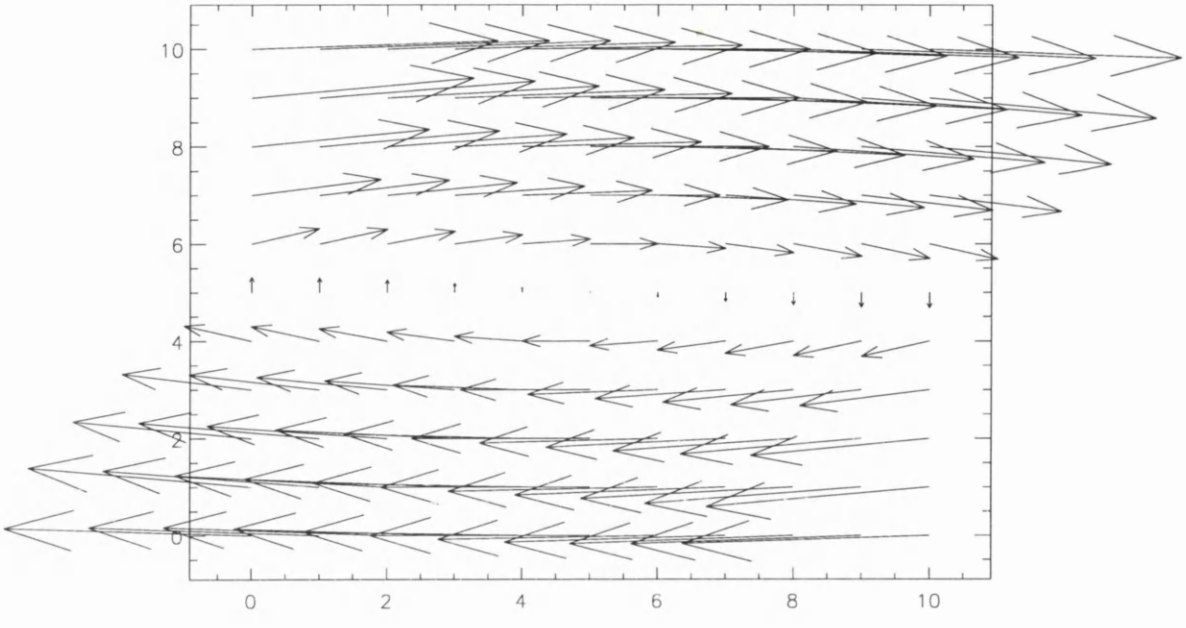

Figure 3.13: Magnetotail model (3.55) with perturbed boundary conditions, δ negative

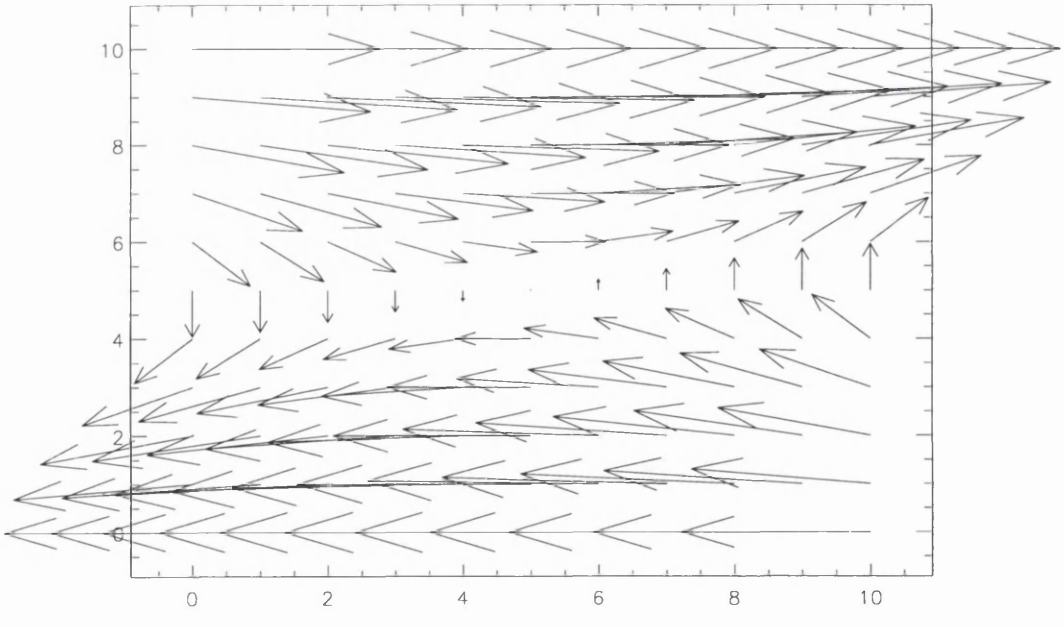

Figure 3.14: Magnetotail model (3.56) with perturbed λ, ϵ negative and within approximation

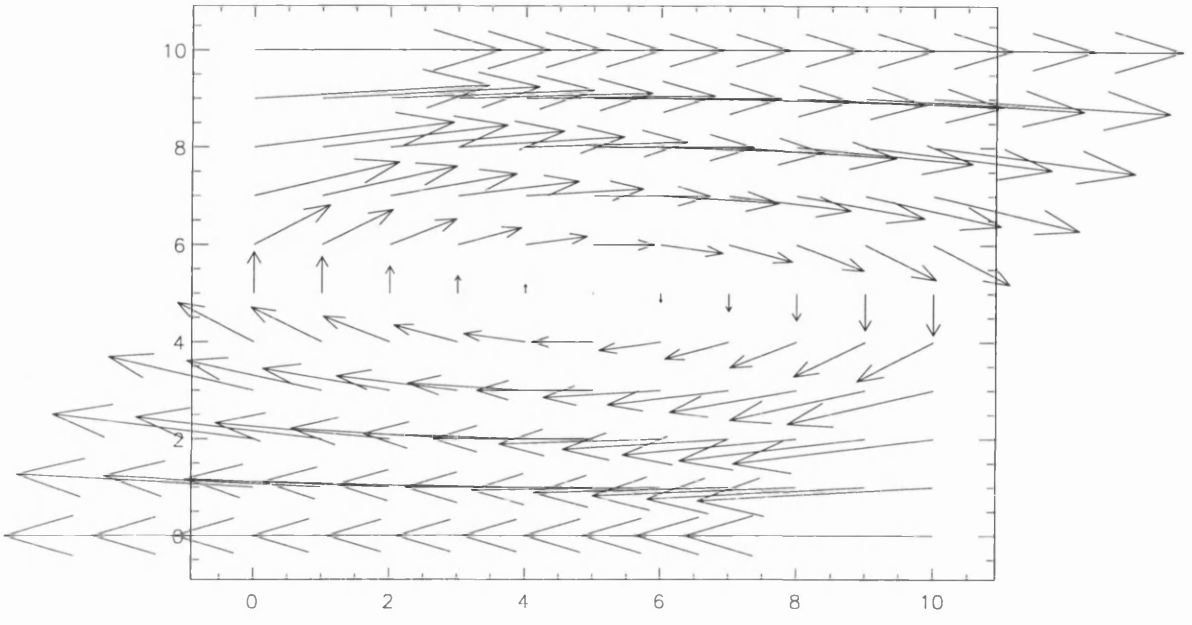

Figure 3.15: Magnetotail model (3.56) with perturbed λ, ϵ positive and within approximation.

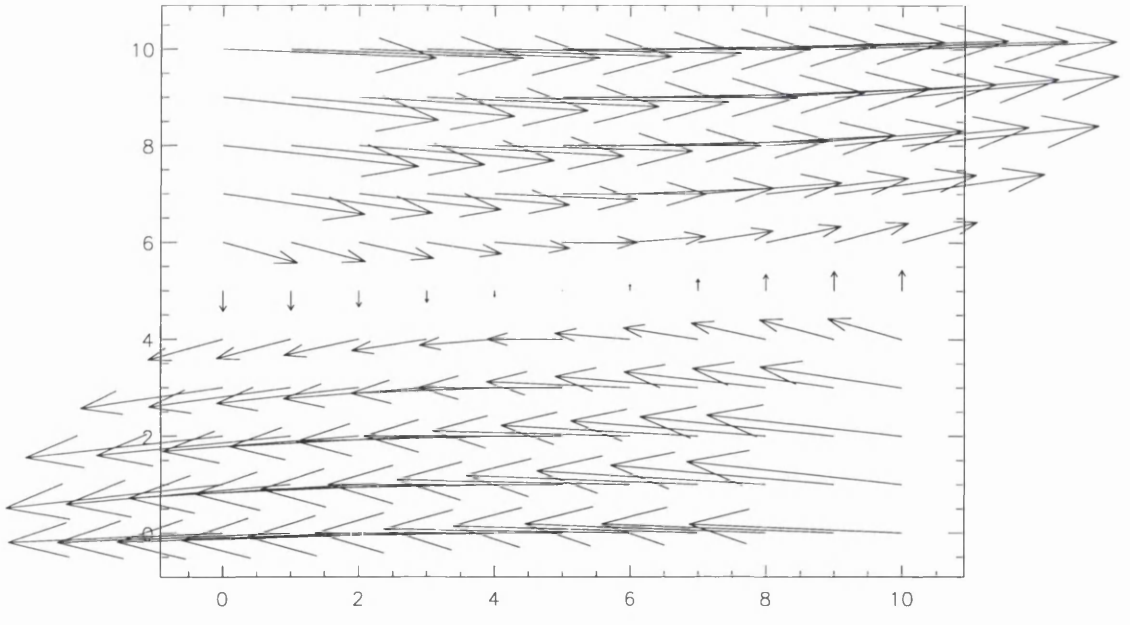

Figure 3.16: Magnetotail model (3.56) with perturbed boundary conditions, δ positive

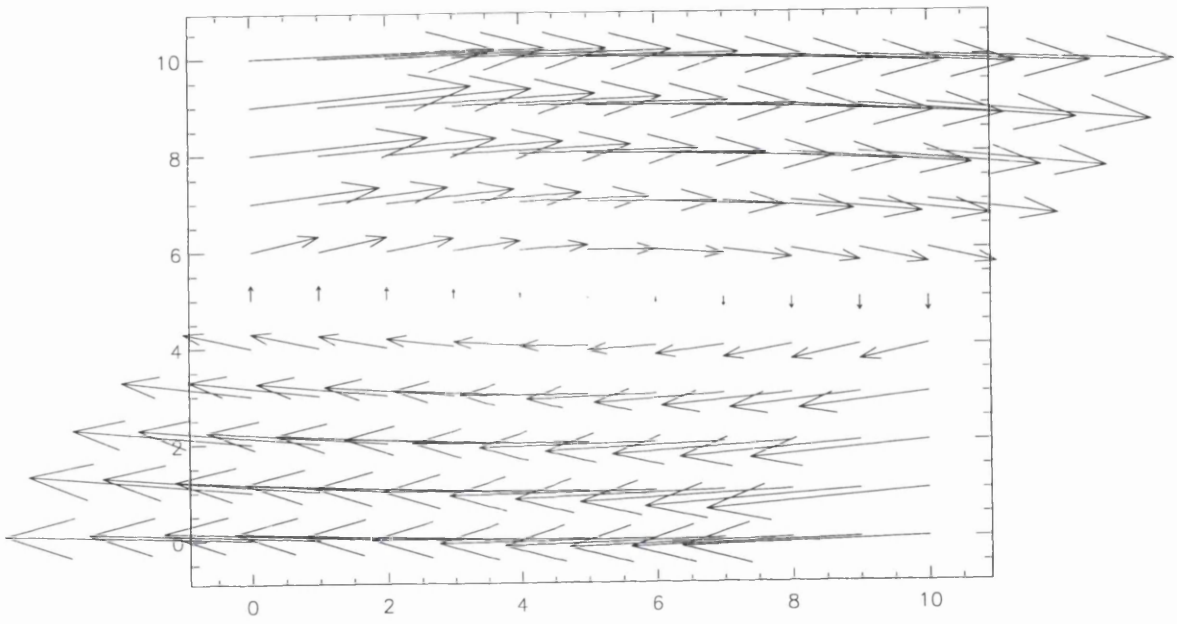

Figure 3.17: Magnetotail model (3.56) with perturbed boundary conditions, δ negative

Chapter 4

Time Dependent Field Aligned Flow

The field aligned equations of chapter 2 are considered with f time dependent. Two approaches are taken to their solution. The first method specifies the variable dependency of f (as in chapter 3) and derives conditions that the magnetic field must satisfy. The second method fixes the magnetic field and looks at the resulting equation for f. Note also that we will sometimes denote the partial derivative $\frac{\partial f}{\partial x}$ by f_x.

4.1 The Flow Function and the Governing Equation

Fully time dependent field aligned flow in this model is governed by two equations: (2.36)

$$(\mathbf{B}.\nabla)\, f = 0$$

and equation (2.38)

$$\frac{\partial}{\partial t}[\nabla \times (f\mathbf{B})] = (1-f^2)\mathbf{B} \times \nabla^2\mathbf{B} + \mathbf{B} \times [\nabla(1-f^2).\nabla]\mathbf{B}$$

both of which were derived in section 2.4. Although written in vector form, only the **z**-component of (2.38) is nonzero. Hence, in full we may write

$$B_\theta \frac{\partial^2 f}{\partial r \partial t} - \frac{B_r}{r}\frac{\partial^2 f}{\partial \theta \partial t} + \left[\frac{1}{r}\frac{\partial (rB_\theta)}{\partial r} - \frac{1}{r}\frac{\partial B_r}{\partial \theta}\right]\frac{\partial f}{\partial t} - \left\{ c\left[1-f^2\right] + \frac{\phi_2}{r}\frac{\partial\left(1-f^2\right)}{\partial r} + \frac{\phi_1}{r^2}\frac{\partial\left(1-f^2\right)}{\partial \theta}\right\} = 0 \tag{4.1}$$

where ϕ_1, ϕ_2 and c are defined in section (3.1). In this chapter we shall examine time dependent solutions to (2.36) and (4.1).

4.2 Solutions to the Full Flow Equation

We attempt here to look for solutions by specifying at the outset the variable dependency in the flow function f. It is found again that specifying f chooses very particular magnetic fields.

4.2.1 $f = f(r,t)$

By equation (4.1), $B_r(r,\theta)$ is identically zero. By the Maxwell equations, this chooses fields of the form $\mathbf{B} = B(r)\hat{\theta}$. Since now both the magnetic field and the velocity do not depend on θ, we must make sure that the pressure does not either: it is unreasonable to expect the pressure profile to depend on θ when nothing else does. Hence, $\phi_1 = B^2$ and $\phi_2 = 0$. Equation (4.1) reduces to the solution of

$$\frac{\partial}{\partial t}\left[\nabla \times (f\mathbf{B})\right] = \mathbf{0}$$
$$\text{i.e., } \nabla \times (f\mathbf{B}) = \mathbf{c}(r,\theta) \tag{4.2}$$

for some function $\mathbf{c}$. This may be solved by setting $\mathbf{c} = [0, 0, c(r)]$, $c(r)$ arbitrary, yielding,

$$f(r,t) = \frac{1}{rB}\int^r sc(s)\,ds + \frac{\tau(t)}{rB} \tag{4.3}$$

for arbitrary function $\tau(t)$. The corresponding fluid velocity is

$$\mathbf{u} = \left[\frac{\tau(t)}{r} + \int^r sc(s)\,ds\right]\hat{\theta} \tag{4.4}$$

When $c(r) = 0$, such a velocity profile is known in fluid mechanics as a line vortex. The underlying magnetic field geometry may be a complicated function of r, but the fluid motion in this case is well known, and is independent of the magnetic field. The pressure associated with this flow is found by substitution into (2.21), and is formally given by

$$\mu_0 p + \frac{B^2(r)}{2} = constant + \int_{r_0}^r \left[f^2(s) - 1\right]\frac{B^2}{s}ds - \tau'\theta \tag{4.5}$$

Again, this is the formal form that p must take in order to solve (2.21). Note, however, that this formal solution carries a θ dependence, which we have already disallowed on physical grounds. We can rectify the situation and balance the momentum equation in a satisfactory way by postulating the existence of an additional external force $\mathbf{F}$ that will drive the system in the correct fashion. Suppose the force is of the form $\mathbf{F} = F\hat{\theta}$: this choice is reasonable since the fluid is moving in the θ direction also. On examining the components of (2.21) we obtain

$$\hat{\mathbf{r}}: \quad -\frac{f^2B^2}{r} = -\frac{\partial p}{\partial r} - \frac{B}{r}\frac{d(rB)}{dr}$$
$$\hat{\theta}: \quad \frac{\partial f}{\partial t}B = -\frac{1}{r}\frac{\partial p}{\partial \theta} + F$$

Here we see the need for an external driving force. If $F = 0$, then integration of the θ-component of the momentum equation means the pressure function picks up a θ dependence, which is not permitted. However, if

$$F = \frac{\partial f}{\partial t} B$$

then the θ-component of (2.21) merely tells us that p is a function of r and t, as required. This force is a driver to the system which sets up the field aligned flow we have asked for. It ensures that the pressure profile is a function of r and t only, the same variables as magnetic field and the flow. One can imagine that the force F drives material around the concentric circles described by the field. The pressure is given by

$$\mu_0 p = -\frac{B^2(r)}{2} + \int_{r_0}^{r} \left[f^2(s) - 1\right] \frac{B^2}{s} ds + T(t)$$

where $T(t)$ is an arbitrary function of time to be determined by boundary conditions.

4.2.2 $f = f(\theta, t)$

When we ask for this variable dependency we find that this chooses fields $\mathbf{B} = \frac{\Theta(\theta)}{r}\hat{\mathbf{r}}$, by (2.36) and (2.20). However, when we substitute $f = f(\theta, t)$ with the correct field into (4.1) then we find that there exists no function f solving (4.1) for any choice of $\Theta(\theta)$. Again, the problem can be resolved by looking at components of the momentum equation (2.21) under the assumptions of this subsection.

$$\hat{\mathbf{r}}: \quad \frac{\partial f}{\partial t}\frac{\Theta}{r} - \frac{f^2\Theta^2}{r^3} = -\frac{\partial p}{\partial r}\frac{\Theta}{r}$$

$$\hat{\theta}: \quad 0 = \frac{\partial p}{\partial \theta} + \frac{\Theta\Theta'}{r^2}$$

Integration of the $\hat{\mathbf{r}}$ component will give a form for p and an arbitrary function $q_1(\theta, t)$: similarly, the θ-component also yield a form for p and a different arbitrary function $q_2(r, t)$. The two answers for p must be identical and therefore we must have

$$q_1(\theta, t) - q_2(r, t) = \frac{\Theta^2}{2r^2}\left[f^2 - 1\right] + \frac{\partial f}{\partial t}\Theta \log(r)$$

No such functions q_1, q_2 exist and therefore the momentum equation cannot be balanced by a suitably defined pressure p. Again, if we include an external force $\mathbf{F} = G\hat{\mathbf{r}}$ (in the same direction as the flow and the field) we can define a pressure for this field aligned system. The equivalence of the integrals of each components means that G, q_1 and q_2 must satisfy

$$q_1(\theta, t) - q_2(r, t) + \int^{r} G(s, \theta, t) ds = \frac{\Theta^2}{2r^2}\left[f^2 - 1\right] + \frac{\partial f}{\partial t}\Theta \log(r)$$

The pressure in this system is now given by

$$p = \int^{r} G(s, \theta, t) ds + q_1(\theta, t) - \frac{\Theta^2}{2r^2} f^2 - \frac{\partial f}{\partial t}\Theta \log(r)$$

and so (2.21) is balanced.

So by postulating an external force we can solve the system consistently. These external forces (F and G) can be imagined as drivers that push the fluid in the required manner in order to achieve a field aligned flow.

4.2.3 $f = f(t)$

With this choice of f, equation (2.36) is redundant and only (2.38) need be considered; this may be solved by separation of variables. If the constant of separation is c_1 then

$$\frac{\partial f}{\partial t} = c_1 \left(1 - f^2\right) \tag{4.6}$$

$$[\mathbf{B}.\nabla - c_1]\mathbf{j} = \mathbf{0} \tag{4.7}$$

The time dependency is now completely described by $f = \tanh(c_2 + c_1 t)$, $c_2 = constant$. The magnetic fields that support this flow function are given by solving (4.7). Note that as $t \to \pm\infty$, then $|f| \to 1$: in the limit, $|f| = 1$ which we know implies that any magnetic field topology will suffice. But the geometry is fixed by (4.7) since we originally considered $f = f(t)$ and not $f = \pm 1$. A field which solves this is $\mathbf{B} = \left(\frac{c_1 a}{r}, \frac{ba}{r} \exp\left[\frac{r^2}{2a}\right], 0\right)$, where a and b are arbitrary constants.

4.2.4 A Linear Analysis of the Full Equation

We examine first a linearised version of (4.1). We perturb the system by modifying the velocity field only. If we have an initial condition of time independence in the unperturbed flow function f_0 (the unperturbed velocity field is therefore time independent also) then we preserve field aligned flow by perturbing f_0 by a small, but time dependent function, i.e.,

$$f = f_0 + \epsilon g(r, \theta, t) \tag{4.8}$$

where ϵ is a small parameter. The total fluid velocity is then given by

$$\mathbf{u} = [f_0(r, \theta) + \epsilon g(r, \theta, t)] \frac{\mathbf{B}}{\sqrt{\mu_0 \rho}} \tag{4.9}$$

$\mathbf{B}$ is still time independent, but we allow time dependent flow. Substituting (4.8) into (4.1) and linearising to first order in ϵ, then by our choice of initial condition we obtain

$$\frac{\partial}{\partial t}[\nabla \times (g\mathbf{B})] + (f\mathbf{B}.\nabla)[\nabla \times (g\mathbf{B})] + (g\mathbf{B}.\nabla)[\nabla \times (f\mathbf{B})] = \mathbf{0} \tag{4.10}$$

If we further suppose that $g(r, \theta, t) = \tau(t) h(r, \theta)$ then we obtain by separation of variables

$$\tau(t) \propto \exp(ct) \tag{4.11}$$

and

$$c[\nabla \times (h\mathbf{B})] + (f\mathbf{B}.\nabla)[\nabla \times (h\mathbf{B})] + (h\mathbf{B}.\nabla)[\nabla \times (f\mathbf{B})] = \mathbf{0} \tag{4.12}$$

where c is the constant of separation. Since we are perturbing the velocity we must also satisfy

$$(\mathbf{B}.\nabla)\, h = 0 \tag{4.13}$$

to maintain incompressibility up to order ϵ, and hence exactly. It is at this point that the choice of initial conditions becomes important to further progress. As we have seen, fixing f chooses $\mathbf{B}$ and whether a properly defined h exists that can solve (4.12, 4.13) is not obvious. If we choose $f = constant \neq \pm 1$ then we know we have chosen fields such that $(\mathbf{B}.\nabla)\mathbf{j} = \mathbf{0}$ by section (3.3.1). This in turn may be satisfied by putting $\mathbf{j} = \nabla \times \mathbf{B} = \mathbf{0}$, leaving the equation

$$(c + f\mathbf{B}.\nabla)\,[\nabla h \times \mathbf{B}] = 0 \tag{4.14}$$

along with (4.13). A simple solution is $h = constant$, yielding

$$\mathbf{u} = [f + \epsilon h \exp(ct)]\,\frac{\mathbf{B}}{\sqrt{\mu_0 \rho}} \tag{4.15}$$

The fluid moves along the field lines, chosen by the initial conditions by defining f, with a velocity either exponentially decreasing or increasing, depending on the sign of c. For c positive, the analysis will become invalid when the perturbing function $\epsilon g(r, \theta, t)$ becomes too large. When c is negative, the fluid will eventually become static everywhere.

In general, however, to continue with this analysis we have to solve 4 equations simultaneously: (2.36), (2.38), (4.12) and (4.13). We can sidestep (2.36) and (2.38) by explicitly choosing $f = \pm 1$ as the equilibrium condition. This means that $\mathbf{B}$ is not fixed by the unperturbed state, leaving us free to find a suitable $g, \mathbf{B}$ pair that solve (4.12) and (4.13). Therefore, this analysis asks for those fields that can support a first order time dependency in a field aligned flow travelling close to the Alfvén velocity.

Again we have two equations to solve simultaneously, and we may proceed in a similar way as in the time independent case, as considered in section 3.1. Although (4.12) is a second order differential equation in h, when we preferentially remove the partial derivatives of h, using (4.13), the second order derivatives drop out. By (4.13), $\frac{\partial h}{\partial r} = -\vartheta \frac{\partial h}{\partial \theta}$ (where ϑ is defined in section 3.1) leaving

$$\frac{\partial h}{\partial \theta} = C(r, \theta)\, h \tag{4.16}$$

where

$$C = -B_r \times \frac{C_1}{C_2}$$

$$C_1 = 2rB_\theta \frac{\partial^2 B_\theta}{\partial\theta\partial r} - 2rB_r \frac{\partial^2 B_r}{\partial\theta\partial r} + 2r^2 B_r \frac{\partial^2 B_\theta}{\partial r^2} - 2B_\theta \frac{\partial^2 B_r}{\partial\theta^2} + (B_r - cr)\frac{\partial B_r}{\partial\theta}$$

$$+r(2B_r + cr)\frac{\partial B_\theta}{\partial r} + 2B_\theta \frac{\partial B_\theta}{\partial\theta} + B_\theta(cr - 2B_r)$$

$$C_2 = -2B_r B_\theta \left[\frac{\partial B_r}{\partial \theta} + r\frac{\partial B_\theta}{\partial r}\right] + (B_r^2 - B_\theta^2)\left[\frac{\partial B_\theta}{\partial \theta} - r\frac{\partial B_r}{\partial r}\right] + (B_r^2 + B_\theta^2)(B_r - cr)$$

Similarly, with $\frac{\partial h}{\partial \theta} = -\frac{1}{v}\frac{\partial h}{\partial \theta}$ we get

$$\frac{\partial h}{\partial r} = D(r,\theta)\, h \tag{4.17}$$

where

$$D = -\frac{B_\theta}{r} \times \frac{D_1}{D_2}$$

with

$$D_1 = -2rB_r \frac{\partial^2 B_r}{\partial \theta \partial r} - 2B_\theta \left[\frac{\partial^2 B_r}{\partial \theta^2} - \frac{\partial B_\theta}{\partial \theta}\right] + r^2 \left[2B_r \frac{\partial^2 B_\theta}{\partial r^2} + c\frac{\partial B_\theta}{\partial r}\right]$$
$$+(2B_r - cr)\left[\frac{\partial B_\theta}{\partial r} - B_\theta\right]$$

$$D_2 = 2B_r B_\theta \left[\frac{\partial B_r}{\partial \theta} + r\frac{\partial B_\theta}{\partial r}\right] + (B_r^2 - B_\theta^2)\left[r\frac{\partial B_r}{\partial r} - \frac{\partial B_\theta}{\partial \theta}\right]$$

We make another 'smoothness' assumption, this time on h, i.e.,

$$\frac{\partial}{\partial r}\left[\frac{\partial h}{\partial \theta}\right] = \frac{\partial}{\partial \theta}\left[\frac{\partial h}{\partial r}\right]$$

Substituting in (4.16) and (4.17) we arrive at

$$\frac{\partial D}{\partial \theta} - \frac{\partial C}{\partial r} = 0 \tag{4.18}$$

The fields which satisfy (4.18) support a first order time dependency in field aligned flow, when perturbed away from the Alfvén velocity. A simple solution to (4.18) is $\mathbf{B} = B(r)\,\hat{\theta}$ which yields

$$g = \frac{a}{rB}$$

where $a = constant$. This has a perturbed flow profile of

$$\mathbf{u} = \left[1 + \frac{\epsilon a \exp(ct)}{rB}\right] B\hat{\theta} = \left[B + \frac{\epsilon a \exp(ct)}{r}\right] \hat{\theta}$$

This solution is only valid in the region of time and space where $\frac{fB\exp(-ct)}{a} \gg \epsilon$. The fluid motion is the original fluid motion with a line vortex motion superposed.

The above analyses suggest that time dependent flow along time independent field lines is possible. Instead of considering the full equation for time dependent field aligned flow we next consider a reduced form by specifying at the outset a particular class of magnetic fields.

4.3 A Nonlinear Flow Equation

A major drawback of the linear analysis in section (4.2.4) is that it is not possible to let the linearised quantities generate any 'sharp' or 'large' features in such a system without the analysis breaking down. Such features in the flow function f would indicate a position in space and time where the magnetohydrodynamic model breaks down. We are drawn to attempt a more demanding treatment to find if such features exist in this model. We consider a reduced form of (2.38), making the assumption that

$$(\mathbf{B}.\nabla)\mathbf{j} = \mathbf{0} \tag{4.19}$$

This drops the $(1-f^2)$ term in (2.38). This choice of field may appear rather arbitrary, but has been seen before in section 3.3.1. We have chosen to work with those fields that support $f = constant$; this can permit a linear analysis making the flow function weakly time dependent, in a similar fashion to that seen above in section 4.2.4. Allowing $f = f(r,\theta,t)$ equation (2.38) becomes

$$\frac{\partial}{\partial t}[\nabla \times (f\mathbf{B})] = \mathbf{B} \times [\nabla(1-f^2).\nabla]\mathbf{B} \tag{4.20}$$

or, in scalar form

$$\begin{aligned} B_\theta \frac{\partial^2 f}{\partial r \partial t} - \frac{B_r}{r}\frac{\partial^2 f}{\partial \theta \partial t} + \left[\frac{1}{r}\frac{\partial(rB_\theta)}{\partial r} - \frac{1}{r}\frac{\partial B_r}{\partial \theta}\right]\frac{\partial f}{\partial t} \\ - \left\{\frac{\phi_2}{r}\frac{\partial\left(1-f^2\right)}{\partial r} + \frac{\phi_1}{r^2}\frac{\partial\left(1-f^2\right)}{\partial \theta}\right\} = 0 \end{aligned} \tag{4.21}$$

Equation (4.21) is a nonlinear partial differential equation in three variables which must be solved in conjunction with (3.8). Using (3.8) we can eliminate f_r in preference to f_θ and vice versa. This process yields two equations

$$f_{rt} + \alpha_1(r,\theta) f_t + \beta(r,\theta) f f_r = 0 \tag{4.22}$$

$$f_{\theta t} + \alpha_2(r,\theta) f_t + \beta(r,\theta) f f_\theta = 0 \tag{4.23}$$

where

$$\beta = \frac{B_\theta \phi_1 - B_r \phi_2}{r\left(B_r^2 + B_\theta^2\right)} \tag{4.24}$$

$$\alpha_1 = -\vartheta \alpha_2 \tag{4.25}$$

$$\alpha_2 = -\frac{B_r}{B_r^2 + B_\theta^2}\left[\frac{\partial(rB_\theta)}{\partial r} - \frac{\partial B_r}{\partial \theta}\right] = -\frac{rB_r}{B_r^2 + B_\theta^2}\hat{\mathbf{z}}.\mathbf{j} \tag{4.26}$$

Both (4.22) and (4.23) must give the same answer on integration and this generates conditions that $\mathbf{B}$ must satisfy in order that this is so. Again, this is done by imposing a condition on f, namely,

$$[f_{\theta t}]_r = [f_{rt}]_\theta \tag{4.27}$$

This gives a rather complicated set of conditions on $\mathbf{B}$,

$$\frac{\partial \alpha_2}{\partial r} - \frac{\partial \alpha_1}{\partial \theta} = 0 \tag{4.28}$$

and

$$\frac{\partial \beta}{\partial r} = -\alpha_1 \beta \tag{4.29}$$

$$\frac{\partial \beta}{\partial \theta} = -\alpha_2 \beta \tag{4.30}$$

If we assume

$$\frac{\partial}{\partial r}\left[\frac{\partial \beta}{\partial \theta}\right] = \frac{\partial}{\partial \theta}\left[\frac{\partial \beta}{\partial r}\right] \tag{4.31}$$

then we can drop either (4.29) or (4.30), since this effectively restates condition (4.28).

In the following work, we do not expressly find a field that solves (4.28) and (4.29), but merely look for a region in a known field where the conditions hold approximately. Some of the fields that solve $(\mathbf{B}.\nabla)\mathbf{j} = \mathbf{0}$ described in chapter 3 also solve (4.28) and (4.29) approximately in certain regions of space. These regions were found by substituting a test field and numerically calculating the left hand side of each equation. By specifying an upper limit to the deviation away from a perfect solution, one can find regions of approximate solution. It was found that the Bessel function field (3.24) contained regions where (4.28) and (4.29) held approximately (where we assume (4.31)).

Any solutions to the nonlinear equations (4.22),(4.23) may be applied in these regions. They are also consistent with a linear analysis where the flow function equation used is also approximate (see section (4.3.1)). Further comment on these conditions is reserved for section (4.3.5). We may attempt to solve each of (4.23,4.22) separately: treating θ as a parameter in the coefficients of (4.23), for example, and similarly with r (4.22). Assuming that we have a suitable magnetic field, this approach leads naturally to the study of an equation of the form

$$f_{xt} + \alpha(x) f_t + \beta(x) f f_x = 0 \tag{4.32}$$

as the main equation of interest; a second order nonlinear, hyperbolic partial differential equation. The coefficients depend on the magnetic field and contain information on the time and space gradients in the problem. It is easily seen that

$$\alpha(x) \sim \frac{1}{lengthscale} \sim \text{space gradient}$$

$\alpha(x)$ carries information on the space evolution of the flow function. $\beta(x)$ is more subtle and entertains an aspect of nonlinearity

$$\beta(x) \sim \frac{1}{time} \times \frac{1}{size\ of\ f} \sim \text{characteristic frequency}$$

β carries not only the time evolution information, but also a solution 'amplitude'. Equation (4.32) forms the basis for further investigation into time dependent field aligned flow.

In the following sections (4.3.1 -4.3.4) we shall consider (4.32) under a hierarchy of increasingly sophisticated assumptions.

4.3.1 Linear Analyses

Linear analysis of (4.32) will provide some understanding of its full nonlinear behaviour. The nonlinear term is simple and the equation is already in canonical form. We will use a variety of assumptions and methods to draw information out about the nature of the flow function.

Suppose f is of the form

$$f = f_0(x,t) + \epsilon\psi(x,t) \tag{4.33}$$

where $f_0(x,t)$ solves (4.32). On substitution, ϵ is a small parameter we can use to linearise equation (4.32). By the initial conditions we arrive at the linear equation

$$\psi_{xt} + \alpha\psi_t + \beta f_0\psi_x + \beta(f_0)_x\psi = 0 \tag{4.34}$$

Equation (4.34) governs a small linear perturbation to the main flow function $f_0(x,t)$, and we can use this to investigate modes of behaviour for various values of α, β. The last two terms carry the original, fully time and space dependent background flow. We have gone from a nonlinear partial differential equation with space dependent coefficients to a linear inhomogeneous equation with time and space dependent coefficients. It is arguable that the linear equation is not much of an improvement on the nonlinear one, as we have made the coefficients dependent on both time and space. If we make the further simplification

$$f_0 = constant$$

then (4.34) becomes

$$\psi_{xt} + \alpha\psi_t + (\beta f_0)\,\psi_x = 0 \tag{4.35}$$

which is a much friendlier equation. We shall use (4.35) as a basis for further study. Note that we have bracketed the coefficient of g_x. It can now be seen that $\beta f_0 \sim \frac{1}{time}$ and represents a natural frequency in this problem. It depends on both the background dominant flow and β the time evolution parameter. Equation (4.35) now means we are looking at small time dependent perturbations away from a constant background flow function; in effect, we are considering the linear onset of time dependent field aligned flow from a time independent background.

In the following sections 4.3.2-4.3.5 we shall make various assumptions on α, β and ψ in order to draw out some of the behaviour possible from solutions to equation (4.35). We shall also show how these assumptions relate to each other, and how the linear analysis can be linked to solutions of the nonlinear flow problem, equation (4.32).

4.3.2 Solution by separation of variables

Using the trial solution $\psi = \tau(t)\,X(x)$ in (4.35) we can separate the variables to get

$$\psi(x,t) \propto \exp\left[ct - c\int_{x_0}^{x} \frac{\alpha(x')}{c + f_0\beta(x')}dx'\right] \tag{4.36}$$

where c is the separation constant. This solution is valid where $\mid \frac{f}{\psi} \mid \gg \epsilon$. We have a rather complicated space dependence and hence this solution does not tell us much that is immediately useful about the flow function. If we put $c = m + in$, permitting evanescent/oscillatory behaviour, then the resulting flow function is

$$\psi(x,t) \propto \exp\left[(m+in)t\right] \times \exp\left\{ -\int_{x_0}^{x} \alpha(x') \left[\frac{m^2 + n^2 + m f_0 \beta(x') + i n f_0 \beta(x')}{\left[m + f_0 \beta(x')\right]^2 + n^2} \right] dx' \right\} \tag{4.37}$$

Retaining the full space dependence of α and β means that this is really only formally useful and almost certainly not integrable; the flow behaviour in this linearisation is still not easily understood. A more measured approach is required that will yield relevant answers. However, (4.36) is of primary importance when considering the nonlinear analysis of section 4.3.5, where the conditions for validity are discussed.

4.3.3 $\alpha = constant$ and $\beta = constant$

Consider (4.34) with $\alpha = constant$ and $\beta = constant$. These are the simplest forms that α, β can take. With these values, (4.34) becomes very simple

$$\psi_{xt} + \alpha \psi_t + \beta f_0 \psi_x = 0$$

We can substitute a linear travelling wave form

$$\psi \sim \exp\left[i\left(kx - \omega t\right)\right] \tag{4.38}$$

By choosing this form, we are effectively looking at the Fourier components of a disturbance in the plasma. Hence the relation between ω and k will tell us what happens to particular frequencies in the Fourier decomposition of the disturbance.

On substitution, we have

$$\omega k - i\omega\alpha + ik\beta f_0 = 0 \tag{4.39}$$

In general, ω and k may be complex; therefore if we substitute

$$\omega = \omega_R + i\omega_I \quad k = k_R + ik_I \tag{4.40}$$

then, on expressing k_R, k_I in terms of ω_R, ω_I we get

$$k_R = \alpha \left[\frac{\omega_R \beta f_0}{\omega_R^2 + \left(\omega_I + \beta f_0\right)^2} \right], \quad k_I = \alpha \left[\frac{\omega_R^2 + \omega_I^2 + \omega_I \beta f_0}{\omega_R^2 + \left(\omega_I + \beta f_0\right)^2} \right] \tag{4.41}$$

We are now in a position to analyse the behaviour of these waves and their relation to other solution forms.

1. **Equivalence to equation 4.37**

This treatment is entirely equivalent to equation (4.37) when we set $\beta(x) = \beta = constant$ and $\alpha(x) = \alpha = constant$. With this, (4.37) becomes

$$\psi(x,t) \propto \exp\left[(m+in)t\right] \times \exp\left\{\alpha\left[\frac{m^2+n^2+mf_0\beta}{n^2+\left[m+f_0\beta\right]^2}\right]x - i\alpha\left[\frac{nf_0\beta}{n^2+\left[m+f_0\beta\right]^2}\right]x\right\} \tag{4.42}$$

By comparing the time dependencies of (4.42) with (4.38) we find that $m \equiv \omega_I$ and $n \equiv -\omega_R$. This means that the evanescent and oscillatory coefficients of x in (4.42) are equal to those of (4.41), i.e,

$$k_R = \alpha\frac{-nf_0\beta}{n^2+\left[m+f_0\beta\right]^2} \quad k_I = \alpha\frac{m^2+n^2+mf_0\beta_0}{n^2+\left[m+f_0\beta\right]^2}$$

2. **'Long lasting' waves**

We may more usefully express k_R, k_I in terms of dimensionless variables scaled to the natural frequency. Therefore with $a = \frac{\omega_R}{\beta f_0}, b = \frac{\omega_I}{\beta f_0}$

$$\frac{k_R}{\alpha} = \frac{a}{a^2+(b+1)^2}, \quad \frac{k_I}{\alpha} = \frac{a^2+b^2+b}{a^2+(b+1)^2} \tag{4.43}$$

For a long lasting wave, i.e., one that remains for a number of wavelengths we require that the evanescent time factor ω_I is much greater than the oscillatory frequency ω_R. This will tell us the components of any disturbance that propagate the furthest without significant exponential growth or decay.

In terms of the frequencies ω_I and ω_R we must have

$$\left|\frac{\omega_I}{\omega_R}\right| \ll 1 \tag{4.44}$$

Therefore

$$\left|\frac{b}{a}\right| \ll 1 \tag{4.45}$$

We also need the motion to be weakly damped in space (characterised by the value of k_I) compared to the oscillatory wavenumber k_R i.e.,

$$\left|\frac{k_I}{k_R}\right| \ll 1 \tag{4.46}$$

or,

$$\left|\frac{a^2+b^2+b}{a}\right| \ll 1 \tag{4.47}$$

We can guarantee this by setting

$$|a| \ll 1 \tag{4.48}$$

This simply says that the wave must have an oscillatory frequency much less than the natural frequency, and that the evanescent decay constant ω_I must be much less this.

We can also carry out the same analysis by expressing ω_R, ω_I in terms of k_R, k_I. Using the dimensionless variables $c = \frac{k_R}{\alpha}, d = \frac{k_I}{\alpha}$,

$$\frac{\omega_R}{\beta f_0} = \frac{c}{c^2 + (1-d)^2}, \quad \frac{\omega_I}{\beta f_0} = \frac{d - d^2 - c^2}{c^2 + (1-d)^2} \tag{4.49}$$

Using the long lasting wave conditions (4.44), (4.46) we obtain similar conditions

$$| c | \ll 1, \quad | \frac{d}{c} | \ll 1 \tag{4.50}$$

Hence to have long lasting waves we need to be in a region of parameter space where the evanescent terms are very much smaller than the oscillatory terms which themselves are very much smaller than the natural, background scales α and βf_0. This demands that the wave changes more slowly in time and space than the medium it is moving in. The 'larger' scale wave effectively does not see the medium in the limit.

3. **Wave behaviour in limits of ω_R, ω_I and βf_0**

   We can force the system into extreme situations if we take one of the parameters to be very much larger than the other.

   (a) When $\beta f_0 \to \infty$, then the natural frequency increases. This forces $a \to 0$ and $b \to 0$ meaning that $k_I, k_R \to 0$. Hence the space dependent part of the wave disappears and we are left with an oscillatory/evanescent motion, $\psi \sim \exp[i(\omega_R + \omega_I)t]$.

   (b) As $\omega_R \to \infty$, $a \to \infty$. The wavenumber k tends to $i\alpha$, i.e., $k_R = 0$, $k_I = \alpha$ in the limit. Again the wave nature of the solution is destroyed as $\psi \sim \exp(-\alpha x)\exp(i\omega t)$; the wave is damped according to the natural scale. By asking for too high an oscillatory frequency, we have damped out the wave.

   (c) Similarly, as $\omega_I \to \infty$, the wave tends to $\psi \sim \exp(-\alpha x)\exp(i\omega t)$. We can conclude that if we want the wave to change quickly in time, we will destroy the wave nature of the motion. The motion will either die or grow with increasing x according to the sign of α, meaning that this linear analysis is no longer adequate to handle the increasing amplitude of the perturbed solution.

Hence those waves with ω_R and ω_I very much larger than the natural background frequency βf_0 undergo either exponential growth or decay at the rate given by α. If growth is seen, then one must move to a nonlinear treatment to properly take account of the growing components.

4.3.4 Assumed form for $\alpha(x), \beta(x)$

In this section we assume that $\alpha(x)$ and $\beta(x)$ depend linearly on x, i.e., we have

$$\alpha(x) = \alpha_0 + \alpha_1(x - x_0)$$

and

$$\beta(x) = \beta_0 + \beta_1(x - x_0)$$

exactly everywhere. Just as we have already assumed α, β to be constant everwhere (see section 4.3.3), we now assume that they are exactly linear functions of the independent variable x.

We can specialise this solution to the case of $\alpha(x)$ and $\beta(x)$ both slowly varying simply by assuming that $\left|\frac{\alpha_1[x-x_0]}{\alpha_0}\right| \ll 1$ and $\left|\frac{\beta_1[x-x_0]}{\beta_0}\right| \ll 1$ for some x_0. This would be equivalent to saying that the medium is nearly homogeneous. If we do *not* constrain the x dependence, we can look at the behaviour of wavelike solutions in a strongly inhomogeneous medium.

With the forms of α and β above, equation (4.35) becomes

$$\psi_{xt} + [\alpha_0 + \alpha_1 (x - x_0)]\,\psi_t + f_0\,[\beta_0 + \beta_1\,(x - x_0)]\,\psi_x = 0 \tag{4.51}$$

The x-dependent coefficients mean that we cannot use a simple linear travelling wave. However, we may still have a harmonic time dependence, $e^{-i\omega t}$. Therefore we postulate a solution of the form

$$\psi(x,t) \sim \phi(x) e^{-i\omega t} \tag{4.52}$$

with $\omega = \omega_R + i\omega_I$

On substitution we get an equation for ϕ;

$$\frac{\phi_x}{\phi} = E(x) + iF(x) \tag{4.53}$$

where

$$E(x) = -[\alpha_0 + \alpha_1(x - x_0)]\left[\frac{\omega_R^2 + \omega_I[\omega_I + f_0\beta_0 + f_0\beta_1(x - x_0)]}{\omega_R^2 + [\omega_I + f_0\beta_0 + f_0\beta_1(x - x_0)]^2}\right] \tag{4.54}$$

and

$$F(x) = [\alpha_0 + \alpha_1(x - x_0)]\left[\frac{\omega_R[f_0\beta_0 + f_0\beta_1(x - x_0)]}{\omega_R^2 + [\omega_I + f_0\beta_0 + f_0\beta_1(x - x_0)]^2}\right] \tag{4.55}$$

Equation (4.53) can be integrated in closed form, yielding

$$\phi(x) \propto \exp[Q_1(x)] \exp[iQ_2(x)] \tag{4.56}$$

where, with $n = 1, 2$,

$$Q_n(x) = -\frac{B_n D_n}{G^2}(x - x_0) + \frac{Y(x)}{2G^3}\left[2B_n H D_n - GA_n D_n - GB_n C_n\right]$$
$$+\frac{Z(x)}{G^3\omega_R}\left[EB_n D_n + HA_n D_n + GHB_n C_n - G^2 A_n C_n - H^2 B_n D_n\right]$$

and

$$Y(x) = \tan^{-1}\left[\frac{H + G(x - x_0)}{\omega_R}\right], \quad Z(x) = \log\left\{E + [F + G(x - x_0)]^2\right\}$$

where $E = \omega_R^2$, $H = \omega_I$, $G = f_0\beta_1$ and

$$A_1 = \alpha_0 \quad A_2 = -\alpha_0$$
$$B_1 = \alpha_1 \quad B_2 = -\alpha_1$$
$$C_1 = \omega_R^2 + \omega_I^2 + \omega_I f_0\beta_0 \quad C_2 = \omega_R f_0\beta_0$$
$$D_1 = \omega_I f_0\beta_1 \quad D_2 = \omega_R f_0\beta_1$$

Consider (4.53) as $|x - x_0| \to 0$; the evanescent and oscillatory factors $E(x)$ and $F(x)$ become

$$E(x) \to -\alpha_0 \left[\frac{\omega_R^2 + \omega_I^2 + \omega_I f_0\beta_0}{\omega_R^2 + [\omega_I + f_0\beta_0]^2}\right] \stackrel{def}{\equiv} E_0$$

$$F(x) \to \alpha_0 \left[\frac{\omega_R\beta_0 f_0}{\omega_R^2 + [\omega_I + f_0\beta_0]^2}\right] \stackrel{def}{\equiv} F_0$$

which recovers the results of the $\alpha, \beta = constant$ analysis (section 4.3.3). Locally, the medium is homogeneous and not position dependent. In the opposite limit of $|x - x_0| \to \infty$ the coefficients $\alpha(x)$, $\beta(x)$ tend to infinity also for α_1, β_1 nonzero. Equations (4.54) and (4.55) become

$$E(x) \to -\frac{\alpha_1\omega_I}{f_0\beta_1} \stackrel{def}{\equiv} E_\infty$$

$$F(x) \to \frac{\alpha_1\omega_R}{f_0\beta_1} \stackrel{def}{\equiv} F_\infty$$

respectively. The numerical values of the space oscillatory and evanescent factors ($F(x)$ and $E(x)$ respectively) depends on their time equivalents ω_R and ω_I, i.e., if the wave is damped only very slightly in time then this scale percolates through to the space evanescent properties.

