Parallel Multigrid Solution of Boundary Value Problems

Nicholas Keith Spencer

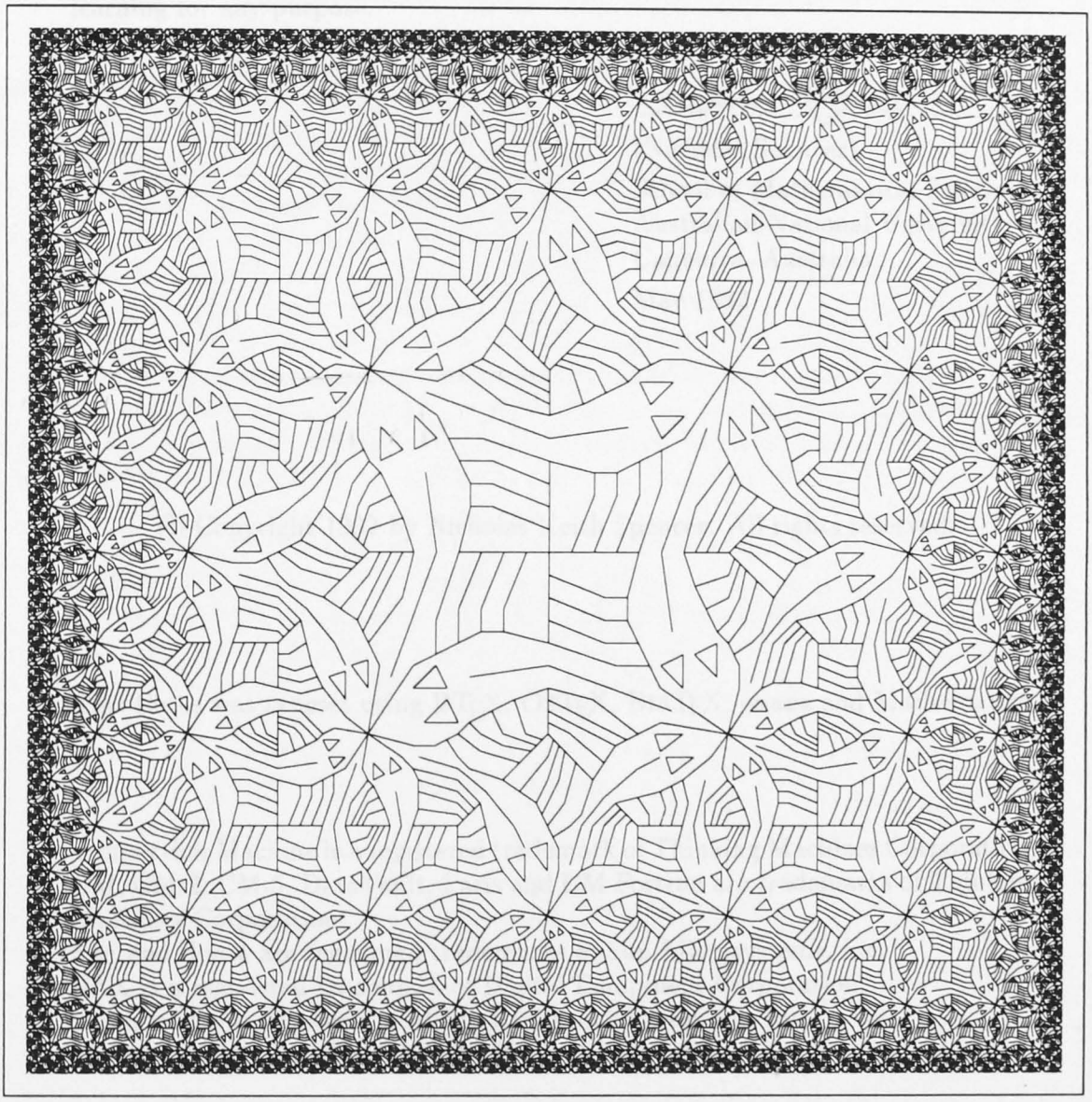
May 1992

A thesis submitted for the degree of
Master of Science (by Research)
at the Australian National University.

So, naturalists observe, a flea
Hath smaller fleas that on him prey;
And these have smaller fleas to bite 'em,
And so proceed ad infinitum.

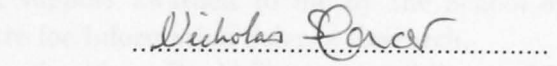
Thus every poet, in his kind,
Is bit by him that comes behind.

— Jonathan Swift, *On Poetry*



Declaration

The work presented in this thesis is my own, except where explicitly indicated. None of the work contained herein has been submitted to any other institute of learning for any purpose.



(Nicholas Spencer)
School of Mathematical Sciences
Australian National University
Canberra, Australia
May 1992

This thesis was typeset using LaTeX, OzTeX, BibTeX, `emacs` and Mathematica.

Acknowledgements

It is my pleasure to acknowledge the assistance, support and information I received from various people to help bring this research project to fruition.
I thank my supervisors Richard Brent and Stephen Roberts for accepting my studentship, and for their considerable patience.
I am very grateful for the financial support awarded to me by the School of Mathematical Sciences and the Centre for Information Science Research.
I greatly appreciate the assistance I received from David Singleton, a fellow multigridder, including the proverbial "hours of illuminating discussion."
I sincerely thank my supervisors, and especially Peter Leviton and David Singleton, for the time and effort they put into proof-reading this thesis.
Thanks also go to Peter Leviton, Matthew Spillane and Robert Bartnik for some good ideas and useful suggestions.
I am grateful to Robert Whaley and Jacek Myczkowski of Thinking Machines Corporation for their expertise on aspects of the Connection Machine.
Help on many facets of the ANU's various computer systems was provided by the system managers Zdzisław Meglicki, John Barlow, Lindsay Hood and others.
Thanks to Donald Knuth and Leslie Lamport for providing the public domain typesetting programs TEX and LATEX.
Thanks also go to Andrew Trevorrow for providing TEXnical advice, and for writing OzTEX, the public domain version of TEX for the Macintosh.
I appreciate the moral support given to me by Andrew Trevorrow, Shanti Rajagopalan, David Singleton, Matthew Spillane, Peter Leviton and my family.
Inspiration for this research was generously provided by Larry Smarr, Donna Cox and Danny Hillis.
Finally, let me wish good luck to my office-mates, Matthew Spillane and Ben Andrews, for the remainder of their PhD's.