Suppose now we assume that the wave has no evanescent time component, i.e., $\omega_I = 0$. In the limit of $|x - x_0| \to \infty$, we have

$$F_\infty \to \frac{\alpha_1\omega_R}{f_0\beta_1}$$

but

$$E_\infty \to 0$$

The wavelength of the motion transforms slowly to a fixed value while the evanescent behaviour disappears completely. The new wavelength is entirely independent of the wavelength at $|x - x_0| \to 0$ since it depends on the gradient terms α_1 and β_1. If we let $\beta_1 \to 0$ keeping α_1 nonzero and $\frac{\omega_R}{f_0} \neq 0$ then

$$|F_\infty| \to \infty \tag{4.57}$$

This means that as $\beta(x) \to constant$ with $\alpha(x)$ dependent on x, the wavenumber $F(x)$ will tend to infinity, meaning that the effective wavelength of the motion will tend to zero.

If we have

$$\left|\frac{\beta_0\alpha_1}{\alpha_0\beta_1}\right| > \frac{(\beta_0 f_0)^2}{\omega_R^2 + (\beta_0 f_0)^2} \tag{4.58}$$

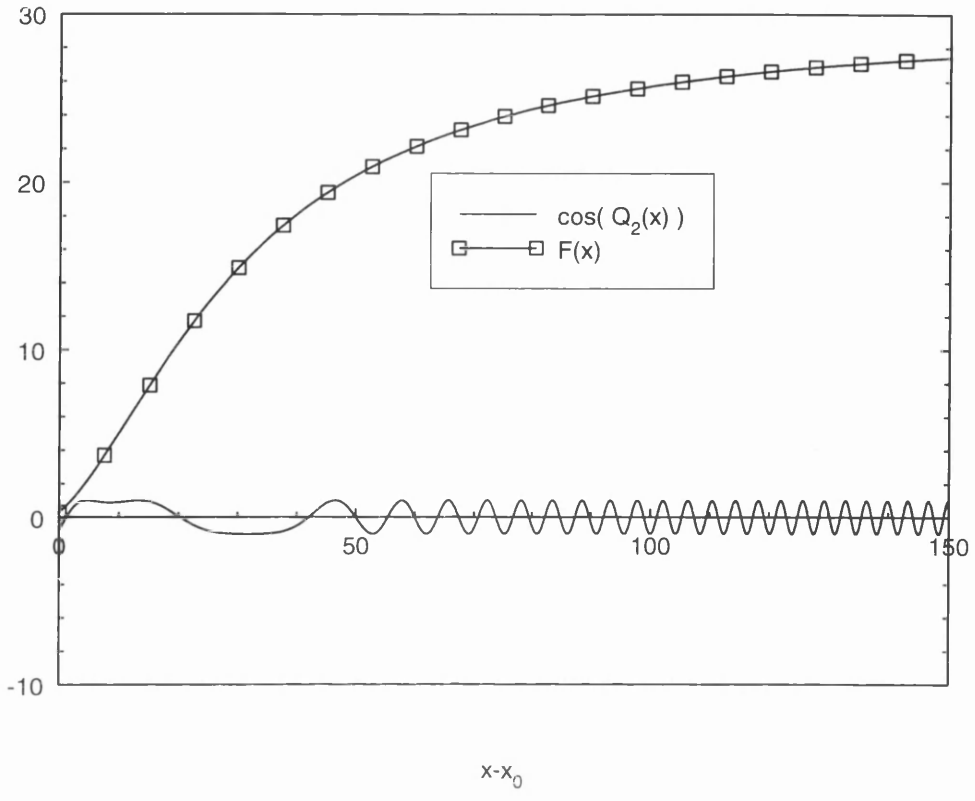

Figure 4.1: $F(x)$, $\cos[Q_2(x)]$ for $\alpha_0 = 1$, $\alpha_1 = 1$, $\beta_0 = 10$, $\beta_1 = 1$, $\omega_R = 30.1$ and $f_0 = 1$

then the equivalent of the wavenumber $F(x)$ will increase smoothly via equation (4.55) from F_0 to F_∞, a value numerically larger than F_0. Since F increases, we are effectively making the wavelength of the motion shorter. Figure 4.1 shows the behaviour of $F(x)$ and $\cos(\int F(x)dx)$ when (4.58) is true. The 'wavenumber' F transforms smoothly into a larger positive number, causing the wavelength of the motion to decrease with increasing x.

This amounts to moving the energy present in the motion onto a different length scale. If (4.58) is true, then the energy is moving to a shorter length scale. If not, then the energy is being redistributed onto longer scales.

This analysis brings out yet another feature of (4.35). Remember that this equation describes the linear onset on time dependent field aligned flow. Therefore, we have shown that if we give the correct variable dependence to the coefficients of the differential terms in (4.35) (see equations (4.51) and (4.58)), the motion can effectively redistribute energy onto different length scales, in this linear limit. In the limit $\beta_1 \to 0$, this length scale tends to zero since $|F_\infty| \to \infty$. This, coupled with the possible exponential growth behaviour seen in section 4.3.3 points to something interesting in the behaviour of the nonlinear flow equation, (4.32). The following section deals with this equation and shows how it may be linked to the linear analyses of sections 4.3.1-4.3.4. See figure 4.4 for a schematic outline of treatments given for equation (4.32).

4.3.5 Nonlinear Analysis

We turn away now from linear analyses to considering equation (4.32) directly, a hyperbolic equation with non-constant coefficients. Unfortunately, we do not possess an analytic solution to (4.32) for α and β both fully varying in x. To ease the problem slightly, we suppose that $\beta = constant$ or at least $\frac{\partial \beta}{\partial x} \ll \frac{\partial \alpha}{\partial y}$. This can be seen as a nonlinear treatment of the case seen in the previous section, where $\beta(x) = constant$ but $\alpha \neq constant$ (see equation (4.57)). Since the equation is hyperbolic, we postulate solutions of the form $f = \psi[\eta]$, where $\eta = c_1(x) + c_2 t$ On substitution into (4.32) we obtain

$$\psi_{\eta\eta} + \frac{\alpha(x)}{c_1'}\psi_\eta + \frac{\beta}{c_2}\psi\psi_\eta = 0$$

If we set $\frac{\alpha(x)}{c_1'} = k = constant$ and $\frac{\beta}{c_2} = z$ then we have a second order ordinary differential equation with constant coefficients,

$$\psi_{\eta\eta} + k\psi_\eta + z\psi\psi_\eta = 0$$

or,

$$\frac{\partial}{\partial \eta}\left[\psi_\eta + k\psi + \frac{z}{2}\psi^2\right] = 0$$

which, on integration yields

$$\psi_\eta + k\psi + \frac{z}{2}\psi^2 = w \tag{4.59}$$

where $w \neq w(\eta)$. We have reduced the problem from a nonlinear hyperbolic partial differential equation to solving a Ricatti equation, some solutions of which are detailed below. We will examine a particular class of solution by specifying that

$$\psi = \frac{p}{q + h(\eta)} \tag{4.60}$$

Equation (4.59) now becomes an equation in h, i.e.,

$$h_\eta + h\left[\frac{2qw}{p} - k\right] + h^2\left[\frac{w}{p}\right] = kq + \frac{zp}{2} - \frac{wq^2}{p} \tag{4.61}$$

Although we still have a Ricatti equation, we can manipulate this form to generate analytic answers relatively easily. Additionally, we will only consider the simpler solutions of the large spectrum available.

1. Consider (4.59),(4.61) with $w = 0$.

   Equation (4.61) is now a linear equation since this specification drops the h^2 term:

   $$h_\eta - kh = kq - \frac{zp}{2}$$

   Additionally, if we set $k = 1$ and $kq - \frac{zp}{2} = 0$ then $h = \exp(\eta)$. Working back through the variables we find that this fixes c_1 and c_2 to be

   $$c_1 = \int^x \alpha(x')dx', \quad c_2 = \frac{\beta p}{2q}$$

with final solution

$$f(x,t) = \frac{p}{q + \exp\left[\int_{x_0}^{x} \alpha(x')\,dx' - \frac{\beta p}{2q}t\right]} \tag{4.62}$$

Detailed comment on this solution can be found later in this section.

2. Consider (4.59),(4.61) with $\frac{2qw}{p} - k = 0$.

This drops the term proportional to h in equation (4.61), so we still have a nonlinear equation to solve, namely

$$h_\eta + \frac{w}{p}h^2 = \frac{q^2 w}{p} - \frac{zp}{2} \tag{4.63}$$

which has two solutions depending on the sign of the coefficient of h^2.

(a) Suppose that $h = mQ$, for $m = constant \neq 0$ and $Q = Q(\eta)$. Then on substitution into (4.63) we have

$$Q_\eta + \frac{mw}{p}Q^2 = \frac{1}{m}\left[\frac{q^2 w}{p} - \frac{zp}{2}\right]$$

This factor m allows us to rewrite the equation in a clearer fashion:

$$Q_\eta + \delta Q^2 = \delta \tag{4.64}$$

where $\delta = \frac{mw}{p} = \frac{1}{m}\left[\frac{q^2 w}{p} - \frac{zp}{2}\right]$. The second equality defines m in terms of the variables already given. Equation (4.64) has solution $Q = \tanh(\delta\eta)$ meaning that $h = m\tanh(\delta\eta)$. Resubstitution of this trial solution into (4.32) fixes $m = 1$ and $\delta = \frac{w}{p} = \frac{1}{2q}$ which in turn determines c_1 and c_2 as

$$c_1 = \frac{1}{2q}\int^{x} \alpha(x')dx, \quad c_2 = \frac{\beta p}{2(1-q^2)}$$

The final form of the flow function f is

$$f(x,t) = \frac{p}{q + \tanh\left[\frac{1}{2q}\int_{x_0}^{x} \alpha(x')\,dx' + \frac{\beta p}{2(1-q^2)}t\right]} \tag{4.65}$$

The behaviour of this solution is described in Table 4.2.

(b) As has been noted, the sign of the coefficient of h^2 is important in the determination of a solution. We can obtain a sign change in this coefficient by performing a similar analysis. This time we put $h = -mQ$ which gives

$$Q_\eta - \frac{mw}{p}Q^2 = -\frac{1}{m}\left[\frac{q^2 w}{p} - \frac{zp}{2}\right]$$

when substituted into (4.63). This time we define a quantity δ such that $-\delta = -\frac{mw}{p} = \frac{1}{m}\left[\frac{q^2 w}{p} - \frac{zp}{2}\right]$; therefore (4.63) becomes

$$Q_\eta - \delta Q^2 = \delta \tag{4.66}$$

with solution $Q = \tan(\eta\delta)$. Again, resubstitution into (4.32) fixes the superfluous variables to $m = 1$ and $\delta = \frac{w}{p} = -\frac{1}{2q}$. This turn fixes the original quantities c_1, c_2 to be

$$c_1 = -\frac{1}{2q}\int^x \alpha(x')dx, \quad c_2 = \frac{\beta p}{2\left(1+q^2\right)}$$

The final flow function is

$$f(x,t) = \frac{p}{q + \tan\left[\frac{-1}{2q}\int_{x_0}^{x}\alpha\left(x'\right)dx' + \frac{\beta p}{2(1+q^2)}t\right]} \tag{4.67}$$

The parameter dependent behaviour of this solution is described in Table 4.3.

Other solutions to (4.61) certainly exist and may be easily found if we already have a solution. If $h_1(\eta)$ is a known solution to (4.61) then with $u = u(\eta)$,

$$h = h_1 + \frac{1}{u}$$

is also a solution. The function u is given by the solution to the linear ordinary differential equation

$$\frac{du}{d\eta} - \frac{w}{p} + \left[k - \frac{2qw}{p} - \frac{2w}{p}h_1\right]u = 0$$

Each of the solutions detailed above have two free parameters p and q. Equations (4.62,4.65,4.67) are exact solutions when $\beta = constant$ and are approximate where β varies with x much more slowly than α. The parameter p takes the form of an amplitude and contains directional information in its sign: for a fixed point in space and time, the sign of p controls the fluid flow relative to the magnetic field. The parameter q is a time scale and can be used to control the appearance (or otherwise) of singularities in the solution, since it appears in the denominator of each solution.

To more clearly illuminate the role of p, q, α and β we describe solution (4.62) when α, β are both constant. Consider first the case $q > 0$: the denominator of (4.62) can never be zero and the solution is governed by the exponential. The case $q < 0$ is more interesting. The denominator of (4.62) is zero when

$$x = \frac{\log(-q)}{\alpha} + \left(\frac{\beta p}{2q\alpha}\right)t \tag{4.68}$$

i.e., the position of the singularity is a function of time. It corresponds to a point in space and time where the fluid velocity is infinite. Now, $t \geq 0, x \geq 0$ for any physical set of (x,t) pairs. These conditions allow us to determine which values of p, q, α and β generate which particular behaviours. Table (4.1) displays the combination of parameter ranges which yields $x \geq 0$ in (4.68), and the times, if any, at which a singularity may be seen.

The fact that a singularity is present is consistent with the linear result (4.57) of section 4.3.4. That result predicted that given certain conditions, the effective wavelength would decrease to zero, moving energy onto smaller length scales. One would expect that the appearance of smaller length scales in the system would mean that such scales would become important to the system

α	q	βp	singularity occurs at time t [Eq.(4.62)]
>0	$q<-1$	>0	$0<t<t_m$
		<0	$t>0$
	$0>q>-1$	>0	*does not appear*
		<0	$t>t_m$
<0	$q<-1$	>0	$t>t_m$
		<0	*does not appear*
	$0>q>-1$	>0	$t>0$
		<0	$0<t<t_m$

Table 4.1: Singularity behaviour for values of p, q, α and β in (4.62)

as a whole. This seems to be borne out by the appearance of the singularity in this nonlinear solution, where the model breaks down at a single point.

Note that we define $t_m = -\frac{1}{\beta p} 2q \log(-q)$ for this solution. This quantity is important in determining the solution behaviour. The factor $1/(\beta p)$ is a characteristic time of the solution, combining both the amplitude of the solution and the time scaling in the original equation. It can be seen that when it exists, the singularity has three characteristic time dependencies. If t_m is negative, then $x \geq 0$ for all times $t \geq 0$ and we get a singularity appearing initially at $(x,t) = \left(\frac{\log(-q)}{\alpha}, 0\right)$, and thereafter moving through the fluid following equation (4.68). If t_m is positive then there are two possible behaviours. Firstly, the singularity can appear after time t_m has elapsed, i.e., the singularity first appears at $(x,t) = \left(0, \frac{-2q\log(-q)}{\beta p}\right)$ and then moves via (4.68). Alternatively, the singularity exists at times $t, 0 \leq t \leq t_m$: in this case, the singularity first appears at $(x,t) = \left(\frac{\log(-q)}{\alpha}, 0\right)$ and then moves to its final position $(x,t) = \left(0, \frac{-2q\log(-q)}{\beta p}\right)$ at time $t > t_m$ where it becomes unphysical due to the demand $x \geq 0$. Table (4.1) tells us that the sign of p is crucial: if the sign of p is changed (for any given $\alpha, \beta, q < 0$) then we move from one type of behaviour to another. Hence the direction of the fluid flow relative to the magnetic field controls when, where and if at all the singularity appears. The reason for this behaviour lies in equation (4.20); if we substitute f with $-f$ then we get

$$-\frac{\partial}{\partial t}[\nabla \times (f\mathbf{B})] = \mathbf{B} \times [\nabla(1-f^2).\nabla]\mathbf{B} \tag{4.69}$$

Equation (4.69) is a different equation from (4.20) and hence we should expect it to have a different behaviour. This shows up in tables 4.1, 4.2 and 4.3 where changing the sign of p changes the behaviour of the singularity. A sign change from $\psi \rightarrow -\psi$ does not affect equation (4.34), i.e., changing the direction of the perturbed flow relative to the background flow does not change (4.34). However, changing the background flow $f_0 \rightarrow -f_0$ does change (4.34): the coefficient of ψ_x

changes from $\beta f_0 \rightarrow -\beta f_0$. Note also that we can obtain (4.69) from (4.20) simply by reversing the direction that time runs in. If we let $t = -\tau$ then $\frac{\partial}{\partial t} = -\frac{\partial}{\partial \tau}$. Hence

$$-\frac{\partial}{\partial \tau}[\nabla \times (f(r,\theta,\tau)\mathbf{B})] = \mathbf{B} \times [\nabla(1 - f^2(r,\theta,\tau).\nabla]\mathbf{B}$$

which is simply equation (4.69) with τ instead of t. Therefore, changing the relative direction of the fluid flow compared to the magnetic field direction is equivalent to changing the direction of time.

Equivalence to (4.33) and (4.36)

We have already shown that sections 4.3.3–4.3.4 may be reduced to considering special cases of equation (4.36). By looking at these special cases we have elucidated some features of the flow function. We can show also that (4.62) contains (4.33) in the correct limit, with (4.36) governing the time and space behaviour.

Previously, we linearised a nonlinear equation (see section 4.3) and solved the resulting linear equation. We will now linearise the result of solving the same nonlinear equation and show that the two are equal. Note also that we can only consider $\beta(x) = \textit{constant}$ since we do not have analytic solutions to equation (4.32) with $\beta(x)$ nonconstant.

Consider equation (4.62) with the extra condition

$$|q| \gg \exp\left[\int_{x_0}^{x} \alpha\,(x')\,dx' - \frac{\beta p}{2q}t\right] \tag{4.70}$$

This immediately excludes us from regions in time and space where singularities can exist, since there is no way we can make the denominator approach zero. Hence

$$f(x,t) = \frac{p}{q}\left\{\frac{1}{1 + \frac{1}{q}\exp\left[\int_{x_0}^{x} \alpha\,(x')\,dx' - \frac{\beta p}{2q}t\right]}\right\} \tag{4.71}$$

which becomes on Taylor expansion to the first order only,

$$f(x,t) \approx \frac{p}{q} - \frac{p}{q^2}\exp\left[\int_{x_0}^{x} \alpha\,(x')\,dx' - \frac{\beta p}{2q}t\right] \tag{4.72}$$

We compare this to equation (4.33), i.e.,

$$f(x,t) = f_0 + \epsilon\psi(x,t)$$

and using equation (4.36) to describe $\psi(x,t)$ we have

$$f(x,t) = f_0 + \epsilon \exp\left[ct - \frac{c}{c + f_0\beta}\int_{x_0}^{x} \alpha(x')dx'\right] \tag{4.73}$$

as the full perturbed flow function.

Firstly, we must have $f_0 \equiv \frac{p}{q}$. Now, equation (4.73) is valid where $\left|\frac{f_0}{\psi}\right| \gg \epsilon$ for $\epsilon \ll 1$. Examining equation (4.72) we find equivalently that

$$\left|\frac{\frac{p}{q}}{\frac{p}{q^2}\exp\left[\int_{x_0}^{x} \alpha\,(x')\,dx' - \frac{\beta p}{2q}t\right]}\right| = \left|q\exp\left\{-\left[\int_{x_0}^{x} \alpha\,(x')\,dx' - \frac{\beta p}{2q}t\right]\right\}\right| \gg 1$$

by assumption (4.70). This is certainly bigger than the required ϵ. Therefore we also satisfy the same criterion for the validity for the analysis. Further if we compare the time dependent parts then we must also have $c \equiv \frac{-\beta p}{2q}$. Using this value for c and the expression for f_0, the coefficient of the integrand in (4.73) is

$$\frac{-c}{c + f_0\beta} = \frac{\frac{\beta p}{2q}}{\frac{-\beta p}{2q} + \beta\frac{p}{q}} = 1$$

which is precisely the factor required in (4.72). Hence the solution to the linearised equation ((4.73)) is equal to the linearisation of the nonlinear solution ((4.72)) when we are 'far away' from any singularities.

4.4 Summary and Conclusions

Fixing a form for the flow function f chooses a particular class of fields, in a similar way to chapter 3. The magnetic fields concomitant with flow functions $f = f(r,t)$, $f = f(\theta,t)$ and $f = f(t)$ are described for the full flow equation (4.1), but the case $f = f(r,\theta,t)$ has only been treated for reduced situations, e.g. the linearisation study of section 4.2 which showed that a 'first order' time dependent field aligned flow was possible for only a particular class of fields, if the flow was close to the Alfvén velocity.

Progress has been swiftest using the reduced flow equation (4.3). Here we specified a class of magnetic fields and looked in detail at the nonlinear hyperbolic partial differential equation governing the flow function. A hierarchy of analysis was developed in order to elucidate the features present in the equation. Figure 4.4 represents the relationship between all the analyses relevant to equation (4.20).

We have found that the natural scales in the system α and $\beta_0 f_0$ (space gradient and frequency respectively) control the behaviour of the wavelike solutions to a linearised version of (4.32), equation (4.35). Equation (4.35) represents the linear onset of time dependent field aligned flow from a constant background flow function.

The linearisation treatments show that a wavelike fluid motion can persist in the flow, against a constant background flow. These treatments also show that the motion appears to move wave energy around onto different length scales. For instance, in section 4.3.4 (for the case $|x - x_0| \to \infty$, $\beta_1 \to 0$) we saw that the effective wavelength of the motion tended to zero (equation (4.57)) , meaning that the motion, and hence the energy, was moving to different length scales.

This led to the consideration of analytic and exact solutions to (4.32) with the assumption of $\beta(x) = constant$. It was found that a singularity could appear in the flow function f and hence in the flow, given the correct parameter range. The fluid accelerates towards the singular point, where the model must break down. This acceleration seems to indicate that energy is being moved onto length scales which cause the breakdown of the model. Therefore we may conclude that,

α	q	βp	singularity occurs at time t [Eq.(4.67)]
> 0	$q > 0$	> 0	$t > t_m$
		< 0	*does not appear*
	$q < 0$	> 0	*does not appear*
		< 0	$t > t_m$
< 0	$q > 0$	> 0	$t < t_m$
		< 0	$t > 0$
	$q < 0$	> 0	$t > 0$
		< 0	$t < t_m$

Table 4.2: Singularity behaviour for values of p, q, α and β, defining $t_m = -\frac{2(1+q^2)}{\beta p}\tan^{-1}(-q)$ in (4.67)

given the correct parameter range, a time dependent field aligned flow will generate a singularity in the fluid flow. We have found, via wave steepening, a possible mechanism for the generation of localised fluid acceleration.

$$f_{xt} + \alpha(x) f_t + \beta(x) f f_x = 0$$

Linear Anaylses	**Nonlinear Analysis**
See sections 4.3.1-4.3.4 for more detail	See section 4.3.5 for more detail
$f = f_0 + \epsilon\psi(x,t)$	Put $\beta(x) = constant$
Put $f_0 = constant$	$\downarrow$
$\downarrow$	$\downarrow$
$\psi_{xt} + \alpha(x)\psi_t + \beta(x) f_0 \psi_x = 0$	$f_{xt} + \alpha(x) f_t + \beta f f_x = 0$
$\downarrow$	$\downarrow$
1. α, β fully varying: eq.(4.36)	Analytic Solutions
	see equations (4.62)
2. α, β linearly dependent on x:	(4.67), (4.65)
see eq.(4.53): special case of (4.36)	$\downarrow$ *can be reduced to* $\downarrow$
$\downarrow$ *can be reduced to* $\downarrow$	$\downarrow$
3. α, β both constant: eq.(4.38)	Eq.(4.72): equivalent to (4.33)
and (4.41)	and (4.36) with $\beta(x) = constant$
Results	**Results**
Wave steepening and	*Possible appearance*
exponential amplitude variation	*of singularity in flow*
Asymptotic change of	*- breakdown of model*
wavelength	

Figure 4.2: Relationship between analyses in section 4.3

α	q	βp	singularity occurs at time t [Eq.(4.65)]
> 0	$0 < q < 1$	> 0	*does not appear*
		< 0	$t > t_m$
	$0 > q > -1$	> 0	$t > t_m$
		< 0	*does not appear*
< 0	$q < -1$	> 0	$t > 0$
		< 0	$t < t_m$
	$0 > q > -1$	> 0	$t < t_m$
		< 0	$t > 0$

Table 4.3: Singularity behaviour for values of a, b, α and β, defining $t_m = -\frac{2(1-q^2)}{\beta p} \tanh^{-1}(-q)$. in (4.65)

Chapter 5

An Application of Genetic Algorithms to Differential Equations

There once was a brainy baboon,

Who always breathed down a bassoon,

For he said, 'It appears

That in billions of years

I shall certainly hit on a tune.'

Sir Arthur Stanley Eddington, New Pathways in Science, 1935

We now consider the calculation of numerical solutions to two second order differential equation's that arose from a problem in field aligned flow in chapter 3. Genetic algorithms are introduced by analogy with the Darwinian theory of evolution coupled with the DNA representation of genetic information. Detailed consideration is given to a variation on an existing method of applying genetic algorithm's to ordinary differential equations [38]. Genetic answers are compared to both analytical results and existing numerical methods.

5.1 Finite Difference Solutions

In chapter 3 we introduced two methods of solving the equations describing a set of magnetic field topologies that support a flow of $f = constant$, namely equation 3.21

$$\nabla^2 A = \lambda(r, \theta) A$$

and equation 3.22,

$$\frac{\partial A}{\partial \theta}\frac{\partial \lambda}{\partial r} - \frac{\partial A}{\partial r}\frac{\partial \lambda}{\partial \theta} = 0$$

where A is the vector potential of the field $\mathbf{B} = \nabla \times (A\hat{\mathbf{z}})$.

We may solve (3.22) by setting $\lambda = g(A)$ leaving

$$\nabla^2 A = Ag(A) \tag{5.1}$$

For any choice of f other than $f = c_1 + \frac{c_2}{A}$, $c_1, c_2 = constants$ we are left with solving the nonlinear Poisson equation (5.1) with the relevant boundary equations (which are derived from the physical situation we wish to emulate). As analytic solutions to nonlinear Poisson equations are few and far between we must use numerical and computational techniques to solve (5.1). A number of library routines are available to solve the Poisson equation numerically by using a finite difference replacement. These routines set up matrix equations where the unknowns are the values of A at each grid point. An initial guess at the answer can be supplied by the user to give the routine starting values to work with: NAG routine D03EBF offers this facility. Matrix methods and manipulations are then used to solve the (commonly) linear equations. This approach is described in more detail in section 6.1.1.

Alternately, with $A = h(\lambda)$ we are left to solve an ordinary differential equation, (3.25)

$$q_1(\lambda)h'' + q_2(\lambda)h' - h\lambda = 0$$

where the dash $'$ refers to differentiation with respect to λ, and $q_1(\lambda) = \frac{1}{r^2}\left(\frac{\partial \lambda}{\partial \theta}\right)^2 + \left(\frac{\partial \lambda}{\partial r}\right)^2$, and $q_2(\lambda) = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial \lambda}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 \lambda}{\partial \theta^2}$. Given a suitable λ such that q_1, q_2 are functions of λ we still have to solve a second order ordinary differential equation. Iterative numerical routines exist to solve such equations. And again, some of these routines can be given an 'initial guess' (for example, NAG D02RAF) to start the routine off.

One possible way through which we can improve the performance of these routines is to generate a better initial guess to the final answer. If we have a 'good enough' initial guess then the solver (whichever one we wish to use) can finish the task. Finding a good guess is not easy or obvious in all cases (indeed, for NAG routine D03EBF the recommended action is to simply set the initial guess to be everywhere zero!). It is at this point we can introduce genetic algorithms as a possible way of generating a 'good enough initial guess'.

5.2 The Genetic Algorithm Concept.

A genetic algorithm (GA) is a computational technique that solves search/ minimisation problems by applying the ideas of Darwinian evolution. By the above definition then, most genetic algorithms must follow the same outline [39] which is summarised in figure (5.1). Each of the statements in

Procedure Genetic Algorithm

begin

initialise population $P(0)$

evaluate $P(0)$

$t = 1$

repeat

select best *npar* parents from population $P(t-1)$

recombine parents to create *nchild* children and new population $P(t)$

mutate entire population $P(t)$

evaluate entire population $P(t)$

$t = t + 1$

until (termination condition satisfied)

final population of fit individuals $P(finish)$

end

Figure 5.1: Scheme for general genetic algorithm

figure (5.1) has a close analogy in the theory of evolution, and it is profitable to consider this first.

In the theory, species evolve by a combination of mechanisms to produce individuals better adapted to their environment. This ultimately means that we must change the DNA coding, or *genotype* of the individual in order to change its physical characteristics, or *phenotype*. This leads to the first requirement for a genetic algorithm that is, a suitable encoded representation of the problem we wish to solve. In a genetic algorithm each candidate is represented by a string of symbols, in analogy with the AGTC alphabet that encodes genes in a DNA molecule [40] . (Each of the letters stands for a chemical structure basic to all DNA molecules: adenine, guanine, thymine and cytosine.) The representation used in a genetic algorithm allows us to evolve new phenotypes by appropriate imitation of the mutation and recombination mechanisms available in nature. We may change the distribution of genes in a population and their contents by

1. Reproduction. This introduces variety into the population by recombining genetic material from the previous generation in new orders, creating new genotypes and hence new phenotypes.

2. Mutation. The content of the genetic code may be changed by a number of mechanisms; for example, copying errors during DNA manipulation can cause the deletion, reordering or multiplication of genes. Ionising radiation can also damage and hence change the DNA content. The classic example of mutation conferring a phenotypic advantage is sickle cell

anaemia, which is prevalent in many human populations in Africa. Although the affected individuals are rendered anaemic, they are also much less susceptible to malaria, which is found to be endemic in those areas where sickle cell anaemia is common. The mutation is due to an error in only one nucleotide in the DNA which causes a subsequent structural change in the protein that it encodes [41].

3. 'Survival of the fittest'[1]. The environment that an individual finds itself in determines whether its genes will survive to the next generation. If it well adapted to the environment then it is more likely to survive to a breeding age than one that is less well adapted. This is summarised in the phrase 'survival of the fittest'.

By these mechanisms, populations may evolve into forms that are better suited to their environment, and may implement analogous operations in a genetic algorithm.

At some initial time we have a population of candidates $P(0)$, encoding the relevant features of the problem in an appropriate representation. The 'quality' of each candidate solution, i.e., how well it solves the problem may be evaluated by the use of a suitable 'fitness' or 'weighting' function. This is the analogue of the environment in nature: this function determines the viability of individuals. Using this weighting operator, we can assign a weight to each candidate and then rank them accordingly. Usually we wish to generate individuals with as low a weighting as possible. Having ranked the population from best to worst (lowest weights being more 'fit' candidates), we can now select the top *npar* individuals of the ranked population as the breeding parents for the next generation. This may be likened to survival of the fittest in an environment causing the best genes to propagate into the next generation.

By allowing the best individuals to breed and by the introduction of new genes via mutation we create new daughter candidate solutions. Also, by keeping the breeding parents we ensure that the new population is no worse than the previous one. The daughters are now in direct competition with their parents. If we now re-rank the entire population we can repeat the process, breeding from the best and generating new daughters until some specified condition is fulfilled, usually the completion of a fixed maximum number of iterations, or the achievement of a particular weight. On exit, the population $P(finish)$ is ranked from the 'best' solution to the worst.

5.2.1 An example application of Genetic Algorithms

One well known example that may be treated by a genetic algorithm is the travelling salesman problem [39]. The problem is: given a set of N cities to visit, find the shortest route that visits each city exactly once. Although simply stated, the travelling salesman problem is an example of an NP-hard problem, i.e., the solution time increases exponentially with N, the number of elements

[1] Due to Herbert Spencer (1820- 1903), *Principles of Biology* III

in the problem. Such problems probably (the question is open at the time of writing) do not have any polynomial time - solution time varying as N^p, $p > 0$ - algorithmic solution.

A natural representation for this problem is to number each of the cities and form a string of numbers which describe the order in which each city is to be visited. The weighting function which may be used to rank the population is the total route distance each string represents. Hence by a suitable implementation of recombination and mutation operators, one may create a genetic algorithm to solve the problem, or at least generate close to optimal routes.

A practical application of this particular problem is circuit board building by robot. Instead of cities, we have circuit component slots and soldering points to visit, To complete as many circuits as possible, the robot must take as little time as possible on each and hence must move as short as distance as possible over each board. This is merely a restatement of the travelling statement problem and has been successfully solved by genetic algorithms.

5.3 A Genetic Boundary Value Ordinary Differential Equation solver

To show that it is at least worthwhile to apply genetic algorithms to differential equations let us first consider a basic ordinary differential equation

$$C_0(x)\,y'' + C_1(x)\,y' + C_2(x)\,y + C_3(x)\,y^2 + C_4(x) = 0 \tag{5.2}$$

where $' \equiv \frac{d}{dx}$. The equation lies over the range $x \in (x_0, x_{end})$ with boundary conditions

$$y(x_0) = y_0, \quad y(x_{end}) = y_{end} \tag{5.3}$$

This equation has a number of features that make it a good test bed for learning about the problems associated with a genetic algorithm type solution to differential equations. Firstly, there are many analytic solutions available, solving both linear and nonlinear versions of (5.2) with which we can check the algorithms' final answer. Secondly, being a one dimensional problem, the 'size' of the genetic code will be small enough to allow a large number of iterations in a reasonable time. This was important as we wanted to optimise convergence strategies in as short a development time as possible before moving onto larger scale Poisson equation problems. Many of the problems associated with the application of genetic algorithms to differential equations have been solved already by Diver [38] in the algorithm GENODE. However there are a number of improvements possible that speed up program performance and convergence times. Therefore, in describing the new algorithm, reference shall be made to GENODE and to the changes that have been made in an attempt to improve algorithm performance

To create a genetic algorithm to solve ordinary differential equations we need 3 basic components: a representation, a weighting function and a complement of operations that can change

the genetic content of the candidates. At each iteration level there are *npar* parents that will generate *nchild* children by breeding, giving a total population at any one time of *npar* + *nchild*. The program itself is called ODE and the following subsections discuss some of the features of its design.

5.3.1 Representation

We represent the genetic algorithm solution to (5.2) as a set of *npoint* points, $y_i^{algorithm}$ at discrete points in the range $x_i \in (x_0, x_{end})$, $1 \leq i \leq npoint$ spaced at intervals $h = \frac{x_{end}-x_0}{npoint-1}$. This choice is intimately tied up with the weighting function. Diver chooses to describe each $y_i^{algorithm}$ as integer multiples of two real numbers μ and ν with $\mu > \nu$. GENODE then finds integers $m, n \in (-50, 50)$ such that

$$y^k = m_i^k \mu + n_i^k \nu \tag{5.4}$$

The numbers μ, ν are chosen in such a way as to span a reasonable range of possible y values, and to allow some fine tuning to be done using the ν graining. The program then creates a genetic code for the k'th candidate in the population by forming a string from the integers m_i^k and n_i^k. The k'th candidate now resembles

$$y^k \equiv (m_1^k n_1^k) \cdots (m_i^k n_i^k) \cdots (m_{npoint}^k n_{npoint}^k)$$

where $1 \leq i \leq npoint$, $1 \leq k \leq npar + nchild$. The motivation behind this representation can be understood if one considers the way in which the ordinary differential equation is represented computationally. The ordinary differential equation is rendered via a centred finite difference representation of the differentials, i.e.,

$$\begin{aligned} y'(x_i) &= \frac{y_{i+1} - y_{i-1}}{2h} + O(h^2) \\ y''(x_i) &= \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + O(h^2) \end{aligned} \tag{5.5}$$

where $h = \frac{x_{end}-x_0}{npoint-1}$. The differentials are represented up to order $O(h^2)$, and any numerical solution cannot do better than this. Hence it makes sense not to demand a greater accuracy from a genetic algorithm solution. Since we cannot do better than $O(h^2)$ the choice of ν should reflect this and therefore ν should be larger than $O(h^2)$.

Although this adequately describes the candidate y^k there are two major problems with this approach. Firstly, there are large time overheads involved in translating from the genetic code for a point y_i^k to a floating point number and vice versa. (This translation must be performed because the weighting function uses floating point numbers). For one iteration the routine as it stands must perform $(npar + nchild) \times npoint$ of these translations: this is computer time that is not usefully spent. Secondly, there is an upper limit to the best possible error at each point, namely

$$\left| y_i^{\text{true value}} - y_i^{\text{k'th algorithm candidate}} \right| \leq \max\left(\frac{\nu}{2}, O(h^2)\right) \tag{5.6}$$

where the true value is either from an analytic answer or from another numerical routine having at least the same $O(h^2)$ accuracy. If $\frac{\nu}{2}$ is smaller than $O(h^2)$ then the maximum error is dominated by the discretisation procedure. This can result in GENODE (and any other differential equation solving genetic algorithm) trying to minimise the error where no useful benefits can be had, since the discretisation is only $O(h^2)$ accurate. To span the range of gene values, one can imagine a situation where a poor choice of μ means that ν is much larger than $O(h^2)$. If $O(h^2)$ is smaller than $\frac{\nu}{2}$ then GENODE cannot reach the limit of the discretisation error, which is what we want to aim for. This is directly a result of the representation chosen in GENODE. It means that GENODE can waste time trying to improve the fitness of a point where no improvement is possible because of the constraints of the ν graining: the error for this point has reached a minimum. It also means that there must be (unless we are fortunate enough to choose a problem in which all the points on the curve y can be exactly represented by (5.4)) a nonzero minimum to the fitness value which cannot be reduced. Therefore it is not clear if the routine has reached the optimum answer or if we are in a genetic cul-de-sac from which the algorithm cannot remove itself. The problem of convergence to sub-optimised candidates requires special treatment, and is detailed below in section 5.3.4.

Clearly then it would be profitable to consider alternative representations. In their 1993 review paper, Beasley, Bull and Martin [42] report on a paper by Janikow and Michalewicz [43] in which a comparison is made between binary and floating point representations. Binary numbers have for a long time been seen as the only reasonable and indeed understandable problem representation in the genetic algorithm community. The binary expression of integers form the basis of these genetic codes, the individual bits being manipulated by the processes described above. This is in effect analogous to the GENODE treatment as again we are necessarily working with a reduced alphabet of m's and n's. It was found that the floating point version gave faster, more consistent and more accurate results. Although this result tends to run counter to accepted wisdom in genetic algorithm circles, it was decided to move over to a floating point representation

There are considerable advantages to be had from using floating point numbers to describe the points $y_i^{k,algorithm}$. The members of the population may be easily stored as a list of floating point numbers in an array, i.e.,

$$y^k \equiv y_1^k \cdots y_i^k \cdots y_{npoint}^k$$

In biological terms, the phenotype and genotype have now become identical. The time penalties incurred by any translation process from genotype to phenotype are now simply not present. A floating point representation also addresses one half of the accuracy problem (5.6) by effectively setting ν, the smallest incremental change to be machine precision. Crucially, the number and type of meaningful mutation operators available to us is increased, as floating point numbers may be manipulated in a variety of ways to yield another floating point number (see the section below).

5.3.2 Weighting Operator

In every generation, natural selection demands that only the 'best' individuals be selected as the breeding parents for the next generation. In a genetic algorithm this process is carried out using a weighting operator that assigns a weight to every individual. In an ordinary differential equation solver, we must assess how well each candidate solves the differential equation along with the boundary conditions. Constant value boundary conditions are handled easily, as at each iteration all individuals in the population simply have the boundary conditions written directly into their genetic representation.

The ordinary differential equation is represented numerically by substituting the differentials using central finite difference replacements (5.5). The differential equation (5.2) is implemented as

$$\left[\frac{C_0(x_i)}{h^2}+\frac{C_1(x_i)}{2h}\right]y_{i+1}+\left[C_2(x_i)-2\frac{C_0(x_i)}{h^2}\right]y_i+ \left[\frac{C_0(x_i)}{h^2}-\frac{C_1(x_i)}{2h}\right]y_{i-1}+C_3(x_i)\,y_i^2+C_4(x_i) \tag{5.7}$$

One must take special care at the endpoints as clearly at the left and right hand sides there are no points corresponding to y_{i-1} and y_{i+1} respectively. GENODE solves this problem by using quadratic extrapolants to generate estimates for y_1' and y_1''.

$$\begin{aligned} y'(x_1) &= \frac{-3y_1+4y_2-y_3}{2h} \\ y''(x_1) &= \frac{y_1-2y_2+y_3}{h^2} \end{aligned} \tag{5.8}$$

Similar expressions are calculated for y'_{npoint} and y''_{npoint}. Using (5.8) and (5.7) equation (5.2) may be implemented numerically in a discretised form. If we use the $y_i^{algorithm}$ values for the k'th candidate in the population then we can calculate an error R_i^k for every point in every candidate solution. This error tells us how well that particular point fits the discretised form of the ordinary differential equation. Hence,

$$R_i^{k,algorithm}=\left[\frac{C_0(x_i)}{h^2}+\frac{C_1(x_i)}{2h}\right]y_{i+1}^k+\left[C_2(x_i)-2\frac{C_0(x_i)}{h^2}\right]y_i^k+ \left[\frac{C_0(x_i)}{h^2}-\frac{C_1(x_i)}{2h}\right]y_{i-1}^k+C_3(x_i)\left(y_i^k\right)^2+C_4(x_i) \tag{5.9}$$

We can now calculate some measures of the fit, ϵ_j^k, $1 \le j \le 4$ of each individual, based on the above R_i^k values, and use these to calculate an overall weighting. Again, following Diver we use fitting measures

1.

$$\epsilon_1^k=\sum_{i=1}^{npoint} \mid R_i^k \mid \tag{5.10}$$

ϵ_1^k is a measure of the overall 'distance' the candidate is away from the actual solution.

2.

$$\epsilon_2^k = \prod_{i=1}^{npoint} \mid R_i^k \mid \tag{5.11}$$

A candidate that has a number of points with $R_i^k \approx 0$ will have a low value of ϵ_2^k. Therefore candidate with low ϵ_2^k contain at least some points that fit the discretised operator (5.7) very well.

3.

$$\epsilon_3^k = \exp\left[f_1 \times \left|(y_1')^k - gvr\right| + f_1 \times \left|(y_{npoint}')^k - gvl\right|\right] - 1 \tag{5.12}$$

Gradient information is supplied for each of the endpoints, gvr and gvl. This is a well known and accepted method in the numerical solution of differential equations: one overspecifies the problem by supplying both gradient and boundary value information at both ends of the range. However, only two of these pieces of information are exact and both cannot hold at the same point. The other two pieces are then used as an estimate in such numerical routines. The currently described genetic algorithm fixes the boundary information but generates penalties if the candidate gradients at the endpoints differ from the user supplied values. The factor $f_1 > 0$ allows the strength of this error to be controlled; a higher f_1 penalises poor gradients more. Thus ϵ_3^k is a measure of how well candidate k satisfies the gradients demanded by the user.

4.

$$\epsilon_4^k = \max_{i=1,npoint} \mid R_i^k \mid \tag{5.13}$$

This is a crude measure of the continuity of the k'th candidate. Candidates with a large ϵ_4^k have at least one point that does not fit the operator well. These candidates can be selected against as it would be relatively harder to relax to a lower weighted individual from one that has poor continuity.

The above measures are combined to form a final overall weight for the k'th candidate in the formula

$$W^k = p_1\epsilon_1^k + p_2\epsilon_2^k + p_3\epsilon_3^k + p_4\epsilon_4^k \tag{5.14}$$

The factors p_1, p_2, p_3 and p_4 allow the user of the algorithm to choose which features are to be selectively penalised over others. The weighting factors chosen can play a crucial role in determining good convergence times (see section 5.3.5). By assigning a weight W^k to all the candidates, one may now rank the entire population from best (lowest W^k) to worst (highest W^k). This allows one to choose the best individuals to form the breeding stock for the next generation.

5.3.3 Breeding, Mutation and Combination Operators

By choosing a floating point representation we increase the number of operations we can perform on the genes. Any operation we perform on the gene, or pair of genes in the case of the combination operators, must also produce a valid gene. This would exclude most of these operators from an integer treatment as an integer would not be returned. One could always perform these operators and choose the nearest integer value but this takes up computer time needlessly and for a large number of operations, produce no change in the genetic content. This is because the changes would be too small for the integer representation to notice even if the floating point number generated conferred some advantage to the individual; the information is lost in the coarseness of the implementation. Some possible functionality is listed below [42].

1. Combination Operators
   (a) *Average* - take the (weighted) average of a pair of genes
   (b) *Geometric Mean* - take the square root of the product of a pair of genes
   (c) *Extension* - take the absolute difference between a pair of a pair of genes and subtract it from the lower or add it to the higher
2. Mutation Operators
   (a) *Random replacement* - replace the gene with a random value
   (b) *Creep* - add or subtract a small (smaller than the gene value itself) randomly generated number
   (c) *Geometric Creep* - multiply the gene by a random number close to 1

GENODE already implements operations 2(a) and 2(b) but a floating point representation permits a wider choice of possible functionality. There are many alternative ways of implementing these: for instance, one can choose the type of random distribution, say, random or Gaussian, used in the mutation operators. These operators are really only feasible in a floating point representation: it is not hard to see that these new operators would be very unlikely to work in an integer representation. In the code the six operators were implemented under full user control.

As described above, we now have a genetic code representing each member of the population. In order to redistribute the existing parental genetic code in new ways amongst the children one must implement a form of reproduction between individuals i.e., distinct candidates produce children with genetic codes made up from their parents. There are very many schemes available to do this, coming under the generic title of *crossover schemes*. This refers to the swapping of the genetic material. In a crossover scheme each candidate is broken into two or more subsections, and these subsections are then recombined to create new children. In a *one-point crossover* [14] scheme,

One point crossover

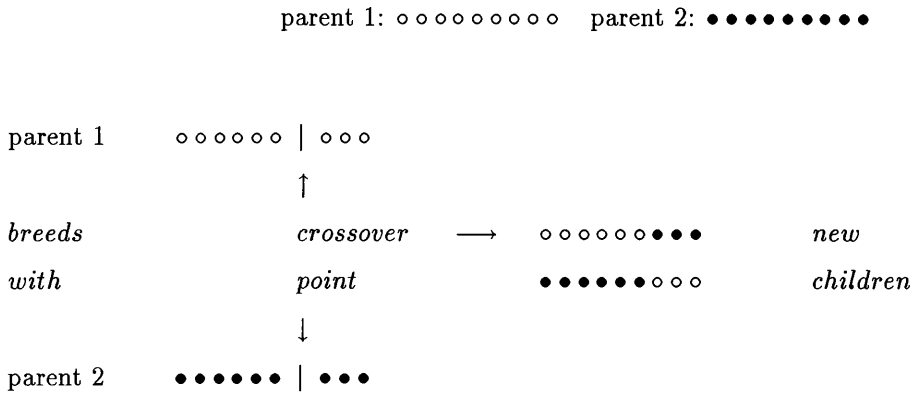

Figure 5.2: One point crossover breeding scheme

the one implemented in the program each string is split at the same point and the fragments are swapped. This is most easily seen pictorially (see figure 5.2). There is some justification for choosing a one point scheme as opposed to the random scheme implemented in GENODE. GENODE creates new individuals by taking a randomly sized piece of genetic material from a random position in the first parent and adding it to a similarly chosen piece of genetic material from the second parent. The routine then makes sure the child is of the correct size, which goes on to form part of the population. The motivation for using this random scheme is that it allows the children to have a more mixed up genetic structure when compared to their parents, hence promoting the distribution of genes across the genotype. However, if we consider the problem that we are trying to solve, we would hope that after a few iterations all the breeding population would at least show the approximate shape of the final solution. From this point onwards the routine should now relax the candidates down to the true solution.

Indeed, this fact is displayed in [38], where after a few iterations, the best candidate of each generation has a very good approximate shape. Thus a random scheme that swaps the *ordering* in the string, while maintaining a good distribution of genetic material, will be inefficient in generating new combinations that are better than the parents as inappropriate curve segments will be swapped. This in turn means that a large number of children will be very unfit as they may be highly discontinuous, inhibiting the overall improvement of the population.. A random scheme will be very good in earlier iterations to quickly get to a reasonable shape, but a one point scheme will permit efficient evaluation of the appropriate curve segments in their correct positions.

5.3.4 Convergence strategies

It frequently occurs in genetic algorithms that a sub-optimal candidate comes to dominate the population. Such an individual need not be even a remotely accurate solution. It survives because the genotypic variation strategies of section 5.3.3 create an insufficient spectrum of gene values to allow the weighting function to move out of its local minimum. Therefore the routine is stuck with a poor best candidate.

ODE employs a number of strategies to avoid premature convergence of the population onto a poor candidate, aside from the essential components of selection, breeding and mutation. By themselves, they are not enough to ensure good convergence, and so something else must be done. (These routines are also implemented in POISGEN, which is described in chapter 6.)

1. Sometimes the entire breeding population can consist of very similar parents. This is characterised by each member having identical or near identical weights, which we assume implies that the parents are very similar. The *genetic diversity* of the breeding population is measured by

$$mtot = \frac{1}{npar} \sum_{n=1}^{npar} \left[1 - \frac{weight(1)}{weight(n)} \right]$$

   If *mtot* is close to zero then the weights assigned to the breeding population are all very similar and the parents are assumed to be genetically similar. This is an assumption, as it is conceivable that genetically dissimilar candidates have similar weights. However, in practice this situation is not seen to arise. If *mtot* is close to 1 then candidates $2 \rightarrow npar$ are assumed to be genetically dissimilar compared to the best in the population (see appendices A, B). Ideally *mtot* should lie somewhere between the two extremes, indicating a healthy spread of potentially useful genes.

   It is difficult to obtain a genetically diverse population from genetically similar parents; hence the evolution towards better individuals will normally proceed very slowly, if at all (excepting the freak appearance of a dramatically better candidate). The routine has converged on a sub-optimised answer. When this happens ODE throws away the entire ranked population except the very best individual by overwriting those candidates ranked 2 to $npar + nchild - 1$ with completely random genotypes. The population is then re-ranked and bred as normal. This new genetic material may contain genes that are beneficial to evolutionary progress, allowing the best candidate to evolve in a manner previously unavailable to it since the required genetic material simply was not present. (This routine is called **tweak** in POISGEN.)

2. Frequently one point in the best candidate has a far worse fit than any other. ODE takes remedial action by making two copies of the best individual and mutating the worst point very slightly, every few generations. A geometric creep is performed on the point, one candidate having new gene value $gene \times (1 - small)$ at the worst point, the other being $gene \times (1 +$

small) at the worst point. These individuals are then placed in the last two positions in the population to ensure that minimum damage is done to the best genes. The population is then re-ranked and the program continues. (This strategy is called via the profiling subroutine in POISGEN.)

3. If the routine is very successful early on, ODE reaches a low best weight relatively quickly. In this case, large scale changes to the breeding population are unlikely to create dramatically better individuals, as they are already highly evolved. Since they have low weights, they are unlikely to have a single gene much worse than any other: it is more likely that each gene has a similar fitting error as determined by equation (5.9). In order to promote further evolution, two copies of the best individual are made but this time every gene value (excepting the boundaries) are changed. One copy consists of genes $gene \times (1 - small)$ (where *gene* is the gene value of the best individual), the other $gene \times (1 + small)$. The value of *small* should be very small compared to 1 as we do not want to perturb the gene values too much. This routine is known as **wobble** in both ODE and POISGEN.

4. We want the routine to converge as quickly as possible to an optimised answer. This implies that the best weight should decrease by a 'reasonable' amount every few iterations. What constitutes 'reasonable' is left to the user to decide by changing the relevant program variables. If the evolutionary history is flat - by this we mean that the best weight has not changed much over a certain number of iterations - then it makes sense to try to influence progress by changing the breeding stock slightly. The evolutionary history can be flat in a breeding population that is genetically diverse. Therefore, convergence strategy 2 will not be activated. Hence ODE has another strategy to combat this situation. When the evolutionary history is flat, the individuals ranked $2 \rightarrow npar$ i.e., the top $(npar - 1)$ individuals excluding the very best, have a genotype-wide geometric creep performed on them. The value of *small* in $gene \times (1 \pm small)$ is larger than in strategy 3 in order to introduce some new genes that are not very distant or too similar to the current genes in the breeding population. This is different from strategy 3 where we do not want to change the gene values very much, since the breeding population is already highly evolved.

   This particular strategy (known as **jiggle** in both ODE and POISGEN). is also activated after a user determined number of iterations in order to inject some diversity into the population at regular intervals.

5. It is possible to run the genetic algorithm using a breeding population composed entirely of randomly selected individuals. If this is done, then it is likely that most of the population will be assigned very high weights since it is unlikely that such individuals will present a good solution to the ordinary differential equation - one random candidate will look much

like another. To improve convergence, we seed the routine with a specially defined candidate (see routine **special** in appendix A). Given boundary conditions $y = y_0$ at $x = x_0$ and $y = y_{end}$ at $x = x_{end}$ we define

$$y_i^{special} = \frac{x_i - x_0}{x_{end} - x_0} + R \times \left[\frac{highbdry - lowbdry}{frac}\right]$$

to be the seeding individual. R is a random number between -1 and 1. *highbdry* and *lowbdry* are the maximum and minimum permitted gene values in the routine. The variable $frac > 1$ allows the second 'noise' term to be scaled against the range of permitted gene values. This is added for two reasons: to prevent a possibly 'spurious' solution from taking over the routine immediately and to allow some guided variability to these special gene values.

It is not clear that in all cases this individual will be an improvement over a completely randomly defined one. However, compare the results of a crossover between two random individuals and a random candidate with the special. Two 'randoms' mating are likely to produce random-looking children. Hence the chances for improvement are poor, as they are likely to be similar to their parents. However, the two children of a special-random crossover are very different from either parent. Therefore, we have increased the genetic diversity by creating very different children. This leads to a much more extensive search of the solution space, which improves convergence.

5.3.5 Experimental Results and Algorithm Behaviour

It is naïve to expect that such a general technique as genetic algorithms will yield a perfect answer every time. By its very nature, we cannot expect exactly repeatable answers. 'Reasonably' accurate answers (at least exhibiting a similar shape to the true solution) in a 'reasonable' time would constitute satisfactory behaviour i.e., a 'low' weight after a fixed number of iterations.

However, if we can generate answers accurate enough to allow more traditional solvers to continue, then this too would constitute a successful application of genetic algorithms.

The quality of the final answer is assessed by both the arithmetic and geometric means of the differences between the true and genetic answer.

$$E_1 = \frac{1}{npoint - 2} \sum_{i=2}^{npoint-1} \mid y_i^{true} - y_i^{algorithm} \mid \tag{5.15}$$

$$E_2 = \sqrt[n]{\prod_{i=2}^{npoint-1} \mid y_i^{true} - y_i^{algorithm} \mid} \tag{5.16}$$

where $n = npoint - 2$. The values y_i^{true} come from either the analytic solution, tables, or NAG routine D02HAF. For all the results in this section, $npar = 20$ and $nchild = 900$. This fairly arbitrary choice reflects the experience gained while using the program: these values were chosen because they gave good convergence in most cases.

The first four sets of results describe ODE's attempts to solve

$$y'' + y = 0 \tag{5.17}$$

with boundary conditions,

$$y(0) = 0, \quad y(2\pi) = 0 \tag{5.18}$$

The values of the more influential variables for each experiment are listed in Tables (5.3.5)→(5.6).

1. Initial attempt: Figures (5.3),(5.4), Table (5.3.5)

   The first thing to notice about Figure (5.3) is that the answer is genetic answer is very poor: although it does have the general shape of the true answer, it does not have the magnitude. This is seen time and time again in the experimental results of both ODE and POISGEN later; shapes seem to be easy to come by, but the final magnitude is very much more difficult. This loss of size is caused by two effects: firstly, the manner in which the gradient fitting is implemented, and secondly, the slow percolation of boundary information into the genotype. Equation (5.17) is linear with general solution $y = A\sin(x) + B\cos(x)$. The boundary values given should choose only the *sin* solution, leaving only the amplitude A to be determined. The constant A is determined by the gradient at one of the endpoints. (The gradients given ensure that A can be defined consistently.) However, the gradients are not fixed in the routine: the values gvl, gvr are merely good suggestions given to the routine in order for it to have something to work on. The measure ϵ_3 assigns a weight to how well the gradients fit at the endpoints, but in a manner that allows some leeway since gvl, gvr are meant to be guesses.

   This means that if a gradient other than the suggested one happens to confer a greater chance of survival to an individual, then that individual will have a greater chance of surviving as a breeding parent for the next generation, despite the fact that its genotype does not solve the exact problem well.

   This problem can be handled by increasing the weighting assigned to candidates that fit the required gradients poorly. This may done by changing the values of both p_3 and f_1. Table (5.2) shows the new values of p_3, f_1, and figures (5.5), (5.6), the corresponding genetic answer and evolutionary history respectively.

   Figure (5.4) displays the evolutionary history of the solution, and is typical of many runs using ODE and POISGEN. The best candidate is relatively poor, but quickly evolves into a fitter organism. Unfortunately, the evolution seems to be largely complete by iteration 50, the remaining iterations only changing the best weight very slightly. Given the eventual answer, we seem to have reached a genetic cul-de-sac.

2. Increased boundary weighting: Figures (5.5), (5.6), Table (5.2)

   As can be seen from figure (5.6), the best weight starts off at a much higher value than in figure (5.4). This is because we have changed the weighting function; we are in some sense exaggerating the difference between individuals by penalising the gradient fitting at the boundaries. Figure (5.6) also demonstrates another evolutionary feature. Between iterations 10 and 20 the best weight is approximately constant. After the twentieth iteration, the best weight drops dramatically by over two orders of magnitude. This indicates that a very much more fit genotype has been created which quickly dominates the population. Its genes are propagated through the population by breeding, improving the overall quality of the stock. This effect has been seen in many natural populations: for instance, the widespread use of DDT in areas where malaria is endemic has led to the emergence of mosquitoes that have developed resistance to many pesticides. This is an example of selection pressure choosing individuals better adapted to their environment.

3. Point doubling: Figures (5.7), (5.8), Table (5.3)

   As has been said, the naïve approach yielding results similar to (5.17) does not yield satisfactory answers. Although one can increase the weighting associated with the boundaries this is unlikely to be satisfactory when we are using a very long genotype. This is because the boundary information will take a long time to influence the centre of the genotype.

   This leads to an important feature in genetic algorithm application. At large values of *npoint*, convergence to low weights is prohibitively poor. A far more efficient method is to start at a low value of *npoint*, run the genetic algorithm to find an approximate answer, and then double the length of the code by interpolating new points in between the genetically determined ones. By this process, we arrive at a longer genotype, describing a larger number of points more accurately than one could reasonably expect from the same number of iterations at the longer code length.

   By starting at a lower value of *npoint* we gain greatly in the time taken per iteration speed, but lose out on accuracy. Note also that the centre portion of the genotype is now very close to the boundaries and hence influences the centre of the range much more strongly than a genotype of twice its length. For a given genotype of length *npoint*, the doubled by interpolation genotype has length $2 \times (npoint) - 1$. Figure (5.7) shows the result of this point doubling tactic. The final answer is much improved on that shown in Figure (5.3). The effect of this point doubling is very marked in the evolutionary history, Figure (5.4). The sudden spike in weight is generated by the increased number of points causing (naturally enough) an increase in the error. This is because these new linearly interpolated points are very unlikely to lie on the true curve, and hence create a fitting error that show up in the final overall

weight.

4. Comparison of answers for long genotype: Figures (5.9), (5.10), Table (5.4)

   Figure (5.9) compares the two genetic solutions to (5.17). Solution (a) is very poor when compared to the true answer; it has such a low weight because most of the points are zero, fooling the routine into thinking that the candidate is solving the problem well. Solution (b) is much more realistic: once again, the shape is very good and it is quite close to the true solution. This demonstrates the power of combining the point doubling and boundary weighting strategies to get reasonable answers for long genotypes. From here on, the solutions were generated using these approaches.

5. ODE solution to

$$y'' + xy' + y = 0 \tag{5.19}$$

   with boundary conditions,

$$y(0) = 1, \quad y(5) = 0 \tag{5.20}$$

   (See Figures (5.11),(5.12) and Table (5.5))

   The program is set up to calculate the subdominant solution $y = \exp\left(-\frac{x^2}{2}\right)$ to (5.19), a stiff ordinary differential equation. Again, the shape is good and is close to the true solution. The NAG routine outperforms ODE, but we are close enough to the true solution to demonstrate that the essentially random, or *blind* genetic algorithm is a reasonable approximation to the geometric methods of traditional numerical solvers.

6. ODE solution to

$$y'' + xy' + x^2y + 2y^2 = 0 \tag{5.21}$$

   with boundary conditions,

$$y(-1) = -1, \quad y(1) = -1 \tag{5.22}$$

   (See Figures (5.13),(5.14) and Table (5.6))

   Here ODE is tackling a nonlinear second order ordinary differential equation. The evolution at various point resolutions is shown in figure (5.13). The routine quickly gets the shape at $npoint = 6$. At $npoint = 11$, the symmetry of the final answer is evident and it is only lacking in magnitude, which is obtained by the end of the program run at $npoint = 21$.

We have shown that genetic algorithms may be successfully applied to ordinary differential equations to at least generate a reasonable answer. However, it seems fair to say that traditional numerical methods for solving ordinary differential equations remain the first choice in all but the most highly nonlinear situations. This still leaves a vast range equations that may be tackled genetically.

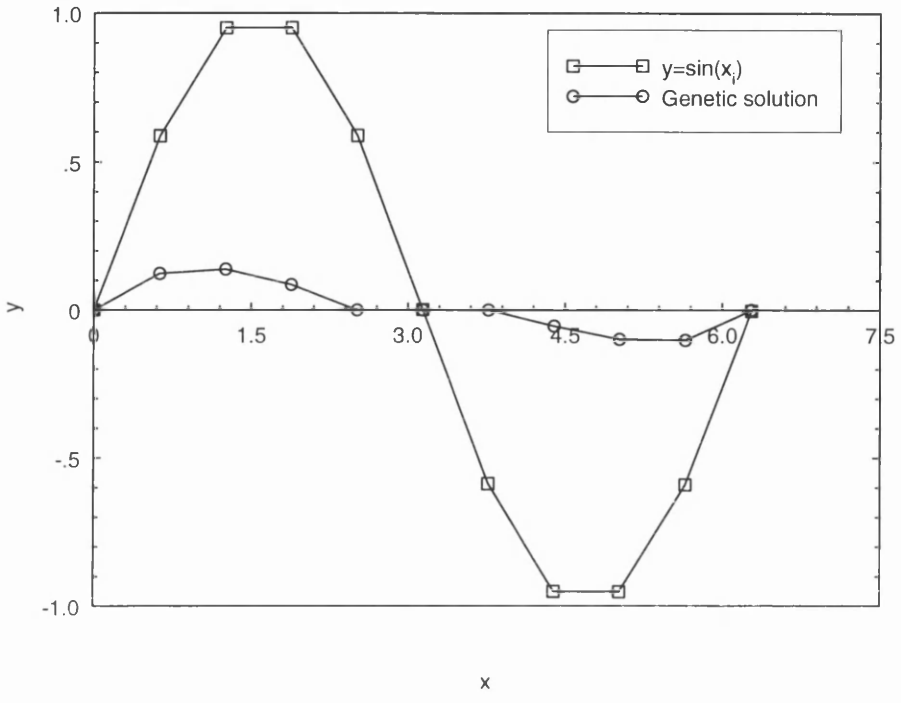

Figure 5.3: Genetic versus true solution for eqn.(5.17)

The genetic algorithm experience gained here suggests that it would be profitable to apply genetic algorithmic techniques to more demanding situations where traditional methods fall down. In chapter 6, we discuss the implementation of a genetic algorithm to solve Poisson's equation.

variable	value
npoint	11
f_1	1.0
p_3	1.0
gvl	1.0
gvr	1.0

Table 5.1: ODE variable values for Fig. (5.3),(5.4)

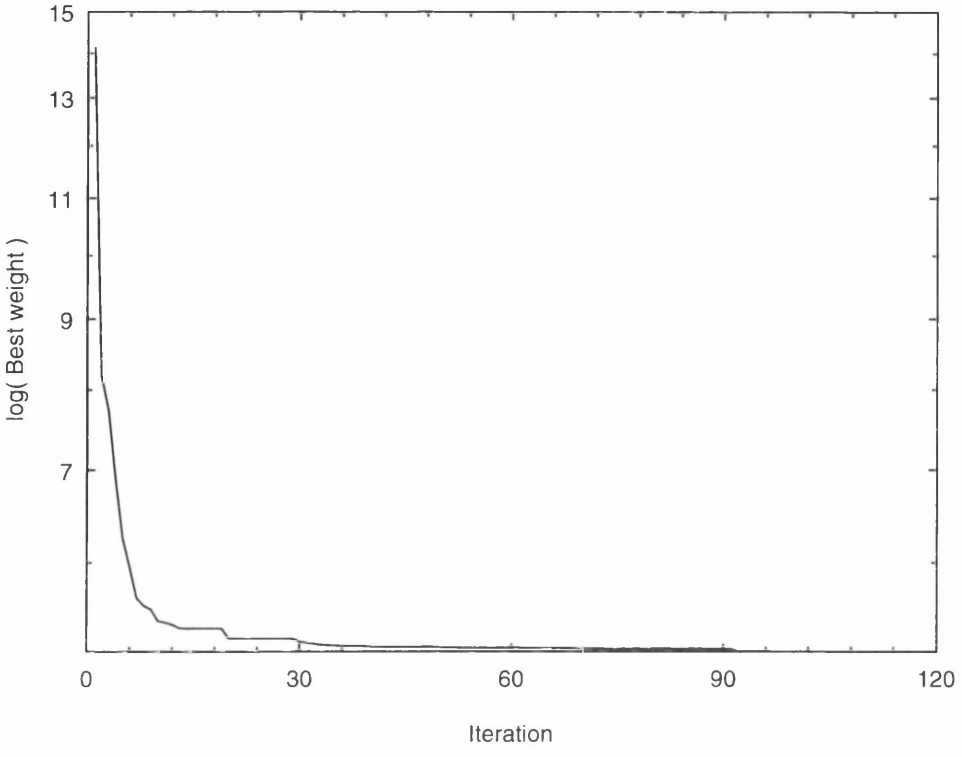

Figure 5.4: Best candidate weight at each iteration for eqn.(5.17)

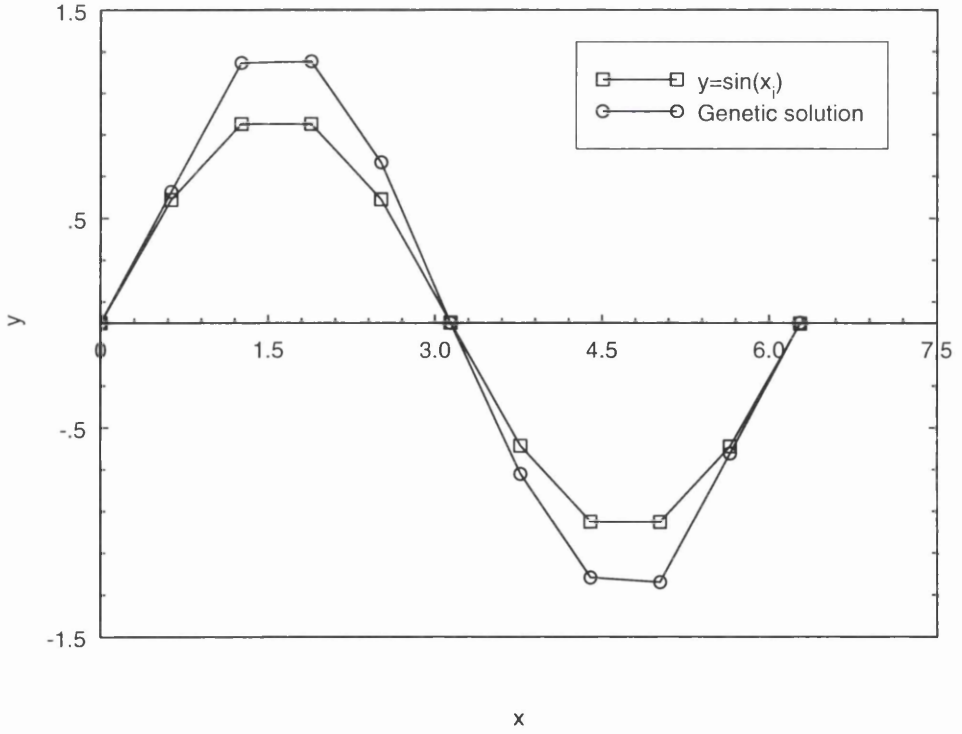

Figure 5.5: Genetic versus true solution for eqn.(5.17) with large boundary penalties

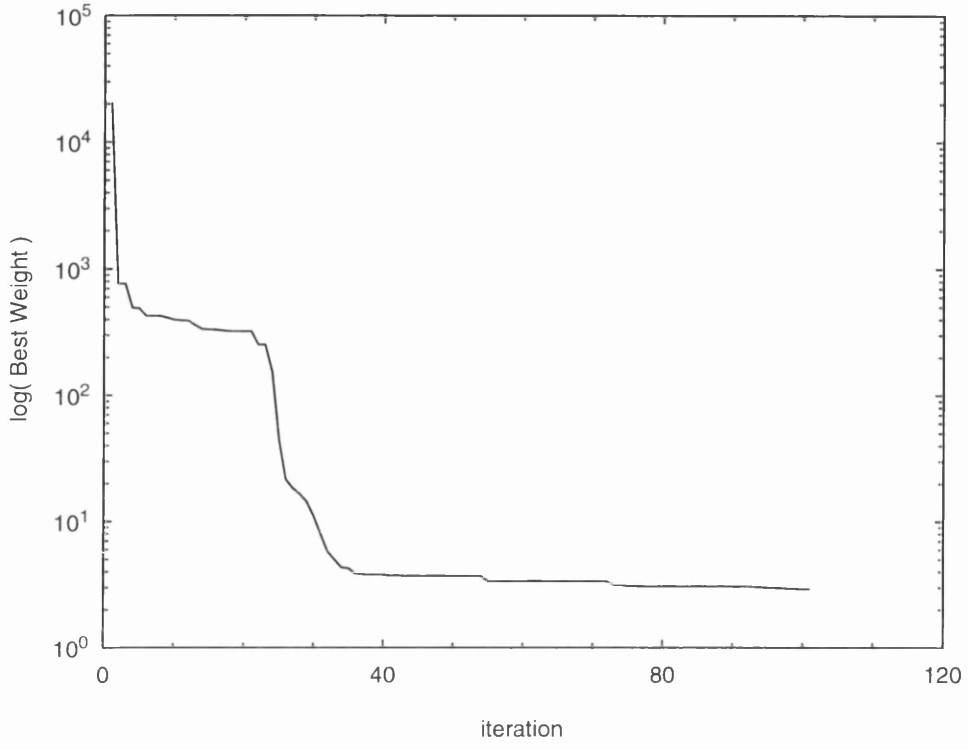

Figure 5.6: Best candidate weight at each iteration for eqn.(5.17) with large boundary penalties

variable	value
npoint	11
f_1	10.0
p_3	10.0
gvl	1.0
gvr	1.0

Table 5.2: ODE variable values for Fig. (5.5),(5.6)

variable	value
npoint	6→11
f_1	1.0
p_3	1.0
gvl	1.0
gvr	1.0

Table 5.3: ODE variable values for Fig. (5.7),(5.8)

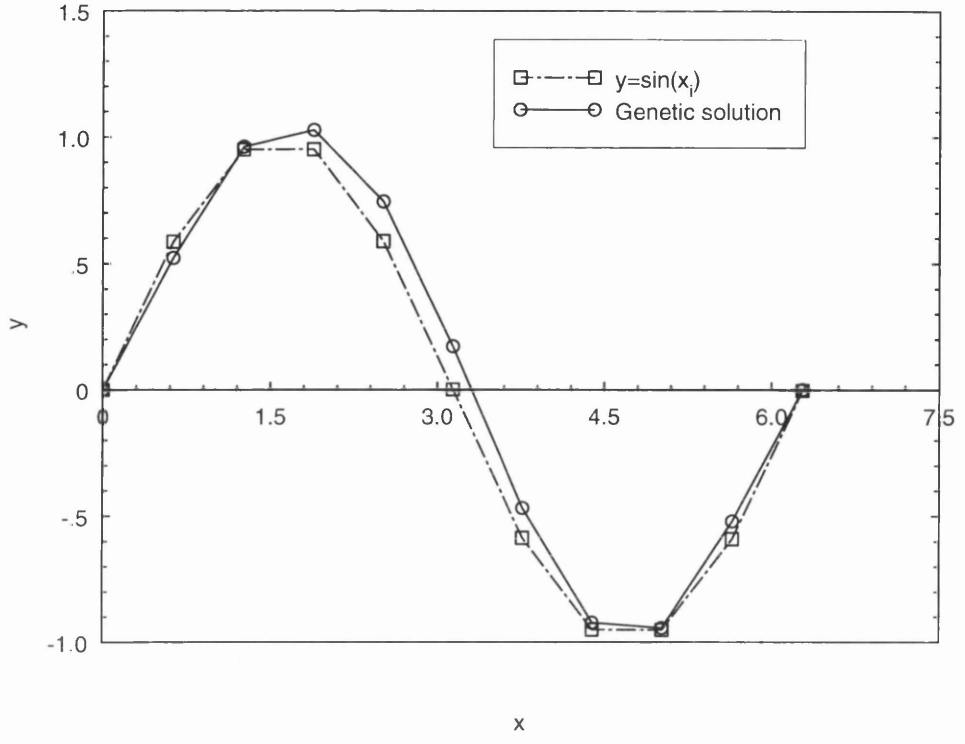

Figure 5.7: Genetic versus true solution for eqn.(5.17) with point doubling

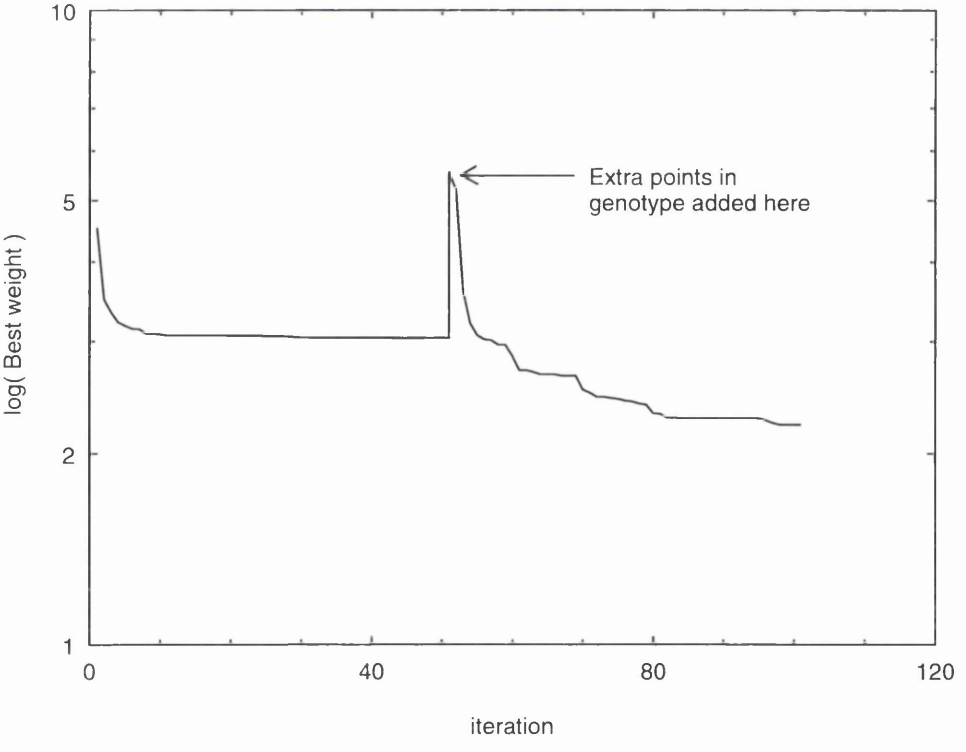

Figure 5.8: Best candidate weight at each iteration for eqn.(5.17) with point doubling.

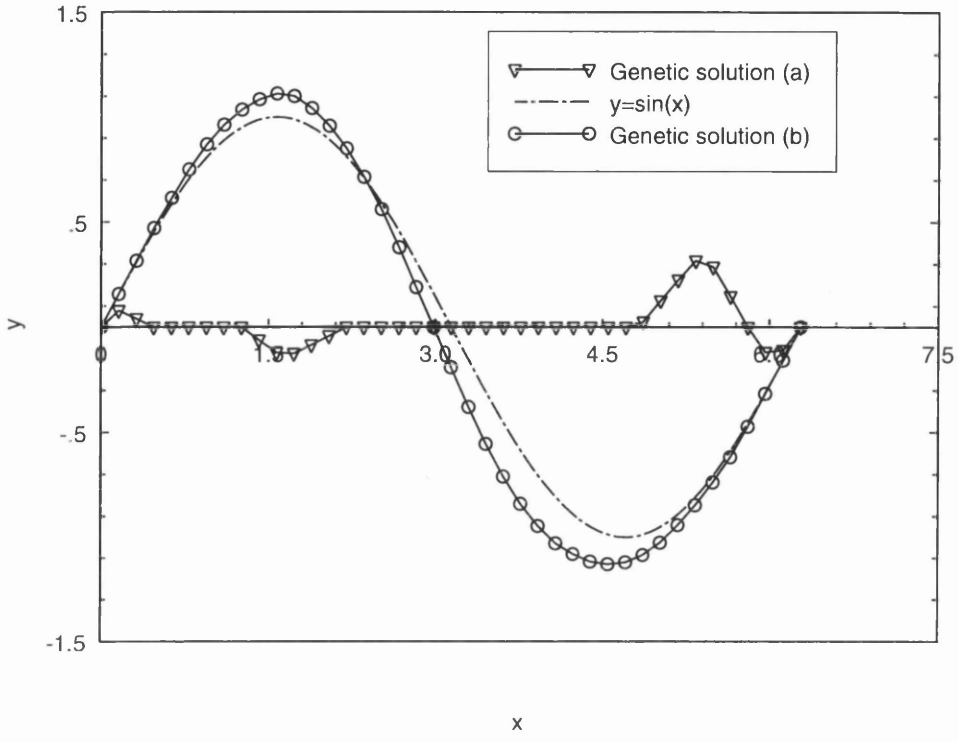

Figure 5.9: Genetic versus true solution for eqn.(5.17) with and without point doubling

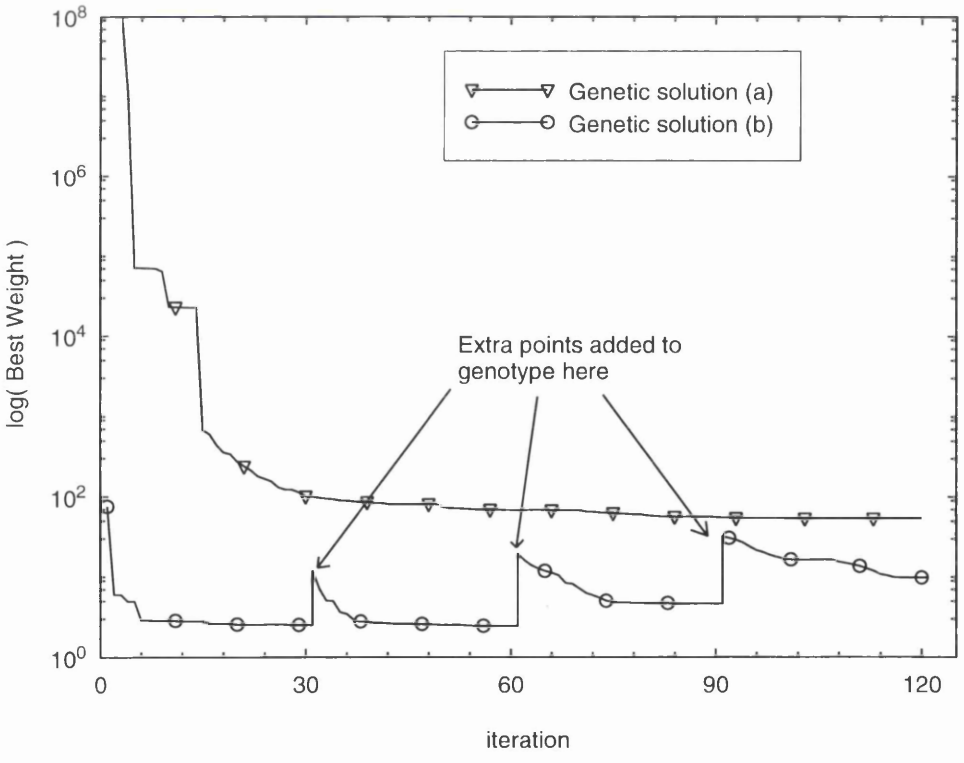

Figure 5.10: Best candidate weight at each iteration for eqn.(5.17) for point doubling run in figure (5.9)

variable	genetic solution (a) value	genetic solution (b) value
npoint	41	6→11→21→ 41
f_1	10.0	10.0
p_3	10.0	10.0
gvl	1.0	1.0
gvr	1.0	1.0

Table 5.4: ODE variable values for Fig. (5.9),(5.10)

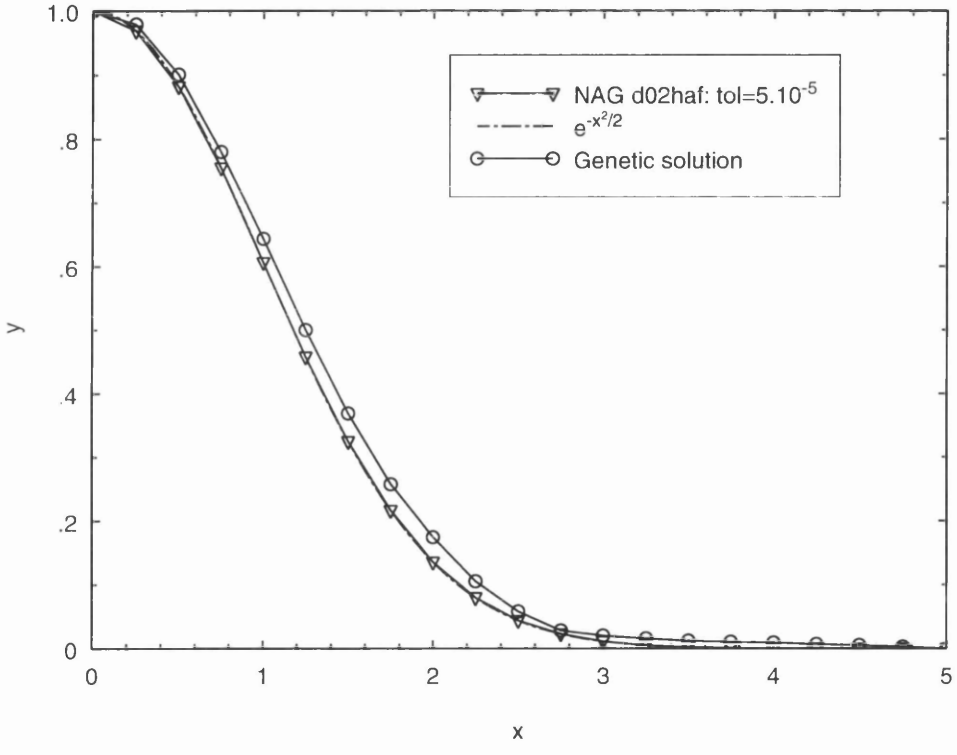

Figure 5.11: Genetic versus true solution for eqn.(5.19)

variable	value
npoint	6→ 11
f_1	10.0
p_3	10.0
gvl	0.0
gvr	0.0

Table 5.5: ODE variable values for Fig. (5.11),(5.12)

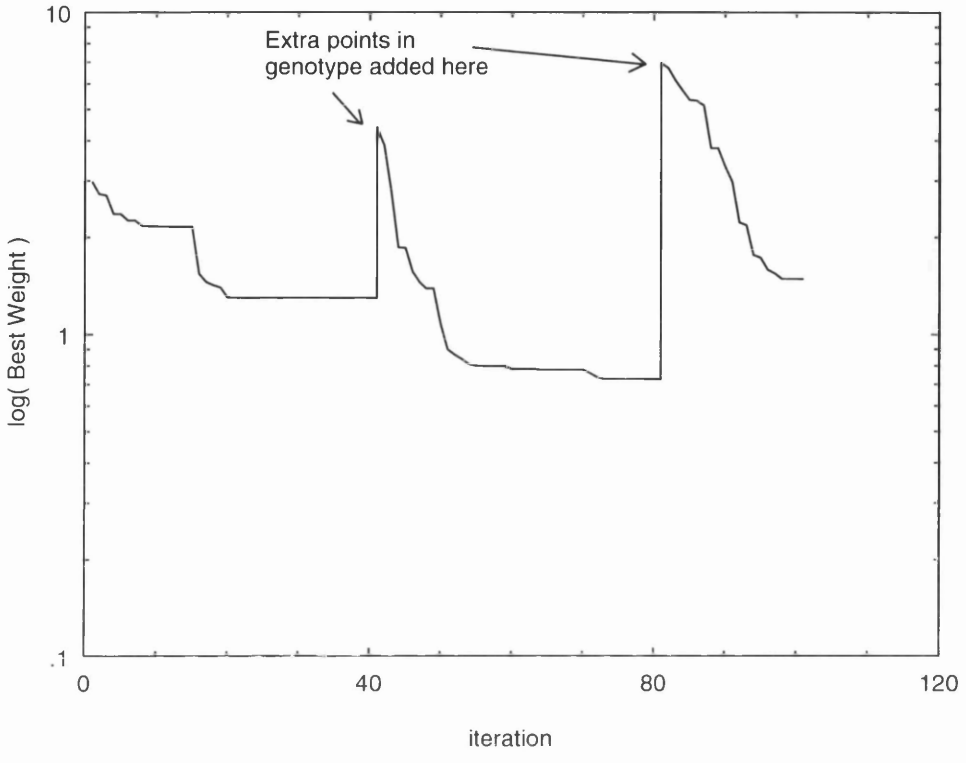

Figure 5.12: Best candidate weight at each iteration for eqn.(5.19)

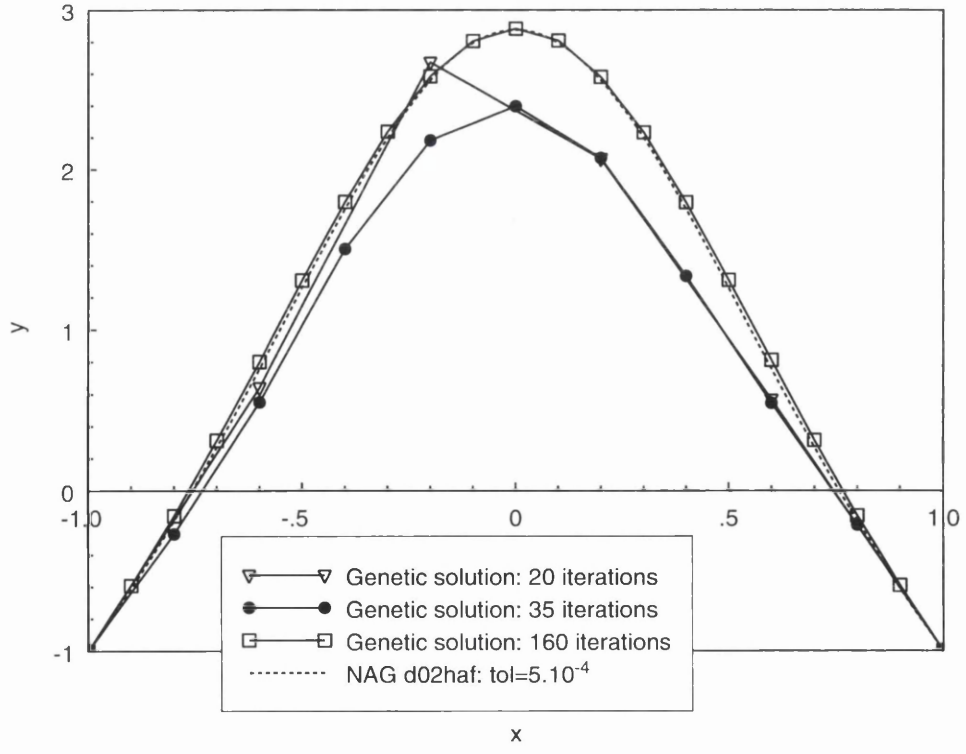

Figure 5.13: Genetic versus true solution for eqn.(5.21)

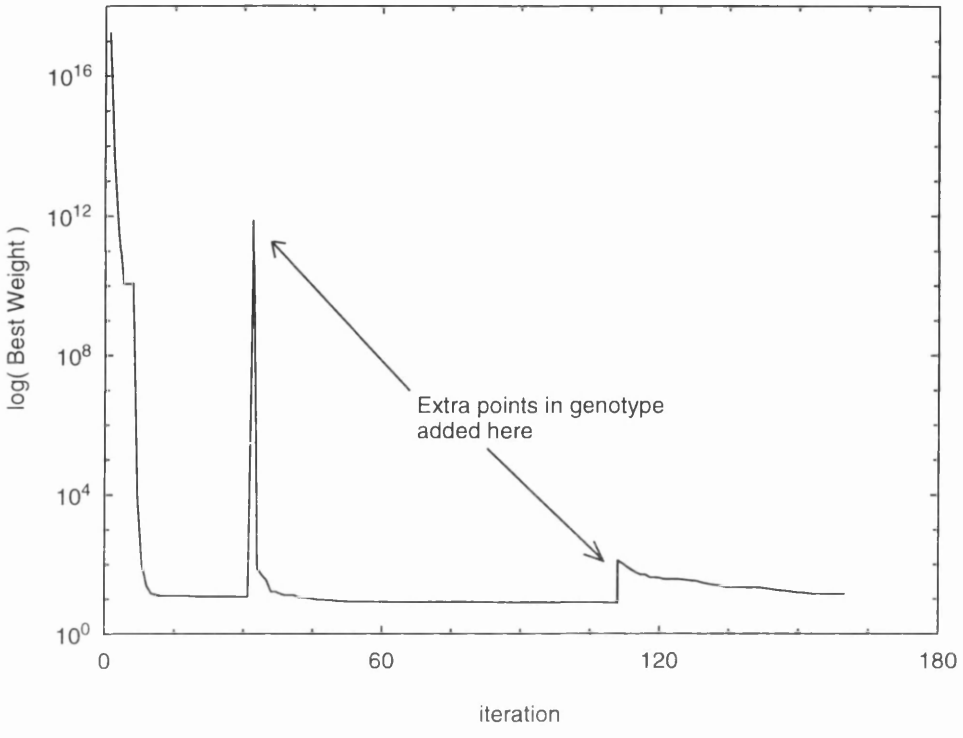

Figure 5.14: Best candidate weight at each iteration for eqn.(5.21)

variable	value
npoint	6→ 11→ 21
f_1	1.0
p_3	100.0
gvl	4.0
gvr	-4.0

Table 5.6: ODE variable values for Fig. (5.13),(5.14)

Chapter 6

A Genetic Poisson Equation solver

A genetic algorithm is developed for solving Poisson's equation on a rectangular region with equally spaced grid points in both the x and y directions. A number of new genetic operators are introduced to better cope with the 2-dimensional nature of this problem. The genetic answers are compared principally against NAG routine D03EBF.

6.1 Poisson's Equation

An obvious next step is to go up to a 2 dimensional version of (5.2), moving from ordinary differential equations to partial differential equations. In view of equation (3.21) arising in chapter 3, we therefore consider a genetic algorithm to the solution of

$$\nabla^2 A = S\left(x, y, A, \frac{\partial A}{\partial x}, \frac{\partial A}{\partial y}\right) \tag{6.1}$$

for arbitrary function S, on a rectangular region $x \in (x_0, x_{end})$, $y \in (y_0, y_{end})$ with Dirichlet boundary conditions

$$A = g\,(x, y) \tag{6.2}$$

on the boundary of the above rectangular region with g also arbitrary. Ideally, the genetic algorithm will provide initial answers for a more exact solver to refine. Hence, we will first examine an existing numerical solution routine from the NAG library, NAG D03EBF [44]. This program is based on matrix manipulation, and is described in section 6.1.1.

6.1.1 A Matrix Method Routine: NAG D03EBF

We can apply NAG D03EBF to the solution of Poisson equations via the 5-point molecule finite difference replacement of ∇^2 on a square grid

$$\nabla^2 A \approx \frac{1}{h^2}\left[A_{i,j+1} + A_{i+1,j} + A_{i,j-1} + A_{i-1,j} - 4A_{i,j}\right] \tag{6.3}$$

where h is the distance between nearest gridpoints [37], i.e.

$$\begin{array}{ccccc} & & A_{i,j+1} & & \\ & & | & & \\ A_{i-1,j} & — & A_{i,j} & — & A_{i+1,j} \\ & & | & & \\ & & A_{i,j-1} & & \end{array} \tag{6.4}$$

This routine solves a set of simultaneous (possibly nonlinear) equations of the form

$$MA = Q \tag{6.5}$$

M is a $N_1 \times N_2$ by $N_1 \times N_2$ matrix and Q is a known matrix of size $N_1 \times N_2$ by 1. The matrix A is the same size as Q and represents the unknowns. Such a set of equations can be generated by a finite difference representation of a two dimensional partial differential equation and can therefore be applied to solve (6.9) with boundary conditions (6.10).

In general we have a set of equations which may be written in the form

$$s_{i,j}A_{i,j-1} + t_{i,j}A_{i-1,j} + u_{i,j}A_{i,j+1} + v_{i,j}A_{i+1,j} + w_{i,j}A_{i,j} = q_{i,j} \tag{6.6}$$

for $i = 1, ..., N_1, j = 1, ..., N_2$ where s, t, u, v and q may depend on x, y and A. When any one of the coefficients s, t, u or v depend on any of the $A_{i,j}$'s then (6.6) is a set of nonlinear equations in $A_{i,j}$. The system is solved iteratively, from a starting approximation T^1 by the formulae

$$R^n = Q - MT^n$$

$$MS^n = R^n$$

$$T^{n+1} = T^n + S^n$$

R^n is the residual error matrix of the n'th approximate solution T^n, and S^n is the updating vector. The user must supply the initial approximation T^1 and the matrix M.