Abstract

This thesis is concerned with the finite-difference multigrid solution of static scalar nonlinear two-dimensional elliptic Dirichlet boundary value problems, and aspects of the implementation of the multigrid algorithm on a parallel processing system, namely the Connection Machine CM-2.

The numerical solution of boundary value problems is a very important area of computational mathematics, as such problems frequently arise from the modelling and simulation of various physical systems.

The multigrid philosophy for the problems under consideration is in principle a simple one: approximations with smooth errors are efficiently obtained by applying suitable relaxation methods; because of the error smoothness, corrections to these approximations may then be calculated by projecting a corresponding equation to a coarser grid. This idea is recursively applied to a hierarchy of successively coarser grids, and leads to an asymptotically optimal iteration for the numerical solution of a wide class of partial differential equations.

Despite the excellent convergence properties of these algorithms for elliptic problems however, the multigrid process cannot be thought of as a fixed method — researchers often experiment a great deal with the many multigrid parameters available to them before obtaining suitable convergence properties. For this reason, we have written a software package called MGLAB which aims to provide a robust user-friendly "laboratory" environment, allowing the researcher to quickly and simply experiment with the multigrid solution of boundary value problems. The package is available in a standard FORTRAN 77 version for use on conventional serial computers and also in a CM Fortran version to gain the benefits of parallel data processing on the Connection Machine.

With the advent of new computer technologies, it is imperative that numerical algorithms keep up to date. The increasingly widespread use of parallel processing systems in advanced computing technologies creates a certain challenge for multigrid algorithms, for they are not readily adapted to such machines.

In this report, we discuss the fundamentals of the multigrid method for linear and nonlinear equations, including theoretical aspects which are of concern to both the multigrid programmer and practitioner. We then present numerical results obtained from MGLAB for certain model problems; these cover a range of complexity from Poisson's equation to the nonlinear geometric problem of finding surfaces of prescribed curvature. We also consider the special requirements of implementing a multigrid scheme on the Connection Machine CM-2. Finally we discuss how to generalise the problems under investigation here to the solution of time-dependent vector boundary value problems in higher dimensions.

Contents

Preface: An Essay on Three Revolutions in Science

Humanity's greatest achievement has been the invention of Science. It has completely transformed our lives, our society and our planet, and is inextricably linked with our very idea of advancement and progress. Indeed, the word *science* is derived from the Latin word meaning *knowledge*.

The ancient Greek empire provided a stable environment for the birth of a great school of early philosophers, which arose in Alexandria, Egypt. Of all the ancient peoples, it was the Greeks who left the greatest heritage to present-day science by laying the foundations of the scientific method. One of a long line of Greek scholars, Aristotle (384-322 BC) epitomised this new systematic thinking. He studied all areas of science and familiarised himself with the entire development of Greek thought preceding him. Aristotle developed deductive logic as a means of reaching conclusions. In this method, one reasons from known scientific principles in order to draw a conclusion relating to a specific case. In his books *Physics* and *Metaphysics*, Aristotle defined natural philosophy and investigated the most general and basic principles of reality and knowledge.

Aristotle and his contemporaries instigated the first scientific revolution by bringing to bear the full weight of abstract reasoning and logic upon the natural world. In fact, Aristotle went too far with this theoretic formalism by distancing himself and his school from nature itself; they tried to create a great metaphysical framework for explaining the nature of the universe, which was too abstract. He and his disciples concentrated on intellectual debate regarding form, change, elementary being and so on, while neglecting actual observation and experimentation. Although Aristotle greatly contributed to the birth of classical science, many of his deductive conclusions were false, since he based his arguments on mistaken ideas and premises, rather than on experiments.

In about 150 AD, Ptolemy developed a unified model of the universe. His theories and observations are preserved in a 13-volume work called *Mathematike Syntaxis* (*Mathematical Composition*), but which became widely known as the *Almagest* (an Arabic word meaning "the greatest") due to the admiration and acclaim that it won. Ptolemy believed that the Sun, Moon, stars and planets all moved at various speeds about a stationary Earth, which he placed at the centre of the universe.

In this same period, the Greek anatomist Galen practised medicine in Rome. Just as Aristotle had emphasised abstract reasoning, Galen similarly based his

anatomical conclusions on thoughts rather than observations, though for a different reason: dissection of human cadavers was prohibited in ancient Rome. In any event, Galen also had many mistaken ideas. Nevertheless, the schools of thought founded by Aristotle, Ptolemy, Galen and their contemporaries were accepted as authoritative throughout Europe until the sixteenth century.

The teachings of the ancient Greeks were lost to Europe in the Dark Ages, being reintroduced through the Arab empire in the 11th century. (In fact, it was Aristotle who taught the young Alexander the Great, who later carried Greek culture to Egypt, Asia Minor and Persia.)