The boundary conditions are slotted in to Q via $q_{i,j}$ and for a source term linearly dependent on A, $\frac{\partial A}{\partial x}$ or $\frac{\partial A}{\partial y}$ the matrix M is suitably modified by the required finite difference representation. This may be most easily explained in terms of an example. Consider the Poisson equation

$$\nabla^2 A = A - \frac{\partial A}{\partial x} + 5\sin(xy) \tag{6.7}$$

The A in the source term may be accounted for by expressing it as $A_{i,j}$ and moving it into the matrix M by changing the value of the coefficient $w_{i,j}$. We can handle the partial derivative of A similarly by using the finite difference representation $\frac{\partial A}{\partial x} \approx \frac{A_{i,j+1} - A_{i,j-1}}{2h}$. The sinusoidal source

term can be expressed in the matrix Q. With this equation the routine need only be called once, and will generate an answer for the $A_{i,j}$'s on the grid hence solving (6.7).

But if we have a source term nonlinear in A, $\frac{\partial A}{\partial x}$ or $\frac{\partial A}{\partial y}$ then we have a choice as to how best to represent and solve the equation. A common technique used to solve a set of nonlinear equations is to solve the current set, say $A_{i,j}^{unknown}$ with coefficients that depend on the previous iteration, say $A_{i,j}^{old}$. Hopefully - although this is not guaranteed - the solution will converge, and converge to something acceptable. The equation

$$\nabla^2 A = A^2 \tag{6.8}$$

may be implemented in at least two ways, following the method above of using a previous approximate solution. One can express A^2 as the square of the $A_{i,j}$'s of the previous level: this means that the matrix M is just the finite difference representation of ∇^2 whereas the source term looks like a function of x, y only, and not A: we have effectively changed the above nonlinear equation (6.8) to the linear $\nabla^2 A = k_1(x, y)$ at each iteration. for some function k_1. Alternatively, one can express A^2 as the product of the previously determined value multiplied by the unknown at the grid point, $A_{i,j}^{old} A_{i,j}^{unknown}$. This resembles the linear equation $\nabla^2 A = k_2(x, y) A$ for some function k_2.

Obviously the number of possible representations depends on the nature of the nonlinear source term and it is entirely likely that different implementations may yield differing answers or indeed, none at all.

This approach is very successful when both the source term and the boundary conditions are in some sense, 'simple' for instance, constant source term and 'smooth' boundary conditions. However in some cases, these routines have very great difficulty in converging to an answer, as is shown in section 6.1.2.

6.1.2 Example application

Consider the following Poisson equation

$$\nabla^2 A = A^2 \tag{6.9}$$

on a square grid in Cartesian co-ordinates with boundary conditions

$$A = \begin{cases} 4\sin\left(\frac{\pi x}{4}\right) & \text{y=-2,-2} \\ 4\sin\left(\frac{\pi y}{4}\right) & \text{x=-2,-2} \end{cases} \tag{6.10}$$

This system was implemented using NAG D03EBF [44]. The square grid, lying in the region $-2 < x < 2$, $-2 < y < 2$, consisted of $N_1 = 11$ points in the x-direction and $N_2 = 11$ points in the y-direction. Following the advice given in the NAG manual [44], the initial approximate answer was set to be $T_{i,j}^0 = 0$.

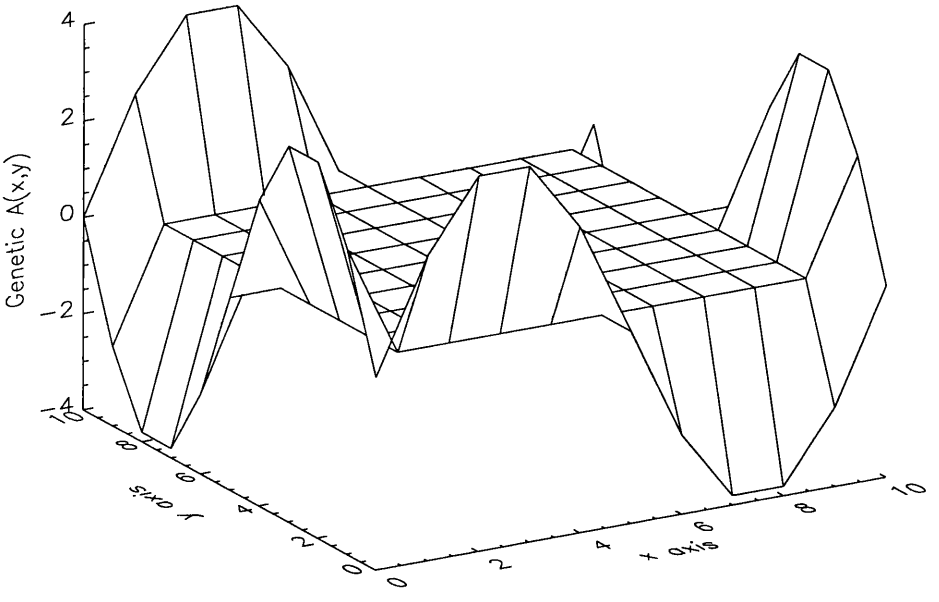

Figure 6.1: NAG solution to equation (6.9) when set up as $\nabla^2 A = k_2(x, y)A$

Convergence is measured against two user set criteria, the NAG variables *conres* and *conchn*. *conres* specifies the convergence criterion to be used on the maximum absolute value of the normalised residual vector (matrix R in NAG DO3EBF - see section 6.1.1) components. The components are normalised by the coefficient of $A_{i,j}$ stored in the array M when that coefficient is non-zero. If the coefficient is zero, the residual is taken from the vector as it is.

The NAG variable *conchn* is the convergence criterion used on the maximum absolute value of the change made at each iteration to the elements of the array T (see section 6.1.1 for more details), namely the dependent variable.

The routine was initially run with $conres = conchn = 0.1 \times 10^{-5}$. The routine was set up to successfully exit when both criteria were satisfied, as recommended. Only when (6.9) was set up as $\nabla^2 A = k_2(x, y)\, A$ did the routine write out anything even remotely sensible (see figure 6.1). With the alternate representation $\nabla^2 A = k_1(x, y)$ NAG D03EBF exited containing an array of non-numbers, indicating the complete collapse of the routine.

Both implementations did not work and both cases converged to unphysical answers for all internal grid points $2 < i, j < 10$. If we take the physical interpretation of the Poisson equation as the temperature distribution in a plate then we see that the NAG derived answer must be wrong.

Varying the tolerances *conres*, *conchn* does not aid convergence. It may be that NAG D03EBF is seriously hampered by the initial guess: it is just not close enough to the final answer to allow it to proceed. This is where a genetic algorithm may be useful in generating an initial guess.

In the following sections we describe the features and philosophy behind the construction of POISGEN, a genetic algorithm for the solution of Poisson's equation.

6.2 Representation

As we are now working with a partial differential equation, the code length per candidate will be approximately the square of the size of a candidate in ODE. We use a floating point number representation of the genetic code. If we have *xpoint* points in the x-direction and *ypoint* points in the y-direction (including boundaries) then each candidate contains $xpoint \times ypoint$ points, although only $(xpoint - 2) \times (ypoint - 2)$ of these need to determined by the genetic algorithm, the rest being boundary information. But along with the extra dimension, we must now decide how to store the genetic code. In order to make good use of the existing code ODE, it was decided to store the code in a similar fashion, as *ypoint* strips of *xpoint* points going in the x-direction across the square grid, i.e., for a typical p'th candidate $A^p_{i,j}$, $1 \leq i \leq xpoint$, $1 \leq j \leq ypoint$, (where we drop the superscript denoting candidate number for clarity), the grid of points look like

$$\begin{array}{ccccc} A_{1,ypoint} & A_{2,ypoint} & \rightarrow & \rightarrow & A_{xpoint,ypoint} \\ A_{1,ypoint-1} & A_{2,ypoint-1} & \rightarrow & \rightarrow & A_{xpoint,ypoint-1} \\ \uparrow & \uparrow & \nearrow & \nearrow & \uparrow \\ A_{1,2} & A_{2,2} & \rightarrow & \rightarrow & A_{xpoint,2} \\ A_{1,1} & A_{2,1} & \rightarrow & \rightarrow & A_{xpoint,1} \end{array} \tag{6.11}$$

The genotype is now formed from the x-strips

$$A_{1,1}A_{2,1}\cdots A_{xpoint,1}A_{1,2}A_{2,2}\cdots A_{xpoint,2}\cdots\cdots A_{1,ypoint}A_{2,ypoint}\cdots A_{xpoint,ypoint} \tag{6.12}$$

As ODE performs genetic operations on an array of floating point numbers, the genotype above is basically equivalent to that in ODE. The existing genetic algorithm mutation, combination and breeding routines will not notice any difference. However, changes are required to the profiling subroutine and the boundary condition writing routine, and are detailed below.

6.3 Weighting Operator

In analogy with the ordinary differential equation routine above, we implement a weighting function via the finite difference representation of (6.1). The ∇^2 term is easily handled via the five point replacement of (6.3). Central differences were used in the expression of S when necessary. We generate a fitting error $R^p_{i,j}$ associated with point $A^p_{i,j}$ (the point (i,j) in the p'th candidate) using

$$R^p_{i,j} = \frac{1}{hk}\left[A^p_{i,j+1} + A^p_{i+1,j} + A^p_{i,j-1} + A^p_{i-1,j} - 4A^p_{i,j}\right] - S^p_{i,j} \tag{6.13}$$

where $S^p_{i,j}$ is the value of the source term applied to the p'th candidate at the point (i,j), given the finite difference representation of S and $h = \frac{x_{end}-x_0}{xpoint-1}$, $k = \frac{y_{end}-y_0}{ypoint-1}$. We can now calculate measures of the fitness of each candidate. This are similar to those seen in ODE. The operator (6.13) is only applied at internal points, $2 \leq i \leq xpoint-1$, $2 \leq j \leq ypoint-1$. For this system it is not necessary to introduce extrapolants to points outside the region of interest. Previously, extrapolants were used to calculate estimates for y' and y'' at the boundaries, and these values were used to generate a measure of the fit of the candidate. This was as much a feature of the original problem as the mimicry of conventional numerical techniques (over specification of boundary information). However, in this problem we can assess the continuity of a candidate at the boundary by other means.

1.
$$\zeta^p_1 = \sum_{i=2}^{xpoint-1} \left[\sum_{j=2}^{ypoint-1} |R^p_{i,j}| \right] \tag{6.14}$$
This is a global estimate of the error of the p'th candidate, and is simply the extension into two dimensions of (5.10).

2.
$$\zeta^p_2 = \prod_{i=2}^{xpoint-1} \left[\prod_{j=2}^{ypoint-1} | R^p_{i,j} | \times \begin{cases} f_1 & f_1 > 1, \quad \text{if i=2}or\text{xpoint-1} \\ & \text{or j=2}or\text{ypoint-1} \\ 1 & \text{otherwise} \end{cases} \right] \tag{6.15}$$
In the definition of ζ^p_2 we attempt to measure the continuity of the p'th candidate, as well as the number of well fitting points in the string. By multiplying $R^p_{i,2}$, $R^p_{2,j}$, $R^p_{i,ypoint-1}$ and $R^p_{xpoint-1,j}$ by a factor f_1 we are increasing the penalty awarded to more discontinuous candidates. Hence, such candidates will have higher weights and will be preferentially discarded. This favours the survival of candidates that are more continuous with the boundary.

3.
$$\zeta^p_3 = \max_{i=1,xpoint}^{j=1,ypoint} | R^p_{i,j} | \tag{6.16}$$
Again we extend (5.13) into two dimensions. This enables us to penalise those candidates that have one large error that dominates the rest, making it difficult to relax to lower weighted individuals

There is no direct analogue to (5.12) as we do not need to supply an initial guess to the gradient of A on the boundary. Boundary continuity is examined in ζ^p_2. We define an overall weight assigned to the p'th candidate,
$$W^p = q_1\zeta^p_1 + q_2\zeta^p_2 + q_3\zeta^p_3 \tag{6.17}$$
where q_1, q_2, q_3 and f_1 are at the users' discretion.

6.4 Breeding, Mutation, Combination and Transcription Operators

In ODE, there was an entire class of genotypic operator missing from the code that come under the general title of *transcription errors.* These 'errors' form a vital part of POISGEN's functionality. When a DNA molecule is unzipped for reproduction, errors may occur. For instance, the copied molecule may have one or more bases missing - deletion. Alternatively, the copied molecule may have one or more genes repeated too often; this is known as addition. Entire segments of DNA may also be swapped over with others. At other times, the order of a particular set of genes can be reversed, which is known as inversion. Also seen is the random rearrangement of a set of genes in the DNA. Crucially, these errors change the content or order of the information carried by the code and by doing so maintain diversity of genotype (and hence phenotype) in the population [40, 45]. The following operators were included in the POISGEN code (see appendix A for more detail): genes appearing in new positions are denoted by a $\circ$ symbol.

1. Deletion

   A randomly sized segment of code from a random point in the array is removed. The array now has a chunk of code missing. Say for example the genotype consists of 10 genes and we delete those numbered 6 and 7. The genotype now looks like

   $$\bullet_1 \bullet_2 \bullet_3 \bullet_4 \bullet_5 - - \bullet_8 \bullet_9 \bullet_{10}$$

   We need two genes to fill the gap. The gap is filled by taking a copy of the material towards the nearest end of the string and moving it along the string to fill the gap, i.e.,

   $$\bullet_1 \bullet_2 \bullet_3 \bullet_4 \bullet_5 \circ_8 \circ_9 \circ_{10} \bullet_9 \bullet_{10}$$

   This is done in order to combat the problem of the effective movement of boundary information into the centre of the genotype. Although there is no guarantee that these genes will be of any use, it does crudely move information to where it is needed.

2. Copying

   This operation is superficially similar to deletion. A randomly sized segment of genotype is copied and placed back in the array, in a position closer to the centre. Consider the example below; genes 4 and 5 are earmarked for this operation.

   $$\bullet_1 \bullet_2 \bullet_3[\bullet_4 \bullet_5] \bullet_6 \bullet_7 \bullet_8 \bullet_9 \bullet_{10} \overset{copy}{\rightarrow} \bullet_1 \bullet_2 \bullet_3 \bullet_4 \bullet_5 \circ_4 \circ_5 \bullet_8 \bullet_9 \bullet_{10}$$

   Since we want to move genes towards the centre, genes 4 and 5 now take the place of 6 and 7, overwriting their values. If this were a deletion operation, 4 and 5 would be removed

completely. We would have

$$\bullet_1 \bullet_2 \bullet_3[\bullet_4\bullet_5] \bullet_6 \bullet_7 \bullet_8 \bullet_9\bullet_{10} \overset{deletion}{\rightarrow} \bullet_1 \bullet_2 \circ_1 \circ_2 \circ_3 \bullet_6 \bullet_7 \bullet_8 \bullet_9\bullet_{10}$$

Deletion involves shifting the genes along the array from one end to cover up the deleted region. Copying is implemented by only moving segments of genes towards the centre without affecting the ends.

3. Inversion

   A randomly selected piece of genotype of random length is inverted and placed back in the same place in the array i.e.,

$$\bullet_1 \bullet_2 \bullet_3 \bullet_4 \bullet_5[\bullet_6 \bullet_7 \bullet_8\bullet_9]\bullet_{10} \overset{inversion}{\rightarrow} \bullet_1 \bullet_2 \bullet_3 \bullet_4 \bullet_5 \circ_9 \circ_8 \circ_7 \circ_6\bullet_{10}$$

4. Mixing

   The genes in a randomly sized piece of array are taken and rearranged in a random order.

$$\bullet_1 \bullet_2 \bullet_3[\bullet_4 \bullet_5 \bullet_6 \bullet_7 \bullet_8\bullet_9]\bullet_{10} \overset{mix}{\rightarrow} \bullet_1 \bullet_2 \bullet_3 \circ_5 \circ_9 \circ_8 \circ_4 \circ_7 \circ_6\bullet_{10}$$

5. Swap

   Two randomly sized pieces of code from non-overlapping regions in the genotype are swapped intact without any further operations acting on them such as mixing etc.

$$\bullet_1[\bullet_2 \bullet_3 \bullet_4] \bullet_5 \bullet_6 \bullet_7 [\bullet_8 \bullet_9 \bullet_{10}] \overset{swap}{\rightarrow} \bullet_1 \circ_8 \circ_9 \circ_{10} \bullet_5 \bullet_6 \bullet_7 \circ_2 \circ_3\circ_4$$

It was found that these operators constituted an important improvement in POISGEN's performance when coupled with a code re-ordering on entry to the transcription subroutines.

We have chosen to represent the genetic code in such a way that we can use the existing code ODE with the minimum of refurbishment, and in doing so we have imposed a directionality on the code. The normal code order is that described by (6.12). Hence, the transcription operators can only shuffle material along one direction. This reduces their efficacy, as their is no reason in the general case why one direction should be preferred over another in the true solution. On entry to the transcription subroutines, the program randomly decides to re-order the code to

$$A_{1,1}A_{1,2}\cdots A_{1,ypoint}A_{2,1}A_{2,2}\cdots A_{2,ypoint}\cdots\cdots A_{xpoint,1}A_{xpoint,2}\cdots A_{xpoint,ypoint} \qquad (6.18)$$

This allows the transcription error operators the chance to move genes around in both directions, increasing gene mobility and genotypic variability. On exit, the code is decrypted back to the normal form of (6.12). The weighting routine only accepts normally directed code of the type shown in (6.12).

Care must be exercised in generating individuals with the correct boundary conditions as the information now no longer appears simply at either end of the string: the boundary information is written into the array at the correct points.

The mutation and combination operators described above for ODE can be carried over without change to POISGEN, as these merely act on the genes, regardless of what they mean to the solution of the problem, i.e., they are blind to the origin of the genetic code. For a uniform distribution of mutation (or combination) operations in the code, there is a chance that the boundary information will be altered. Obviously, such an individual cannot be permitted to exist in this form and remedial action must be taken. Therefore, before a weight is assigned to each individual, the boundary information must be overwritten into the code at the correct place. This is done *after* the transcription operations have taken place, as although incorrect for this point in the code, a gene not very different from the boundary genes may be useful, especially near the boundary.

ODE breeds children using one point crossover (see section 5.3.3) by copying the relevant genetic material from each parent into a new entry of the population array. It then assigns a weight to the child. The program stores the entire population in memory at all times.

6.5 Convergence strategies

The techniques used for varying the content (section 5.3.4) of the gene pool remain largely the same as in ODE. Where and to what extent they are used is based on experience gained while using the program. Since the routine is still at an experimental stage it should be understood that the current listing is a snapshot of a program in flux. The major new features as compared to ODE are described below.

1. The population is seeded by a special individual calculated via the boundary conditions. A straight line is calculated between opposing pairs of boundary points. For instance, the straight line joining $A_{i,1}$ and $A_{i,ypoint}$ is given by $v_i(y_j) = A_{i,1} + \frac{y_j - y_0}{y_{end} - y_0}[A_{i,ypoint} - A_{i,1}]$. Similarly, for points $A_{1,j}$ and $A_{xpoint,j}$ we have $w_j(x_i) = A_{1,j} + \frac{x_i - x_0}{x_{end} - x_0}[A_{xpoint,j} - A_{1,j}]$. This allow one to define the internal points of the special individual as

$$A_{i,j}^{special} = \frac{v_i(y_j) + w_j(x_i)}{2} + R \times \left[\frac{highbdry - lowbdry}{frac}\right]$$

where $2 \leq i \leq xpoint$ and $2 \leq j \leq ypoint$. R is a random number between -1 and 1 and $frac > 1$, a scaling which allows one to control what fraction of the range of gene values. *highbdry* and *lowbdry* are the maximum and minimum permitted gene values respectively (see appendices A, B for details). This acts as a seed from which the algorithm can start to evolve more advanced individuals.

2. The breeding population replacement routine **tweak** is kept intact with only minor changes to allow for the larger size of this new genotype.

3. POISGEN uses the strategies of ODE detailed in section 5.3.4, but with some important changes. Those strategies that changed gene values (aside from replacing the entire population) all used a geometric creep to effect this. In POISGEN, the same strategies are used, but a random choice is made between using a geometric creep or an arithmetic creep. This was found to aid convergence rates, as we have access to another important operator.

4. Decision making.

   POISGEN has a subroutine called **decide** (see appendix A) that allows the user to set tolerances for the activation of convergence strategies. After a fixed number of iterations, the subroutine is called which takes stock of the recent behaviour of the routine. The other convergence strategies are called while the routine is still iterating to an answer. This routine looks at a finalised answer and decides on what course of action to take. If the answer is not good enough, it takes the breeding population $2 \rightarrow npar$ and adds a small amount of random 'noise' over the entire genotype. This is set to be a fraction of the difference between the highest and lowest gene values in the current genotype multiplied by a random number between -1 and 1. The noise is therefore scaled to each genotype. Additionally, some of the previous strategies are implemented at this point but with larger values of the permitted range of variation. This is done in attempt to introduce some bigger gene variations in the population outside the main loop.

   If the answer is within the set tolerances, then the point resolution along each direction is doubled. For the doubled genotype, a new set of tolerances are required to control its evolution.

5. The worst point in the best candidate is also treated here. This time four copies of the best are made, each having the points around it (see four-point molecule (6.4)) as well as the point itself changed. This was done to recognise the fact the points surrounding a best point contribute to its fitting error to some extent. Hence the fit at a particular point can be improved by making changes to its neighbouring points.

6.6 Experimental Results and Algorithm Behaviour

Again, it is unreasonable to expect that a random technique will give us perfect results every time. To aid the assessment of results, we introduce some numerical measures POISGEN's performance. The quality of each answer was measured using the extension into two dimensions of the arithmetic

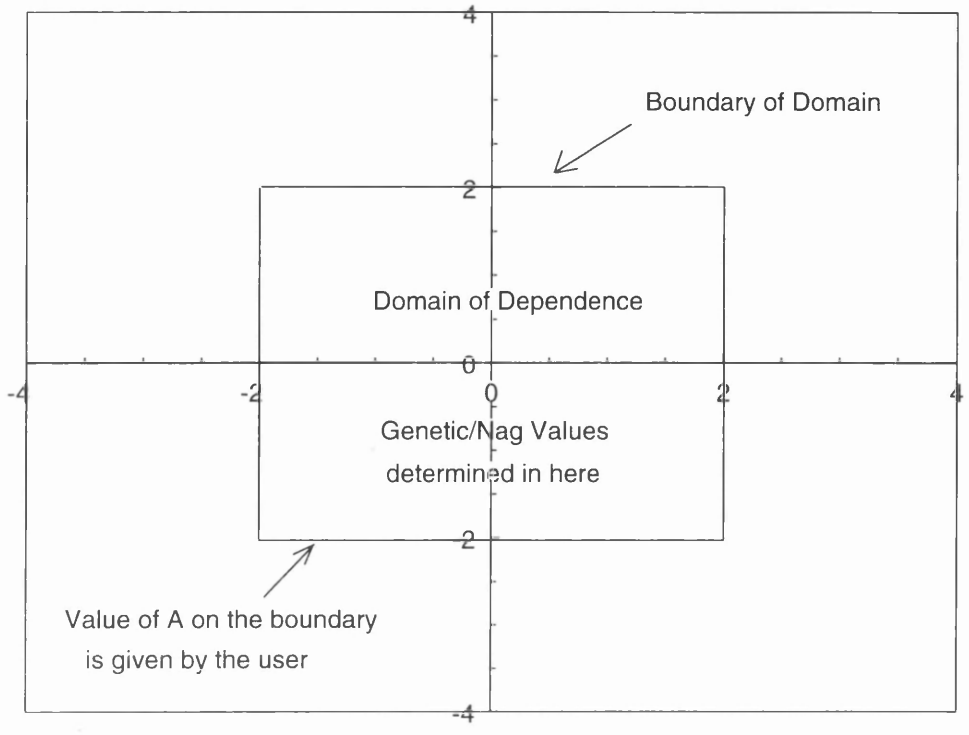

Figure 6.2: Region of dependency for equations (6.21) to (6.25)

and geometric means (5.15), (5.16).

$$\varepsilon_1 = \frac{1}{(xpoint-2)\,(ypoint-2)} \sum_{i=2}^{xpoint-1} \left[\sum_{j=2}^{ypoint-1} \left| A_{i,j}^{true} - A_{i,j}^{algorithm} \right| \right] \tag{6.19}$$

$$\varepsilon_2 = \sqrt[n]{\prod_{i=2}^{xpoint-1} \prod_{j=2}^{ypoint-1} | A_{i,j}^{true} - A_{i,j}^{algorithm} |} \tag{6.20}$$

where $n = (xpoint-2) \times (ypoint-2)$. The values $A_{i,j}^{true}$ come from either the analytic solution or NAG routine D03EBF, fed with the genetic answer as an initial guess, or with zero initial guess, as specified. Note that these are absolute errors and measure the average distance between the true solution and the genetic answer.

Although these are all crude measures they do give an idea of both algorithm performance and the features inherent in the equation chosen: for example, the sizes of the numbers involved and the difficulty of the equation. Below is a list of some equations tackled by POISGEN: each of them was solved on the region of space displayed in figure (6.2). Unless otherwise specified, POISGEN first calculates an answer with $xpoint = ypoint = 6$ and then doubles to $xpoint = ypoint = 11$. The program was run using $lowbdry = -10, highbdry = 10$: this defines the minimum and maximum gene values possible (see appendix B for details). Evolutionary histories are not quoted in general, but some examples are shown.

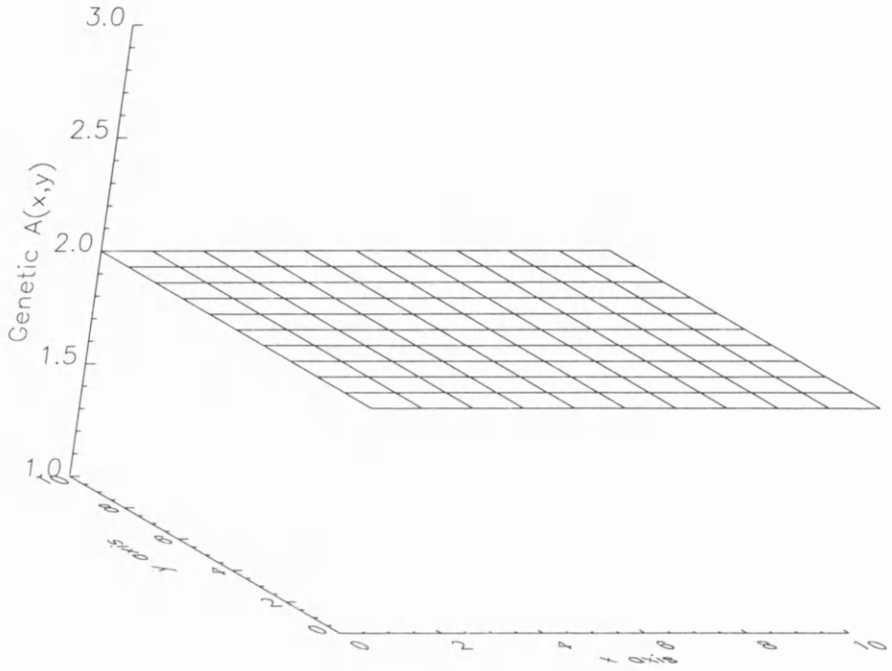

Figure 6.3: Genetic solution to equation (6.21)

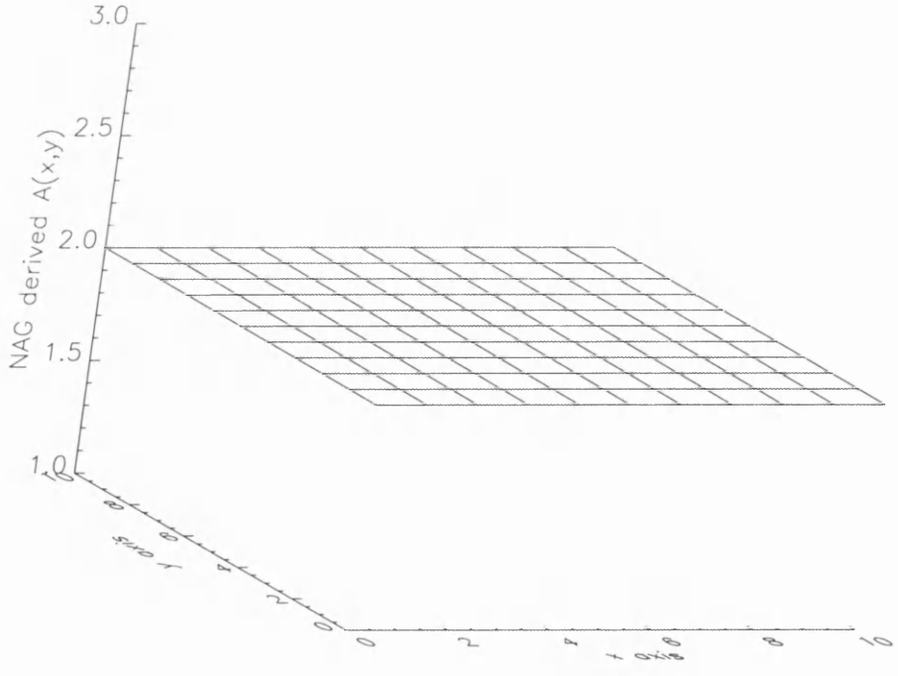

Figure 6.4: NAG routine solution to (6.21)

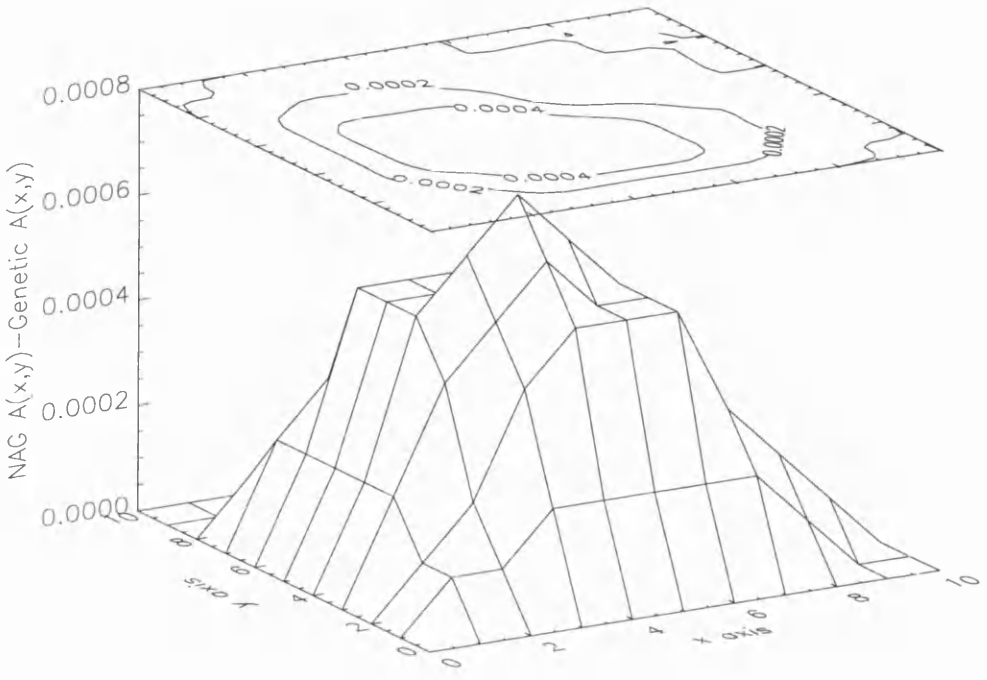

Figure 6.5: Difference between (6.3) and (6.4)

1.

$$\nabla^2 A = 0, \quad A = 2 \text{ on all boundaries} \tag{6.21}$$

$$\text{Errors:} \quad \varepsilon_1 = 2.40 \times 10^{-4} \quad \varepsilon_2 = 0.0$$

See figures (6.3),(6.4), (6.5) and (6.6).

One can see at a glance that the genetic answer (figure (6.3)) appears identical to the analytic answer (figure(6.4)), $A = 2$ everywhere. The difference between the two, shown in figure (6.5), bears this out, as do the error measures. (Machine precision was exceeded in the second error and therefore zero was given). That they are so close is very encouraging. This shows that the routine has promise, but this should be tempered by the fact that equation (6.21) is a very simple Laplace equation.

An obvious feature in the difference between the two solutions is the increase in error as we move away from the boundaries, exactly as expected. Hence we will expect most of the difference figures to peaked approximately in the middle of the region. This also denotes the region where the algorithm has the greatest difficulty in finding good points.

By using a floating point representation, the smallest permitted deformation to each point is limited by machine precision: this means that evolution - by evolution we mean the emergence of a candidate with a lower best weight - may proceed by smaller increments. It

is not necessarily true that a small change to a gene value leads to a small change in the weight of that candidate. But for most points away from the boundary, this is approximately true. Hence evolution may proceed in jumps smaller than previously available when using the GENODE representation.

The evolutionary history is shown in figure (6.6).

2.

$$\nabla^2 A = A, \quad A = \begin{cases} \exp(y-1), & \text{if } 1 < y < 2, x = -2 \\ \exp(|x|-1), & \text{if } -2 < x < -1, y = 2 \\ 1, & \text{everywhere else} \end{cases} \tag{6.22}$$

$$\text{Errors:} \quad \varepsilon_1 = 1.56 \times 10^{-2} \quad \varepsilon_2 = 9.79 \times 10^{-3}$$

See figures (6.7),(6.8) and (6.9).

Equation (6.22) is slightly harder than (6.21), being a Poisson equation with nonconstant boundary conditions. POISGEN's performance is very good: there is a small absolute distance between the NAG and genetic answers. Again, the worst errors are towards the centre of the region. The equivalence of problem difficulty in both directions is also demonstrated in the relatively symmetrical distribution of error. Examination of the difference between the NAG and genetic answers shows that close to the peak in A, the error is the least, suggesting that the high boundary weighting is working well with the relatively high value of the peak as compared to the rest of the surface. The NAG routine also had no trouble obtaining this solution, as expected. This particular answer was generated using the coding $\nabla^2 A = A^{old}_{i,j}$ - see section 5.1 for details. When the NAG answer is submitted submitted to POISGEN a very small weight is generated - very much less than 1. This indicates that we the NAG routine is generating good answers we can trust. If the NAG routine answer gave a 'large' non-zero weight then this would indicate a poor NAG supplied answer.

3.

$$\nabla^2 A = xy, \quad A = 3 \text{ on all boundaries} \tag{6.23}$$

$$\text{Errors:} \quad \varepsilon_1 = 2.80 \times 10^{-2} \quad \varepsilon_2 = 2.01 \times 10^{-2}$$

See figures (6.10),(6.11) and (6.12).

This is an inhomogeneous problem, and again, POISGEN shows no great difficulty in finding a very close answer within 2000 iterations. The answer is symmetrical, as are the point by point errors in figure (6.5); this is merely a reflection of the symmetry in the original problem.

4.

$$\nabla^2 A = xy, \quad A = \begin{cases} 4\sin\left(\frac{\pi x}{2}\right), & \text{if } y = -2, 2 \\ 4\sin\left(\frac{\pi y}{2}\right), & \text{if } x = -2, 2 \end{cases} \tag{6.24}$$

Errors: $\varepsilon_1 = 0.144 \quad \varepsilon_2 = 0.102$

See figures (6.13),(6.14) and (6.15).

This is the most demanding problem yet considered, and this shows up in the error measures. The measured errors are beginning to become significant, even although the two solutions look quite similar. Again, the NAG supplied answer yielded a very low ($\ll 1$) POISGEN weight, indicating a genuine solution. Figure (6.15) tells the story; symmetrical and large errors exist at all nonboundary points. This indicates a feature of genetic algorithms applied to differential equations first mentioned in chapter 5, namely, the shape finding qualities of the program. The general shape is similar to the true answer, but the actual values are not particularly accurate. However, this does suggest that we are on the right track for using genetic algorithms to find approximate solutions to more difficult Poisson equations.

5.

$$\nabla^2 A = A^2, \quad A = \begin{cases} 4\sin\left(\frac{\pi x}{2}\right), & \text{if } y = -2, 2 \\ 4\sin\left(\frac{\pi y}{2}\right), & \text{if } x = -2, 2 \end{cases} \tag{6.25}$$

Errors: $\varepsilon_1 = 0.952 \quad \varepsilon_2 = 0.434$

See figures (6.16),(6.17), (6.18) and (6.19).

There are several features to this problem that make it an interesting example.

(a) It seems to be a relatively difficult problem to solve. The NAG routine failed completely for implementations equivalent to $\nabla^2 A = k_1(x, y)A$ and $\nabla^2 A = k_2(x, y)$ (for some functions k_1, k_2 - see sections, 6.1.1, 6.1.2 for more detail). Changing the various parameters in the NAG routine did not help in finding a solution.

(b) The evolutionary history (figure (6.19)) also reveals the difficulty of this particular problem: there is a very high initial best weight (the first 10 weights have been omitted as they beyond the range of the plotting program) which drops quickly to less than 100. From here, evolution is slow and shallow, indicating that this candidate is relatively stable. On doubling the point resolution, the best weight increases by 6 orders of magnitude: this reflects the large number of new and relatively unfit interpolated points that have been created. Obviously, the more points we have, the greater error we are likely to have. Again, evolution is initially very fast and the best weight falls to about

300. A flat phase is encountered, broken only by the emergence of a noticeably fitter candidate at about 1420 iterations.

(c) After 2000 iterations, POISGEN generated the candidate solution shown in figure (6.16). This was then supplied to the NAG routine as an approximate initial guess. This 'guess' was good enough for the NAG routine to proceed, the result of a which is shown in figure 6.17.

However, when this answer (figure 6.17) is resubmitted to the genetic algorithm we find that it has a large enough non-zero weight associated with it for us to state that this is not the final answer. POISGEN generates an initial approximate answer of typical weight 160 (see figure 6.19) for 11×11 grid points. When NAG returns an answer having been given the genetic answer as an initial guess, the weight as measured by POISGEN (keeping the weights constant - see section B.1) is typically less by about 20. Therefore, the NAG routine has improved the solution, but not by very much. Although NAG appears to successfully complete the problem, this cannot be so, because if it did, the weight assigned to it by POISGEN would be substantially lower than the original genetic guess.

A more detailed examination of the NAG answer is required. If one examines the residuals calculated by the NAG routine in the problem, then one finds an oscillatory behaviour. Typically, the NAG routine initially generates residuals of the order $0.1 \rightarrow 10$, which fails the 'successful exit' criteria. However, as the iterations continue, the error measures in both the residuals and maximum change drop dramatically to below the exit criteria set by *conres* and *conchn*. This is what triggers the apparently successful determination of an answer. As the routine progresses, the convergence measures increase again to values greater than 10^{50}. From now on the routine oscillates between satisfying the convergence criteria and grossly exceeding them.

It is evident that this is a very difficult problem, and that the NAG routine *does* improve the solution, but not by as much as expected.

6.7 Further comments on the genetic solution of Equation (6.25)

Figure (6.18) reveals how much different the genetic and NAG derived answers are. There are very large errors present, yet they were small enough to permit NAG to continue. This indicates that the solution set $A_{i,j}^{best,genetic}$ is 'close' enough to a solution of the nonlinear equations represented by (6.25) to permit standard numerical techniques to continue. Therefore there is no question that the genetic algorithm is generating spurious answers - if one accepts that an answer to (6.25)

can be reasonably represented by the point to point nature of a finite difference technique. The genetic algorithm is merely solving a set of nonlinear equations arising from the finite difference representation of (6.25). Therefore, the same caveat applies to these solutions as applies to all finite difference answers.

This effect is amply demonstrated by the results described in figures (6.20), (6.21), (6.22) and (6.23). These were generated in attempt to solve (6.25) again, but this time with a different range of permitted A values, as governed by the program variables $lowbdry, highbdry$ (see appendices A,B for more detail). With $lowbdry = -5, highbdry = 5$, POISGEN generated (6.20).

Although very similar to the earlier solution there are a number of differences. Firstly, the range of gene values is much less than in (6.16): for instance, there are regions where $A > 4$ in figure (6.16), whereas no such values exist in (6.20). At first sight, these are truly different solutions to the same equation.

This may be possible, as we are attempting to solve a nonlinear problem: it seems that by reducing the range of values to search through we appear to have found another set of numerical values that solve the nonlinear problem set, both of which were unobtainable by NAG alone. Figures (6.23) and (6.22) also show the difference between this program run and the previous one. The best weight drops quickly to lower values which, and on point resolution doubling, the best weight jumps by only one order of magnitude. This may be attributed to the lower un-doubled weight, coupled with the lower range of gene values. Evolution is then remarkably steady, showing that the genetic algorithm is steadily improving the candidate. The error values in (6.22) are also far lower than in (6.18). This indicates that the genetic solution represented by figure 6.20 is in fact closer to the final NAG answer than that obtained previously with $lowbdry = -highbdry = -10$.

The same answer was produced when each of these candidates were fed to the NAG routine D03EBF. The final answer is shown in figure 6.21. This is a rather important feature of this genetic algorithm: it seems to find an approximate answer almost independent of the range of gene values permitted.

This leads to a very interesting question in this particular application of genetic algorithms. The original run had $lowbdry = -highdry = -10$ and therefore included the possibility of determining the genes of the second run with $lowbdry = -highdry = -5$. If this is so, why did the algorithm choose one and not the other? It may be that the original population was biased in such a way to prefer the creation of higher valued solutions and so the genetic algorithm would then enhance this bias. Or it may be that the higher value solution is somehow 'preferred' over the other, and that unless the range is restricted, one will never see alternate solutions. This means that the higher value solution is acting as some form of attractor for the algorithm.

Whether the solution finally generated has anything to do with what is meant physically by A is another question, and is one that will not be addressed here. That the POISGEN solution

was good enough to permit NAG to continue vindicates the application of genetic algorithms to partial differential equations. Although the NAG routine is much faster in finding an answer, its performance is dramatically disabled by a poor initial guess, or by a difficult equation. We can use POISGEN to home in on an answer, and then let NAG polish it off.

6.8 Summary and Conclusions

In chapters 5 and 6 we have developed genetic algorithmic solutions to two commonly occurring differential equations based on the techniques originally implemented in the FORTRAN77 code GENODE. We have shown that it is feasible to use a floating point genotypic representation, coupled with an expanded range of possible genetic operators to solve Poisson's equation. The resulting program POISGEN can be used as a stand alone solver, or as an initial guess provider for more accurate routines. It was found that POISGEN generated answers that were consistent with other methods, although not as accurate. It was also found that in one particular case (equation (6.25)), no accurate answer could be generated by either POISGEN or NAG D03EBF. The NAG routine did manage to improve the genetic answer in this case, suggesting that a solution to the finite difference equations representing (6.25) does exist.

The major benefit inherent in this approach is its independence from matrix manipulations and/or geometric techniques. These methods can give rise to unrealistic solutions due to numerical instability. Genetic algorithms offer the possibility of increased stability, but at the expense of accuracy and time. Badly fitting answers are simply thrown away; their poor genes are not allowed to propagate into further iterations. In more traditional methods, numerical errors can propagate disastrously into higher itearions, creating spurious answers.

The time taken to generate a 'reasonable' answer is the biggest factor weighing against the application of genetic algorithm's to differential equation's. This was especially noticeable in the re-design of ODE to create POISGEN. POISGEN had to be run on SUN/SPARC workstations in order to generate answers in a reasonable time, whereas both ODE and GENODE may confortably run on PC's. This was largely due to the much increased length of genotype and to the more sophisticated convergence strategies used.