During the Middle Ages, Europe was controlled by Church States and the development of science stagnated. Roger Bacon, an English monk living in the 1200's for example, was greatly interested in advancing scientific knowledge. He criticised the deductive method of obtaining knowledge, seeing the need for experiment, measurement and mathematics. He was imprisoned for criticising dependence on accepted authority.

In general, European scholars of that time preferred theology to the study of nature. During the thirteenth century, theologians organised the knowledge in the Greek writings so as to agree with their own religious views. Scholars saw no need for direct observation of nature. The writings of Aristotle, Ptolemy and Galen were considered Truth; to disagree with them was considered heresy.

The great scientific renaissance can be traced to precisely 1543, when two revolutionary works were published. Nicolaus Copernicus, a Polish astronomer, published *De Revolutionibus Orbium Coelestium* (*On the Revolutions of the Celestial Orbs*), which contradicted Ptolemy's model of the universe. Simultaneously and independently, the Belgian anatomist Andreas Vesalius published *De Humani Corporis Fabrica* (*On the Fabric of the Human Body*), which described his own anatomical observations, rather than merely repeating Galen's statements.

While each volume was conceived in terms of the corresponding work of classical antiquity, each was consciously novel in its approach, and proved to be revolutionary in its implications. A new philosophy — Copernicanism — was emerging, one which was rational, radical and profoundly anti-Aristotelian and anti-Ptolemaic. It also became a symbol of the struggle for free unconventional intellectual development.

Some fifty years later, Galileo Galilee made fundamental astronomical discoveries with the newly-invented telescope, and Copernicanism was publicised in a highly dramatic manner. Galileo and Francis Bacon became the patron saints of the new philosophy in England, where the scientific revolution reached its pinnacle in the work of the Royal Society and especially of its most celebrated Fellow, Sir Isaac Newton. The important ingredient of this revolution was a deep concern with methodology. A profound belief arose in the discovery of a scientific method, which was presumed to be unique and universal. The most influential writers in this regard were Descartes and Bacon, who each had a grand vision of its potential. The science historian Marie Boas Hall writes [54]:

> Both Bacon and Descartes saw natural philosophy as the most urgent and

> profitable field for investigation; both saw it yielding its secrets rapidly if the proper methods of research were applied; both insisted that the world is utterly rational, composed of nothing but matter and motion, and that only rational, anti-mystical methods of thought could therefore comprehend it.

Whereas Descartes began with self-evident principles and deduced complex metaphysical laws ("I think, therefore I am"), Bacon began with empirically-determined fact. Bacon rejected *a priori* hypotheses in favour of those based on sense experience. Further, he insisted that all hypotheses be subject to crucial deciding experiments, by which they are accepted or not.

Some time later in 1660, the Royal Society of London was formed by scientists who saw themselves as being inspired by Bacon and Galileo, men who had given them a profound belief in experiment, and a conviction of the importance of mathematical physics.

The greatest English scientist, Newton, drew the profound vision of a mechanical universe from Descartes; the experimental method from Boyle and Hooke; the distrust of hypotheses not based on empirical evidence and a fascination for induction from Bacon, and from Galileo he derived the concept of a mathematical universe to be properly described only in the language of mathematics. In this way Newton's *Philosophiæ Naturalis Principia Mathematica* (*The Mathematical Principles of Natural Philosophy*) differs from Descartes' *Principia Philosophiæ*.

Thus the great minds of the scientific revolution had formulated a mechanical universe of matter and motion, reasonable, rational, obeying fixed mathematical laws, to be ascertained by means of theory *and* experiment.

For the last three or four centuries, the scientific method has relied upon these twin paradigms of abstract thought and observation. A great deal of progress has been achieved through the methodical application of the scientific method, a process which may be summarised as follows:

1. state the problem
2. form the hypothesis
3. observe/experiment
4. interpret the data
5. draw conclusions

The third great revolution in science began in this century, although it was presaged much earlier by Charles Babbage. He saw the need for automatic, speedy and error-free calculation of such things as tables of logarithms. Babbage devised and built a mechanical calculating device called a difference engine in 1822. His later, more ambitious project of a fairly general calculating machine called an analytical engine was never completed because of financial constraints and a tool industry which lacked sufficient precision.

The true era of the computer dawned during the 1940's, when automatic computing machines were first built, a technological leap spurred on by the war effort. Indeed, these early computers were designed for specific military purposes, such as the calculation of missile trajectories, to aid in the investigation

of implosion and detonation, and to break codes and ciphers. The first electronic computer is generally recognised to be ENIAC (the Electronic Integrator and Calculator), constructed in the Moore School of Engineering, Philadelphia in 1946.

John von Neumann was an early prophet of the importance of computing to the field of science, especially mathematics. It is incredible that as early as 1946, he and Herman Goldstine wrote [46]:

> We could, of course, continue to mention still other examples to justify our contention that many branches of both pure and applied mathematics are in great need of computing instruments to break the present stalemate created by the failure of the purely analytical approach to nonlinear problems. Instead we conclude by remarking that really efficient high-speed computing devices may, in the field of nonlinear partial differential equations as well as in many other fields which are now difficult to access or are entirely denied of access, provide us with those heuristic hints which are needed in all parts of mathematics for genuine progress. In the specific case of fluid dynamics these hints have not been forthcoming for the last two generations from the pure intuition of mathematicians, although a great deal of first-class mathematical effort has been expended in attempts to break the deadlock in that field. To the extent to which such hints arose at all (and that was much less than one might desire), they originated in a type of physical experimentation which is really computing. We can now make computing so much more efficient, fast, and flexible that it should be possible to use the new computers to supply the needed heuristic hints. This should ultimately lead to important analytical advances.

It is sad that von Neumann did not live long enough to see his vision of computer-assisted science come true.

The rise of the computer to pre-eminent importance in our modern world has taken place because, just as the birth of science involved abstraction, the computer is a tool for processing information in an abstract form, and so it can be utilised for an exceedingly broad range of activities.

Within the last decade or so, it has become clear that scientific computation has progressed to a stage where it can now be considered as a new and significant part of the scientific method. This is because computer hardware, software and methodologies are now sufficiently advanced to allow scientists to experiment with physical systems in the abstract space of a computer, rather than being restricted to observations of the real world. Modern techniques in numerical analysis, simulation, modelling and visualisation now give us a new window on nature. New areas of science have recently appeared, such as fractals, chaotic systems and cellular automata. The computer serves as a new observing instrument for these newly-discovered worlds. The impact of the computer on mathematics is not limited to so-called applied mathematics and engineering; consider for example Appel and Haken's proof-by-computer of the four-colour theorem in 1976 [6]. The mathematical community has still not come to terms with the philosophical ramifications of such automated reasoning.

The ability to rapidly solve nonlinear partial differential equations allows us to study time-dependent simulation of many physical systems. These and other difficult numerically-intensive problems are becoming more accessible to scientists with the advent of parallel and other advanced computing technologies. The following list is a sample of the many and various areas of science which modern supercomputers are helping to advance:

- fluid and gas dynamics: aerodynamics, study of shocks, jets, boundary layers, recirculating flows, instability and turbulence
- N-body problems: particle and swarm simulations, galaxy formation and evolution, cosmology
- theoretical physics: gauge theories, particle and field interaction, quantum electrodynamics and chromodynamics, condensed matter research
- applied physics: properties of materials and defects, surface physics, optical fibre technology, fusion and plasma research
- astrophysics: galactic jets, gas-matter interactions, accretion disks and black holes, magnetohydrodynamics
- cellular automata: complexity theory, artificial life, statistical mechanics, bush-fire simulation
- artificial intelligence: robotics, neural networks, knowledge-based reasoning, automated theorem proving, pattern recognition
- commercial applications: database searching, industrial design
- device and circuit simulation: VLSI design and layout, propagation of electrons through semiconductors
- theoretical chemistry: biomolecular design, quantum simulations, superconductors, macroscopic phenomena
- environmental science: climate modelling, weather forecasting, ozone distribution, oceanography, tidal modelling
- structure dynamics: solid mechanics, engineering, stress and fracturing characteristics, structural mechanics, elastodynamics
- geophysics: reservoir simulation, flow through porous media, seismic analysis, geodetic networks, tectonics
- image processing: medical imaging, remote sensing, astronomical imaging, tomography, image restoration
- biology: gene sequencing, molecular geometry, drug design.

To summarise, the birth of Western science took place in the ancient Greek empire in the 300's BC, when Aristotle and other scholars developed abstract reasoning and theory as the foundation-stone of the scientific method. Little progress was made then until the 1500's, when Copernicus and others brought about a great renaissance in science. This second revolution was based on the perceived importance of observation and experiment, tied with theory and mathematics. The third great revolution in science began in the 1940's when the electronic computer was invented. By the 1980's, the speed and capability of computers enabled scientists to effectively simulate the workings of nature, creating a crucial new element of the scientific method.

Chapter 1

Introduction

Science has become a collection of mathematical theories adorned with a few physical facts. Further, if one can speak of the goal of modern scientific theory, it is to subsume all its results under one mathematical principle whose implications would describe the multifarious operations of nature.

— Morris Kline [63]

The computer revolution is the greatest challenge facing mathematics in the coming years. Mathematicians have set great stock in abstract mathematics, in which concepts and rigour have been the dominant things, but now algorithms are really important.

— Albert Tucker [71]

1.1 Overview

In this research report, we will be concerned with the finite-difference multigrid solution of nonlinear elliptic boundary value problems (BVP's), and consider the implementation of multigrid algorithms on the Connection Machine CM-2, a parallel processing supercomputer in relatively widespread use throughout the scientific world. To keep matters reasonably uncomplicated, we generally discuss only time-independent scalar two-dimensional partial differential equations, with Dirichlet boundary conditions on the unit square. This will enable the exposition to be clear and concise; we consider extensions to this class of problem in Chapter 9. We will discuss the fundamentals of the multigrid method for linear and nonlinear equations, presenting results for certain model problems obtained from a new and original multigrid software package. This package is implemented on both standard serial computers and on the CM-2. We look at issues arising from serial and parallel implementation of multigrid algorithms. We also consider how these numerical results reflect upon the structure of the multigrid iteration.

Many problems of mathematical physics lead to boundary value problems involving partial differential equations of elliptic type. This type of equation, to be carefully defined in Chapter 2, is characterised by *global* properties of its solution: the solution at every point depends upon the solution at every other point in the domain. The physical law behind this "collective behaviour" is usually a force field of some type, where the forces (instantly propagated) depend on the solution and vice versa. In contrast, hyperbolic equations have a diminished sense of collectiveness: their behaviour is much more local. These underlying properties mean that numerical techniques for solving the various types of equations are likewise different.

Elliptic BVP's arise in areas such as fluid dynamics, electromagnetics, solid state and materials science, general relativity, applied mechanics and engineering. These areas encompass many different types of physical processes, for example vibration, conduction, convection, diffusion, particle and field interaction, stress and fracture. The challenge to the computational mathematician is often not merely to solve such a problem, but to do so rapidly and efficiently, thus allowing researchers to study the behaviour of parabolic systems.