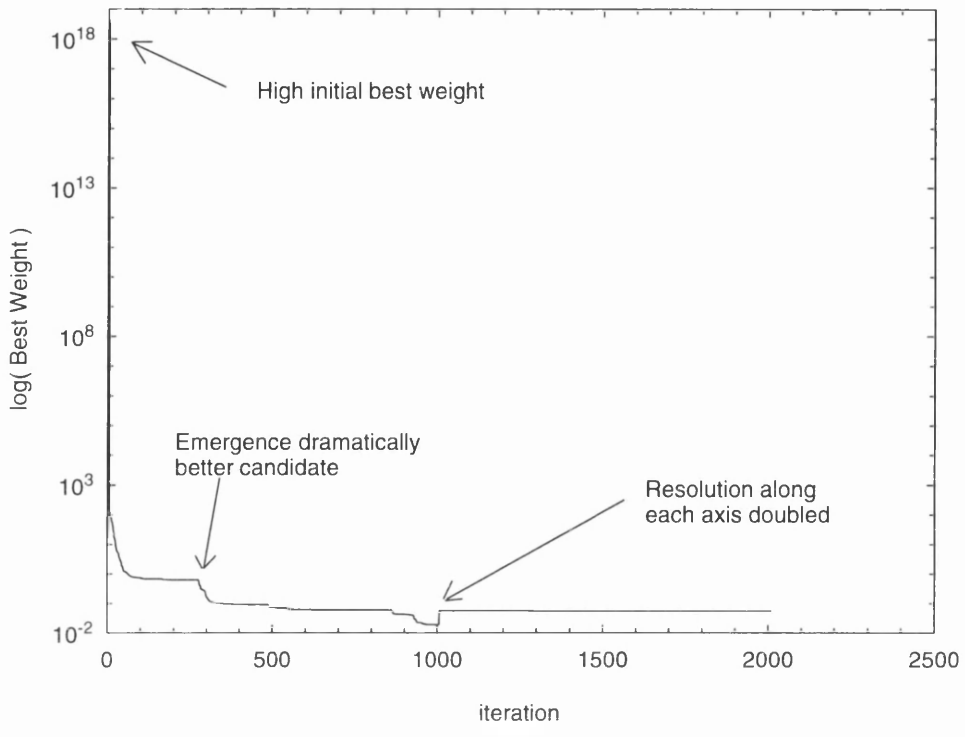

Figure 6.6: Evolutionary history of fig.(6.3)

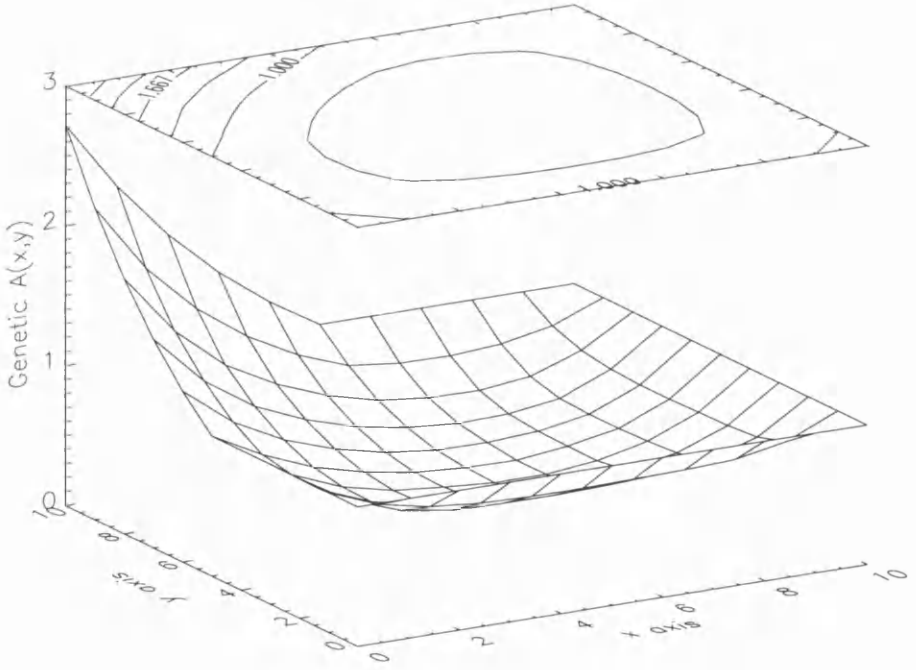

Figure 6.7: Genetic solution to eqn.(6.22)

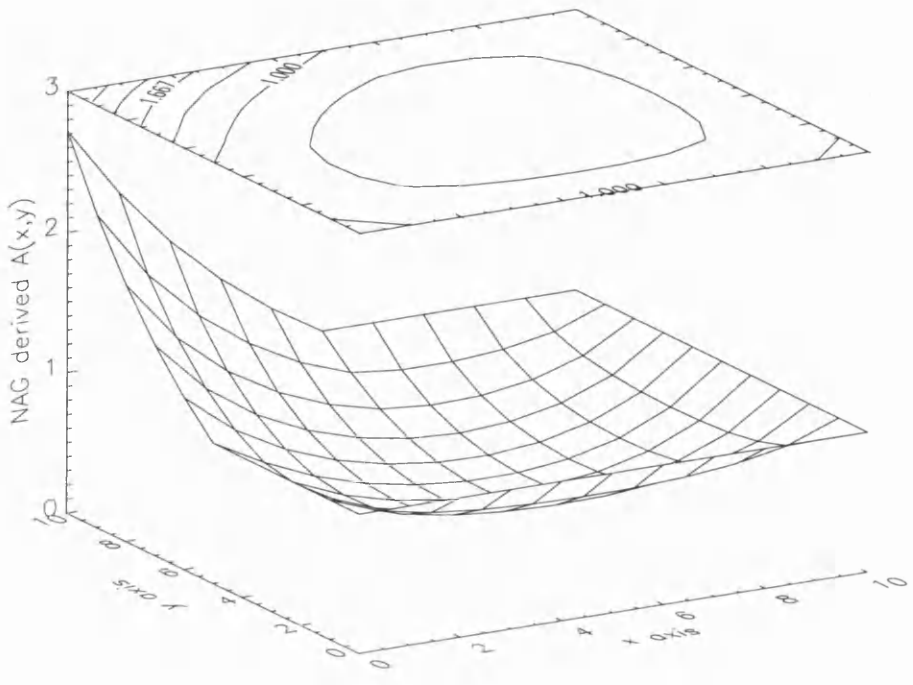

Figure 6.8: NAG routine solution to eqn.(6.22)

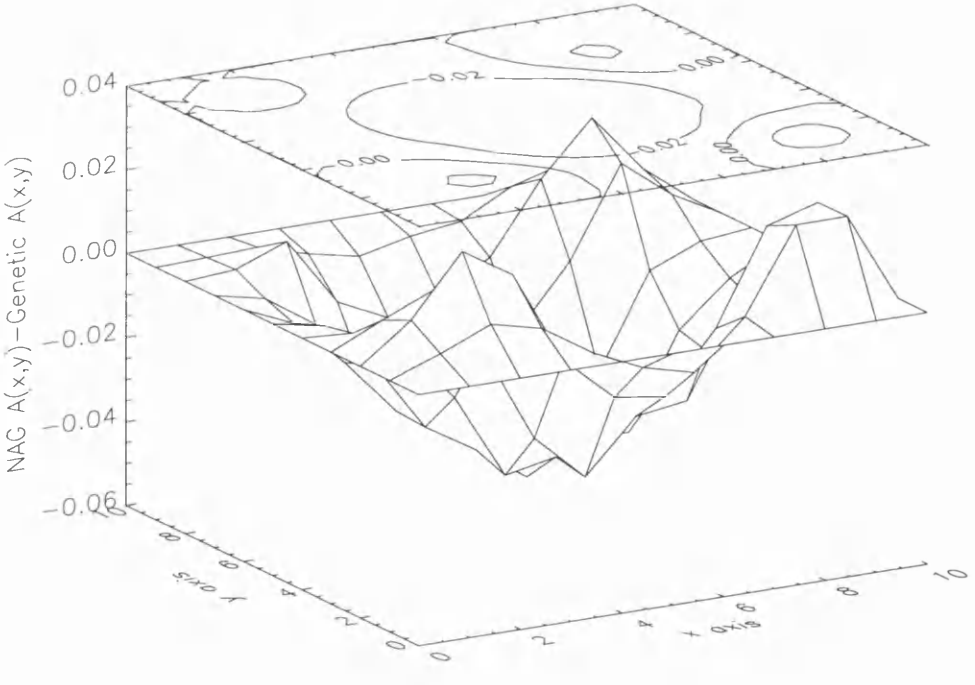

Figure 6.9: Difference between fig.(6.7) and fig.(6.8)

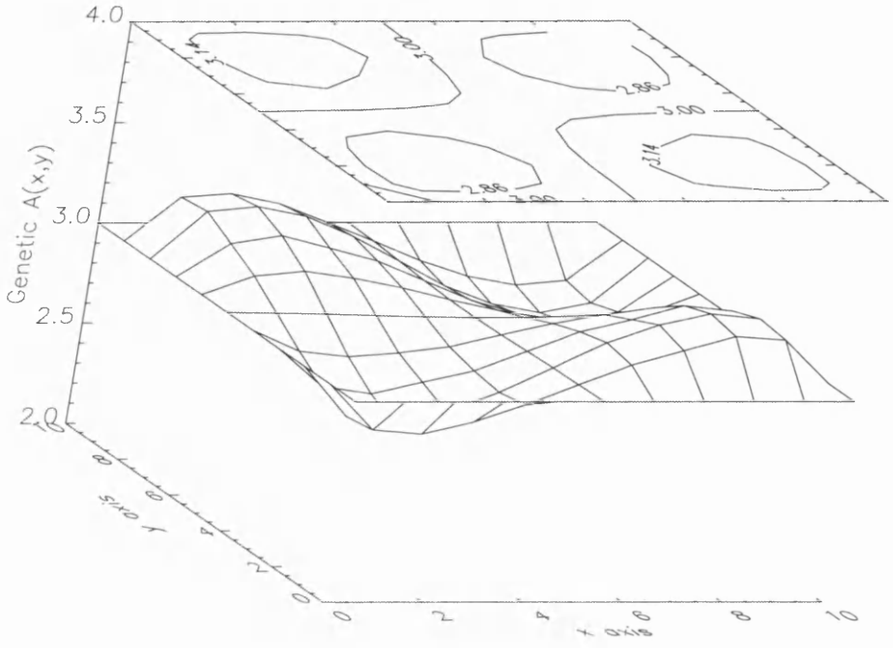

Figure 6.10: Genetic solution to eqn.(6.23)

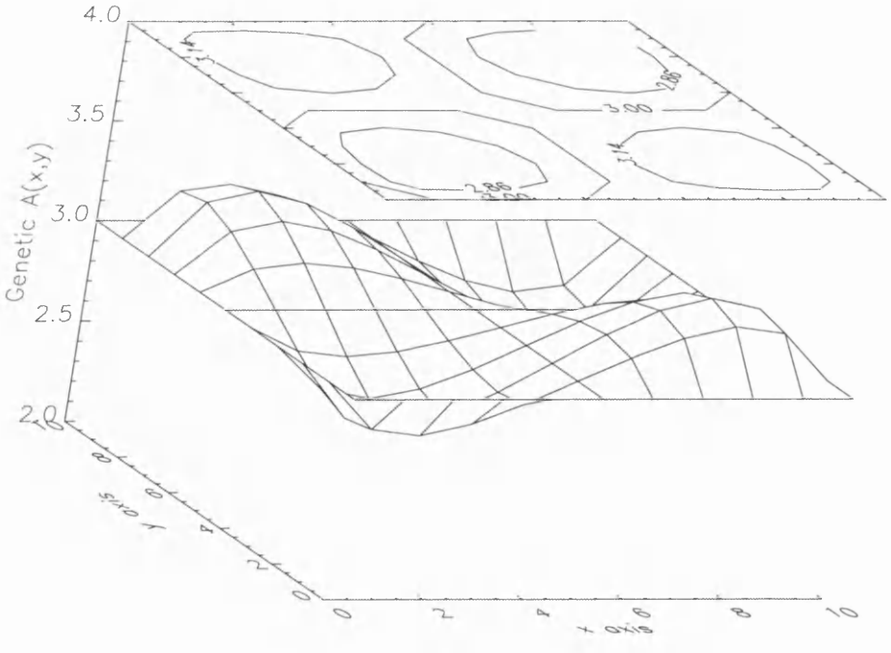

Figure 6.11: NAG routine solution to eqn.(6.23)

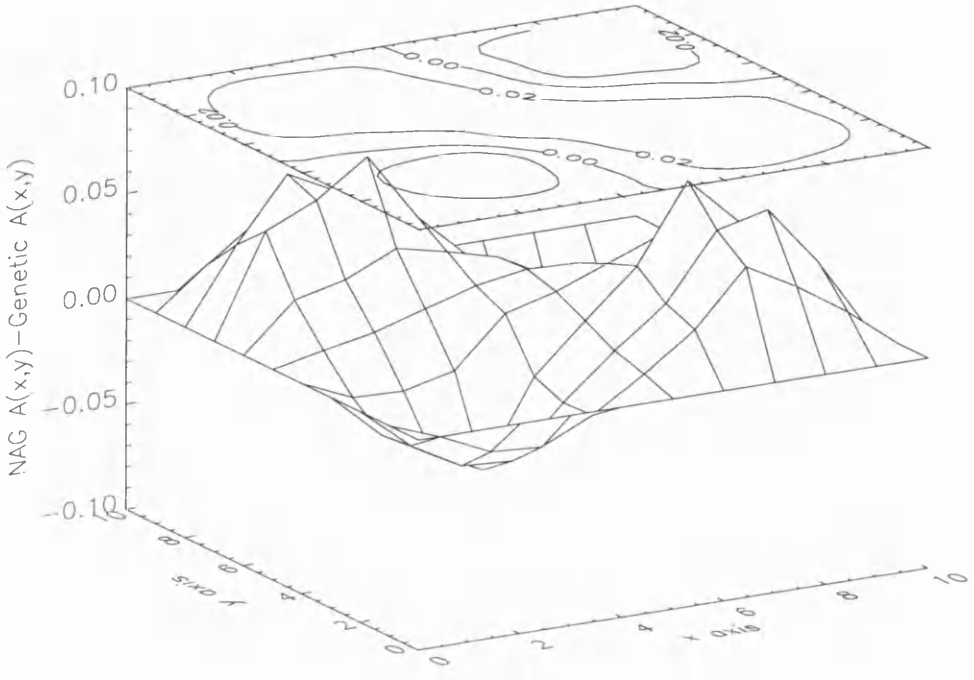

Figure 6.12: Difference between fig.(6.10) and (6.11)

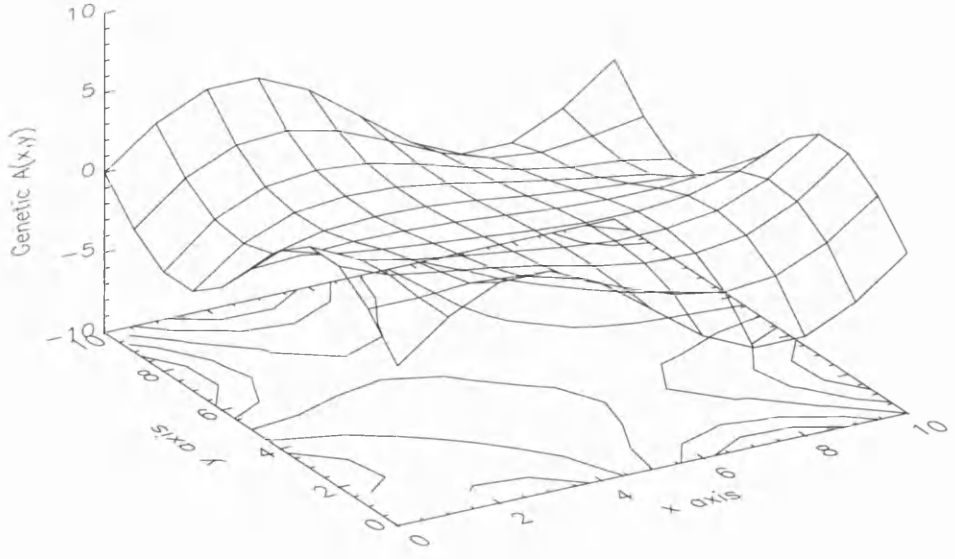

Figure 6.13: Genetic solution to eqn.(6.24)

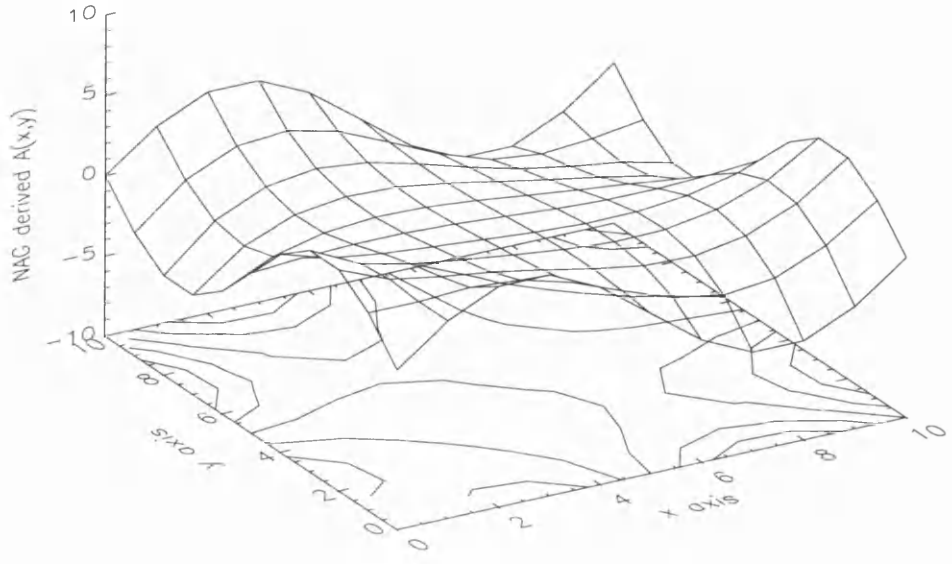

Figure 6.14: NAG routine solution to eqn.(6.24)

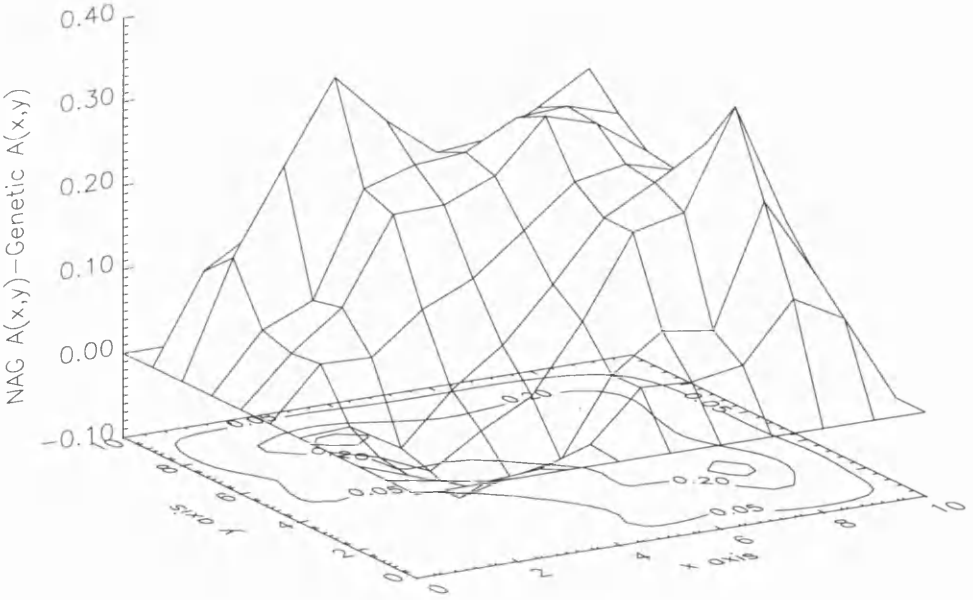

Figure 6.15: Difference between fig.(6.13) and fig.(6.14)

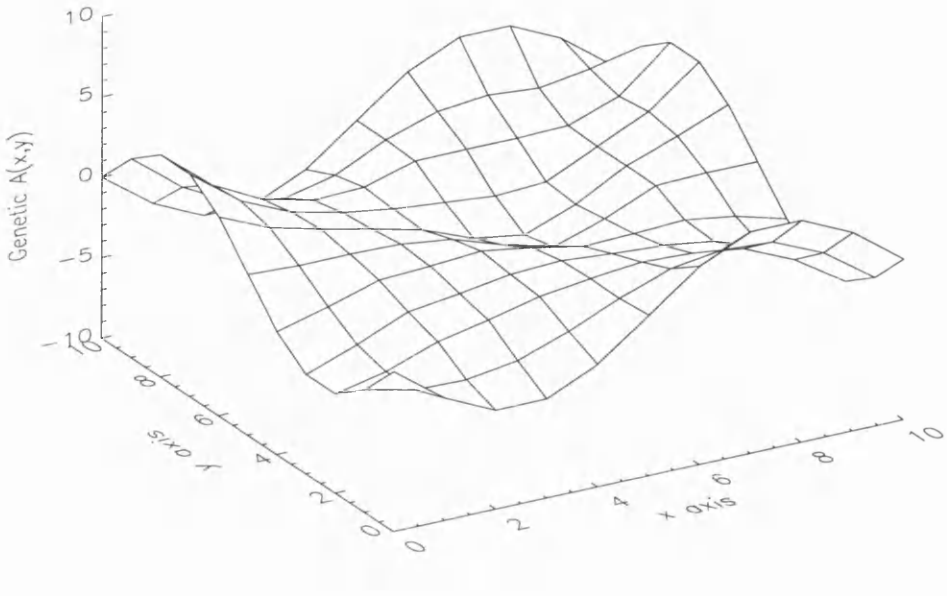

Figure 6.16: Genetic solution to eqn.(6.25)

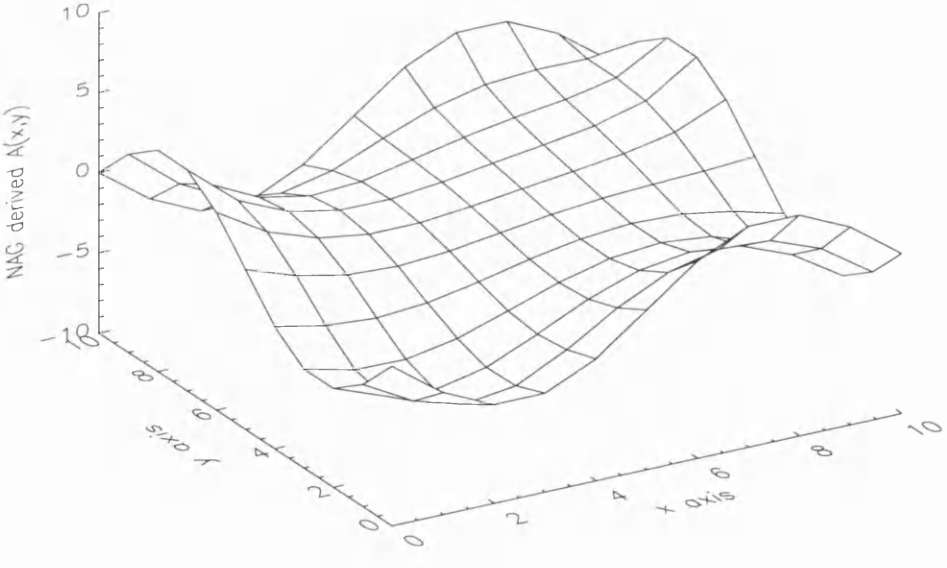

Figure 6.17: NAG routine solution to eqn.(6.25)

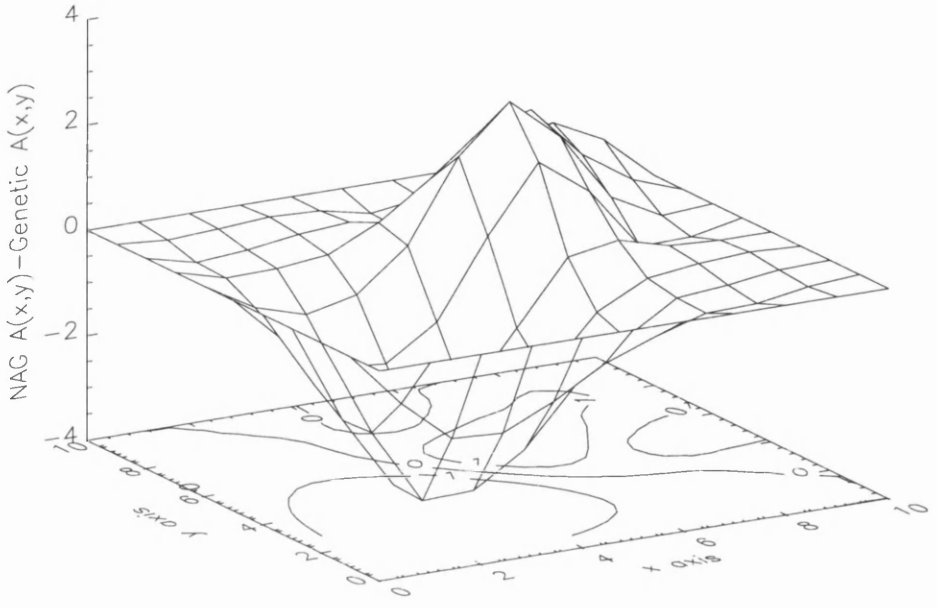

Figure 6.18: Difference between fig.(6.16) and fig.(6.17)

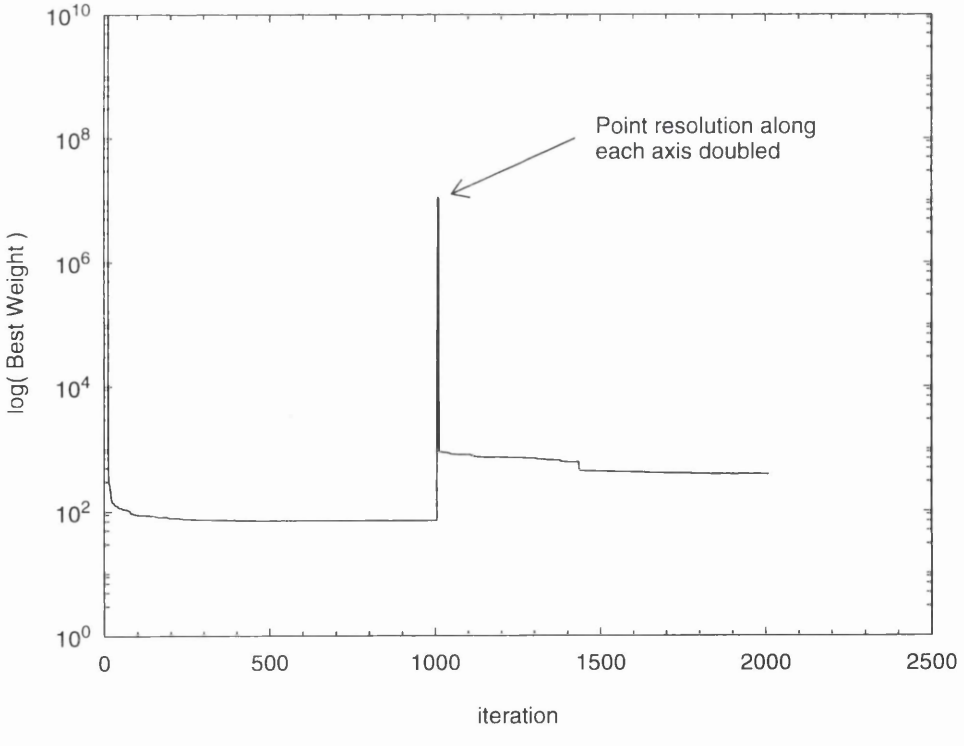

Figure 6.19: Evolutionary history of figure (6.16)

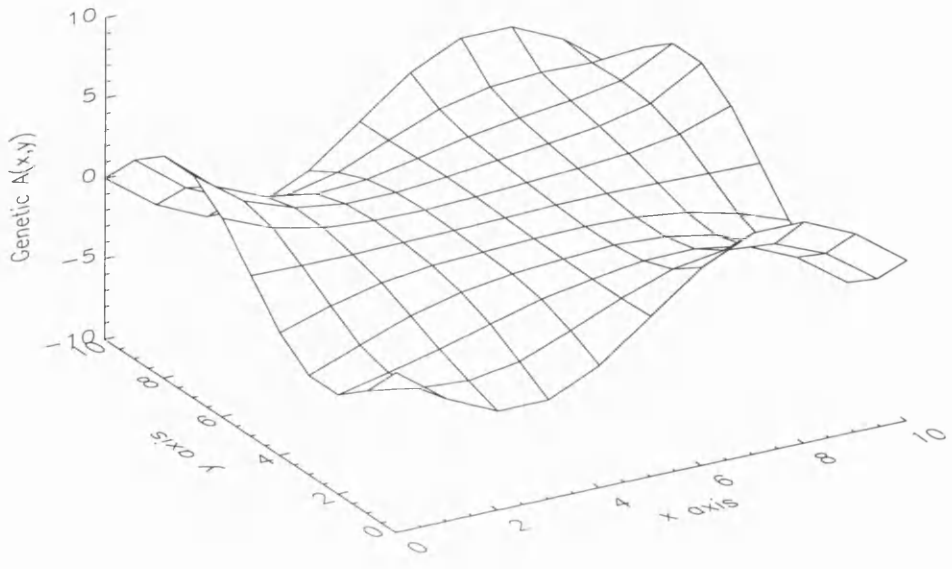

Figure 6.20: Genetic solution to eqn.(6.25) with different range of gene values $lowbdry = -highbdry = -5$

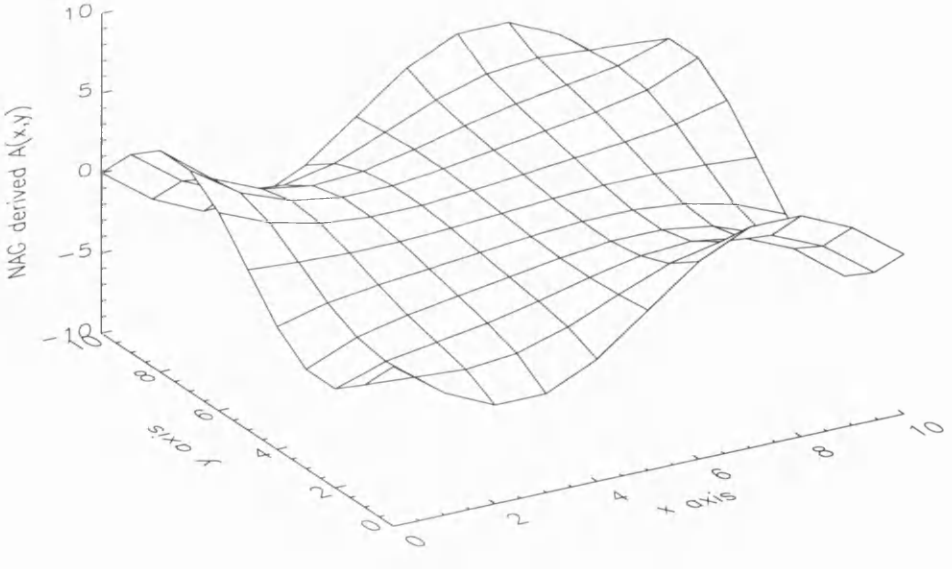

Figure 6.21: NAG routine solution to eqn.(6.25) with different range of gene values $lowbdry = -highbdry = -5$

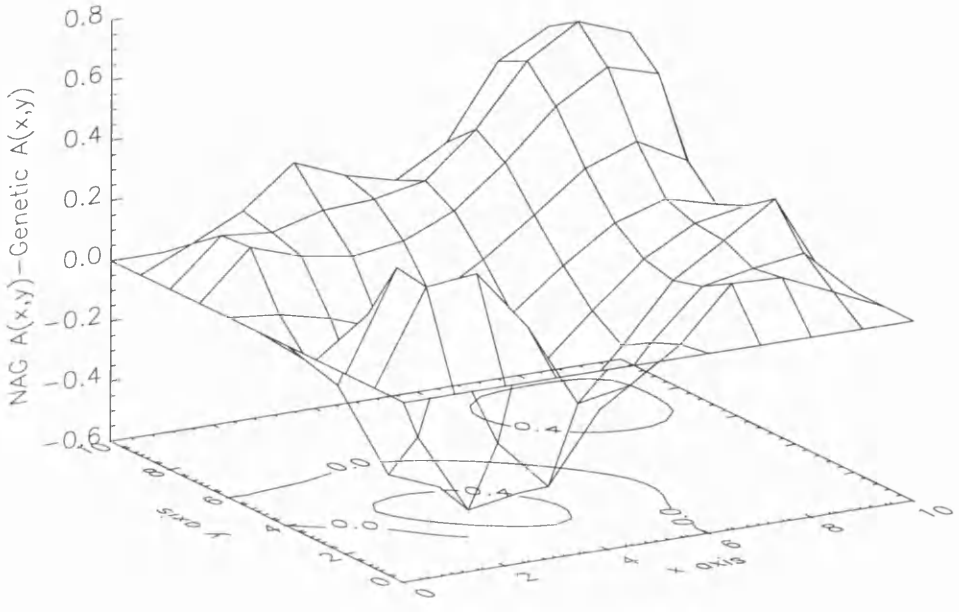

Figure 6.22: Difference between fig.(6.20) and fig.(6.21)

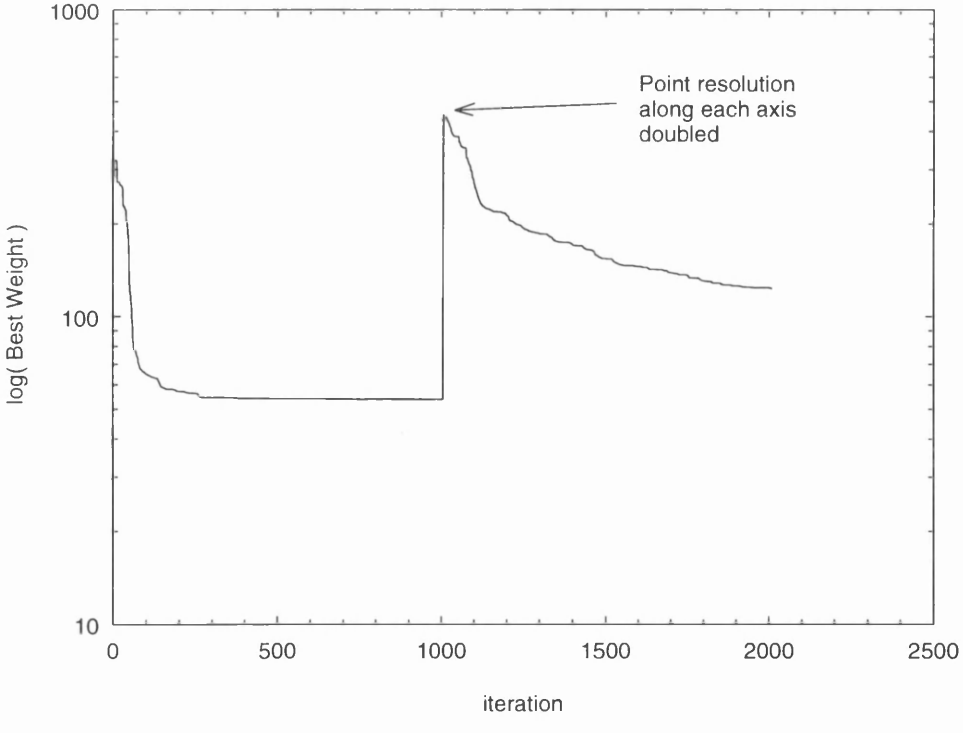

Figure 6.23: Evolutionary history of fig.(6.20)

Chapter 7

Future Work

The work of this thesis splits into two sections, genetic algorithms and field aligned flow. Genetic algorithmic techniques for the solution of differential equations were developed out of a need to generate better initial guesses to solve an equation in field aligned flow, aside from their intrinsic interest. As this application of genetic algorithm's is relatively new, there are several interesting routes down which research could go. Also, both time dependent and time independent field aligned flow appear to offer some unusual features (a possible acceleration mechanism and unusual field topologies, respectively) that are worthy of further study.

There are a number of ideas that may be explored in relation to the work already carried out. Some of them are listed below, in various states of genesis.

7.1 Field Aligned Flow

7.1.1 Computational Solutions

Computational methods have not been applied in this thesis in great detail, aside from the development of genetic algorithms. An outstanding problem that may be only accessible via computation is a solution to equation (4.32). This would be a valuable addition to our knowledge of time dependent field aligned flow. However, this is an example of one the trickiest type of equations to solve [7]: a nonlinear hyperbolic partial differential equation. Stability is a problem with such equations and care will be needed to ensure that reasonable solutions are generated. Fortunately, the special case of $\beta(x) = constant$ can be used as a guide. It is difficult to predict what the solution will look like: β is a space dependent time scale that also includes the 'amplitude' of the solution. This suggests that the wave will evolve in a more complicated fashion with time. It is unclear what this will do to the presence of the singularity currently observed in the analytic case.

More generally, a simulation of the 2-dimensional magnetohydrodynamic plasma described in

section 2.4 would give important insight into both the situations we have considered analytically and to more realistic geometries, such as jets and magnetotails. As has been seen, the geometry plays a vital part in exact field aligned flow, and it would be interesting to model, say an arch of plasma in cylindrical polar co-ordinates with field aligned flow. Again, this may be the only way forward to solve (2.38).

7.1.2 Analytical Work

In this thesis we have in the main, considered only exact field aligned flow: no deviation is permitted and hence by the induction equation the magnetic field may not evolve in time. Obviously, this is quite a major restriction if we want to apply field aligned flow to a real situation, as it is very unlikely that a plasma would be purely field aligned. Hence it is natural to consider flows that are predominantly field aligned, but not exactly. Any non field aligned component means that the total field must evolve in time. This will add another, convective time scale to the problem. It may be that certain field aligned configurations can persist with a small non-parallel component. Questions like this could be answered by a perturbation analysis combined with a linearisation procedure.

The question of stability is an important one, particularly in connection with the singularity described in chapter 4. It seems intuitively reasonable to suggest that the plasma would not suffer the creation of such a feature, and that some other process would take over to mitigate this singularity. It seems most reasonable to tackle this problem by including a new term in the model. Following previous work, and guided by physical processes, consideration of a compressible plasma would be the next step. This may allow the material to pile up at a (moving?) point in the fluid, removing the infinite velocity. Alternatively, perhaps the inclusion of another time or space scale - via magnetic diffusitivity or fluid viscosity - would again give the plasma an avenue of escape, rather than becoming singular. The prospect for analytical progress looks best if we consider these effects to be small perturbations to the system at present. The inclusion of resistive effects in a field aligned system would have an important tie-in with reconnection work (both analytic and computational) and the concept of negative inertia (see sections 2.3.2 and 2.3.3).

Recently, we have found that some field aligned flows can generate nonlinear Alfvén waves, propagating in the r, θ plane, but with field components perpendicular to it.

We start with the basic equation set describing the plasma, equations (2.20)→ (2.23). In all previous work, we confined attention to the r, θ plane. Suppose now we describe the system by setting:

$$\mathbf{B} \rightarrow \mathbf{B} + \mathbf{b}, \quad \mathbf{U} \rightarrow \mathbf{U} + \mathbf{w},$$

The new components to the magnetic field and velocity fields are $\mathbf{w} = w\hat{\mathbf{z}}$ and $\mathbf{b} = b\hat{\mathbf{z}}$ respectively. $\mathbf{B}$ and $\mathbf{U}$ are the magnetic and velocity fields in the plane. Incompressibility demands that w is

a function of r and θ only; similarly the Maxwell equations demand that $b = b(r, \theta)$. It turns out that the resulting equations may be split up into two parts: a set describing the behaviour in the plane

$$\frac{\partial \mathbf{B}}{\partial t} + (\mathbf{U}.\nabla)\mathbf{B} = (\mathbf{B}.\nabla)\mathbf{U} \tag{7.1}$$

$$\frac{\partial \mathbf{U}}{\partial t} + (\mathbf{U}.\nabla)\mathbf{U} = -\nabla p + \mathbf{J} \times \mathbf{B} - \mathbf{j} \times \mathbf{b} \tag{7.2}$$

and another describing the z-direction

$$\frac{\partial \mathbf{b}}{\partial t} + (\mathbf{U}.\nabla)\mathbf{b} = (\mathbf{B}.\nabla)\mathbf{w} \tag{7.3}$$

$$\frac{\partial \mathbf{w}}{\partial t} + (\mathbf{U}.\nabla)\mathbf{w} = \mathbf{j} \times \mathbf{B} \tag{7.4}$$

where $\mathbf{j} = \nabla \times \mathbf{b}$ and $\mathbf{J} = \nabla \times \mathbf{B}$ and we have set $\rho = 1$ and $\mu_0 = 1$. These equations are, at the moment, very general. Suppose we are given $\mathbf{B}$ and $\mathbf{U}$, then we can solve for $\mathbf{b}$ and $\mathbf{w}$. Let us assume that

$$\mathbf{w} = \frac{\mathbf{b}}{\sqrt{\mu_0 \rho}}$$

which is true for an Alfvén wave. Since $\mathbf{j} \times \mathbf{B} = (\mathbf{B}.\nabla)\mathbf{b}$, equations (7.3) and (7.4) become identical and equal

$$\left[\frac{\partial}{\partial t} + \mathbf{U}.\nabla\right] \mathbf{w} = (\mathbf{B}.\nabla)\, \mathbf{w} \tag{7.5}$$

a linear hyperbolic equation for w. If we impose the field aligned flow condition (equation (2.35)) then the equation takes on a form

$$\frac{\partial w}{\partial t} - (1 - f)\left[B_r \frac{\partial w}{\partial r} + \frac{B_\theta}{r}\frac{\partial w}{\partial \theta}\right] = 0 \tag{7.6}$$

We have re-expressed (7.5) as a scalar equation. Although it is a linear equation (soluble by the method of characteristics [46]), the waves it describes are not 'linearised' because at no stage in the analysis have we specified any particular size to w. The system supports these nonlinear Alfvén waves [34]. We have yet to include the effect of the 'feedback' term $\mathbf{j} \times \mathbf{b}$ in equations (7.1) and (7.2). This will change the values of $\mathbf{U}$ and $\mathbf{B}$ in the plane which will in turn change $\mathbf{w}$.

The appearance of these waves and their subsequent effects for the system are certainly worthy of further investigation.

7.1.3 Modelling

We have only scratched the surface in applying field aligned flow to feasible, physical geometries. As has been noted above, curved geometries may provide regions of interest for solar physics. This may be best tackled computationally. Even within the confines of the Poisson equation developed in chapter (6), there is room for manoeuvre. One area in need of improvement is the boundary description. It may be more realistic to describe the region with Robbins type boundary

information. On the upper and lower sides of the regions of interest we could set $\frac{\partial A}{\partial x} = 0$, which would kill off the y-component of $\mathbf{B}$, reproducing the y-directed components without fixing a magnitude to A at the boundary.

Solutions involving f nonconstant are harder to compute, but may be more realistic from a modelling point of view. An interesting route to explore would be solutions to the fields allowed by $f = f(t)$. As $t \to \infty$, $f \to \pm 1$. At $f = \pm 1$ exactly, the original equations state that any choice of $\mathbf{B}$ is permitted. This is not true when $f \to \pm 1$, as the time dependent flow function introduces a condition of the magnetic field (see section 4.2.3). Therefore, there is something particular about these fields: in the limit, they are just like another with $f = \pm 1$, but they have arisen from considering a more complicated problem that tends to an easier one. It may be possible to first write a genetic algorithm to obtain some feel for the analytic solution of the field governing equation (4.7), and then let more accurate methods take over.

7.2 Genetic Algorithms

It has been shown that genetic algorithms may be useful in generating approximate numerical solutions to traditionally difficult Poisson equations. These rough solutions are generally good enough to allow standard numerical techniques to calculate an accurate answer. There are very many directions that may be taken to extend and improve the application of genetic algorithms to differential equations, and some are listed below.

7.2.1 Extensions to POISGEN

There are a number of ways in which we can extend the existing code.

1. Meta-Genetic Algorithms

   POISGEN is an example of a complex, multi-parameter system (see appendix A,B) with a not easily described behaviour. The optimisation of its behaviour could be performed by using a *meta*-genetic algorithm to find an optimised set of parameters for use in POISGEN. With a genotype consisting of POISGEN parameters and each genotype being used to run a copy of POISGEN weights would be assigned according to how well each copy reproduced the solution (whether analytic or NAG defined) equation. Such a genetic algorithm would be computationally expensive to run, but may be worthwhile to obtain better performance over a wide range of problems.

2. One obvious extension is to rewrite the code to solve Poisson equations on differently shaped regions in different co-ordinates systems i.e., cylindrical polars.

3. POISGEN solves the Poisson equation for Dirichlet boundary conditions (6.2). One could extend the problem to Robbin's boudary conditions,

$$\alpha(x,y)\,A+\beta(x,y,)\,\frac{\partial A}{\partial n}=\gamma(x,y) \tag{7.7}$$

on the boundary ∂R of a region R. Here $\frac{\partial}{\partial n}$ refers to differentiation along the normal directed away from the interior of R.

7.2.2 Alternate Solution Representations

In both ODE and POISGEN we have used a finite difference expression of the relevant equation to generate an approximate solution. This was done as it allows singularities to develop in a solution, should any be present. Therefore, this representation is the most general. However, it is not the only way we can describe a solution.

A function can also be described as a weighted sum of orthogonal functions, over the desired range. To discuss this implementation properly we must introduce some definitions [47]. The *inner product* of two functions $\psi(x)$ and $\phi(x)$, bounded and integrable over the range $a \leq x \leq b$ is defined as

$$(\phi,\psi)=\int_a^b \rho(x)\,\phi(x)\,\psi(x)\,dx$$

with respect to a weight function $\rho(x)>0$ in $a<x<b$. Commonly, $\rho(x)=1$, but this is not necessarily true; for instance, Chebyshev polynomials of the first kind have $\rho(x)=\left(1-x^2\right)^{-1/2}$. The inner product is clearly symmetric, i.e., $(\phi,\psi)=(\psi,\phi)$. Two functions are said to be orthogonal on the interval $a \leq x \leq b$ with respect to the weight function $\rho(x)$ if

$$(\phi,\psi)=0$$

A set of functions ϕ_k, $k=1,2,\ldots$ form an orthogonal set if

$$(\phi_k,\phi_j)=0, \quad k\neq j \tag{7.8}$$

The norm of a function ϕ is defined by

$$\parallel \phi \parallel = \sqrt{(\phi,\phi)}$$

If $\parallel \phi \parallel < \infty$ then ϕ is said to be square integrable. Also, ϕ may be normalised by defining a new function $\hat{\phi}=\phi/\parallel \phi \parallel$.

Given an orthonormal set of square integrable functions ϕ_k and a square integrable function $\psi(x)$ over $a<x<b$ then the numbers (ψ,ϕ_k) may be called the Fourier coefficients of $\psi(x)$. The formal series

$$\psi(x)=\sum_{k=1}^{\infty}(\psi,\phi_k)\,\phi_k \tag{7.9}$$

is called the Fourier series of ψ. Commonly, Fourier series refer to an expansion in trigonometric functions, but we can generalise this meaning to the expansion of a function in terms of a set of orthonormal functions.

We can of course apply this to the solution of ordinary differential equations, and from there, design a suitable genotype for a genetic algorithm to work with. Consider the ordinary differential equation boundary value problem we used in the code ODE (see section 5.3). We can substitute the expansion (7.9) into (5.2). Writing $A_k = (\phi_k, y_{solution})$ we get

$$\sum_{k=1}^{\infty} A_k \phi_k'' + C_1(x) \sum_{k=1}^{\infty} A_k \phi_k' + C_2(x) \sum_{k=1}^{\infty} A_k \phi_k + \left[C_3(x) \sum_{k=1}^{\infty} A_k \phi_k \right]^2 = C_4(x)$$

The unknowns are the numerical values of A_k, the Fourier coefficients, and it is this that forms the genes for the genotype. Obviously, the genetic algorithm cannot use an infinite number of genes, so we truncate the series at some upper limit n, using

$$A_1 A_2 A_3 \cdots\cdots A_n$$

as the genetic code for each individual. Choosing Fourier coefficients as genes differentiates between genotype and phenotype, a distinction blurred in most of this thesis. The translation between the two is encapsulated in the Fourier decomposition used to encode y as a series of orthonormal functions.

We can generate a local error at each point x_i in the range for the p'th candidate

$$R_i^p = \sum_{k=1}^{n} A_k^p \chi_k(x_i) + \left[C_3(x_i) \sum_{k=1}^{n} A_k^p \phi_k(x_i) \right]^2 - C_4(x_i) \tag{7.10}$$

where $\chi_k(x_i) = \phi_k''(x_i) + C_1(x_i)\phi_k'(x_i) + C_2(x_i)\phi_k(x_i)$. A weight can be assigned to each candidate in a very similar way to section (5.3.2) by generating separate measures of fitness and combining them: we can keep the sum, product and maximum error measures as defined above, but we must treat the boundary conditions differently. From (5.3), $y(x_0) = y_0, y(x_{end}) = y_{end}$. At the left hand boundary, for instance, we must have $y_0 = \sum_{k=1}^{n} A_k \phi_k(x_0)$, and similarly for the other boundary. Therefore, for genes in the p'th candidate we should define

$$\epsilon_3^p = \text{ function of } \left| y_0 - \sum_{k=1}^{n} A_k^p \phi_k(x_0) \right|, \quad \left| y_{end} - \sum_{k=1}^{n} A_k^p \phi_k(x_{end}) \right| \tag{7.11}$$

For a suitable function, this should penalise candidates discontinuous with the boundary. A weight W^p can de defined, completing the required elements for a genetic algorithm.

There are disadvantages with this representation. It is not quite as general as the previous scheme, as it may have trouble trying to describe a single (non-infinite) spike in the solution. Also, it may be that the most important component in the expansion is at high n and the genotype may never be long enough to contain it. This is tempered by the analytic result that $(\phi, \psi) \to 0$

as $k \rightarrow \infty$. and hence higher components provide smaller contributions to the overall expanded series. Also, since the proposed scheme does not explicitly write the boundary conditions into the genotype, then we can expect the fitting errors generated here to be somewhat more important than in the finite difference scheme. Again, this effect can be mitigated by making the penalties associated with ϵ_3 as defined above very high, choosing those parents that best fit the boundary conditions.