The last decade has seen the evolution of a major new technique to tackle some of these difficult problems in scientific computation. Multigrid methods have proven to be powerful PDE-solvers, although multigrid has not yet become a mature field of numerical analysis. In particular, convergence theory lags behind our practical knowledge of multigrid algorithms, which deliver results at speeds comparable to any other numerical method. Nevertheless, it is now well-known that multigrid is an optimal $O(N)$ solver (for problems involving N unknowns) for a wide class of PDE's, including the class of well-behaved discrete elliptic boundary value problems [17, 51, 93].

The past history of numerical analysis and computation has been marked by a sharp division into the finite-difference and the finite-element fields. Multigrid can be viewed as a fairly general acceleration technique, one which is applicable to both these fields. (For an early discussion of multigrid methods for finite elements, see McCormick and Ruge [78].) Moreover, multigrid and *multilevel* techniques in general are now being applied to a very broad range of numerical problems (see Brandt's 1988 Weizmann Report [18]), such as:

- inverse problems,
- integral equations,
- global optimisation,
- n-body interactions,
- linear programming,
- fast Fourier transforms,
- combinatorial optimisation,
- nonlinear non-elliptic PDE's,
- behaviour of statistical fields,

- determinants of systems of equations, and
- approximation of piecewise-smooth functions.

To quote from Brandt [18]:

> Multilevel computations have evolved into an independent discipline, interacting with other computational methodologies, it has its own internal development, gradually increasing the understanding of the many types of multi-scale interactions, their modes of operation and domains of application. The research exhibits the deep interdisciplinary and cross-fertilising role of applied mathematics, in that various underlying relations and algorithmic ideas are carried back and forth between widely varying areas of applications. It is thus quite beneficial that an advanced research group works widely across this discipline, not too limited to some specific applications.

Multilevel solvers can even be constructed when problems have no explicit geometric basis. In these algebraic multigrid (AMG) solvers, we require fine-level variables to be "strongly connected" by the fine-level equations to at least some coarse-level variables. The inter-level transfers may also be based purely on algebraic equations. For more details, see Ruge and Stüben [87].

A revolution in computer science occurred in 1985 with the announcement of the Connection Machine by Thinking Machines Corporation [56]. This was the first computer system to use massive parallelism to achieve supercomputer status. Designed specifically to study artificial intelligence, the CM-1 utilised tens of thousands of primitive single-bit processors in a highly connected network. The hope was to imitate the architecture of the human brain. Ironically, much more interest in the Connection Machine was aroused in numerical fields than in artificial intelligence. We classify the operation of this machine as SIMD (single instruction, multiple data), meaning that identical instructions are performed on all processors in unison, on data sets local to each processor. This classification contrasts with MIMD (multiple instructions, multiple data) machines in which processors execute a local sequence of instructions (see Chapter 8 for further details). The SIMD CM-1 computer was found to be ideal for solving finite-difference and finite-element problems on regular grids, in particular.

Thinking Machines Corporation responded by introducing the CM-2 in 1987, which included double-precision floating-point units, thus giving computational scientists potential gigaflop performance. At the present time, the great majority of applications that are run on the 80 or so Connection Machines throughout the world are mathematical in nature. The remaining applications tend to fall into the two smaller categories of artificial intelligence and database applications [57].

Thinking Machines Corporation has recently announced the CM-5 [98]. It is intended to be a universal computer (that is, dual MIMD/SIMD), operating on "cooperative" principles, and is designed to be scalable to the teraflop performance level (trillions of floating-point operations per second). Up until now, parallel computer users have been forced to choose between MIMD machines, which are good at independent branching but poor at synchronisation and com-

munication, and SIMD machines, which have complementary characteristics. The CM-5 is claimed to combine the best of both these architectures, while taking advantage of the latest developments in VLSI, compiling methods, RISC processors, networking and so on.

Apart from massively-parallel systems, there are many other types of high-performance computer available to researchers. These include machines constructed from extremely advanced solid-state technologies, relying on one or a handful of CPU's with vector pipelines (*eg.* Cray Y-MP and Fujitsu VP). Another class of supercomputer is the moderately-parallel system which employs tens or hundreds of processors to share the computational work (*eg.* Intel iPSC and NCUBE). We will say more about parallel processing systems in Chapter 8.

With the advent of new computer technologies, it is necessary for numerical algorithms to be kept up to date. The increasingly widespread use of parallel processing systems in supercomputers creates a certain challenge for multigrid algorithms, for they are not readily adapted to SIMD machines. On the other hand, finite-difference methodologies are well suited to such an architecture.

Multigrid is a relatively recent numerical method, which presents the numerical practitioner with a large choice of parameters. At this stage of development, multigrid cannot be considered a fixed method. The optimal value of the multigrid parameters is often not known in advance, hence it is common for a certain amount of experimentation to take place, in order to obtain suitable convergence properties. In a survey article of 1991, Frederickson, McBryan, Stüben, Trottenberg *et al* [75] wrote:

> For a wide class of problems in scientific computing, in particular for partial differential equations, the multigrid (more generally, the multilevel) principle has proved to yield highly efficient numerical methods. However, the principle has to be applied carefully; if the multigrid components are not chosen appropriately for the given problem, the efficiency may be far from optimal. This has been demonstrated for many practical problems. Unfortunately, the general theories on multigrid convergence do not give much help in constructing really efficient multigrid algorithms, though some progress has been made in bridging the gap between theory and practice during the last few years. Research in finding highly-efficient algorithms for nonmodel applications therefore is still a sophisticated mixture of theoretical considerations, transfer of experiences from model to real-life problems, and systematic experimental work.

One of the goals of this research project was to write a software package for the CM-2 suitable for solving arbitrary PDE's in the class specified above, using the multigrid method in a "laboratory-style" environment. This would provide a suitable platform for experimenting with the various multigrid parameters, in contrast to a "black-box" solver. For this reason, we have sometimes sacrificed performance for the sake of flexibility and ease-of-programming. This has resulted in the user's ability to program and solve PDE's from scratch in as little as ten minutes. We have also emphasised user-friendliness, to a degree that the user is

easily able to perform a thorough parameter-space investigation.

This software package, called MGLAB, was initially written in standard FORTRAN 77, then ported to the Connection Machine using the CM Fortran language. Results will be presented for various model problems which were obtained from serial machines as well as from the CM-2 located at the Australian National University. These problems begin with simple linear constant-coefficient equations, and progress to more interesting nonlinear geometric problems. We will relate these results to the multigrid process and to considerations of parallel multigrid implementation.

1.2 A Brief History of Multigrid

Virtually every problem of mathematical physics leads naturally to solving one or more functional equations of the form

$$\mathcal{A}\,u = f, \tag{1.1}$$

where $\mathcal{A}$ is an operator from some space $\mathcal{X}$ into some space $\mathcal{Y}$, f is given in $\mathcal{Y}$, and u is the desired solution in $\mathcal{X}$. Examples of such classes of problems are ordinary differential equations, partial differential equations and integral equations. In general, it is not possible to determine the solution explicitly, or its explicit form may be so complicated as to be practically useless, so that one is interested in an approximate solution of the equation.

The standard numerical procedure for solving the problem (1.1) is to first discretise the problem, using some discretisation parameter h, thereby constructing approximating algebraic equations on finite-dimensional approximation subspaces $\mathcal{X}_h$ and $\mathcal{Y}_h$, and then to create some numerical process to approximately solve this system of discrete equations, which we write as

$$\mathcal{A}_h u_h = f_h \tag{1.2}$$

In the case of differential equations, such discretisation algorithms typically give rise to a large sparse system of algebraic equations. If this system is linear then we write it in the matrix form $A_h u_h = f_h$.

The study of numerical analysis investigates the following fundamental questions which immediately arise (see for example [94]):

1. Do the discrete equations (1.2) converge to the continuous equations (1.1) as $h \to 0$?
2. Is existence and uniqueness guaranteed for the approximate equations (1.2)?
3. Equations (1.2) may be nonlinear or otherwise difficult. Can we develop an algorithm to solve them?
4. Will this process be stable, convergent and consistent? (See Chapter 2 for definitions of these three terms.)
5. How do we implement an efficient algorithm on a computer (in terms of CPU time, memory usage, *etc*)?

The last few decades has seen an enormous amount of research into the problem of numerically solving such systems. Typically these algorithms fall into two classes: *direct* and *iterative* methods.

Direct methods calculate the solution exactly (to within machine precision) in a finite number of algorithmic steps, an amount of computation that can be determined in advance. The archetypal direct method is Gaussian elimination; other methods are based on FFT's (fast Fourier transforms), cyclic reduction or special properties of certain classes of matrices. Applied to a dense problem on an $n \times n$ grid, these algorithms require $O(n^2 \log n)$ arithmetic operations, and so approach the optimal $O(n^2)$ operation count. Unfortunately, direct methods are rather specialised, are best applied to separable self-adjoint BVP's, and can break down in other cases. They also require inordinate amounts of computer memory when applied to large dense problems.

In contrast, iterative (or *relaxation*) methods start from an initial guess and proceed to obtain more and more accurate approximations from some computational cycle. This iteration may be repeated as often as necessary to achieve a desired accuracy, hence the amount of computation depends on the required accuracy. This sequence of simple updating iterations converges to the numerical exact solution of the system (that is, within machine precision). The prototypes of this class are the Jacobi and Gauss-Seidel methods of relaxation. While direct methods can be applied to any non-singular problem, iterative methods are generally more efficient for multidimensional problems; moreover, they are often better-suited to computer solution, as they consist of a repetition of simple steps. They also take good advantage of the sparse nature of the finite-difference system of equations, and may require no computer memory in addition to the storage of the discretised domain. The two classic references on iterative methods are Varga [101] and Young [105].

Relaxation suffers from two deficiencies, however, which are manifestations of the same underlying property. (These are discussed in detail in Chapter 2.) The first is that the convergence factor is $1-O(h^2)$ or $1-O(h)$, where h is the size of the grid or mesh; hence convergence deteriorates as the grid is refined. This is unfortunate, as accurate results require the use of fine grids, especially in regions where there are small-scale changes. The second difficulty is that smooth (*ie.* long frequency) error modes are very slowly reduced. Hence relaxation quickly stalls once high-frequency errors are removed. Multigrid is a hybrid process which overcomes these difficulties. It combines the following three elements:

1. relaxation to obtain smooth errors,
2. calculation of corrections on increasingly coarse grids, and
3. incorporation of *nested iteration* (using coarser grids to obtain good initial approximations on finer grids).

Multilevel numerical processes have been independently devised by many investigators. While the above three processes have been separately known for many years, it is only the multigrid combination which results in an optimal iteration.

As early as 1935, Southwell [90, 91] recognised that coarser grids hold computational benefits. In [91] he describes a process whereby a problem is first (approximately) solved on a coarse grid (involving a much smaller amount of computational work), then this solution is interpolated to a finer grid, serving as a good initial guess for subsequent relaxation on that grid. In his words:

> Our techniques will provide for steadily increasing accuracy, attainable at the cost of proportionally increased labour; and to this end they will utilise results obtained on one size of net as a starting assumption in relation to a net of smaller mesh. This device will be termed *advance to a finer net.*

The natural next step is to apply this idea to successively finer grids. In 1954, Allen [29] wrote (still using arcane language):

> Successively finer nets are always chosen, each to have strings one-half as long as the previous (coarser) net. The solution should always be worked first on the coarsest net, for it will contain the fewest nodes, and the liquidation process will be correspondingly easier ... Indeed, in some problems it may be that less work is involved in the end if the solution is first found on the coarsest possible net, followed by several advances to successively finer nets, until sufficient accuracy is achieved eventually on a very fine net.