However, it is not all bad news. A major advantage should be the global nature of the orthonormal functions, compared to the necessarily 'local' character of the discretisation approach. Each of the orthonormal functions span the entire range of equation dependence, and hence the corresponding Fourier coefficient approximates the solution everywhere. The discretisation procedure is purely local which can cause problems when generating a fitting error at each point. This can be explained by *epistasis* [48] , which is the effect of one gene on the expression (in the phenotype) of another. This effect can be particularly severe in both ODE and POISGEN and can fool the routine into propagating poor genes. This is because both codes rely on next nearest neighbour interaction to assess fitting errors. If by chance, the i'th gene has two neighbours that happen to give a low value of R_i^p even although when compared to the true solution they are very poor, then gene i in the p'th candidate will be judged to be 'good' and will therefore have an improved chance of surviving to the next generation. The finite difference genetic algorithm technique relies on the efficient propagation of boundary information into the genotype to compensate for this effect.

There should be no interaction of this type between neighbouring genes in this new scheme, guaranteed by the orthogonality of the expansion functions. There are also a large number of orthogonal function sets to choose from, which could make a big difference to the length of the genotype and therefore convergence times. Such a program could also be easily modified to decompose, say, experimentally derived data into a Fourier series of trigonometric functions.

Extending into a Poisson type problem using a suitable Fourier decomposition would follow much the same lines. It is envisaged that these programs would be used with other routines, either as decomposition programs or straightforward equation solvers.

7.2.3 Other Differential Equations

Of the three types of partial differential equation, we have tackled only the elliptic type. Parabolic and hyperbolic equations are fundamentally different equations.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \tag{7.12}$$

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \tag{7.13}$$

either as initial value problems, or as boundary value problems. A possible hyperbolic scheme involves generating an implicit finite difference scheme for the next time step, using the previous time levels. The coefficients would depend on the values in the previous time level. One could use a genetic algorithm to determine a rough solution, and then let a traditional solver finish the job. This hybrid technique would combine the best of both worlds - an accurate solution to the equations representing the partial differential equation in a reasonable time, free from gross numerical instability.

Another, perhaps even more important class of problems that we can examine genetically is the system

$$\frac{dy_i}{dx} = f_i\left(x, y_j\right), \quad 1 \leq i, j \leq n \tag{7.14}$$

a set of $n \geq 2$ ordinary differential equations in the independent variable x. A study of this system would also aid the study of hyperbolic/parabolic equations, and would yield important information on the effect of epistasis, which was cited as a major concern above.

The solution y_i to each equation depends on the other equations, which is very important from a genetic point of view. The problem of epistasis now becomes much worse as genes (points) from different genotypes must necessarily affect the viability of candidates. This is like saying that your health depends on the health of your neighbour. This problem also raises some difficult questions on how to view the set i.e. is it best approached as n separate but interacting populations, the best individuals from each representing the best overall solution. Or perhaps the best method is to simply concatenate the points required in each equation and call this the genotypic representation.

It is however, a very open question at present: the only way to find for sure is to try it out!

Appendix A

POISGEN: listing and documentation

A.1 Outline of the code

A.1.1 Developmental History

POISGEN is a FORTRAN 77 code designed to solve Poisson equations by genetic algorithms on rectangular regions using a Cartesian co-ordinate system. It is largely based on a previously written code, an ordinary differential equation solver, ODE, which was itself inspired by [38]. The method is explained in greater detail in chapter 5. It was developed initially on a PC running SALFORD FTN77, and then refined on SUN/SPARC workstations. It is however, still very much in its infancy, and should be regarded as an research tool to test the feasibility of the application of genetic algorithms to differential equations. Many of the routines described are experimental, and are included because they seemed like a good idea at the time. The code is not optimised in any particular fashion, but is written in as modular a fashion as possible to (hopefully) allow easy reading and modification. There are also a liberal number of comments sprinkled in the code to aid understanding of its functionality.

The program may be sectioned in three parts. The first part describes the genetic algorithm itself by calling on various routines later in the code. This part controls the behaviour of the entire program. The second, and bulkiest part, implements the required genetic algorithm behaviour: it is here that the actions of mutation, breeding, sorting, etc. are actually performed on the population. The final part describes the equation we are trying to solve, that is, the boundary conditions and source term.

Variable Name(Size), Type	Description
npar, integer	Number of breeding parents per generation
nchild, integer	Number of children bred per generation
xpoint, integer	Number of points in x-direction, including boundaries
ypoint, integer	Number of points in y-direction, including boundaries
npoint, integer	$npoint = (xpoint - 2) \times (ypoint - 2)$ Total number of points(genes) per candidate
$y\,(1000, 1000)$,real array	Genes of the entire population $y\,(n, i)$ holds the i'th point of the n'th candidate
$weight\,(1000)$, real array	$weight\,(i)$ is the weight of the i'th candidate
highbdry, real	maximum permissible gene value
lowbdry, real	minimum permissible gene value

Table A.1: Some POISGEN variables

A.1.2 Important variables

Some of the most important program variables are listed in Table A.1. The time taken per generation varies as *npoint* which in turn depends on $xpoint, ypoint$. These variables govern the overall speed and behaviour of the algorithm. The variables $npar, nchild, xpoint, ypoint$ are user defined and loaded into POISGEN from the input file GA.IN (see appendix B).

The program has a maximum upper capacity, limited by the size of the y array. Currently, the program runs with $npar + nchild \leq 1000$ and $npoint \leq 1000$. This memory limit was set by the PC that POISGEN was originally developed on. Computation time is roughly proportional to $npoint \times (npar + nchild)$. However, a large population of coarsely grained individuals will tend to outweigh the benefits of a population of more finely grained individuals (see comments on point doubling in chapters 5 and 6).

A.2 Description of POISGEN functions and subroutines

Below is a list of the subroutines and functions as they appear in the POISGEN listing. Figure (A.1) denotes the layout of the description. Further functionality details may be found in the listing itself, and in appendix (B).

1. **profile***(number)* *(integer)*

SUBROUTINE NAME*(variables)* *(variable types)*

Description of functionality

FUNCTION NAME*(variables)* **return type**,*(variable types)*

Description of functionality

Figure A.1: Key to function and subroutine descriptions

If *number=-1* then go to **zoomer** subroutine, else assign a weight to candidates numbered $1 \rightarrow number$.

2. **sort**

   sort the array *weight* by heapsort method in descending order: $weight(1) = lowest$ and $weight(npar + nchild) = highest$. Also reorders y array from best (lowest weight) to worst (highest weight).

3. **zoomer***(xwor,ywor)* *(integer,integer)*

   vary the value of $A^1_{xwor,ywor}$, the worst point in the best candidate, in copies of the best candidate using **gcreep.**

4. **gener8***(n)* *(integer)*

   fill $y(n, 1 \rightarrow npoint)$ with a uniformly random distribution of numbers between *lowbdry*, *highbdry*.

5. **continuity***(n)* *(integer)*

   write boundary information into the genotype of candidate n.

6. **special***(n,frac)* *(integer,frac)*

   generate a special individual. Straight lines are constructed between pairs of opposing edge grid points. The special individual is calculated by adding the values of both the x and y directed lines at a specified internal point.

7. **special2***(n,frac)* *(integer,real)*

   add an amount $p \times \frac{highbdry - lowbdry}{frac}$, $-1 \leq p \leq 1$ p random, to all the points in the n'th candidate.

8. **special3***(n,frac)* *(integer,real)*

   add an amount $p \times \frac{high - low}{frac}$, $-1 \leq p \leq 1$ p random, to all the points in the n'th candidate. $high, low$ are the largest and smallest gene values in the n'th candidate.

9. **breed**

   breed two children from two parents by one point crossover

10. **mutate***(nlow,nup,mzlev)* *(integer,integer,integer)*

perform *mzlev* mutations on candidates numbered from *nlow* $\rightarrow$ *nup* in *y*. The types of mutation operators used are decided by the variables loaded in from *mutate.in*.

11. **combine***(nlow,nup,czlev)* *(integer,integer,integer)*

perform *czlev* combinations on candidates numbered from *nlow* $\rightarrow$ *nup* in *y*. The types of combination operators used are decided by the variables loaded in from *combine.in*.

12. **ranrep***(dummy)* **real**,*(real)*

generates a uniformly random number in the range $lowbdry \leq ranrep \leq highbdry$. *dummy* variable to maintain legal FORTRAN.

13. **creep***(gene,czmax,rantype)* **real**,*(real,real,integer)*

add a number $p \times czmax$ to the gene value *gene*. If $rantype = 0$, then $p = \pm 1$ (decided uniformly randomly). If $rantype = 1$, p is uniformly random in the range $-1 \leq p \leq 1$.

14. **gcreep***(gene,gczmax,rantype)* **real**,*(real,real,integer)*

multiply gene value *gene* by $(1 + p \times gczmax)$. If $rantype = 0$, then $p = \pm 1$ (decided uniformly randomly). If $rantype = 1$, p is uniformly random in the range $-1 \leq p \leq 1$.

15. **wav***(gene1,gene2,avzwt1,avzwt2)* **real**,*(real,real,real,real)*

calculate the weighted arithmetic average of two genes *gene*1 and *gene*2. The average is defined as $wav = \frac{gene1.avzwt1 + gene1.avzwt1}{abs(avzwt1)+abs(avzwt2)}$

16. **gav***(gene1,gene2,gzop)* **real**,*(real,real,integer)*

take the geometric average of the gene values *gene*1, *gene*2. The exact mode of operation of this function is governed by the value of *gzop*.

17. **trans***(nlow,nup,tzlev)* *(integer,integer,integer)*

perform *tzlev* transcription operations on candidates numbered from *nlow* $\rightarrow$ *nup* in *y*. The types of transcription operators used are decided by the variables loaded in from *transcr.in*.

18. **swap***(parent,place,noof)* *(integer,integer,integer)*

swap genes from $place \rightarrow place + noof - 1$ with $place - noof \rightarrow place - 1$ in candidate *parent*.

19. **reverse***(parent,place,noof)* *(integer,integer,integer)*

reverse the order of *noof* genes from $place \rightarrow place + noof - 1$ in candidate *parent*.

20. **mix***(parent,place,noof)* *(integer,integer,integer)*

randomly rearrange the order of *noof* genes from $place \rightarrow place + noof - 1$ in candidate *parent*.

21. **copy***(parent,place,noof)* *(integer,integer,integer)*

takes a copy of the genes from $place \rightarrow place + noof - 1$ in candidate *parent* and moves them to the next nonaffected region closest to the centre of the genotype, overwriting the genes already present.

22. **del***(parent,place,noof)* *(integer,integer,integer)*

deletes the genes from $place \rightarrow place + noof - 1$ by copying the segments from the nearest end to $place + noof$. Therefore, one end of the code will have 2 identical copies of *noof* genes in order.

23. **check**

measure the genetic diversity of the breeding population i.e., the top *npar* parents of the ranked population. Genetic Diversity $mtot = \frac{1}{npar} \sum_{n=1}^{npar} 1 - \frac{weight(1)}{weight(n)}$

24. **tweak***(mtot)* *(real)*

keep top *twzpar* parents and fill the rest of the population with completely random individuals.

25. **jiggle***(ran0)* *(real)*

take each gene in the (ranked) candidates numbered $2 \rightarrow npar + nchild$ and randomly decide to perform a **creep** or **gcreep** operation. *ran*0 is either *czmax* or *gczmax*.

26. **wobble***(ran0)* *(real)*

make three copies of the best candidate, but perform a **creep** or **gcreep** on all the points of the genotype. *ran*0 is either *czmax* or *gczmax*. The first copy has a **creep** or **gcreep** (randomly decided) of randomly changing size performed on each gene. The second copy has a **creep** or **gcreep** (randomly decided) operation performed on each gene, but this time the size of the operation is fixed at $gene + p_f \times ran0$ or $gene \times (1 + p_f \times ran0)$. p_f is fixed for all genes in the genotype and is a random number, $-1 \leq p_f \leq 1$. The third copy is similar to the second except this time $-p_f$ is used.

27. **double**

double the point resolution in the **x** and **y** directions. Genotype length of every individual is increased by interpolation between known points.

28. **unifran***(nlow,nup)* **integer**,*(integer,integer)*

generate a uniformly random integer number in the range $nlow \rightarrow nup$.

29. **te***(gene)* **integer**,*(real)*

test if *gene* is within the range $lowbdry \leq gene \leq highbdry$. $te = 0$ if within the range, $te = -1$ if less than *lowbdry* and $te = +1$ if above *highbdry*.

30. **pl***(i,j)* **integer**,*(integer,integer)*

calculate the normal storage position in a candidate for the point $A_{i,j}$.

31. **pl2***(i,j)* **integer**,*(integer,integer)*

calculate the normal storage position in a candidate that is being doubled in size for the point $A_{i,j}$.

32. **plcd***(i,j)* **integer**,*(integer,integer)*

calculate the alternate storage position in a candidate for the point $A_{i,j}$.

33. **xx1pts2***(xpints2,ypints2)* *(integer,integer)*

double the number of points in the x and y directions

34. **apts2***(n,xpints2,ypints2,npints2)* *(integer,integer,integer,integer)*

interpolate new points in candidate n to go along with doubled number of points in the x and y directions. When $dbl = 0$ interpolate using **ranrep** points. If $dbl = 1$ then use linear interpolation.

35. **ncode***(n)* *(integer)*

Reorder normally stored candidate n genotype into alternative storage. See section 6.4.

36. **ncode***(n)* *(integer)*

Reorder alternately stored candidate n genotype into normal storage. See section 6.4.

37. **loadzin**

load in and check user set data - see appendix B. Sets up common blocks.

38. **setup**

setup initial population and $x, x1$ arrays. Open monitoring and output files.

39. **dumpzout**

dump out algorithm information to output files

40. **update***(noutzwt,noutzh,noof)* *integer,integer,integer*

update evolutionary history files

41. **source***(i,j,a,ax,ay)* **real***,(integer,integer,real,real,real)*

    calculate the value of the source term $S_{i,j}$ - see section 6.6.

42. **lobd***(i,j)* **real***,(integer,integer)*

    boundary conditions $g_{i,j}$ at the lower edge of the rectangular region -see section 6.6.

43. **upbd***(i,j)* **real***,(integer,integer)*

    boundary conditions $g_{i,j}$ at the upper edge of the rectangular region -see section 6.6.

44. **lhs***(i,j)* **real***,(integer,integer)*

    boundary conditions $g_{i,j}$ at the left hand edge of the rectangular region -see section 6.6.

45. **rhs***(i,j)* **real***,(integer,integer)*

    boundary conditions $g_{i,j}$ at the right hand edge of the rectangular region -see section 6.6.

46. **decide***(iternow,bzwt,evolav)* *(integer,real,real)*

    given the best weight *bzwt* and average evolutionary gradient *evolav* at iteration *iternow*, decide what to do next.

47. **bias***(i,j)* **real***,(integer,integer)*

    biassing function for mutation/combination functions.

A.3 POISGEN listing

Below is a listing of the version of POISGEN used to calculate solutions to the Poisson equations listed in chapter 6. The necessary input parameters are described in appendix B. Although not reproduced in strict FORTRAN77 format, it should be legible.