The modern term for this technique is nested iteration.

Coarse-grid acceleration techniques were used in the 1950's and 1960's by Stiefel [92], Fedorenko [37], Wachspress [102] and de la Vallée Poussin [30], for example. These were all two-grid methods.

A more advanced idea is to consider how the current fine-grid approximation can be improved by further references to the coarse grid problem, a procedure which exploits the "proximity" between the fine and coarse grids.

Brandt [15] and Stüben and Trottenberg [93] give a description of the content of two pivotal Russian papers, published in the 1960's. In 1964, Fedorenko [38] introduced the multigrid method in a narrow sense by introducing the idea of a nested hierarchy of grids (that is, using elements (1) and (2) above), so as to prove a theoretical result. In the case of Poisson's equation on a rectangular grid containing N points, using the standard five-point discretisation, Fedorenko proved that the number of operations required to reduce the residuals by a factor ϵ is $O(N\,|\log\epsilon|)$. This is asymptotically the optimal result, although the constants in his estimate are very large; in fact they are four orders of magnitude larger than those we find in practice. Fedorenko also proved that multigrid has a rate of convergence bounded by some number less than unity, which is independent of the grid size h. This is a remarkable result, one which highlights the optimal nature of the multigrid method.

Bakhvalov [9] later generalised this result to any second-order elliptic PDE with continuous coefficients; but derived even larger constants in his theoretical bounds. Because these estimates were far worse than those of other methods, the multigrid idea of Fedorenko languished for some time. He himself did not seem to realise the true practical potential of the method, even though he also indicated the possibility of combining all three multigrid elements.

It was not until 1973 that the first full multigrid algorithms and results were published by Brandt [14], who seems to be the first person to have recognised the true practical efficiency of multigrid. Moreover, he introduced a general two-pronged method called the multilevel adaptive technique (MLAT) for solving partial differential boundary value problems. MLAT consists of adaptive discretisation and "multilevel iterative procedures in which coarser grids constantly participate in solving the equations on finer grids," that is, multigrid methods. (For more information on adaptive grids, see [48, 76, 79].) Over the next decade, more and more researchers became enthusiastic about multigrid, noting in particular the very high rates of convergence that can be achieved in practice.

In 1975 Hackbusch independently re-invented the elements comprising multigrid. He began to systematise convergence analyses of general multigrid methods, a process which continued for some years [49, 50, 51]. In 1977, Brandt published what many regard as the seminal paper in the field, *Multi-level Adaptive Solutions to Boundary Value Problems* [15]. As the method was developed and refined in the 1980's, it was seen how to apply multigrid and multilevel techniques to diverse areas of computational mathematics, such as finite-element problems, vector-valued BVP's, eigenvalue and bifurcation problems, parabolic and other time-dependent PDE's, hyperbolic problems in fluid dynamics (transonic flow, shocks, full Navier-Stokes equations), singular perturbation phenomena (see [16, 61]), and integral equations. With this increasing interest, the number of papers published on multigrid methods snowballed in the last decade. At the present time we may describe multigrid as a maturing new field of computational mathematics, one which holds great promise for certain types of problems facing the researchers of today.

Good introductions to multigrid may be found in Briggs [20] and Jespersen [58].

1.3 Thesis Outline

Chapter 2 introduces definitions and notation used in this thesis. It discusses the basic material required for subsequent chapters, in particular the fundamental concepts of finite-difference discretisation and relaxation. Various methods of relaxation are discussed, along with their convergence rates and other properties. This material comprises the classical techniques for the numerical solution of partial differential equations.

Chapter 3 presents the basic ingredients of the multigrid method for linear equations: intergrid transfer, coarse-grid correction and nested iteration. We consider theoretical smoothing and convergence rates and how these vary with certain parameters. The multigrid elements are brought together to form a synergistic union. Various types of cycles and other multigrid options are presented, giving rise to a large multigrid parameter space.

Chapter 4 reports on the results of applying the multigrid method to our model linear problems, which are chosen to be representative of their type. These

results come from our "laboratory-style" multigrid software package MGLAB. This gives us the opportunity to investigate the effect of the multigrid parameters on convergence properties of different types of problems. The standard FORTRAN 77 implementation of MGLAB and the structure of the package is discussed, as are aspects of debugging and algorithm correctness.

Chapter 5 is concerned with the modifications we make to the linear multigrid process in order to solve nonlinear PDE's — the so-called full approximation scheme (FAS). We extend the concepts discussed in Chapter 3 to the nonlinear case, and look at the implementation of nonlinear multigrid in MGLAB.

Chapter 6 presents the results of our multigrid solution of some model nonlinear problems. We make similar observations and interpretations to those in Chapter 4 for these nonlinear equations.

Chapter 7 considers some interesting nonlinear problems from geometry, namely surfaces of prescribed curvature. These equations are derived from physical models of soap films and bubbles. After discussing some theoretical considerations, we present results obtained using MGLAB. A theorem of Serrin provides a convenient sharp test for our numerical method.

Chapter 8 discusses the implementation details for MGLAB on the Connection Machine CM-2 parallel supercomputer. Relevant aspects of the CM-2's architecture are presented, in particular, the hypercube communications network. Performance and timing results are given for serial codes on a selection of conventional computers, in addition to the CM Fortran version of MGLAB. These results are interpreted to give insights into aspects of machine architecture and efficient implementation thereon.