1.51.0 program POISGEN

```
      C SUN/SPARC VERSION
      C One point crossover routine
      C To solve Laplacian/Poisson equations
      C V 0.0: no runs yet! Based on ngq.for
      C V 0.1: runs based on constant valued bdry
      C V 1.0: Laplacian/Poisson solver on rectangular grid
      C This program uses NO extrapolants in the profile
      C function and NO difference replacements at the
      C boundary. The boundary is only directed considered
      C when we consider points next to the boundary.
      C NOTE also that this is the interactive version of
      C of lap1.for.
      C This version is controlled by a control file, control.in
      C V 1.1: Position dependent biassing function giving more muta-
tion
      C to those points furthest away from the boundaries
      C V 1.2: Errors in Ncode,Dcode corrected. Commenting out of
      C extrapolant functionality. Continuity with boundary
      C conditions now penalised - more continuous solutions
      C are given lower weights.
      C V 1.3: Source term now contains gradients of the function A
      C i.e., Ax and Ay. When we have a large number of points
      C in a candidate length, then the weight of that
      C candidate can be large, even although a visual comparison
      C with the true solution tells you that it is in fact a
      C good approximation. It is found
      C that moving to a 21x21 grid -large!- almost guarantees
      C that all the weights will 'top out' at the maximum
      C weight. Hence if all candidates, including completely
      C random ones, have the same weight there is nothing to choose
      C between them all. Therefore, gibberish answers are the
      C final result.
      C To counter this the weighting given to each measure
      C of the fitness should change according to the number
      C of points in the code.
      C V 1.4: It is found that an especially fit candidate can dominate
      C the population for a large number of iterations. This
      C means that the evolutionary gradient is flat and action
      C must be taken to generate better candidates. So various
      C schemes to combat this are being tried to combat the
      C flatness. Now we increase the weighting parameters
      C very slightly in a bid to differentiate better from
      C worse candidates
      C
      C xpoints == points in the x direction
      C xpoints == points in the y direction
      C npoints == xpoints*ypoints total number of points <=2000
      C npar == number of parents for the next generation
      C nchild == number of children to be generated
      C
      C Operator Information
      C Combination Operators act on the genes of the breeding stock
      C Mutation Operators act on the genes of the breeding stock
      C Transcription Operators act on the genes of the breeding stock
      C This version introduces a biasing function
      C to the maximun permitted value of the creep and geometric
      C creep operators. See subroutine bias
      C
      C.....
      C
      C GENETIC ALGORITHM: this is the top level of the program
and
      C it is here that the structure of the GA
      C is formed
      C
      C.....
      C
      C Variable Statements
      C Specific variables
      C
      C Evolution monitoring variables
        real pzwt,grad,flat(100),flatav
        real egrad,evol(100),evolav
        integer evolback
      C
      C  Do-loop variables
        integer ii,k
      C
      C  In/out-put variables
        integer noutzwt
      c  integer,noutzh
      C
      C  Decision variables
        integer kzey
      C
      C  Iteration variables
        integer izlow,izup
      C
      C  Other Variables
        real weight(1000),y(1000,2000)
      C
      C  Genetic Algorithm: 'Size' of GA variables
        integer npar,nchild,ntotal,npoints,itermax
        integer xpoints,ypoints
      C
      C  Genetic Algorithm: operators' variables
        real jig,wob,wobif
        integer loadin,jignow,wobnow
        integer mzlev,czlev,tzlev
        integer worzp
        integer swazs,revzs,mixzs,copzs,delzs
      C
      C  Profiling variables
        real pen1,pen2,dterm,factor,prod0
      C
      C  Common Blocks
        common /wts/ weight
        common /zysz/ y
      C
      C  Genetic algorithm data
        common /ga1/   npar,nchild,npoints
        common /ga2/   itermax
        common /ga3/   ntotal
        common /ga5/   jig,jignow,wob,wobnow,wobif
        common /ga6/   loadin
        common /ga7/   worzp
        common /ga11/  xpoints,ypoints
        common /mctzdat/ mzlev,czlev,tzlev
        common /trazdat1/ swazs,revzs,mixzs,copzs,delzs
        common /dec/   kzey,izlow,izup
      C
      C  Profiling data
        common /prof/  pen1,pen2,dterm,factor,prod0
      C
      C  Parameter statements
        parameter(noutzwt=20)
        parameter(noutzh=21)
      C
      C.....
      C
      C  INITIALISATION AND SETTING UP DATA
      C
      C.....
      C
      C  Initialise evolution monitoring stuff
        do 108 ii=1,50
        flat(ii)=10000.0
        evol(ii)=0.0
      108 continue
      C
      C  Load and Check data
        Call Loadzin
      C
```

```
C  Define Program Variables
C  Iteration variables
   kzey=0
   izlow=1
   izup=itermax
C  Set previous weight pzwt to default
C  large value and start...
   pzwt=1.0D100
   flatav=1.0
   evolback=5
C
C  Setup initial data:: x points and initial stock
C  Load in previous data if loadin=1 in file ga.in
C  Initial stock dumped to ADAMEVE.OUT
   if (loadin.eq.1) then
   print*,'Setup initial data:: best parent from BZPARZN.OUT'
   Call Setup
c   print*,'Data in ga.in must agree with the size of parent'
c   print*,'Size of input parent must be =',xpoints*ypoints
    open(unit=noutzwt,file='BZPARZN.OUT'
    ,access='sequential',
  &   status='unknown')
     do 107 k=1,npoints
      read(unit=noutzwt,fmt=97000) y(1,k)
107   continue
   close(unit=noutzwt)
   Call Profile(1)
   print*,'Loaded parent weight =',weight(1)
   else
   print*,'Setup initial data:: initial stock'
   Call Setup
   end if
C
C  Open files to monitor the history of the evolution
   open(unit=noutzwt,file='BZWTZH.OUT',access='sequential',
  &   status='unknown')
C
C  Initialise entire population
   print*,'Breeding from initial stock'
   Call Breed
C
C  Keep the first individual. At this point it is either the one
C  generated by special2, a previously loaded one.
   Call Keep(1)
C
C.....
C.....
C
C  MAIN ITERATION LOOP: THE GA ITSELF
C
C.....
C
678 print*,'Entering main iteration loop'
C
C   Start of iteration do-loop
   do 100 ii=izlow,izup
C
C   Make the whole stock continuous
   do 156 k=1,npar+nchild
     Call Continuity(k)
156   continue
C
C   profile(npar+nchild) the entire stock
   Call Profile(npar+nchild)
C
C   Sort the entire stock lower weights, higher entry
   Call Sort
c     if ((level.eq.2).and.(ii.eq.izlow)) then
c     open(unit=14,file='test4',status='unknown',
c  &     access='sequential')
c     do 295 k=1,npoints
c     write(unit=14,fmt=97000) y(1,k)
c295   continue
c     close(unit=14)
c     end if
C
C   Vary point in best parent having worst fit
   if ( mod(ii,worzp).eq.0 ) then
     Call Profile(-1)
     Call Profile(npar+nchild)
     Call Sort
   end if
C
C   Jiggle breeding stock every jignow iterations
   if ( mod(ii,jignow).eq.0 ) then
     print*,'Jiggle:: jig=',jig
     Call Jiggle(jig)
     Call Profile(npar+nchild)
     Call Sort
   end if
C
C   Evolution history may be flat! Check and take action
C   with a jiggle,special3 and zoom
   if (flatav.lt.0.0000001) then
     print*,'Evolution is flat:: Jiggle jig=',jig
C   Do a jiggle
     Call Jiggle(jig)
     Call Profile(npar+nchild)
     Call Sort
C   Add scaled randomness to the breeding stock
     do 146 k=2,npar
     Call Special3(k,50.0)
146   continue
     Call Profile(npar+nchild)
     Call Sort
C   Do a zoom on the worst point in the best candidate
     Call Profile(-1)
     Call Profile(npar+nchild)
     Call Sort
C   Wobble the best candidate
     Call Wobble(wob)
     Call Profile(npar+nchild)
     Call Sort
   end if
C
C   Switch on gene wobble operator
C   When triggered, it will wobble the parent genes
C   VERY SLIGHTLY, hopefully keeping it in the
C   breeding set but adding slightly different material
   if ((ii.ge.wobnow).or.(weight(1).lt.wobif)) then
     print*,'Wobble:: wob=',wob
     Call Wobble(wob)
     Call Profile(npar+nchild)
     Call Sort
   end if
C
C   Current best weight
   print*,ii,' Best weight =',weight(1)
C
C   Calculate measures of the evolution
   grad=weight(1)-pzwt
   egrad=1.0-weight(1)/pzwt
   do 109 k=1,evolback-1
     flat(k)=flat(k+1)
     evol(k)=evol(k+1)
109   continue
   flat(evolback)=abs(grad)
   evol(evolback)=abs(egrad)
   flatav=0.0
   evolav=0.0
   do 110 k=1,evolback
     flatav=flatav+abs(flat(k))
     evolav=evolav+abs(evol(k))
110   continue
```

```
      evolav=evolav/evolback
      flatav=flatav/evolback
C
C     If the new best weight is bigger - error in routine - stop
      if ( (pzwt.lt.weight(1)).and.(ii.ne.izlow) ) then
        print*,' Something wrong in routine! '
        stop
      end if
c     print*,'Change in best weight =',grad
c     print*,'Relative change in best weight E=',egrad
c     print*,'Average change in best weight Egrad=',evolav
C
C     Set new previous best weight
      pzwt=weight(1)
C
C     Breed top npar entries in y to generate nchild children
      Call Breed
C
C     Update history of evolution files
      Call Update(noutzwt,noutzh,ii)
C
C     Check genetic diversity of parents - if too low, tweak.
      Call Check
C
C     Mutate the stock indicated
      Call Mutate(2,ntotal,mzlev)
C
C     Combination operations performed on indicated stock
      Call Combine(2,ntotal,czlev)
C
C     Transcription operations performed on indicated stock
      Call Trans(2,ntotal,tzlev)
100 continue
C
C.....
C
C  END OF MAIN LOOP.
C.....
C.....
C  Profile,sort and update the entire population
      do 111 k=1,npar+nchild
      Call Continuity(k)
111 continue
c     Call Profile(npar+nchild)
c     Call Sort
c     Call Update(noutzwt,noutzh,ii)
C
C  Current best weight
      print*,'Best weight =',weight(1)
C
C  Dump out the current generation of best candidates
      Call Dumpzout
C
C  Decide what to do now. This sets kzey and the iteration
C  size izlow and izup
      Call Decide(ii,weight(1),evolav)
C
C  Options
C     kzey=0 :: Quit and exit program
C     kzey=1 :: Restart with best of previous level
C     kzey=2 :: Continue at this point resolution level
C     kzey=3 :: Double points and continue
      if (kzey.eq.0) then
C     Close files that monitor the history of the evolution
       close(unit=noutzwt)
c      close(unit=noutzh)
       print*,'History of best weight per generation in BZWTZH.OUT'
c      print*,'History of best candidate'
c      print*,'per generation in BZCANZH.OUT'
      end if
C
      if (kzey.eq.1) then
        pzwt=1.0D100
C     Reset evolution monitoring stuff
      do 1081 ii=1,50
        flat(ii)=10000.0
        evol(ii)=0.0
1081 continue
      goto 678
      end if
C
      if (kzey.eq.2) then
      goto 678
      end if
C
      if (kzey.eq.3) then
      if (((2*xpoints-1)*(2*ypoints-1)).gt.2000) then
C     Close files that monitor the history of the evolution
      close(unit=noutzwt)
c      close(unit=noutzh)
      print*,'Not enough array space to double points'
      print*,'Closing files....'
      print*,'History of best weight per generation in BZWTZH.OUT'
      print*,'History of best candidate'
      print*,'per generation in BZCANZH.OUT'
      print*,'Exiting program....ByeBye, ByeBye, ByeBye!'
      stop
     else
      Call Double
      pzwt=1.0D100
      open(unit=14,file='test3',status='unknown',
     &     access='sequential')
      do 299 k=1,npoints
      write(unit=14,fmt=97000) y(1,k)
299   continue
      close(unit=14)
C     Reset evolution monitoring stuff
      do 1082 ii=1,50
      flat(ii)=10000.0
      evol(ii)=0.0
1082   continue
C
C     Vary profile parameters according to level
C     of point resolution
1083   Call Profile(npar+nchild)
      Call Sort
      if (log(weight(1)).ge.(300.0*log(10.0)) ) then
      print*,'Change in point resolution:'
      print*,'change profile parameters'
      prod0=prod0/2.0
      goto 1083
      end if
     goto 678
     end if
     end if
     print*,'prod0=',prod0
     print*,'pen1=',pen1
     print*,'pen2=',pen2
C
C  Format Statements
97000 format(f15.8)
C
     END
C
C.....
C
C  GENETIC ALGORITHM ROUTINES
C  These routines operate directly on the strings to provide
C  the genetic algorithm functionality. Once the strings have
C  been set up these routines exist as the top level of behaviour,
C  and call on other routines/functions to implement the re-
quired
C  genetic algorithm.
C
```

```
C.....
C
C  Calculate the weights of all the candidates
   Subroutine Profile(number)
C
C  Subroutine specific variables
   real fit(2000),sum,worst,worstp
   real ed2
   integer i,number,numb,zoom,xwor,ywor
   real rrs(2000)
   integer nn,n
   integer place
   real dzmax,small
C
C  Other variables
   real y(1000,2000),x(100),x1(100)
   integer npar,nchild,npoints
   integer xpoints,ypoints
   real weight(1000)
   real pen1,pen2
   real dterm,factor,prod0,prod
   real hstep,kstep
C
C  Other functions
   integer te
   integer pl
   real source
C
C  Common statements required
   common /ga1/   npar,nchild,npoints
   common /ga11/  xpoints,ypoints
   common /zysz/  y
   common /zdimz/ x,x1
   common /wts/   weight
   common /disc/  rrs
   common /prof/  pen1,pen2,dterm,factor,prod0
   common /size/  hstep,kstep
C
C  Must evaluate pde at every point.
C  Poisson/laplacian equations
C  NB 2nd order finite difference.
C
C  Routine
   dzmax=300.0*log(10.0)
   small=1.0e-15
   zoom=0
   if (number.eq.-1) then
   numb=1
c   print*,'Varying worst point in best candidate'
   zoom=1
   else
   numb=number
   end if
   do 3000 i=1,numb
   ed2=hstep*kstep
   sum=0.0
   worst=0.0
   prod=0.0
   do 3500 n=2,ypoints-1
     do 3501 nn=2,xpoints-1
     place=pl(nn,n)
     if ( te( y(i,place) ).ne.0 ) then
      print*,i,place,' ',y(i,place),' Outside bdry...'
      stop
     end if
C      Construct the operator here
C      Nonbdry points
     fitc=(( y( i,pl(nn+1,n) )+y( i,pl(nn,n+1) )+
    &       y( i,pl(nn-1,n) )+y( i,pl(nn,n-1) )-
    &     4.0*y( i,place   )
    &     )/ed2)-source( nn,n,y(i,place),
    &     (y(i,pl(nn+1,n))-y(i,pl(nn-1,n)))/(2.0*hstep),
    &     (y(i,pl(nn,n+1))-y(i,pl(nn,n-1)))/(2.0*kstep)
    &          )
     rrs(place)=abs(fitc)
     fit(place)=abs(fitc)+small
C      Gives us the pos'n of worst departure from ode soln
     worstp = amax1(worst,fit(place))
     if (worstp.gt.worst) then
      xwor=nn
      ywor=n
     end if
     worst=worstp
C      Calculate measures of the fitness
     if( (nn.eq.2).or.(n.eq.2).or.
    &     (nn.eq.(xpoints-1)).or.(nn.eq.(ypoints-1))
    &   ) then
      prod=prod+prod0*log(fit(place))
      sum=sum+fit(place)
     else
      prod=prod+log(fit(place))
      sum=sum+fit(place)
     end if
3501   continue
3500 continue
C
C  Now do the boundary points, cornors excepted since they
C  play no direct role in the evaluation of the laplacian
C  Lower bdry
c    n=1
c    do 3502 nn=2,xpoints-1
c    place=pl(nn,n)
c    fitc=(( y( i,pl(nn+1,n) )+y( i,pl(nn,n+1) )+
c   &      y( i,pl(nn-1,n) )+extrap( i,nn,n )-
c   &     4.0*y( i,place   )
c   &    )/ed2)-source( nn,n,y(i,place) )
c    rrs(place)=abs(fitc)
c    fit(place)=abs(fitc)+small
cC   Gives us the pos'n of worst departure from ode soln
c    worstp = amax1(worst,fit(place))
c    if (worstp.gt.worst) then
c      xwor=nn
c      ywor=n
c    end if
c    worst=worstp
cC   Calculate measures of the fitness
c    prod=prod+log(fit(place))
c    sum=sum+fit(place)
c3502 continue
C
C  Upper bdry
c    n=ypoints
c    do 3503 nn=2,xpoints-1
c    place=pl(nn,n)
c    fitc=(( y( i,pl(nn+1,n) )+extrap( i,nn,n )+
c   &      y( i,pl(nn-1,n) )+y( i,pl(nn,n-1) )
c   &     -4.0*y( i,place   )
c   &    )/ed2)-source( nn,n,y(i,place) )
c    rrs(place)=abs(fitc)
c    fit(place)=abs(fitc)+small
cC   Gives us the pos'n of worst departure from ode soln
c    worstp = amax1(worst,fit(place))
c    if (worstp.gt.worst) then
c      xwor=nn
c      ywor=n
c    end if
c    worst=worstp
cC   Calculate measures of the fitness
c    prod=prod+log(fit(place))
c    sum=sum+fit(place)
c3503 continue
cC
cC   Left hand bdry
c    nn=1
```

```
c    do 3504 n=2,ypoints-1
c    place=pl(nn,n)
c c    fitc=(( y( i,pl(nn+1,n) )+y( i,pl(nn-1,n) )+
c  &       extrap( i,nn,n )+y( i,pl(nn,n-1) )
c  &     -4.0*y( i,place   )
c  &    )/ed2)-source( nn,n,y(i,place) )
c    rrs(place)=abs(fitc)
c    fit(place)=abs(fitc)+small
cC    Gives us the pos'n of worst departure from ode soln
c    worstp = amax1(worst,fit(place))
c    if (worstp.gt.worst) then
c      xwor=nn
c      ywor=n
c    end if
c    worst=worstp
cC    Calculate measures of the fitness
c    prod=prod+log(fit(place))
c    sum=sum+fit(place)
c3504 continue
cC
cC   Right hand bdry
c    nn=xpoints
c    do 3505 n=2,ypoints-1
c    place=pl(nn,n)
c    fitc=(( extrap( i,nn+1,n)+y( i,pl(nn-1,n) )+
c  &       y( i,pl(nn,n+1) )+y( i,pl(nn,n-1) )
c  &     -4.0*y( i,place   )
c  &    )/ed2)-source( nn,n,y(i,place) )
c    rrs(place)=abs(fitc)
c    fit(place)=abs(fitc)+small
cC    Gives us the pos'n of worst departure from ode soln
c    worstp = amax1(worst,fit(place))
c    if (worstp.gt.worst) then
c      xwor=nn
c      ywor=n
c    end if
c    worst=worstp
cC    Calculate measures of the fitness
c    prod=prod+log(fit(place))
c    sum=sum+fit(place)
c3505 continue
C
C   Calculate final weight for the i'th candidate
C   Make sure that prod does not exceed machine limit.
C   IF it does, set to be very large.
C
    if ( (prod).gt.dzmax )then
      prod=exp(dzmax)
    else
      prod=exp(prod)
    end if
    weight(i)=pen1*worst+pen2*sum + prod
3000 continue
    if (zoom.eq.1) then
      Call Zoomer(xwor,ywor)
    end if
    return
    end
C.....
C   Sort out all the candidates and put them in order
    Subroutine Sort
C
C   Subroutine specific variables
    integer nsort,l,ir,ii,i,j
    real savewt
    real savey(2000)
C
C   Other variables
    integer npar,nchild,ntotal,npoints
    real y(1000,2000)
    real weight(1000)
C
C   Common statements required
    common /ga1/    npar,nchild,npoints
    common /ga3/    ntotal
    common /zysz/   y
    common /wts/    weight
C
C   Routine: sort by heapsort - see Numerical Recipes
    nsort=ntotal
    l=(nsort/2)+1
    ir=nsort
101 continue
    if (l.gt.1) then
    l=l-1
    savewt=weight(l)
    do 1011 ii=1,npoints
      savey(ii)=y(l,ii)
101 1 continue
    else
    savewt=weight(ir)
    do 1012 ii=1,npoints
      savey(ii)=y(ir,ii)
1012   continue
    weight(ir)=weight(1)
    do 1013 ii=1,npoints
      y(ir,ii)=y(1,ii)
1013   continue
    ir=ir-1
    if (ir.eq.1) then
      weight(1)=savewt
      do 1014 ii=1,npoints
      y(1,ii)=savey(ii)
1014    continue
      return
    endif
    endif
    i=l
    j=2*l
202 if (j.le.ir) then
    if (j.lt.ir) then
      if (weight(j).lt.weight(j+1)) then
      j=j+1
      end if
    end if
    if (savewt.lt.weight(j)) then
      weight(i)=weight(j)
      do 2021 ii=1,npoints
      y(i,ii)=y(j,ii)
2021    continue
      i=j
      j=2*j
    else
      j=ir+1
    endif
    goto 202
    endif
    weight(i)=savewt
    do 2022 ii=1,npoints
    y(i,ii)=savey(ii)
2022 continue
    goto 101
    end
C.....
C   Zoom in on a bad point
    Subroutine Zoomer(xwor,ywor)
C
C   Routine specific variables
    integer ii,xwor,ywor
    real ran1,gene1,gene2
C
C   Other variables
    real y(1000,2000),zoomy
    integer npar,nchild,npoints
```

```
      integer xpoints,ypoints
C
C  Other functions
      integer te,pl
      real g05caf
C
C  Common blocks
      common /zysz/ y
      common /ga1/ npar,nchild,npoints
      common /ga11/ xpoints,ypoints
      common /ga8/ zoomy
C
C  Routine
C  Copy the best parent into the lowest positions
      do 7100 ii=2,npoints-1
      y(npar+nchild,ii)=y(1,ii)
      y(npar+nchild-1,ii)=y(1,ii)
7100 continue
      ran1=zoomy*2*( g05caf(1.0)-0.5 )
C
C  Where is the point?
C  Bottom left hand cornor
      if ((xwor.eq.1).and.(ywor.eq.1)) then
      gene1=y(1,pl(xwor+1,ywor+1))*(1.0+ran1)
      gene2=y(1,pl(xwor+1,ywor+1))*(1.0-ran1)
      if ( te(gene1).ne.0 ) then
        gene1=y(1,pl(xwor+1,ywor+1))
      end if
      if ( te(gene2).ne.0 ) then
        gene2=y(1,pl(xwor+1,ywor+1))
      end if
      y( npar+nchild,pl(xwor+1,ywor+1) )=gene1
      y( npar+nchild-1,pl(xwor+1,ywor+1) )=gene2
      end if
C
C  Bottom right hand cornor
      if ((xwor.eq.xpoints).and.(ywor.eq.1)) then
      gene1=y(1,pl(xwor-1,ywor+1))*(1.0+ran1)
      gene2=y(1,pl(xwor-1,ywor+1))*(1.0-ran1)
      if ( te(gene1).ne.0 ) then
        gene1=y(1,pl(xwor-1,ywor+1))
      end if
      if ( te(gene2).ne.0 ) then
        gene2=y(1,pl(xwor-1,ywor+1))
      end if
      y( npar+nchild,pl(xwor-1,ywor+1) )=gene1
      y( npar+nchild-1,pl(xwor-1,ywor+1) )=gene2
      end if
C
C  Top left hand cornor
      if ((xwor.eq.1).and.(ywor.eq.ypoints)) then
      gene1=y(1,pl(xwor+1,ywor-1))*(1.0+ran1)
      gene2=y(1,pl(xwor+1,ywor-1))*(1.0-ran1)
      if ( te(gene1).ne.0 ) then
        gene1=y(1,pl(xwor+1,ywor-1))
      end if
      if ( te(gene2).ne.0 ) then
        gene2=y(1,pl(xwor+1,ywor-1))
      end if
      y( npar+nchild,pl(xwor+1,ywor-1) )=gene1
      y( npar+nchild-1,pl(xwor+1,ywor-1) )=gene2
      end if
C
C  Top right hand cornor
      if ((xwor.eq.xpoints).and.(ywor.eq.ypoints)) then
      gene1=y(1,pl(xwor-1,ywor-1))*(1.0+ran1)
      gene2=y(1,pl(xwor-1,ywor-1))*(1.0-ran1)
      if ( te(gene1).ne.0 ) then
        gene1=y(1,pl(xwor-1,ywor-1))
      end if
      if ( te(gene2).ne.0 ) then
        gene2=y(1,pl(xwor-1,ywor-1))
      end if
      y( npar+nchild,pl(xwor-1,ywor-1) )=gene1
      y( npar+nchild-1,pl(xwor-1,ywor-1) )=gene2
      end if
C
C  Lower bdry
      if( ((xwor.gt.1).and.(xwor.lt.xpoints))
     &    .and.(ywor.eq.1)
     & ) then
      gene1=y(1,pl(xwor,ywor+1))*(1.0+ran1)
      gene2=y(1,pl(xwor,ywor+1))*(1.0-ran1)
      if ( te(gene1).ne.0 ) then
        gene1=y(1,pl(xwor,ywor+1))
      end if
      if ( te(gene2).ne.0 ) then
        gene2=y(1,pl(xwor,ywor+1))
      end if
      y( npar+nchild,pl(xwor,ywor+1) )=gene1
      y( npar+nchild-1,pl(xwor,ywor+1) )=gene2
      end if
C
C  Upper bdry
      if( ((xwor.gt.1).and.(xwor.lt.xpoints))
     &    .and.(ywor.eq.ypoints)
     & ) then
      gene1=y(1,pl(xwor,ywor-1))*(1.0+ran1)
      gene2=y(1,pl(xwor,ywor-1))*(1.0-ran1)
      if ( te(gene1).ne.0 ) then
        gene1=y(1,pl(xwor,ywor-1))
      end if
      if ( te(gene2).ne.0 ) then
        gene2=y(1,pl(xwor,ywor-1))
      end if
      y( npar+nchild,pl(xwor,ywor-1) )=gene1
      y( npar+nchild-1,pl(xwor,ywor-1) )=gene2
      end if
C
C  Left hand bdry
      if( ((ywor.gt.1).and.(ywor.lt.ypoints))
     &    .and.(xwor.eq.1)
     & ) then
      gene1=y(1,pl(xwor+1,ywor))*(1.0+ran1)
      gene2=y(1,pl(xwor+1,ywor))*(1.0-ran1)
      if ( te(gene1).ne.0 ) then
        gene1=y(1,pl(xwor+1,ywor))
      end if
      if ( te(gene2).ne.0 ) then
        gene2=y(1,pl(xwor+1,ywor))
      end if
      y( npar+nchild,pl(xwor+1,ywor) )=gene1
      y( npar+nchild-1,pl(xwor+1,ywor) )=gene2
      end if
C
C  Right hand bdry
      if( ((ywor.gt.1).and.(ywor.lt.ypoints))
     &    .and.(xwor.eq.xpoints)
     & ) then
      gene1=y(1,pl(xwor-1,ywor))*(1.0+ran1)
      gene2=y(1,pl(xwor-1,ywor))*(1.0-ran1)
      if ( te(gene1).ne.0 ) then
        gene1=y(1,pl(xwor-1,ywor))
      end if
      if ( te(gene2).ne.0 ) then
        gene2=y(1,pl(xwor-1,ywor))
      end if
      y( npar+nchild,pl(xwor-1,ywor) )=gene1
      y( npar+nchild-1,pl(xwor-1,ywor) )=gene2
      end if
C
C  Internal point
      if( ((ywor.gt.1).and.(ywor.lt.ypoints)).and.
     &  ((xwor.gt.1).and.(xwor.lt.ypoints))
```

```
   & ) then
    do 7101 ii=2,npoints-1
      y(npar+nchild-2,ii)=y(1,ii)
      y(npar+nchild-3,ii)=y(1,ii)
7101  continue
    gene1=y(1,pl(xwor,ywor+1))*2.0*( g05caf(1.0)-0.5 )
    gene2=y(1,pl(xwor,ywor-1))*2.0*( g05caf(1.0)-0.5 )
    gene3=y(1,pl(xwor+1,ywor))*2.0*( g05caf(1.0)-0.5 )
    gene4=y(1,pl(xwor-1,ywor))*2.0*( g05caf(1.0)-0.5 )
    if ( te(gene1).ne.0 ) then
      gene1=y(1,pl(xwor,ywor+1))
    end if
    if ( te(gene2).ne.0 ) then
      gene2=y(1,pl(xwor,ywor-1))
    end if
    if ( te(gene3).ne.0 ) then
      gene3=y(1,pl(xwor+1,ywor))
    end if
    if ( te(gene4).ne.0 ) then
      gene4=y(1,pl(xwor-1,ywor))
    end if
    do 7102 ii=0,3
      if ( g05caf(1.0).gt.0.5 ) then
      y( npar+nchild-ii,pl(xwor,ywor+1) )=gene1
      else
      y( npar+nchild-ii,pl(xwor,ywor+1) )=y(1,pl(xwor,ywor+1))
      end if
      if ( g05caf(1.0).gt.0.5 ) then
      y( npar+nchild-ii,pl(xwor,ywor-1) )=gene2
      else
      y( npar+nchild-ii,pl(xwor,ywor-1) )=y(1,pl(xwor,ywor-1))
      end if
      if ( g05caf(1.0).gt.0.5 ) then
      y( npar+nchild-ii,pl(xwor+1,ywor) )=gene3
      else
      y( npar+nchild-ii,pl(xwor+1,ywor) )=y(1,pl(xwor+1,ywor))
      end if
      if ( g05caf(1.0).gt.0.5 ) then
      y( npar+nchild-ii,pl(xwor-1,ywor) )=gene4
      else
      y( npar+nchild-ii,pl(xwor-1,ywor) )=y(1,pl(xwor-1,ywor))
      end if
7102  continue
    end if
C
C   Continuity
    Call Continuity(npar+nchild)
    Call Continuity(npar+nchild-1)
    Call Continuity(npar+nchild-2)
    Call Continuity(npar+nchild-3)
    return
    end
C.....
C   Generate a completly random string of numbers
    Subroutine Gener8(n)
C
C   Routine specific variables
    integer n,i
C
C   Other variables
    integer npar,nchild,npoints
    real y(1000,2000),lowbdry,highbdry
    real g05caf
C
C   Common blocks
    common /eqn2/ lowbdry,highbdry
    common /ga1/ npar,nchild,npoints
    common /zysz/ y
C
C   Routine
    do 2000 i=1,npoints
    y(n,i)=lowbdry+ (highbdry-lowbdry)*g05caf(1.0)
2000 continue
    return
    end
C.....
C   Ensure continuity of the nth candidate
    Subroutine Continuity(n)
C
C   Subroutine specific
    integer n
C
C   Other variables
    integer npar,nchild,npoints
    integer xpoints,ypoints
    real y(1000,2000)
C
C   Other functions
    integer pl
    real lobd,upbd,lhs,rhs
C
C   Common blocks
    common /ga1/ npar,nchild,npoints
    common /ga11/ xpoints,ypoints
    common /zysz/ y
C
C   Routine
C   The region is xpoints*ypoints
C   Bdries,not cornors
C   Left and right hand boundaries
    do 765 ii=2,xpoints-1
    y( n,pl(ii,ypoints)   )=upbd(ii,ypoints)
    y( n,pl(ii,1)     )=lobd(ii,1)
765 continue
C   Lower and upper bdries
    do 766 ii=2,ypoints-1
    y( n,pl(1,ii)     )=lhs(1,ii)
    y( n,pl(xpoints,ii)   )=rhs(xpoints,ii)
766 continue
C   Cornors
    y( n,pl(1,1) )=(lobd(1,1)+lhs(1,1))/2.0
    y( n,pl(xpoints,1) )
     =(lobd(xpoints,1)+rhs(xpoints,1))/2.0
    y( n,pl(1,ypoints) )=(upbd(1,ypoints)+lhs(1,ypoints))/2.0
    y( n,pl(xpoints,ypoints) )=
   & (upbd(xpoints,ypoints)+rhs(xpoints,ypoints))/2.0
    return
    end
C.....
C   Calulates an average value over the square plus a little
C   randomness
    Subroutine Special(n,frac)
C
C   Subroutine specific variables
    integer i,j,n
    real frac,xhere,yhere,here
C
C   Other variables required by this routine
    real y(1000,2000)
    real x(100),x1(100)
    real xzstart,xzend
    real yzstart,yzend
    real lowbdry,highbdry
    integer xpoints,ypoints
C
C   Other functions
    integer te,pl
    real g05caf
    real lhs,rhs,lobd,upbd
C
C   Common statements required
    common /eqn2/ lowbdry,highbdry
    common /eqn4/ xzstart,xzend
    common /eqn41/ yzstart,yzend
```

```
      common /ga11/ xpoints,ypoints
      common /zdimz/ x,x1
      common /zysz/ y
C
C  Routine
      do 2500 j=2,ypoints-1
      do 2502 i=2,xpoints-1
2501    xhere=lhs(1,j) +
     &    ( ( x(i)-xzstart)*( rhs(xpoints,j)-lhs(1,j) )
     &          /(xzend-xzstart) )
        yhere=lobd(i,1) +
     &     ( (x1(j)-yzstart)*( upbd(i,ypoints)-lobd(i,1) )
     &          /(yzend-yzstart) )
        here=(xhere+yhere)/2.0 +
     &   ( 2.0*(g05caf(1.0)-0.5)*(highbdry-lowbdry)/frac)
        if ( te(here).ne.0 ) then
         goto 2501
        else
         y(n,pl(i,j))=here
        end if
2502  continue
2500  continue
      Call Continuity(n)
      return
      end
C.....
C  Takes the candidate and adds a little randomness, the size
C  of which is dependent on the range of A-values permitted
      Subroutine Special2(n,frac)
Cs
C  Subroutine specific variables
      integer i,n
      real frac,here
C
C  Other variables required by this routine
      integer npar,nchild,npoints
      integer xpoints,ypoints
      real y(1000,2000)
      real highbdry,lowbdry
C
C  Other functions
      integer te,pl
      real g05caf
C
C  Common statements required
      common /ga11/ xpoints,ypoints
      common /eqn2/ lowbdry,highbdry
      common /ga1/ npar,nchild,npoints
      common /zysz/ y
C
C  Routine
      do 2600 j=2,ypoints-1
      do 2601 i=2,xpoints-1
        here=y(n,pl(i,j)) + (highbdry-lowbdry)*
     &       2.0*(g05caf(1.0)-0.5)/frac
        if ( te(here).eq.1 ) then
        here=highbdry
        end if
        if ( te(here).eq.-1 ) then
        here=lowbdry
        end if
        y(n,pl(i,j))=here
2601  continue
2600  continue
      Call Continuity(n)
      return
      end
C.....
C  Takes a candidate and adds a liitle randomness, the size
C  of which is dependent on the range of values actually
C  stored in the candidate
      Subroutine Special3(n,frac)
C
C  Subroutine specific variables
      integer i,n
      real frac,here
      real high,low
C
C  Other variables required by this routine
      integer npar,nchild,npoints
      integer xpoints,ypoints
      real y(1000,2000)
      real lowbdry,highbdry
C
C  Other functions
      integer te,pl
      real g05caf
C
C  Common statements required
      common /ga11/ xpoints,ypoints
      common /ga1/ npar,nchild,npoints
      common /eqn2/ lowbdry,highbdry
      common /zysz/ y
C
C  Routine
      high=lowbdry
      low=highbdry
C  First find the highest and lowest values
      do 2551 j=2,ypoints-1
      do 2552 i=2,xpoints-1
        if ( y(n,pl(i,j)).gt.high ) then
        high=y(n,pl(i,j))
        end if
        if ( y(n,pl(i,j)).lt.low ) then
        low=y(n,pl(i,j))
        end if
2552  continue
2551  continue
C  Now vary the candidate
      do 2630 j=2,ypoints-1
      do 2631 i=2,xpoints-1
        here=y(n,pl(i,j)) + (high-low)*
     &       2.0*(g05caf(1.0)-0.5)/frac
        if ( te(here).eq.1 ) then
        here=highbdry
        end if
        if ( te(here).eq.-1 ) then
        here=lowbdry
        end if
        y(n,pl(i,j))=here
2631  continue
2630  continue
      Call Continuity(n)
      return
      end
C.....
C  Breed new candidates by simple one point crossover
C  No biasing in favour of any particular candidate
C  Top npar candidates are numbered in the y array in order
C  best in y(1), second best in y(2) etc...
C   [Parents] [Chop]   [Swap]   [Children]
C  1 *******  **** + *** **** + !!! ****!!!
C  2 !!!!!!!  !!!! + !!! !!!! + *** !!!!***
      Subroutine Breed
C
C  Subroutine specific variables
      integer parent1,parent2,chop
      real y1(2000),y2(2000)
      integer reord
C
C  Other variables required by this routine
      integer npar,nchild,npoints
      integer unifran
      real y(1000,2000)
```

```
C
C  Common statements required
   common /ga1/   npar,nchild,npoints
   common /ga3/   ntotal
   common /zysz/  y
C
C  Routine
   do 5000 k=npar+1,ntotal,2
   parent1=unifran(1,npar)
5670  parent2=unifran(1,npar)
   if (parent2.eq.parent1) then
     goto 5670
   end if
   chop=unifran(1,npoints)
C
C   Reorder the code of the parents if reord>50
   reord=unifran(1,100)
   if (reord.gt.50) then
    Call Ncode(parent1)
    Call Ncode(parent2)
   end if
C
C   Store chopped code from chop to npoints
   do 3010 i=chop,npoints
     y1(i)=y(parent1,i)
     y2(i)=y(parent2,i)
3010  continue
C
C   Put crossed over code in child from chop to npoints
   do 3020 i=chop,npoints
     y(k,i)=y2(i)
     y(k+1,i)=y1(i)
3020 continue
C   Retain original parent code from 1 to chop-1
   do 3030 i=1,chop-1
     y(k,i)=y(parent1,i)
     y(k+1,i)=y(parent2,i)
3030  continue
C
C   Reorder the code of the parents if reord>50
   if (reord.gt.50) then
    Call Dcode(parent1)
    Call Dcode(parent2)
    Call Dcode(k)
    Call Dcode(k+1)
   end if
5000 continue
   return
   end
C.....
C  Mutation Operators. Randomly choose which is to be
C  used. Each can be weighted to be preferentially chosen
C  Pick a random number between 1,100. If that number lies
C  within a particular mutation operators' range of influence
C  then that operator will be chosen.
C  Operators can also obey different randomness distributions
C  just to play around
C  This version introduces a biasing function
C  to the maximun permitted value of the creep and geometric
C  creep operators. See subroutine bias
C  Mutants also satisfy continuity
   Subroutine Mutate(nlow,nup,mzlev)
C
C  Mutation Operator Variables
   integer rzlo,rzup,rantype1
   integer czlo,czup,rantype2
   real czmax
   integer gczlo,gczup,rantype3
   real gczmax
    integer mutzlev
C
C  Routine Variables
   integer i,j
   integer nlow,nup,mzlev
   integer parent1,place
   integer mutzop
   real gene,mutant
C
C  Other variables
   integer xpoints,ypoints
   real y(1000,2000)
   integer npar,nchild,npoints,ntotal
C
C  Other functions
   integer te
   integer pl
   integer unifran
   real ranrep,creep,gcreep
c   real bias
C
C  Common blocks required
   common /mutzdata/ rzlo,rzup,rantype1,
  &        czlo,czup,rantype2,czmax,
  &        gczlo,gczup,rantype3,gczmax,
  &        mutzlev
   common /ga1/ npar,nchild,npoints
   common /ga3/ ntotal
   common /zysz/ y
   common /ga11/ xpoints,ypoints
C
C  Routine
   gene=0.0
   do 6000 k=1,mzlev
C
C   Where to mutate
   parent1=unifran(nlow,nup)
   i=unifran(2,xpoints-1)
   j=unifran(2,ypoints-1)
   place=pl(i,j)
C
C   Get the gene to be mutated
   gene=y(parent1,place)
C
C   Which operator
   mutzop=unifran(1,100)
C
C   Ensure at least gene is replaced by itself
   mutant=gene
C
C   Random replacement
   if ((mutzop.ge.rzlo).or.(mutzop.le.rzup)) then
     mutant=ranrep(1.0)
   end if
C
C   Call Creep
   if ((mutzop.ge.czlo).or.(mutzop.le.czup)) then
     mutant=creep(mutant,czmax,rantype2)
   end if
C
C   Call Geometric creep
   if ((mutzop.ge.gczlo).or.(mutzop.le.gczup)) then
     mutant=gcreep(mutant,gczmax,rantype3)
   end if
C
C   This gene is the mutation
   if ( te(mutant).ne.0 ) then
     y(parent1,place)=gene
   else
     y(parent1,place)=mutant
   end if
C
6000 continue
   return
   end
```

```
C.....
C  Mutation functions
C  Random replacement
   Real Function Ranrep(dummy)
C
C  Specific variables
   real dummy
C
C  Other variables
   real lowbdry,highbdry
C
C  Other functions
   real g05caf
C
C  Common Blocks
   common /eqn2/ lowbdry,highbdry
C
C  Routine
   ranrep=lowbdry*(highbdry-lowbdry)*g05caf(1.0)
   return
   end
C  Creep
   Real Function Creep(gene,czmax,rantype)
C
C  Specific variables
   integer rantype
   real gene,czmax
C
C  Other functions
   real g05caf
C
C  Routine
   if (rantype.eq.0) then
   if ( g05caf(1.0).gt.0.5 ) then
     creep=gene+czmax
   else
     creep=gene-czmax
   end if
   end if
   if (rantype.eq.1) then
   creep=gene+ 2.0*czmax*(g05caf(1.0)-0.5)
   endif
   return
   end
C  Geometric Creep
   Real Function Gcreep(gene,gczmax,rantype)
C
C  Specific Variables
   integer rantype
   real gene,gczmax
C
C  Other functions
   real g05caf
C
C  Routine
   if (rantype.eq.0) then
   if ( g05caf(1.0).gt.0.5 ) then
     creep=gene*(1.0+gczmax)
   else
     creep=gene*(1.0-gczmax)
   end if
   end if
   if (rantype.eq.1) then
   gcreep=gene*(1.0 + 2.0*gczmax*(g05caf(1.0)-0.5))
   endif
   return
   end
C.....
C  Combination Operators. Randomly choose which is to be
C  used. Each can be weighted to be preferentially chosen
C  Pick a random number between 1,100. If that number lies
C  within a particular combination operators' range of influence
C  then that operator will be chosen. Three main modes of use,
C  mode=0 - combine genes from same position on different can-
didates
C  mode=1 - combine genes from different position on different
C     candidates
C  mode=2 - randomly flip between the above two modes
C
C  Combined offspring are forced to satisfy continuity
   Subroutine Combine(nlow,nup,czlev)
C
C  Subroutine specific variables
   integer parent1,parent2,place1,place2,combzop
   real comb,gene1,gene2
   integer nlow,nup,czlev
C
C  Other variables required
   integer avezlo,avezup
   real avzwt1,avzwt2
   integer gavzlo,gavzup,gzop
   integer extzlo,extzup
    integer comzlev
    integer mode
   integer npar,nchild,npoints,ntotal
   real y(1000,2000)
   real lowbdry,highbdry
C
C  Other functions
   integer unifran,te
   real wav,gav,extop
C
C  Common blocks
   common /zysz/ y
   common /eqn2/ lowbdry,highbdry
   common /comzdata/ avezlo,avezup,avzwt1,avzwt2,
  &        gavzlo,gavzup,gzop,
  &        extzlo,extzup,
  &        comzlev,
  &        mode
   common /ga1/   npar,nchild,npoints
   common /ga3/   ntotal
C
C  Routine
   do 7000 k=1,comzlev
C
C   Choose the parents
   parent1=unifran(nlow,nup)
   parent2=unifran(nlow,nup)
C
C   Choose the places on the chromosomes
   place1=unifran(1,npoints)
   place2=unifran(1,npoints)
C
C   Choose the operator
   combzop=unifran(1,100)
C
C   Modus Operandii
   if (mode.eq.0) then
     gene1=y(parent1,place1)
     gene2=y(parent2,place1)
     place2=place1
   end if
   if (mode.eq.1) then
     gene1=y(parent1,place1)
     gene2=y(parent2,place2)
   end if
   if (mode.eq.2) then
     if (g05caf(1.0).le.0.5) then
     gene1=y(parent1,place1)
     gene2=y(parent2,place1)
     place2=place1
     else
     gene1=y(parent1,place1)
```

```
      gene2=y(parent2,place2)
      end if
    end if
C
C  At worst, the statement allow will make no change to
C  the gene. We put this in to ensure at least something might
C  happen if the genes we have chosen fall inbetween all the
C  stools below.
C
    comb=gene1
C
C  Arithmetic average
    if ((combzop.ge.avezlo).and.(combzop.le.avezup)) then
      comb=wav(gene1,gene2,avzwt1,avzwt2)
    end if
C
C  Geometric average
    if ((combzop.ge.gavzlo).and.(combzop.le.gavzup)) then
      comb=gav(gene1,gene2,gzop)
    end if
C
C  Extension Operator
    if ((combzop.ge.extzlo).and.(combzop.le.extzup)) then
      comb=extop(gene1,gene2)
    end if
C
C  Make sure new info is within the range permitted
    if ( te(comb).eq.0 ) then
      y(parent1,place1)=comb
      y(parent2,place2)=comb
    end if
    if ( te(comb).eq.1 ) then
      y(parent1,place1)=gene2
      y(parent2,place2)=gene1
    end if
    if ( te(comb).eq.-1 ) then
      y(parent1,place1)=gene2
      y(parent2,place2)=gene1
    end if
7000 continue
    return
    end
C
C  Combination operators
C  Arithmetic Mean Combination Operator
C  Yields a weighted sum of two genes
    Real Function Wav(gene1,gene2,avzwt1,avzwt2)
C
C  Specific varaiables
    real gene1,gene2,avzwt1,avzwt2
C
C  Routine
    wav=(avzwt1*gene1+avzwt2*gene2)/
     (abs(avzwt1)+abs(avzwt2))
    return
    end
C  Geometric Mean Combination Operator
C  If either of the two genes passed to the function
C  are negative then we have three possible outcomes
C  gzop=0 - positive geometric mean
C  gzop=1 - negative geometric mean
C  gzop=2 - randomly assign either the plus or minus g-mean
C  Otherwise, take the positive g-mean
    Real Function Gav(gene1,gene2,gzop)
C
C  Specific varaiables
    integer gzop
    real gene1,gene2
C
C  Other functions
    real g05caf,wav
C
C  Routine
    gav=wav(gene1,gene2,1.0,1.0)
    if ( ((gene1.lt.0.0).and.(gene2.gt.0.0)).or.
   &   ((gene2.lt.0.0).and.(gene1.gt.0.0)) ) then
    if (gzop.eq.0) then
      gav=sqrt(abs(gene1*gene2))
    end if
    if (gzop.eq.1) then
      gav=-sqrt(abs(gene1*gene2))
    end if
    if (gzop.eq.2) then
      if (g05caf(1.0).le.0.5) then
      gav=sqrt(abs(gene1*gene2))
      else
      gav=-sqrt(abs(gene1*gene2))
      end if
    end if
    end if
    if ((gene1.le.0.0).and.(gene2.le.0.0)) then
    gav=-sqrt(abs(gene1*gene2))
    end if
    if ((gene1.gt.0.0).and.(gene2.gt.0.0)) then
    gav=sqrt(abs(gene1*gene2))
    end if
    return
    end
C  Extension Operator
C  Take the difference between two genes and randomly decide
to
C  add it to the higher, or subtract from the lower.
    Real Function Extop(gene1,gene2)
C
C  Specific variables
    real gene1,gene2
C
C  Other functions
    real g05caf
C
C  Routine
    if ( g05caf(1.0).le.0.5) then
    extop=max(gene1,gene2) + abs(gene1-gene2)
    else
    extop=min(gene1,gene2) - abs(gene1-gene2)
    end if
    return
    end
C.....
    Subroutine Trans(nlow,nup,tzlev)
C
C  Specific variables
    integer nlow,nup,tzlev,tzop
    integer k,parent1,place
    integer reord
C
C  Other variables
    integer swazlo,swazup,swazs
    integer revzlo,revzup,revzs
    integer mixzlo,mixzup,mixzs
    integer copzlo,copzup,copzs
    integer delzlo,delzup,delzs
    integer npar,nchild,npoints
C
C  Other functions
    integer unifran
C
C  Common Blocks
    common /trazdata/ swazlo,swazup,
   &        revzlo,revzup,
   &        mixzlo,mixzup,
   &        copzlo,copzup,
   &        delzlo,delzup
    common /trazdat1/ swazs,revzs,mixzs,copzs,delzs
```

```
      common /ga1/ npar,nchild,npoints
C  Routine
      do 7500 k=1,tzlev
C
C    Choose parent
      parent1=unifran(nlow,nup)
C
C    Reorder the code of the parent if reord>50
      reord=unifran(1,100)
      if (reord.gt.50) then
       Call Ncode(parent1)
      end if
C
C    Choose operator
      tzop=unifran(1,100)
C
C    Do the transcription 'errors'
C    Reverse the order a sequence of genes
      if ((tzop.ge.revzlo).and.(tzop.le.revzup)) then
       place=unifran(revzs+1,npoints-revzs-1)
       Call Reverse(parent1,place,revzs)
      end if
C
C    Swap the position of one or more genes
      if ((tzop.ge.swazlo).and.(tzop.le.swazup)) then
       place=unifran(swazs+1,npoints-swazs-1)
       Call Swap(parent1,place,swazs)
      end if
C
C    Randomly reorder a sequence of genes
      if ((tzop.ge.mixzlo).and.(tzop.le.mixzup)) then
       place=unifran(mixzs+1,npoints-mixzs-1)
       Call Mix(parent1,place,mixzs)
      end if
7500 continue
C
C  Copy data from edges and move it into the centre
      if ((tzop.ge.copzlo).and.(tzop.le.copzup)) then
       place=unifran(copzs+1,npoints-copzs-1)
       Call Copy(parent1,place,copzs)
      end if
C
C  Delete data from parent and move tail into the centre
      if ((tzop.ge.delzlo).and.(tzop.le.delzup)) then
       place=unifran(delzs+1,npoints-delzs-1)
       Call Del(parent1,place,delzs)
      end if
C
C  Put the code back into normal form if reord>50
      if (reord.gt.50) then
       Call Dcode(parent1)
      end if
      return
      end
C.....
C  Transcription error routines
      Subroutine Swap(parent,place,noof)
C
C  Specific variables
      integer k,parent,place,noof
      real yswap(2000)
C
C  Other variables
      real y(1000,2000)
C
C  Common Blocks
      common /zysz/ y
C
C  Routine
C  Get data from place to place+noof-1
      do 7510 k=1,noof
      yswap(k)=y(parent,place+k-1)
7510 continue
C
C  Move data located at (place-noof)z(place-1)
C  to (place)z(place+noof-1)
      do 7520 k=1,noof
      y(parent,place+k-1)=y( parent,place-noof+(k-1) )
7520 continue
C
C  take the swap data and put it in (place-noof)z(place-1)
      do 7530 k=1,noof
      y( parent,place-noof+(k-1) )=yswap(k)
7530 continue
      return
      end
C
      Subroutine Reverse(parent,place,noof)
C
C  Specific variables
      integer k,parent,place,noof
      real yrev(2000)
C
C  Other variables
      real y(1000,2000)
C
C  Common Blocks
      common /zysz/ y
C
C  Routine
C  Store and reverse from (place)z(place+noof-1)
      do 7540 k=1,noof
      yrev(noof-k+1)=y(parent,place+k-1)
7540 continue
C  Replace segment (place)z(place+noof-1) with its reverse
      do 7550 k=1,noof
      y(parent,place+k-1)=yrev(k)
7550 continue
      return
      end
      Subroutine Mix(parent,place,noof)
C
C  Specific variables
      integer k,parent,place,noof
      integer pt1,pt2
      real y1,y2
C
C  Other variables
      real y(1000,2000)
C
C  Other functions
      integer unifran
C
C  Common Blocks
      common /zysz/ y
C
C  Routine
C  Pick two points in the range and swap them over
      do 7570 k=1,noof
      pt1=unifran(place,place+noof-1)
      pt2=unifran(place,place+noof-1)
      y1=y(parent,pt1)
      y2=y(parent,pt2)
      y(parent,pt1)=y2
      y(parent,pt2)=y1
7570 continue
      return
      end
C  Copy data from outside towards the middle
      Subroutine Copy(parent,place,noof)
C
C  Specific variables
      integer k,parent,place,noof
C
```

```
C  Other variables
   real y(1000,2000)
   integer npar,nchild,npoints
C
C  Common Blocks
   common /zysz/ y
   common /ga1/ npar,nchild,npoints
C
C  Routine
   if (place.gt.(npoints/2)) then
   do 7580 k=1,noof
     y(parent,place-noof+k-1)=y(parent,place+k-1)
7580  continue
   else
   do 7590 k=1,noof
    y(parent,place+noof+k-1)=y(parent,place+k-1)
7590  continue
   end if
   return
   end
C  Delete data and move genes inwards
   Subroutine Del(parent,place,noof)
C
C  Specific variables
   integer k,parent,place,noof
C
C  Other variables
   real y(1000,2000)
   integer npar,nchild,npoints
C
C  Common Blocks
   common /zysz/ y
   common /ga1/ npar,nchild,npoints
C
C  Routine
   if (place.gt.(npoints/2)) then
   do 7800 k=1,npoints-place+1
     y(parent,place-noof+k-1)=y(parent,place+k-1)
7800  continue
   else
   do 7810 k=1,place
     y(parent,place+noof-k+1)=y(parent,place+1-k)
7810  continue
   end if
   return
   end
C.....
C  Check that there is sufficient genetic
C  diversity in the breeding population
C  weight(1)/weight(ii) ==
C           <= 1 by definition
C  m1==
C   = 1 - similarity <= 1
C  mtot=[ (sum ii=1,npar) of m1 ]/npar
C   == measure of the parental similarity
   Subroutine Check
C
C  Subroutine specific variables
   integer ii
   real m1,mtot
C
C  Other required variables
   real weight(1000),gentol
   integer npoints,npar,nchild
C
C  Common blocks
   common /wts/ weight
   common /ga1/ npar,nchild,npoints
   common /ga4/ gentol
C
C  Routine
   mtot=0.0
   m1=0.0
   do 9000 ii=1,npar
   m1=1.0-weight(1)/weight(ii)
   mtot=mtot+m1
9000 continue
   mtot=mtot/(1.0*npar)
   print*,'Genetic Diversity Measure',mtot
   if (mtot.le.gentol) then
   Call Tweak(mtot)
   endif
   return
   end
C.....
C  Tweak the parents
C  The diversity in the breeding population is less
C  than the level tolerated by gentol. Now Tweak!!
   Subroutine Tweak(mtot)
C
C  Subroutine specific variables
   integer ii,k
   real mtot
C
C  Other required variables
   integer npar,nchild,npoints
   integer twzpar
   real y(1000,2000)
C
C  Common blocks
   common /ga1/ npar,nchild,npoints
   common /zysz/ y
   common /ga9/ twzpar
C
C  Routine: tweak all but the twzpar parent(s)
   print*,'Tweaking....',npar+nchild-twzpar,' candidate(s)'
   write(unit=20,fmt=*) 'Tweak here'
   do 10002 ii=twzpar+1,npar+nchild
   Call Gener8(ii)
10002 continue
C
C  Make 'em continuous
   do 1561 k=1,npar+nchild
   Call Continuity(k)
1561 continue
   Call Profile(npar+nchild)
   Call Sort
   return
   end
C.....
C  Jiggle the breeding stock
C  Two methods are used: add a small number
C  to each gene, or multiply by a number close to one.
C  By the same number close to one
   Subroutine Jiggle(ran0)
C
C  Subroutine required variables
   integer i,ii
   real ran,ran0,ytest
C
C  Other variables
   integer npoints,npar,nchild
   real y(1000,2000),lowbdry,highbdry
   integer te
   real g05caf
C
C  Common Blocks
   common /zysz/ y
   common /ga1/ npar,nchild,npoints
   common /eqn2/ lowbdry,highbdry
C
C  Routine:
C  Randomly decide whether to multiply or add a little bit
C  it the parent stock
```

```
      do 8880 i=2,npar
      ran=ran0*2*( g05caf(1.0)-0.5 )
      do 8881 ii=2,npoints-1
        if ( g05caf(1.0).gt.0.5 ) then
        ytest=y(i,ii)*(1.0+ran)
        else
        ytest=y(i,ii)+ ran
        end if
        if ( te(ytest).eq.1 ) then
        ytest=y(i,ii)
        end if
        if ( te(ytest).eq.-1 ) then
        ytest=y(i,ii)
        end if
        y(i,ii)=ytest
8881  continue
      Call Continuity(i)
8880 continue
      return
      end
C.....
C  Wobble the best parent very slightly
C  and put it at the bottom of the heap
      Subroutine Wobble(ran0)
C
C  Subroutine required variables
      integer ii
      real ran1,ran,ran0
      real ytest1,ytest2,ytest3
C
C  Other variables
      integer npoints,npar,nchild
      real y(1000,2000),lowbdry,highbdry
C
C  Other functions
      integer te
      real g05caf
C
C  Common Blocks
      common /zysz/ y
      common /ga1/ npar,nchild,npoints
      common /eqn2/ lowbdry,highbdry
C
C  Copy best parent and wobble VERY slightly
C  in 3 different ways....multiply every gene randomly
C  and 2 copies that are slightly above and below the best
C  parent
      ran1=ran0*2*( g05caf(1.0)-0.5 )
      do 8781 ii=1,npoints
      ytest1=y(1,ii)
      ytest2=y(1,ii)
      ytest3=y(1,ii)
      ran=ran0*2.0*( g05caf(1.0)-0.5 )
      if ( g05caf(1.0).gt.0.5) then
        ytest1=y(1,ii)*(1.0+ran)
      else
        ytest1=y(1,ii) + ran
      end if
      if ( te(ytest1).eq.1 ) then
        ytest1=y(1,ii)
      end if
      if ( te(ytest1).eq.-1 ) then
        ytest1=y(1,ii)
      end if
      y(npar+nchild,ii)=ytest1
C
      if ( g05caf(1.0).gt.0.5) then
        ytest2=y(1,ii)*(1.0+ran1)
      else
        ytest2=y(1,ii) + ran1
      end if
      if ( te(ytest2).eq.1 ) then
        ytest2=y(1,ii)
      end if
      if ( te(ytest2).eq.-1 ) then
        ytest2=y(1,ii)
      end if
      y(npar+nchild-1,ii)=ytest2
C
      if ( g05caf(1.0).gt.0.5) then
        ytest3=y(1,ii)*(1.0-ran1)
      else
        ytest3=y(1,ii) - ran1
      end if
      if ( te(ytest3).eq.1 ) then
       ytest3=y(1,ii)
      end if
      if ( te(ytest3).eq.-1 ) then
        ytest3=y(1,ii)
      end if
      y(npar+nchild-2,ii)=ytest3
8781 continue
C
C  Continuity
      Call Continuity(npar+nchild-2)
      Call Continuity(npar+nchild-1)
      Call Continuity(npar+nchild)
      return
      end
C.....
C
C  GENE & STRING MANIPULATORS/INFORMATION
C  These are not Genetic Algorithm Operators, but
C  manipulate and read the genes to provide new information.
C
C.....
C  Calculate new sizes and double resolution in x and y dirns
      Subroutine Double
C
C  Specific variables
      integer k
      integer xpoints2,ypoints2,npoints2
C
C  Other variables
      real jig,wob,wobif
      integer jignow,wobnow
      integer xpoints,ypoints,npar,nchild,npoints
      integer dbl
      integer mzlev,czlev,tzlev
      integer swazs,revzs,mixzs,copzs,delzs
      real hstep,kstep
      real xzstart,xzend
      real yzstart,yzend
C
C  Common blocks
      common /ga1/   npar,nchild,npoints
      common /ga11/  xpoints,ypoints
      common /ga10/  dbl
      common /mctzdat/ mzlev,czlev,tzlev
      common /trazdat1/ swazs,revzs,mixzs,copzs,delzs
      common /ga5/   jig,jignow,wob,wobnow,wobif
      common /size/  hstep,kstep
      common /eqn4/  xzstart,xzend
      common /eqn41/ yzstart,yzend
C
C  Routine
C  Double point resolutions in each direction
C  These are the new doubled values of the x and y
C  coarseness
      xpoints2=2*xpoints-1
      ypoints2=2*ypoints-1
      npoints2=xpoints2*ypoints2
      print*,'Doubling resolution, quadrupling code length'
      print*,'Points in x-direction ',xpoints2
```

```
      print*,'Points in y-direction ',ypoints2
      print*,'Doubling A-points of population=',npar+nchild
      print*,'Genetic code length per candidate',npoints2
C     Option of doubling method
      if (dbl.eq.0) then
      print*,'Doubling points by random replacement'
      else
      if (dbl.eq.1) then
        print*,'Doubling points by interpolation'
      end if
      end if
C
C     Double the x and y resolution with the function below
      Call XX1pts2(xpoints2,ypoints2)
C     Call the subroutine that actually doubles the points
      do 680 k=1,npar+nchild
      Call Apts2(k,xpoints2,ypoints2,npoints2)
680   continue
C
C     Now assign the temporary variable values to the
C     variables used in the rest of the program.
      xpoints=xpoints2
      ypoints=ypoints2
      npoints=npoints2
C
C     Recalculate step sizes in each direction
      hstep=(xzend-xzstart)/( (xpoints-1)*1.00 )
      kstep=(yzend-yzstart)/( (ypoints-1)*1.00 )
C
C     Now that we have doubled point sizes, we can make
C     the doubled candidates continuous
      do 608 k=1,npar+nchild
      Call Continuity(k)
608   continue
C
C     Introduce a bit of variety to the parents on doubling
c     do 681 k=npar+1,npar+nchild
c     Call Gener8(k)
c     Call Continuity(k)
c681  continue
C
C     Wobble the best candidate
      Call Wobble(wob)
C     Keep the best candidate
      Call Keep(1)
C
C     Double the variables that scale with genetic code size
      print*,'Doubling mutation,combination'
      print*,'and transcription rates...'
      czlev=2*czlev
      mzlev=2*mzlev
      tzlev=2*tzlev
      swazs=2*swazs
      revzs=2*revzs
      mixzs=2*mixzs
      copzs=2*copzs
      delzs=2*delzs
      print*,'czlev',czlev,' mzlev',mzlev,' tzlev',tzlev
      print*,'swazs,revzs,mixzs,copzs,delzs'
      print*,swazs,revzs,mixzs,copzs,delzs
      return
      end
C.....
C     Generate a random integer between mlow,nup
      Integer Function unifran(nlow,nup)
C
C     Subroutine specific
      integer nlow,nup
C
C     Other functions
      real g05caf
C
C     Routine
      unifran=nlow+ nint( (nup-nlow)*g05caf(1.0) )
      return
      end
C.....
C     Test if gene is outside the bounds
      Integer function te(gene)
C
C     Subroutine specific
      real gene
C
C     Other variables
      real highbdry,lowbdry
C
C     Common blocks
      common /eqn2/ lowbdry,highbdry
C
C     Routine
      te=0
      if (gene.gt.highbdry) then
      te=1
      end if
      if (gene.lt.lowbdry) then
      te=-1
      end if
      return
      end
C.....
C     Where to put the value in the string given its i,j co-ords
C     Useing x-directed segments piled up
      Integer Function pl(i,j)
C
C     Specific variables
      integer i,j
C
C     Other variables
      integer xpoints,ypoints
C
C     Common Blocks
      common /ga11/ xpoints,ypoints
C
C     Routine
      pl=i+(j-1)*xpoints
      return
      end
C.....
C     For doubling pts we need a different routine: use x-directed
C     strips
      Integer Function pl2(i,j)
C
C     Specific variables
      integer i,j
C
C     Other variables
      integer xpoints,ypoints
C
C     Common Blocks
      common /ga11/ xpoints,ypoints
C
C     Routine
      pl2=i+(j-1)*(2*xpoints-1)
      return
      end
C.....
C     Alternate storage method:: given (i,j) pair, store the
C     code in strips of y-directed segments
      Integer Function plcd(i,j)
C
C     Specific variables
      integer i,j
C
C     Other variables
```

```
      integer xpoints,ypoints
C
C  Common Blocks
      common /ga11/ xpoints,ypoints
C
C  Routine
      plcd=j+(i-1)*ypoints
      return
      end
C.....
C  Double pts in the x and y directions
      Subroutine XX1pts2(xpoints2,ypoints2)
C
C  Subroutine specific
      real store(100)
      integer ii,xpoints2,ypoints2
C
C  Other variables
      integer xpoints,ypoints
      real x(100),x1(100)
C
C  Common blocks
      common /ga11/ xpoints,ypoints
      common /zdimz/ x,x1
C
C  Routine
C  Do the x-direction: data held in x array
C  Store old points
      do 1231 ii=1,xpoints
      store(ii)=x(ii)
1231 continue
C
C  Delete string
      do 1261 ii=1,xpoints2
      x(ii)=0.0
1261 continue
C
C  Reassign old points in new positions
      do 1241 ii=1,xpoints
      x(2*ii-1)=store(ii)
1241 continue
C
C  Calculate new points in new positions
      do 1251 ii=1,xpoints2
      if ( mod(ii,2).eq.0 ) then
        x(ii)=0.5*( x(ii-1) + x(ii+1) )
      end if
1251 continue
C
C  Do the y-direction: data held in x1 array
C  Store old points
      do 12311 ii=1,ypoints
      store(ii)=x1(ii)
12311 continue
C
C  Delete string
      do 12611 ii=1,ypoints2
      x1(ii)=0.0
12611 continue
C
C  Reassign old points in new positions
      do 12411 ii=1,ypoints
      x1(2*ii-1)=store(ii)
12411 continue
C
C  Calculate new points in new positions
      do 12511 ii=1,ypoints2
      if ( mod(ii,2).eq.0 ) then
        x1(ii)=0.5*( x1(ii-1) + x1(ii+1) )
      end if
12511 continue
      return
      end
C.....
C  Double A-points
      Subroutine Apts2(n,xpoints2,ypoints2,npoints2)
C
C  Subroutine specific
      integer ii,j
      real stozy(2000)
      integer n,xpoints2,ypoints2,npoints2
C
C  Other variables
      integer npar,nchild,npoints
      integer xpoints,ypoints
      integer dbl,nout
      real y(1000,2000)
C
C  Other functions
      integer pl,pl2
      real ranrep
C
C  Common blocks
      common /ga1/ npar,nchild,npoints
      common /ga11/ xpoints,ypoints
      common /zysz/ y
      common /ga10/ dbl
C
C  Routine
      nout=14
C  Store old points
      do 123 ii=1,xpoints*ypoints
      stozy(ii)=y(n,ii)
123 continue
      if (n.eq.1) then
      open(unit=nout,file='test0',status='unknown',
     &   access='sequential')
      do 299 ii=1,npoints
        write(unit=nout,fmt=98009) y(n,ii)
299   continue
      close(unit=nout)
      end if
C
C  Delete string
      do 126 ii=1,npoints2
      y(n,ii)=0.0
126 continue
C
C  Reassign old points in new positions
      do 124 j=1,ypoints
      do 1244 ii=1,xpoints
        y(n,pl2(2*ii-1,2*j-1))=stozy( pl(ii,j) )
1244  continue
124 continue
      if (n.eq.1) then
      open(unit=nout,file='test1',status='unknown',
     &   access='sequential')
      do 298 ii=1,xpoints2*ypoints2
        write(unit=nout,fmt=98009) y(n,ii)
298 continue
      close(unit=nout)
      end if
C
C  Calculate new points in new positions
      if (dbl.eq.0) then
      do 1245 j=1,ypoints2
        do 1246 ii=1,xpoints2
        if( ( (mod(j,2).eq.1).and.(mod(ii,2).eq.0) ).or.
     &      ( (mod(j,2).eq.0).and.(mod(ii,2).eq.1) ).or.
     &      ( (mod(j,2).eq.0).and.(mod(ii,2).eq.0) )
     &    ) then
          y(n,pl2(ii,j))=ranrep(1.0)
        end if
1246    continue
```

```
1245  continue
   end if
   if (dbl.eq.1) then
   do 125 j=1,ypoints2
     do 1255 ii=1,xpoints2
     if ( (mod(j,2).eq.1).and.(mod(ii,2).eq.0) ) then
      y(n,pl2(ii,j))=0.5*( y(n,pl2(ii-1,j))+y(n,pl2(ii+1,j)) )
     end if
     if ( (mod(j,2).eq.0).and.(mod(ii,2).eq.1) ) then
      y(n,pl2(ii,j))=0.5*( y(n,pl2(ii,j-1))+y(n,pl2(ii,j+1)) )
     end if
     if ( (mod(j,2).eq.0).and.(mod(ii,2).eq.0) ) then
      y(n,pl2(ii,j))=0.25*( y(n,pl2(ii-1,j-1))
  &              +y(n,pl2(ii+1,j-1))
  &              +y(n,pl2(ii-1,j+1))
  &              +y(n,pl2(ii+1,j+1))
  &           )
     end if
1255   continue
125   continue
   end if
   if (n.eq.1) then
   open(unit=nout,file='test2',status='unknown',
  &   access='sequential')
   do 297 ii=1,xpoints2*ypoints2
     write(unit=nout,fmt=98009) y(n,ii)
297 continue
   close(unit=nout)
   end if
98009 format( (f8.3,tr1) )
   return
   end
C.....
C  Takes x-directed code and reorders it into y-directed code
   Subroutine Ncode(n)
C
C  Specific variables
   integer n,i,j
   real ystore(2000)
C
C  Other variables
   integer npar,nchild,npoints
   real y(1000,2000)
   integer xpoints,ypoints
C
C  Other functions
   integer pl,plcd
C
C  Common blocks
   common /ga1/ npar,nchild,npoints
   common /zysz/ y
   common /ga11/ xpoints,ypoints
C
C  Routine
C   Copy candidate
   do 7001 i=1,npoints
    ystore(i)=y(n,i)
7001 continue
C   Reorder genetic code
   do 7002 i=1,xpoints
    do 7003 j=1,ypoints
     y( n,plcd(i,j) )=ystore( pl(i,j) )
7003   continue
7002 continue
   return
   end
C.....
C  Takes y-directed stored code and re-orders it into
C  normal x-directed code
   Subroutine Dcode(n)
C
C  Specific variables
   integer n,i,j
   real ystore(2000)
C
C  Other variables
   integer npar,nchild,npoints
   real y(1000,2000)
   integer xpoints,ypoints
C
C  Other functions
   integer pl,plcd
C
C  Common blocks
   common /ga1/ npar,nchild,npoints
   common /zysz/ y
   common /ga11/ xpoints,ypoints
C
C  Routine
C   Copy candidate
   do 7004 i=1,npoints
    ystore(i)=y(n,i)
7004 continue
C   Reorder genetic code
   do 7005 j=1,ypoints
    do 7006 i=1,xpoints
     y( n,pl(i,j) )=ystore( plcd(i,j) )
7006   continue
7005 continue
   return
   end
C.....
C
C  LOADING IN AND SETTING UP: DIAGNOSTIC OUTPUTS:
FILES IN/OUT
C
C.....
C
C  Load in the necessary data
   Subroutine Loadzin
C
C  Subroutine specific variables
   integer i
   integer nin1
C
C  Other functions
   integer te
   real lhs,rhs,lobd,upbd
   real y(1000,2000),x(100),x1(100)
C
C  Genetic algorithm Variables
   integer loadin
   integer ran0
   integer npar,nchild,xpoints,ypoints
    integer itermax
     real jig,wob,wobif
     integer jignow,wobnow
     integer worzp
     real zoomy
     integer twzpar
      integer dbl
C
C  Equation Variables
   real xzstart,xzend
   real yzstart,yzend
   real lowbdry,highbdry
    real hstep,kstep
C
C  Mutation Operator Variables
   integer rzlo,rzup,rantype1
   integer czlo,czup,rantype2
   real czmax
   integer gczlo,gczup,rantype3
   real gczmax
```

```
      real mutzlev1
       integer mutzlev
       integer mzlev
C
C  Combination Operator Variables
      integer avezlo,avezup
      real avzwt1,avzwt2
      integer gavzlo,gavzup,gzop
      integer extzlo,extzup
       real comzlev1
       integer comzlev
       integer mode
       integer czlev
C
C  Transcription Operator variables
      integer swazlo,swazup
      integer revzlo,revzup
      integer mixzlo,mixzup
       integer copzlo,copzup
       integer delzlo,delzup
       integer swazs,revzs,mixzs,copzs,delzs
       real swazsf,revzsf,mixzsf,copzsf,delzsf
       real trazlev1
       integer trazlev
       integer tzlev
C
C  Profile Operator Variables
      real pen1,pen2
      real dterm,factor,prod0
C
C  Control file data
      real cont(5,10)
      integer level,levelm
C
C  Common Blocks
C  Program
      common /zysz/ y
      common /zdimz/ x,x1
C
C  Genetic algorithm data
      common /ga1/ npar,nchild,npoints
      common /ga11/ xpoints,ypoints
      common /ga2/ itermax
      common /ga3/ ntotal
      common /ga4/ gentol
      common /ga5/ jig,jignow,wob,wobnow,wobif
      common /ga6/ loadin
      common /ga7/ worzp
      common /ga8/ zoomy
      common /ga9/ twzpar
      common /ga10/ dbl
C
C  Equation data
      common /eqn4/  xzstart,xzend
      common /eqn41/ yzstart,yzend
      common /eqn2/  lowbdry,highbdry
      common /size/  hstep,kstep
C
C  Mutation data
      common /mutzdata/ rzlo,rzup,rantype1,
     &        czlo,czup,rantype2,czmax,
     &        gczlo,gczup,rantype3,gczmax,
     &        mutzlev
C
C  Combination data
      common /comzdata/ avezlo,avezup,avzwt1,avzwt2,
     &        gavzlo,gavzup,gzop,
     &        extzlo,extzup,
     &        comzlev,
     &        mode
      common /mctzdat/ mzlev,czlev,tzlev
C
C  Transcription data
      common /trazdata/ swazlo,swazup,
     &        revzlo,revzup,
     &        mixzlo,mixzup,
     &        copzlo,copzup,
     &        delzlo,delzup
      common /trazdat1/ swazs,revzs,mixzs,copzs,delzs
C
C  Profile data
      common /prof/ pen1,pen2,dterm,factor,prod0
C
C  Control data
      common /control/ cont,level,levelm
C
C  Routine
      nin1=14
C
C  Profile operator
      print*,'Profiling data in profile.in'
      open(unit=nin1,file='profile.in',access='sequential',
     &   status='unknown')
      read(unit=nin1,fmt=99998) pen1
      read(unit=nin1,fmt=99998) pen2
      read(unit=nin1,fmt=99998) dterm
      read(unit=nin1,fmt=99998) factor
      read(unit=nin1,fmt=99998) prod0
      close(unit=nin1)
C
C  Read in Equation data
      print*,'Loading Equation data from equation.in'
      open(unit=nin1,file='equation.in',access='sequential',
     &   status='unknown')
      read(unit=nin1,fmt=99996) xzstart,xzend
      read(unit=nin1,fmt=99996) yzstart,yzend
      read(unit=nin1,fmt=99996) lowbdry,highbdry
      close(unit=nin1)
      if (highbdry.le.lowbdry) then
      print*,'highbdry less than lowbdry!!'
      stop
      endif
C
C  Read in and check Genetic Algorithm data
      print*,'Loading Genetic Algorithm data from ga.in'
      open(unit=nin1,file='ga.in',access='sequential',
     &   status='unknown')
      read(unit=nin1,fmt=99999) loadin
      read(unit=nin1,fmt=99999) ran0
      read(unit=nin1,fmt=99999) npar
      read(unit=nin1,fmt=99999) nchild
      read(unit=nin1,fmt=99999) xpoints
      read(unit=nin1,fmt=99999) ypoints
      read(unit=nin1,fmt=99999) itermax
      read(unit=nin1,fmt=99998) gentol
      read(unit=nin1,fmt=99991) jig,jignow
      read(unit=nin1,fmt=99990) wob,wobnow,wobif
      read(unit=nin1,fmt=99991) zoomy,worzp
      read(unit=nin1,fmt=99999) twzpar
      read(unit=nin1,fmt=99999) dbl
      close(unit=nin1)
      if (loadin.eq.1) then
      print*,'Loading data from BZPARZN.OUT'
      end if
      if (ran0.ne.1) then
      Call G05ccf
      print*,'...non repeatable random nos'
      else
      print*,'...repeatable random nos'
      endif
      npoints=xpoints*ypoints
      hstep=(xzend-xzstart)/( (xpoints-1)*1.00 )
      kstep=(yzend-yzstart)/( (ypoints-1)*1.00 )
      if ((npoints.le.5).or.(npoints.gt.2000)) then
```

```
      print*,npoints
      print*,'Must have 5 < npoints <= 2000'
      stop
      endif
      ntotal=npar+nchild
      if ((ntotal).gt.1000) then
      print*,'nchild+npar must be less than 1000'
      stop
      end if
      jig=jig*(highbdry-lowbdry)
      wob=wob*(highbdry-lowbdry)
      do 8001 j=1,ypoints
      do 8002 i=1,xpoints
        if( (te(lhs(i,j)).ne.0).or.(te(rhs(i,j)).ne.0).or.
     &    (te(upbd(i,j)).ne.0).or.(te(lobd(i,j)).ne.0)
     &    ) then
        print*,'At least one point on the bdry lies outside'
        print*,'the range of high/lowbdry'
        stop
        end if
 8002  continue
 8001 continue
C
C  Read in Mutation Operator data
      print*,'Loading Mutation Operator data from mutate.in'
      open(unit=nin1,file='mutate.in',access='sequential',
     &    status='unknown')
      read(unit=nin1,fmt=99995) rzlo,rzup,rantype1
      read(unit=nin1,fmt=99994) czlo,czup,rantype2,czmax
      read(unit=nin1,fmt=99994) gczlo,gczup,rantype3,gczmax
      read(unit=nin1,fmt=99998) mutzlev1
      close(unit=nin1)
      print*,'
      mutzlev=nint(mutzlev1*ntotal*(xpoints-2)*(ypoints-2))
      print*,'No.of mutations ',mutzlev
      mzlev=mutzlev
C
C  Check if any operators are switched off
      if ( ((rzlo.lt.0).and.(rzup.lt.0)).or.
     &    ((rzlo.gt.100).and.(rzup.gt.100)) ) then
      print*,'MUTATION: Random Replacement operator OFF'
      endif
      if ( ((czlo.lt.0).and.(czup.lt.0)).or.
     &    ((czlo.gt.100).and.(czup.gt.100)) ) then
      print*,'MUTATION: Creep operator OFF'
      endif
      if ( ((gczlo.lt.0).and.(gczup.lt.0)).or.
     &    ((gczlo.gt.100).and.(gczup.gt.100)) ) then
      print*,'MUTATION: Geometric Creep operator OFF'
      endif
C
C  Read in Combination Operator data
      print*,'Loading Combination Operator data from combine.in'
      open(unit=nin1,file='combine.in',access='sequential',
     &    status='unknown')
      read(unit=nin1,fmt=99993) avezlo,avezup,avzwt1,avzwt2
      read(unit=nin1,fmt=99995) gavzlo,gavzup,gzop
      read(unit=nin1,fmt=99992) extzlo,extzup
      read(unit=nin1,fmt=99998) comzlev1
      read(unit=nin1,fmt=99999) mode
      close(unit=nin1)
      print*,'
      comzlev=nint(comzlev1*ntotal*npoints)
      print*,'No.of combinations',comzlev
      czlev=comzlev
C
C  Check if any combination operators are off
      if ( ((avezlo.lt.0).and.(avezup.lt.0)).or.
     &    ((avezlo.gt.100).and.(avezup.gt.100)) ) then
      print*,'COMBINATION: Weigted Gene Averaging operator OFF'
      endif
      if ( ((gavzlo.lt.0).and.(gavzup.lt.0)).or.
     &    ((gavzlo.gt.100).and.(gavzup.gt.100)) ) then
      print*,'COMBINATION: Geometric Gene Averaging operator
     OFF'
      endif
      if ( ((extzlo.lt.0).and.(extzup.lt.0)).or.
     &    ((extzlo.gt.100).and.(extzup.gt.100)) ) then
      print*,'COMBINATION: Gene Extension operator OFF'
      endif
C
C  Read in Transcription Operator data
      print*,'Loading Transcription Operator data from transcr.in'
      open(unit=nin1,file='transcr.in',access='sequential',
     &    status='unknown')
      read(unit=nin1,fmt=99989) swazlo,swazup,swazsf
      read(unit=nin1,fmt=99989) revzlo,revzup,revzsf
      read(unit=nin1,fmt=99989) mixzlo,mixzup,mixzsf
      read(unit=nin1,fmt=99989) copzlo,copzup,copzsf
      read(unit=nin1,fmt=99989) delzlo,delzup,delzsf
      read(unit=nin1,fmt=99998) trazlev1
      close(unit=nin1)
      print*,'
      trazlev=nint(trazlev1*ntotal*npoints)
      print*,'No.of transcriptions',trazlev
      tzlev=trazlev
      swazs=nint(swazsf*npoints)
      revzs=nint(revzsf*npoints)
      mixzs=nint(mixzsf*npoints)
      copzs=nint(copzsf*npoints)
      delzs=nint(delzsf*npoints)
C
C  Check if any transcription operators are off
      if ( ((swazlo.lt.0).and.(swazup.lt.0)).or.
     &    ((swazlo.gt.100).and.(swazup.gt.100)) ) then
      print*,'TRANSCRIPTION: Swapping operator OFF'
      endif
      if ( ((mixzlo.lt.0).and.(mixzup.lt.0)).or.
     &    ((mixzlo.gt.100).and.(mixzup.gt.100)) ) then
      print*,'TRANSCRIPTION: Mixing operator OFF'
      endif
      if ( ((revzlo.lt.0).and.(revzup.lt.0)).or.
     &    ((revzlo.gt.100).and.(revzup.gt.100)) ) then
      print*,'TRANSCRIPTION: REVERSAL operator OFF'
      endif
      if ( (swazs.gt.npoints).or.(mixzs.gt.npoints).or.
     &    (revzs.gt.npoints) ) then
      print*,'Too many points asked for in one of the'
      print*,'transcription operator functions!'
      stop
      end if
C
C  Read in control program data
      print*,'Loading data from control.in '
      open(unit=nin1,file='control.in',access='sequential',
     &    status='unknown')
      do 7777 i=1,4
      read(unit=nin1,fmt=99987) cont(i,1),cont(i,2)
     &            ,cont(i,3)
      read(unit=nin1,fmt=99988) cont(i,4),cont(i,5)
     &            ,cont(i,6),cont(i,7)
 7777 continue
      read(unit=nin1,fmt=99999) level
      read(unit=nin1,fmt=99999) levelm
      close(unit=nin1)
C  Set itermax to the value of limitn for the first level
      itermax=nint( cont(level,3) )
      if (levelm.gt.4) then
      print*,'levelm must be less than or equal to 4'
      stop
      end if
C
C  Format Statements
99999 format(i5)
```

```
99998 format(f15.8)
c99997 format(3(f15.8))
99996 format(2(f15.8))
99995 format(3(i5))
99994 format(3(i5),f15.8)
99993 format(2(i5),2(f15.8))
99992 format(2(i5))
99991 format(f15.8,tr10,i5)
99990 format(f15.8,tr10,i5,tr10,f15.8)
99989 format(2(i5),f15.8)
99988 format(4(f15.8,tr8))
99987 format(3(f15.8,tr8))
   return
   end
C.....
C  Setup initial parents - see output file ADAMEVE.OUT
   Subroutine Setup
C
C  Subroutine specific variables
   integer i,ii,nout
C
C  Other variables
   integer npoints,npar,nchild
   integer xpoints,ypoints
   real y(1000,2000)
   real x(100),x1(100)
   real xzend,xzstart
   real yzend,yzstart
C
C  Common blocks
   common /zdimz/ x,x1
   common /zysz/ y
   common /eqn4/ xzstart,xzend
   common /eqn41/ yzstart,yzend
   common /ga1/ npar,nchild,npoints
   common /ga11/ xpoints,ypoints
C
C  Routine
   nout=14
   print*,'Setup initial data:: x points'
   do 10 i=1,xpoints
   x(i)=xzstart+(i-1)*(xzend-xzstart)/(1.0*(xpoints-1))
10  continue
   print*,'Setup initial data:: y points'
   do 11 i=1,ypoints
   x1(i)=yzstart+(i-1)*(yzend-yzstart)/(1.0*(ypoints-1))
11  continue
C  Setup initial data:: initial stock
   print*,'Setup initial data:: initial population'
   do 20 i=1,npar
   Call Gener8(i)
   Call Continuity(i)
20  continue
   print*,'Setup initial data:: special individual'
   Call Special(1,10.0)
   Call Continuity(1)
C
C  Dump out initial data
   print*,'Dumping out initial parents in ADAMEVE.OUT'
   print*,'Currently only first 100 entries per candidate'
   open(unit=nout,file='ADAMEVE.OUT',status='unknown',
  &   access='sequential')
   do 30 i=1,npar
   write(unit=nout,fmt=98000) ( y(i,ii), ii=1,100 )
30  continue
   close(unit=nout)
C
C  Format Statements
98000 format(100(f15.8,tr8))
   return
   end
C.....
C  Diagnostics: dump out the relevant information
   Subroutine Dumpzout
C
C  Subroutine specific variables
   integer i,ii,nout
C
C  Other required variables
   integer npoints,npar,nchild
   integer xpoints,ypoints
   integer level,levelm
   real y(1000,2000)
   real weight(1000)
   real x(100),x1(100)
   real rrs(2000)
   real xzstart,xzend
   real yzstart,yzend
   real cont(5,10)
C
C  Common blocks
   common /control/ cont,level,levelm
   common /wts/  weight
   common /zysz/ y
   common /zdimz/ x,x1
   common /ga1/  npar,nchild,npoints
   common /ga11/ xpoints,ypoints
   common /disc/ rrs
   common /eqn4/ xzstart,xzend
   common /eqn41/ yzstart,yzend
C
C  Routine
   nout=14
C
C  Best parent, depending on level
   if (level.eq.1) then
   print*,'Level 1 best parent in BZPARZN1.OUT'
   print*,'Total of ',npoints,' points'
   open(unit=nout,file='BZPARZN1.OUT',status='unknown',
  &   access='sequential')
   do 299 i=1,npoints
     write(unit=nout,fmt=98001) y(1,i)
299   continue
   close(unit=nout)
   end if
   if (level.eq.2) then
   print*,'Level 2 best parent in BZPARZN2.OUT'
   print*,'Total of ',npoints,' points'
   open(unit=nout,file='BZPARZN2.OUT',status='unknown',
  &   access='sequential')
   do 298 i=1,npoints
     write(unit=nout,fmt=98001) y(1,i)
298   continue
   close(unit=nout)
   end if
   if (level.eq.3) then
   print*,'Level 3 best parent in BZPARZN3.OUT'
   print*,'Total of ',npoints,' points'
   open(unit=nout,file='BZPARZN3.OUT',status='unknown',
  &   access='sequential')
   do 297 i=1,npoints
     write(unit=nout,fmt=98001) y(1,i)
297   continue
   close(unit=nout)
   end if
   if (level.eq.4) then
   print*,'Level 4 best parent in BZPARZN4.OUT'
   print*,'Total of ',npoints,' points'
   open(unit=nout,file='BZPARZN4.OUT'
    ,status='unknown',
  &   access='sequential')
   do 296 i=1,npoints
     write(unit=nout,fmt=98001) y(1,i)
296   continue
```

```
      close(unit=nout)
      end if
C
C  Best parent at current level
      print*,'Dumping out current best parent in BZPARZN.OUT'
      print*,'Total of ',npoints,' points'
      open(unit=nout,file='BZPARZN.OUT',status='unknown',
     &    access='sequential')
      do 301 i=1,npoints
      write(unit=nout,fmt=98001) y(1,i)
301   continue
      close(unit=nout)
C
C  Weights of the best parents
      print*,'Dumping out current best parents weights in BZWTZN.OUT'
      open(unit=nout,file='BZWTZN.OUT',status='unknown',
     &    access='sequential')
      do 302 i=1,npar
      write(unit=nout,fmt=*) i,weight(i)
302   continue
      close(unit=nout)
C
C  Discretisation errors as found by the profiler
      print*,'Dumping out current best parent'
      print*,'discretisation errors in BZERRZN.OUT'
      Call Profile(npar+nchild)
      Call Sort
      Call Profile(1)
      open(unit=nout,file='BZERRZN.OUT'
     ,status='unknown',
     &    access='sequential')
      do 304 ii=1,npoints
        write(unit=nout,fmt=98001) rrs( ii )
304   continue
      close(unit=nout)
C
C  Some information about calculation
      print*,'Information for LIST3D.FOR program in linfo.in'
      open(unit=nout,file='linfo.in',status='unknown',
     &    access='sequential')
      write(unit=nout,fmt=98002) xpoints
      write(unit=nout,fmt=98002) ypoints
      write(unit=nout,fmt=98001) xzstart
      write(unit=nout,fmt=98001) xzend
      write(unit=nout,fmt=98001) yzstart
      write(unit=nout,fmt=98001) yzend
      close(unit=nout)
C
C  Format Statements
98001 format(f15.8)
98002 format(i5)
      return
      end
C.....
C  Update the evolution history files
      Subroutine Update(noutzwt,noutzh,noof)
C
C  Subroutine specific variables
c     integer ii
      integer noof
C
C  Other required variables
      integer noutzwt,noutzh
      real weight(1000),y(1000,2000)
C
C  Common blocks
      common /wts/ weight
      common /zysz/ y
C
C  Routine
      write(unit=noutzwt,fmt=*) noof,weight(1)
c     write(unit=noutzh,fmt=98002) noof,( y(1,ii), ii=1,100 )
C
C  Format Statements
98002 format(i5,100(f15.8,tr8))
      return
      end
C.....
C
C  PHYSICS STUFF FOR THE EQUATIONS
C  These functions describe the equation we want to solve,
C  i.e., source and boundaries
C
C.....
C  Source term for poisson equation
      Real Function Source(i,j,a,ax,ay)
C
C  Subroutine specific variables
      integer i,j
      real a,ax,ay
C
C  Other variables
      real x(100),x1(100)
C
C  Common statements required
      common /zdimz/ x,x1
C
C  Routine
C  For details see internal report and analysis no.16
C  This particular run has lambda=1
      source=a**2.0
      return
      end
C.....
C  Boundary Conditions
      Real Function Lobd(i,j)
C
C  Subroutine specific variables
      integer i,j
      real pi
c     real r0
C
C  Other variables
      real x(100),x1(100)
      integer xpoints,ypoints
      intrinsic sin
C
C  Common statements
      common /zdimz/ x,x1
      common /ga11/ xpoints,ypoints
C
C  Routine
c     r0=( x(xpoints)-x(1) )/2.0
c     if ( (x(i)-x(1)).lt.r0 ) then
c     lobd=exp( 1.0-(x(i)-x(1))/r0 )
c     else
c     lobd=1.0
c     end if
      pi=acos(-1.0)
      lobd=4.0*sin( 2.0*pi*( (x(i)-x(1))/(x(xpoints)-x(1)) ))
c     lobd=2.0
      return
      end
C.....
      Real Function Upbd(i,j)
C
C  Subroutine specific variables
      integer i,j
      real pi
C
C  Other variables
      real x(100),x1(100)
      integer xpoints,ypoints
      intrinsic sin
```

```
C
C  Common statements
   common /zdimz/ x,x1
   common /ga11/ xpoints,ypoints
C
C  Routine
   pi=acos(-1.0)
   upbd=4.0*sin( 2.0*pi*( (x(i)-x(1))/(x(xpoints)-x(1)) ))
c   upbd=x(i)
c   upbd=2.0
   return
   end
C.....
   Real Function Lhs(i,j)
C
C  Subroutine specific variables
   integer i,j
C
C  Other variables
   real x(100),x1(100)
   integer xpoints,ypoints
   intrinsic sin
c   real r0
C
C  Common statements
   common /zdimz/ x,x1
   common /ga11/ xpoints,ypoints
C
C  Routine
c   r0=( x1(ypoints)-x1(1) )/2.0
c   if ( (x1(j)-x1(1)).lt.r0 ) then
c   lhs=exp( 1.0-(x1(j)-x1(1))/r0 )
c   else
c   lhs=1.0
c   end if
   pi=acos(-1.0)
   lhs=4.0*sin( 2.0*pi*( (x1(j)-x1(1))/(x1(ypoints)-x1(1)) ))
c   lhs=2.0
   return
   end
C.....
   Real Function Rhs(i,j)
C
C  Subroutine specific variables
   integer i,j
   real pi
C
C  Other variables
   real x(100),x1(100)
   integer xpoints,ypoints
   intrinsic sin
C
C  Common statements
   common /zdimz/ x,x1
   common /ga11/ xpoints,ypoints
C
C  Routine
   pi=acos(-1.0)
   rhs=4.0*sin( 2.0*pi*( (x1(j)-x1(1))/(x1(ypoints)-x1(1)) ))
c   rhs=x1(j)
c   rhs=2.0
   return
   end
C.....
C  Monitor the emergence of the best candidate
C  This routine controls when to double points, continue or exit
C  On entry,iternow should be a multiple of limitn
   Subroutine Decide(iternow,bzwt,evolav)
C
C  Subroutine specific variables
   integer iternow
   real bzwt,evolav
   integer limmax,limitn
   real fact,fact1
   real tzwt,steep,steep1
   integer i
C
C  Other variables
   real y(1000,2000)
   real cont(5,10)
   integer level,levelm
   real kp(2000)
   integer xpointsk,ypointsk
   integer npar,nchild,npoints
   integer xpoints,ypoints
   integer kzey,izlow,izup
   integer mzlev,czlev,tzlev
   integer nout
C
C  Common statements
   common /control/ cont,level,levelm
   common /keep1/ kp
   common /keep2/ xpointsk,ypointsk
   common /zysz/ y
   common /ga1/  npar,nchild,npoints
   common /ga11/ xpoints,ypoints
   common /dec/  kzey,izlow,izup
   common /mctzdat/ mzlev,czlev,tzlev
C
C  Routine
C
   nout=14
C  Setup the variables we need - give them nicer names
   limmax=nint( cont(level,1) )
   tzwt=cont(level,2)
   limitn=nint( cont(level,3) )
   steep=cont(level,4)
   fact=cont(level,5)
   steep1=cont(level,6)
   fact1=cont(level,7)
   Call Keep(1)
C
C
C  WE HAVE NOT REACHED THE UPPER ITERATION LEVEL
C
C
   IF (ITERNOW.LT.LIMMAX) THEN
C   START AGAIN CONDITION
C   [Best weight greater that fact*tzwt] OR
C   [ [evolution is too shallow] AND
C     [ fact1*tzwt < bzwt < fact*tzwt ]
C   ]
   PRINT*,'ITERNOW < LIMMAX'
   if( (bzwt.gt.fact*tzwt).or.
  &    ( (evolav.lt.steep).and.
  &      ( (bzwt.gt.fact1*tzwt).and.(bzwt.lt.fact*tzwt) )
  &    )
  &  ) then
     if (level.eq.1) then
     print*,'LEVEL 1:: STARTING AGAIN'
     print*,'Evolution gradient is too shallow'
     print*,'within upper and lower limits for continuance'
     print*,'OR best weight is above upper limit'
     kzey=1
C      No doubling at level 1 therefore just start again using
C      initial choice
C      Copy the string we kept into the top level
     do 9087 i=1,npoints
      y(1,i)=kp(i)
9087   continue
C      Generate a random set of other candidates
     do 9086 i=2,npar+nchild
      Call Gener8(i)
      Call Continuity(i)
```

```
 9086   continue
C      Reset iteration variables
       izlow=izup+1
       izup=izup+limitn
C      Profile and sort the population
       Call Profile(npar+nchild)
       Call Sort
C      Keep the best individual
       Call Keep(1)
       else
       kzey=1
       print*,'LEVEL ',level,' :: STARTING AGAIN'
       print*,'Evolution gradient is too shallow'
       print*,'within upper and lower limits for continuance'
       print*,'OR best weight is above upper limit'
       do 9085 i=1,npoints
        y(1,i)=kp(i)
 9085   continue
C      Generate a random set of other candidates
       do 9084 i=2,npar+nchild
        Call Gener8(i)
        Call Continuity(i)
 9084   continue
C      Reset iteration levels
       izlow=izup+1
       izup=izup+limitn
C      Profile and Sort the entire population
       Call Profile(npar+nchild)
       Call Sort
C      Keep the best individual
       Call Keep(1)
       end if
     end if
C
C    CONTINUE BECAUSE EVOLUTION IS STEEP EVEN AL-
THOUGH HIGH WEIGHT
C    [evolution is steep enough] AND [ fact1*tzwt < bzwt < fact*tzwt
]
C
     if ( (evolav.gt.steep).and.
    &   ( (bzwt.gt.fact1*tzwt).and.(bzwt.lt.fact*tzwt) )
    &   ) then
       kzey=2
       print*,'LEVEL ',level,' :: CONTINUING'
       print*,'Continuing:high wt BUT gradient good enough!!!'
C      Keep the best individual
       Call Keep(1)
C      Reset iteration levels
       izlow=izup+1
       izup=izup+limitn
     end if
C
C    CONTINUE BECAUSE WEIGHT IS NEARLY THERE EVEN
ALTHOUGH EVOLUTION
C    IS SHALLOW
C    [evolution is shallow] AND [ tzwt < bzwt < fact1*tzwt ]
C
     if ( (evolav.lt.steep1).and.
    &   ( (bzwt.gt.tzwt).and.(bzwt.lt.fact1*tzwt) )
    &   ) then
       kzey=2
       print*,'LEVEL ',level,' :: CONTINUING'
       print*,'Continuing:best wt near and evolution is shallow!!'
C    Reset iteration levels
       izlow=izup+1
       izup=izup+limitn
C    Extra action!!!
       print*,'Extra action!!!'
       do 8999 i=2,npar
       Call Special3(i,20.0)
 8999   continue
       do 9003 i=npar+1,npar+nchild
       Call Gener8(i)
       Call Continuity(i)
 9003   continue
       Call Wobble(0.1)
C    Profile and sort the entire population
       Call Profile(npar+nchild)
       Call Sort
C    Keep the best individual
       Call Keep(1)
     end if
C
C    CONTINUE BECAUSE WEIGHT IS NEARLY THERE AND
EVOLUTION IS STEEP
C    [evolution is steep] AND [ tzwt < bzwt < fact1*tzwt ]
C
     if ( (evolav.gt.steep1).and.
    &   ( (bzwt.gt.tzwt).and.(bzwt.lt.fact1*tzwt) )
    &   ) then
       kzey=2
       print*,'LEVEL ',level,' :: CONTINUING'
       print*,'Continuing:best wt near and evolution is steep!!'
C    Reset iteration levels
       izlow=izup+1
       izup=izup+limitn
C    Keep the best individual
       Call Keep(1)
     end if
C
C    CONTINUE BECAUSE WEIGHT IS GOOD ENOUGH BUT
WE CAN MAKE IT BETTER
C    AT THIS LEVEL.
C    GRADIENT is good enough, so continue
C    [ bzwt < tzwt ]
     if ( (bzwt.le.tzwt).and.(evolav.gt.steep1) ) then
       kzey=2
       print*,'LEVEL ',level,' :: CONTINUING'
       print*,'Continuing:target wt reached below LIMMAX!!'
       print*,'AND evolution is steep enough'
C    Reset iteration levels
       izlow=izup+1
       izup=izup+limitn
C    Keep the best individual
       Call Keep(1)
     end if
C    [ bzwt < tzwt ] AND [ evol< steep1 ]
C    GRADIENT is NOT good enough, so need to fiddle
     if ((bzwt.le.tzwt).and.(evolav.le.steep1)) then
       kzey=2
       print*,'LEVEL ',level,' :: CONTINUING'
       print*,'Continuing:target wt reached below LIMMAX!!'
       print*,'BUT evolution is not steep enough. Action!!'
       do 9002 i=2,npar
       Call Special3(i,50.0)
 9002   continue
       Call Wobble(0.1)
C    Profile and sort the entire population
       Call Profile(npar+nchild)
       Call Sort
C    Reset iteration levels
       izlow=izup+1
       izup=izup+limitn
C    Keep the best individual
       Call Keep(1)
     end if
C
     ELSE
C
C
C    WE ARE AT LIMMAX:: ONLY TWO OPTIONS AVAIL-
ABLE EXIT, OR DOUBLE
C    POINTS AS LONG WE ARE WITHIN THE DOUBLING
LIMITS
```

```
C   TARGET NOT REACHED:: EXIT PROGRAM
C   [ bzwt > tzwt ]
    PRINT*,'ITERNOW > LIMMAX'
    if (bzwt.gt.tzwt) then
      print*,'LEVEL ',level,' :: EXITING'
      print*,'Target weight not reached'
      print*,'Fool about with parameters'
      kzey=0
    end if
C
C   WEIGHT IS GOOD ENOUGH BUT WE ARE AT THE LIMIT
OF POINT
C   RESOLUTION
C   [bzwt>tzwt] AND [ [level<=4] AND [level=levelm] ]
    if( (bzwt.le.tzwt).and.
   &    ( (level.le.4).and.(level.eq.levelm) )
   &    ) then
      print*,'LEVEL ',level,' :: EXITING'
      print*,'Target weight reached'
      print*,'Resolution level reached'
      print*,'Exiting program...'
      kzey=0
    end if
C
C   DOUBLING RESOLUTION AND CONTINUING
C   Limmax and target weight have been reached and level is ok
C   double points and continue
    if ( (bzwt.lt.tzwt).and.
   &    (level.lt.4).and.(level.ne.levelm)
   &    ) then
      print*,'LEVEL ',level,' :: DOUBLING'
      print*,'Target weight reached'
      print*,'Doubling points and continuing'
      level=level+1
      kzey=3
C     Reset iteration levels
      izlow=izup+1
      izup=izup+nint( cont(level,3) )
C     Profile and sort entire population
c       Call Profile(npar+nchild)
c       Call Sort
C     Keep the best individual
      Call Keep(1)
      print*,'NOW AT LEVEL ',level
    end if
    END IF
    return
    end
C.....
C   Keep the nth candidate and dimensions
    Subroutine Keep(n)
C
C   Subroutine specific variables
    integer n,xpointsk,ypointsk
    real kp(2000)
    integer i
C
C   Other variables
    real y(1000,2000)
    integer xpoints,ypoints
    integer npar,nchild,npoints
C
C   Common statements
    common /keep1/ kp
    common /keep2/ xpointsk,ypointsk
    common /zysz/ y
    common /ga1/ npar,nchild,npoints
    common /ga11/ xpoints,ypoints
C
C   Routine
    do 9090 i=1,npoints
    kp(i)=y(n,i)
9090 continue
    xpointsk=xpoints
    ypointsk=ypoints
    return
    end
C.....
    Real Function Extrap(n,i,j)
C
C   Specific variables
    integer n,i,j
C
C   Other variables
    integer xpoints,ypoints
    real y(1000,2000)
C
C   Other functions
    integer pl
C
C   Common Blocks
    common /ga11/ xpoints,ypoints
    common /zysz/ y
C
C   Routine
    if (j.eq.1) then
    extrap=2.0*y( n,pl(i,j) )-y( n,pl(i,j+1) )
    end if
    if (j.eq.ypoints) then
    extrap=2.0*y( n,pl(i,j) )-y( n,pl(i,j-1) )
    end if
    if (i.eq.1) then
    extrap=2.0*y( n,pl(i,j) )-y( n,pl(i+1,j) )
    end if
    if (i.eq.xpoints) then
    extrap=2.0*y( n,pl(i,j) )-y( n,pl(i-1,j) )
    end if
    return
    end
C.....
    Real Function Bias(i,j)
C
C   Specific variables
    integer i,j
    real cx,cx1,r,r0
C
C   Other variables
    integer xpoints,ypoints
    real x(100),x1(100)
C
C   Common Blocks
    common /ga11/ xpoints,ypoints
    common /zdimz/ x,x1
C
C   Routine
C   Find the centre
    cx=( x(1) + x(xpoints) )/2.0
    cx1=( x1(1) + x1(ypoints) )/2.0
C   Calculate straight line from centre to point and
C   scale it to the dimensions we have chosen for this
C   calculation. Call the scale length r0
    r0=amax1( x(xpoints)-x(1),x1(ypoints)-x1(1) )
    r=sqrt( ( cx-x(i) )**2 + ( cx1-x1(j) )**2 )/r0
C   Biasing function:: zero if on bdry, nonzero elsewhere
C   Choice of function available, but say for mutation
C   we could have a larger variation in the permitted range
C   towards the centre, where it is most needed
    if ( (i.eq.1).or.(i.eq.xpoints).or.
   &    (j.eq.1).or.(j.eq.ypoints)
   &  ) then
    bias=0.0
    else
    bias=2.0-exp(r-1.0)
    end if
```

```
      return
      end
C.....
```

Appendix B

POISGEN: sample input files and comments

The philosophy of leaving the algorithm as open as possible to modification is most obviously expressed by the large number of input variables required to control POISGEN's detailed behaviour. This hopefully allows the user the maximum freedom to experiment with the routine. It is also a measure of the current level of sophistication that these algorithms have attained: on the one hand, a large number of variables must mean that we may influence the behaviour in very many ways, but on the other, it means that the truly significant variables are not known. The variables are not divided arbitrarily between the files. Those that bear some loose relation to each other are largely kept together.

1. **ga.in** Genetic algorithm variables: these control global algorithm behaviour i.e., genotypic length and activation conditions for various convergence strategies.

2. **equation.in** equation variables such as the initial number of points in the x and y directions.

3. **mutate.in** variables that control the mutation operations via subroutine **mutate** and related functions/subroutines.

4. **combine.in** variables that control the combination operations via subroutine **combine** and related functions/subroutines.

5. **transcr.in** variables that control the transcription operations via subroutine **trans** and related functions/subroutines.

6. **profile.in** variables that control the weighting given to each individual in the population via subroutine **profile** and related functions/subroutines.

filename

⋮

row.(column letter) *variable name*: short description of use in program.

⋮

Figure B.1: Key to variable descriptions

7. **control.in** these variables are used in subroutine **decide** to decide what action the algorithm should take after a certain number of iterations.

It is entirely likely that not all the variables are equally important. Some may be dropped in later versions, as the program is not yet in a finalised 'black box' state. Brief descriptions of all the user set variables are available in section B.1.

B.1 Description of input variables

Figure B.1 provides a key to reading the variable descriptions. The input files also come with a short note of each variables' use, which is expanded upon below.

ga.in

1. *loadin*: if = 1 then load in old parental stock from BZPARZN.OUT, else start with random individuals.
2. *rep*: if = 1 then use repeatable random numbers, else nonrepeatable.
3. *npar*: number of breeding parents per generation.
4. *nchild*: number of children created per generation.
5. *xpoints*: number of points including boundaries in the x direction.
6. *ypoints*: number of points including boundaries in the y direction.
7. *itermax*: maximum number of iterations (generations) to be performed.
8. *gentol*: minimum genetic diversity (see subroutine **check**) permitted in the breeding stock. If the genetic diversity falls below this level, remedial action is taken.
9. (a) *jig*: $jig \times (highbdry - lowbdry)$ is the maximum value of the factor in subroutine **jiggle**.
   (b) *jignow*: call subroutine **jiggle** every $k \times jignow$'th iteration.
10. (a) *wob*: $wob \times (highbdry - lowbdry)$ is the maximum value of the factor used in subroutine **wobble**.

(b) *wobnow*: call subroutine **wobble** after *wobnow* iterations.

(c) *wobif*: call subroutine **wobble** if best weight is less than *wobif*.

11. (a) *zoomy*: maximum value of factor used in subroutine **zoomer**.

    (b) *worzp*: call subroutine **zoomer** every *worzp*'th iteration.

12. twzpar: number of parents to keep unchanged when calling subroutine **tweak**.

13. dbl: method of interpolation used. If = 0, then call **ranrep** at all new points. If = 1, then user linear interpolation to find values for the new points.

profile.in

1. *pen1*: worst point weighting

2. *pen2*: summation of discretisation errors weighting

3. *dterm*: not used at present

4. *factor*: not used at present

5. *prod0*: used in product of discretisation error measure. This penalises for continuity with the boundary conditions.

equation.in

1. (a) *xzstart*: left hand side of rectangular region.

   (b) *xzend*: right hand side of rectangular region.

2. (a) *yzstart*: lower boundary of rectangular region.

   (b) *yzend*: upper boundary side of rectangular region.

3. (a) *yzmin*: minimum permissible gene value

   (b) *yzmax*: maximum permissible gene value

control.in

The data in this file is used exclusively in the subroutine **decide**. Each pair of rows contains conditions and information on the desired program behaviour at differing point resolutions. The first two rows specify the conditions for carrying on to the next point resolution level and also some conditions for activating remedial action, should the calculation become bogged down in any way.

The design of the controlling mechanism is detailed in appendix A.

1. (a) *limmax*: maximum number of iterations at this point resolution level

   (b) *tzwt*: target best weight after *limmax* iterations.

(c) *limitn*: call **decide** every *limitn*'th iteration.

2. (a) *steep*: minimum evolutionary steepness for best weights 'far' from target weight $tzwt$.

   (b) *fact*: $fact \times tzwt$ is a 'large' or 'far' weight; used to decide how far away the best weight is from the target weight $tzwt$.

   (c) *steep1*: minimum evolutionary steepness for best weights 'close' to the target weight $tzwt$.

   (d) *fact1*: $fact1 \times tzwt$ is a 'low' or 'close' weight; used to decide how close away the best weight is from the target weight $tzwt$.

   The following 6 lines of data are paired off in a similar fashion. The last two variables are

3. level: implement lines $2 \times level$,$2 \times level + 1$ of **control.in** in **decide** as the data for the first level of point resolution

4. level1: implement lines $2 \times level1$,$2 \times level1 + 1$ of **control.in** in **decide** as the data for the last level of point resolution

The mutation, combination and transcription subroutines work in very similar ways. An integer number between 1 and 100 is generated and should this number lie in the range $\cdots lo$ to $\cdots up$, (where $\cdots$ represents a string of symbols; for example rz for the random replacement functiom) then the corresponding operator is implemented. Also, in all cases, if $rantype(n) = 1$, then a uniformly random distribution is used. Other random distributions are not yet fully implemented.

mutate.in

1. variables used in function **ranrep**

   (a) *rzlo*: lower limit of range

   (b) *rzup*: upper limit of range

   (c) *rantype1*: type of random ditribution used.

2. variables used in function **creep**

   (a) *czlo*: lower limit of range

   (b) *czup*: upper limit of range

   (c) *rantype2*: type of random ditribution used.

   (d) *czmax*: maximum arithmetic creep

3. variables used in function **gcreep**

   (a) *gczlo*: lower limit of range

(b) *gczup*: upper limit of range

(c) *rantype3*: type of random ditribution used.

(d) *gczmax*: maximum geometric creep

4. *mutlev1*: percentage of genes to be mutated. Total number of genes=$npoint \times (npar + nchild)$.

combine.in

1. variables used in function **wav**

   (a) *avezlo*: lower limit of range

   (b) *avezup*: upper limit of range

   (c) *avzwt1*: weighting given to first gene

   (d) *avzwt2*: weighting given to second gene

2. variables used in function **wav**

   (a) *gavzlo*: lower limit of range

   (b) *gavzup*: upper limit of range

   (c) *gzop*: what to do in case of a -ve product of genes.

3. variables used in function **ext**

   (a) *extzlo*: lower limit of range

   (b) *extzup*: upper limit of range

4. *comlev1*: number of combination operations expressed as a percentage of the total number of genes. Total number of genes=$npoint \times (npar + nchild)$. The percentage of genes affected by these operations is $2 \times comlev1$.

5. mode: where to choose the genes from on differing candidates.

transcr.in

1. variables used in subroutine **swap**

   (a) *swazlo*: lower limit of range

   (b) *swazup*: upper limit of range

   (c) *swazsf*: percentage of genotype (length *npoint*) to swap with neighbouring portion. Nearest integer value chosen.

2. variables used in subroutine **rev**

(a) *revzlo*: lower limit of range

(b) *revzup*: upper limit of range

(c) *revzsf*: percentage of genotype (length *npoint*) to have its order reversed. Nearest integer value chosen.

3. variables used in subroutine **mix**

(a) *mixzlo*: lower limit of range

(b) *mixzup*: upper limit of range

(c) *mixzsf*: percentage of genotype (length *npoint*) to have its order mixed up. Nearest integer value chosen.

4. variables used in subroutine **copy**

(a) *copzlo*: lower limit of range

(b) *copzup*: upper limit of range

(c) *copzsf*: percentage of genotype (length *npoint*) to be copied. Nearest integer value chosen. Overwrites consecutive genetic information.

5. variables used in subroutine **delete**

(a) *delzlo*: lower limit of range

(b) *delzup*: upper limit of range

(c) *delzsf*: percentage of genotype (length *npoint*) to be deleted. Nearest integer value chosen.

B.2 Sample input files

To run POISGEN, one must supply seven separate files, correctly formatted with the relevant information. Below are the input files used to generate the results of section 6.6.

1. Mutation data: **MUTATE.IN**

```
1       33      1
33      66      0       0.2
1       100     1       0.2
0.01
^------^------^------^---------------------------------------
MUTATE.IN for laplacian/poisson program LAP1.FOR
```

```
Mutation Operator Data
^------^------^------^-------------------------------------
[row]....[operator].........[variables]...[description]
1........random replacement..rzlo          lower bound of d.o.i
                             rzup          upper bound of d.o.i
                             rantype1      random distrib type
2........creep..............czlo          lower bound of d.o.i
                             czup          upper bound of d.o.i
                             rantype2      random distrib type
                             czmax         maximum effect
3........geometric creep.....gczlo         lower bound of d.o.i
                             gczup         upper bound of d.o.i
                             rantype3      random distrib type
                             gczmax        maximum effect
4........---------------.....mutzlev1      % mutations of all
                                           type applied to
                                           entire stock
```

2. Combination data: **COMBINE.IN**

```
1       33        1.0     1.0
34      90        1
67      100
0.01
0
^------^---------^-------^----------------------
COMBINE.IN for laplacian/poisson program POISGEN
Combination Operator Data
^------^---------^-------^----------------------
[row]..[operator].........[variables]...[description]
1......weighted average....avezlo        lower bound of d.o.i
                           avezup        upper bound of d.o.i
                           avzwt1        weight to first gene
                           avzwt2        weight to second gene
2......geometric average...gavzlo        lower bound of d.o.i
                           gavzup        upper bound of d.o.i
                           gzop          what to do with
                                         +ve,-ve genes
```

```
3......extension...........extzlo        lower bound of d.o.i
                           extzup        upper bound of d.o.i
4......---------------.....comzlev1      %combinations of all
                                         types acting on
                                         entire stock
5......---------------.....mode          where to choose the
                                         genes from
                                         ==0 same place on
                                             diff't cands
                                         ==1 diff't places on
                                             diff't cands
                                         ==2 mix of the
                                             above two
```

3. Transcription data: **TRANSCR.IN**

```
1       20      0.05
21      40      0.05
41      60      0.05
61      80      0.05
81      100     0.05
0.01
^------^------^-----------------------------------------
TRANSCR.IN for laplacian/poisson program LAP1.FOR
Transcription Operator Data
^------^------^-----------------------------------------
[row]   [operator]   [variables]     [description]
1       swap         swazlo          lower bound of d.o.i
                     swazup          upper bound of d.o.i
                     swazsf          % genetic code to swap
2       reverse      revzlo          lower bound of d.o.i
                     revzup          upper bound of d.o.i
                     revzsf          % genetic code to reverse
3       mixing       mixzlo          lower bound of d.o.i
                     mixzup          upper bound of d.o.i
                     mixzsf          % genetic code to mix
4       copying      copzlo          lower bound of d.o.i
                     copzup          upper bound of d.o.i
```

```
                    copzsf          % genetic code to copy
5       deletion    delzlo          lower bound of d.o.i
                    delzup          upper bound of d.o.i
                    delzsf          % genetic code to delete
6       trazlev1                    %transcriptions of all
                                    types applied to entire
                                    stock
```

4. Algorithm main vaiables: **GA.IN**

```
0
0
20
100
6
6
5
0.0000001
0.0066667               10
0.0066667               100             200.0
0.05                    2
1
1
^-----------------------^-----------------
GA.IN for laplacian/poisson programs
^-----------------------^-----------------
[Row]   [Variable(s)]   [Description]
1       loadin          if ==1 then load in old parental stock
                        from bzparzn.out
2       rep             if ==1 then repeatable ran nos
3       npar            no.of parents to breed
4       nchild          no.of children to create
5       xpoint          no.of points including endpoints
                                        in the x dirn
6       ypoint          no.of points including endpoints
                                        in the y dirn
7       itermax         maximum number of iterations
8       gentol          minimum total genetic diversity
```

```
                          permitted in the breeding stock
9       jig,jignow        (jig)*(highbdry-lowbdry)
                            - maximum multiplicative jiggle
                              for entire candidates only
                              acts like a gene gcreep except
                              applied over entire candidate
                              every k*jignow'th iteration
10      wob,wobnow,       (wob)*(highbdry-lowbdry)
        wobif               - maximum multiplicative wobble
                              for genes only. acts like
                              a gene gcreep except applied
                              to all non-bdry genes in
                              candidate.  Activated when
                              after wobnow iterations or when the
                              best weight is less than wobif
11      zoomy,worzp       zoom operator: max in gcreep and do a
                          zoom e0very worzp'th times
12      twzpar            tweak operator: activated by level of
                          genetic diversity gentol. twzpar
                          parent(s) to tweak
13      dbl               Interpolation command
                          ==0, random replacement at new points
                          ==1, linear interpolation of y points
```

5. Equation variables + miscellany: **EQUATION.IN**

```
-2.0            2.0
-2.0            2.0
-5.0            5.0
^---------------^--------------^-------------
EQUATION.IN for laplacian/poisson equations
^---------------^--------------^-------------
[Row]  [Variable(s)]    [Description]
1       xzstart,xzend   start and endpoint of equation in x-dirn
2       yzstart,yzend   start and endpoint of equation in y-dirn
3       yzmin,yzmax     max and min y values
```

6. Program control:**CONTROL.IN**

```
1000.0          100.0          15.0
0.001           10.0           0.01           2.0
2000.0          100.0          15.0
0.0001          10.0           0.0001         2.0
1800.0          600.0          19.0
0.0001          13.0           0.0003         4.0
1504.0          200.0          96.0
0.0001          14.0           0.0001         5.0
1
2
^----------------------^------------------
CONTROL.IN for laplacian/poisson programs
^----------------------^------------------
[Row] [Variable(s)]   [Description]
1     limmax,tzwt   ] Level 1 resolution data
      limitn        ] limmax:: max iterations at this level
2     steep,fact,   ] tzwt:: target weight after limmax
      steep1,fact1  ] limitn:: subgoal test limit
                      steep::  steepness for distant weight
                      fact::   multiplication factor for tzwt
                               to judge how far distant but
                               desirable weight is
                      steep1:: steepness for closer weight
                      fact1::  multiplication factor for tzwt
                               to judge how far closer but
                               desirable weight is
Each following pair of rows describes the same data
but for the next resolution level.  The data is stored
in a real array cont(5,10) and any required integer values
are calculated as needed.

9       level          Which level of point resolution
                       to start with
10      levelm         Which level of point resolution
                       to end with
```

7. Profile/Weighting data:**PROFILE.IN**

```
10.0
1.0
1.0
1.0
20.0
-----------------------------------------------------
PROFILE.IN in laplacian/poisson programs
pen1            worst point weighting
pen2            sum of discretisation error weighting
dterm           not used
factor          not used
prod0           product of discretisation error
                weighting for boundary
```

Bibliography

[1] C.K. Birdsall and A.B. Langdon. *Plasma Physics via Computer Simulation.* Adam Hilger, 1991.

[2] J. Wesley D. Boucher. Modelling of ITER Operation. In *21st EPS Conference on Controlled Fusion and Plasma Physics (one page abstracts)*, 1994.

[3] L.J. Lanzerotti C.F. Kennel and E.N. Parker, editors. *Solar System Plasma Physics*, volume 2. North-Holland Publishing Company, 1979.

[4] Graeme Stewart. *Wave Propagation in Equal Mass Plasmas.* PhD thesis, University of Glasgow, Department of Physics and Astronomy, 1992.

[5] R.O. Dendy, editor. *Plasma Physics: an introductory course*, chapter 13. Cambridge University Press, 1993.

[6] C.T. Shaw. *Using Computational Fluid Dynamics.* Prentice Hall International (UK) Ltd., 1992.

[7] A.R. Mitchell and D.F. Griffiths. *The Finite Difference Method in Partial Differential Equations*, chapter 4. John Wiley and Sons, 1987.

[8] Jeffrey R. Sampson. *Adaptive Information Processing: An Introductory Survey.* Springer-Verlag, 1976.

[9] R.J. Groebner D.R. Baker and K.H. Burrell. A neural network for the analysis of DIII-D charge exchange recombination data. *Plasma Physics and Controlled Fusion*, 36(1):109–121, January 1994.

[10] K. MacPherson. Neural network computation techniques applied to solar activity prediction. *Advanced Space Research*, 13(9):447–450, 1993.

[11] T. Toffoli and N. Margolus. *Cellular Automata machines a new environment for modeling.* MIT Press, 1987.

[12] Shiyi Chen Daniel O. Martií and William H. Matthaeus. Lattice boltzmann magnetohydrodynamics. *Physics of Plasmas*, 1(6):1850–1867, June 1994.

[13] Jorg Heitkotter and David Beasley, editors. *The Hitch-hikers guide to evolutionary computation (FAQ in comp.ai.genetic).* obtained by ftp from newsgroup comp.ai.genetic, 20th June,1994.

[14] David R. Bull David Beasley and Ralph R. Martin. An Overview of Genetic Algorithms: part 2, Fundamentals. *University Computing*, 15(2):58–69, 1993.

[15] F.F. Chen. *Introduction to Plasma Physics and controlled fusion*, volume 1. Plenum Press, 2nd edition, 1984.

[16] P.C. Clemmow and J.P. Dougherty. *Electrodynamics of Particles and Plasmas.* Addison-Wesley Publishing Company, Inc., 1990.

[17] N.A. Krall and A.W. Trivelpiece. *Principles of Plasma Physics.* McGraw-Hill Book Company, 1973.

[18] W.J. Duffin. *Electricity and Magnetism.* McGraw-Hill Book Company (UK) Limited, 3rd edition, 1980.

[19] D.P. Corson P. Lorrain and F. Lorrain. *Electromagnetic fields and waves.* W.H. Freeman and Company, 3rd edition, 1988.

[20] E.W. Laing. *Plasma Physics.* Sussex University Press, 1976.

[21] T.J.M. Boyd and J.J. Sanderson. *Plasma Dynamics.* Thomas Nelson and Sons Ltd., 1969.

[22] Wilmot N. Hess. *The Radiation Belt and Magnetosphere.* Blaisdell Publishing Company, 1968.

[23] B.M. McCormac, editor. *Magnetospheric physics*, volume 44. R.Deidel Publishing Company, 1974.

[24] W.K. Peterson T.E. Eastman, L.A. Frank and W. Lennartson. The plasma sheet boundary layer. *Journal of Geophysical Research A*, 89:1553–1572, 1984.

[25] A. Nishida. *Geomagnetic Diagnosis of the Magnetosphere.* Springer-Verlag, 1987.

[26] L.A. Frank T.E. Eastman and C.Y. Huang. The boundary layers as the primary transport regions in the earth's magnetotail. *Journal of Geophysical Research A*, 90:9541–9560, 1985.

[27] R.J. DeCoster and L.A. Frank. Observations pertaining to the dynamics of the plasma sheet. *Journal of Geophysical Research A*, 84:5099–5121, 1979.

[28] U. Gebhardt and M. Kiessling. The structure of ideal magnetohydrodynamics with incompressible flow. *Physics of Fluids B*, 4(7):1689–1701, July 1992.

[29] K. Schindler and J. Birn. On the generation of field-aligned plasma flow at the boundary of the plasma sheet. *Journal of Geophysical Research*, 92(A1):95–107, November 1987.

[30] J. Birn. Stretched three-dimensional plasma equilibria with field-aligned flow. *Physics of Fluids B*, 3(2):479–484, February 1991.

[31] J. Birn. Magnetotail equilibrium theory: The general three-dimensional solution. *Journal of Geophysical Research*, 92(A10):11,101–11,108, October 1987.

[32] R. Young and E. Hameiri. Approximate magnetotail equilibria with parallel flow. *Journal of Geophysical Research*, 97(A11):16,789–16,802, November 1992.

[33] L.C. Lee and M. Yan. Structure of field-aligned plasma jets associated with magnetic reconnection. *Physics of Fluids B*, 4(11):3808–3810, November 1992.

[34] J.A. Shercliff. *A Textbook of Magnetohydrodynamics.* Pergamon Press, 1965.

[35] D.L. Book. *NRL Plasma Formulary.* published by The Office of Naval Research, 1978. aka 'the wee plasma book'.

[36] Grant R. Fowles. *Analytical Mechanics.* Saunders College Publishing, CBS College Publishing, 4th edition edition, 1986.

[37] I.A. Abramowitz M., Stegun, editor. *Handbook of Mathematical Functions.* Dover Publications, Inc., New York., 1965.

[38] D.A. Diver. Applications of genetic algorithms to the solution of ordinary differential equations. *Journal of Physics A*, 26:3503–3513, 1993.

[39] Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing: research notes in artificial intelligence.* Morgan Kaufmann Publishers, inc. by Pitman,London, 1987. chapter 4 by J.J. Grefenstette.

[40] Irwin H. Herskowitz. *Principles of Genetics.* Collier MacMillan Publisher (London), 2nd edition, 1977.

[41] Edward O. Wilson. *The Diversity of Life.* Penguin Books, 1993.

[42] David R. Bull David Beasley and Ralph R. Martin. An Overview of Genetic Algorithms: part 1, Research topics. *University Computing*, 15(4):170–181, 1993.

[43] C.Z. Janikow and Z. Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 31–36. Morgan Kaufmann, 1991. Edited by R.K. Belew and L.B. Booker.

[44] Numerical Algorithms Group. *NAG fortran library manual (mark 13).* Numerical Algorithms Group ltd, 1988.

[45] David E. Goldberg, editor. *Genetic Algorithms in search, optimization and machine learning.* Addison-Wesley Publishing Company, Inc., 1989.

[46] G.F.D Duff. *Partial Differential Equations.* Oxford University Press, 1956.

[47] E. Zauderer. *Partial Differential Equations of Applied Mathematics.* John Wiley and Sons, 2nd edition, 1989.

[48] Yuval Davidor. *Genetic Algorithms and Robotics: a heuristic strategy for optimization.* World Scientific Publishing Company, 1991.

To study, to finish, to publish.

Benjamin Franklin.

The day is done,

And I'm having fun,

I think I'm dumb,

Maybe just happy.

Kurt Cobain (1993), from 'In Utero'.