Chapter 9 describes extensions to the material presented in the previous chapters: we discuss various types of parallel multigrid schemes, including Frederickson and McBryan's PSMG method; time-dependent problems; parametric PDE's; systems of equations; PDE's with Neumann, mixed and periodic boundary conditions; and BVP's on Riemannian manifolds. We also briefly consider finite-volume methods in the multigrid context.

Chapter 10 summarises the material presented in this thesis, and draws conclusions regarding the multigrid solution of boundary value problems.

Appendix A contains a listing of `mg10.fcm`, the CM Fortran implementation of Problem MG10 (see Chapter 6), Poisson's equation with zero boundary conditions. It gives an example of a driver program, which combines with the back-end multigrid library package MGLAB. It demonstrates which subroutines are required to be written by the user when solving a nonlinear problem.

Appendix B contains sample output for Problem MG10 using `mg10.fcm` and MGLAB run on the Connection Machine CM-2.

Appendix C contains a partial listing of `mglab.fcm`, the CM Fortran implementation of the multigrid library MGLAB. A complete listing was not feasible due to its length, nor desirable since much of the code is of little interest. We list the kernel routines essential to the multigrid process: relaxation, restriction, prolongation, coarse-grid correction, *etc.* These demonstrate the ease of programming in the CM Fortran language, given a simple multigrid implementation.

Chapter 2

Fundamental Concepts

Relaxation can be deceptive. It can look as if you aren't doing much at all.

— Bob Montgomery and Linda Evans [81]

In this chapter, we give an overview of the fundamental ideas used in the classical numerical solution of partial differential equations. We begin with some basic definitions and terminology, followed by a discussion of finite-difference discretisation. We then consider various relaxation methods for solving boundary value problems, examining their rates of convergence and other properties. Relaxation methods are important because these iterations form the basis of multigrid algorithms. Indeed, many of the multigrid parameters arise from the various relaxation techniques.

2.1 Definitions and Problem Statement

This research report is concerned with the solution of time-independent (possibly nonlinear) elliptic boundary value problems. We will firstly define the concept of ellipticity for linear, quasilinear and fully-nonlinear equations.

A k^{th}-order partial differential equation in n independent variables is a relation of the form

$$F(\mathbf{x}, u, Du, D^2u, \ldots, D^k u) = 0 \tag{2.1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \boldsymbol{R}^n$, $u : \boldsymbol{R}^n \to \boldsymbol{R}$ and

$$Du = (D_1 u, D_2 u, \ldots, D_n u) \equiv \left(\frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \ldots, \frac{\partial u}{\partial x_n} \right).$$

The general form of a *second-order linear* PDE in n independent variables is [44]

$$\sum_{i,j=1}^{n} \alpha_{ij}(\mathbf{x}) \frac{\partial^2 u(\mathbf{x})}{\partial x_i \partial x_j} + \sum_{i=1}^{n} \beta_i(\mathbf{x}) \frac{\partial u(\mathbf{x})}{\partial x_i} + \gamma(\mathbf{x})\, u(\mathbf{x}) = f(\mathbf{x}),$$

which we write more concisely as

$$\alpha^{ij} D_{ij} u + \beta^i D_i u + \gamma u = f. \tag{2.2}$$

In this notation, a k^{th}-order linear PDE is of the form

$$\alpha_1^{i_1,i_2,\ldots,i_k} D_{i_1,i_2,\ldots,i_k} u + \alpha_2^{i_1,i_2,\ldots,i_{k-1}} D_{i_1,i_2,\ldots,i_{k-1}} u + \ldots + \alpha_k u = f.$$

By definition, the second-order linear PDE of equation (2.2) is *elliptic* in some region $\Omega \in \boldsymbol{R}^n$ if the coefficient matrix $[\alpha^{ij}(\mathbf{x})]$ is positive definite in Ω. In the case of two independent variables (x, y), this second-order PDE reduces to an equation of the form

$$a u_{xx} + b u_{xy} + c u_{yy} + d u_x + e u_y + g u = f \tag{2.3}$$

and so this equation is

$$\text{elliptic} \qquad \text{if} \quad 4ac - b^2 > 0.$$

In addition, we say that equation (2.3) is

$$\begin{aligned} &\text{parabolic} \quad \text{if} \quad 4ac - b^2 = 0, \text{ and} \\ &\text{hyperbolic} \quad \text{if} \quad 4ac - b^2 < 0. \end{aligned}$$

For higher-order linear PDE's of even order, the ellipticity condition $[\alpha^{ij}] > 0$ is generalised to the positivity of the contraction of the highest-order tensor:

$$\alpha^{i_1,i_2,\ldots,i_m,j_1,j_2,\ldots,j_m} \xi_{i_1,i_2,\ldots,i_m} \xi_{j_1,j_2,\ldots,j_m} > 0 \qquad \forall \quad \text{non-zero tensors } \xi_{i_1,i_2,\ldots,i_m}.$$

The prototypes for the class of linear elliptic PDE's are Laplace's equation

$$\Delta u \equiv D_{ii} u = 0$$

and Poisson's equation $\Delta u = f$.

A PDE is said to be *quasilinear* if the functional F in equation (2.1) is linear in the highest order derivatives [59]. A second-order equation is therefore quasilinear if it can be written in the form

$$\alpha^{ij}(x, u, Du)\, D_{ij} u + \beta(x, u, Du) = 0.$$

One of the most widely known quasilinear elliptic PDE's is the minimal surface equation (see Chapter 6).

A *general* second-order equation on a domain $\Omega \in \boldsymbol{R}^n$ can be written in the form [44]

$$F(\mathbf{x}, u, Du, D^2 u) = 0, \tag{2.4}$$

where F is a real function on the set $\Gamma = \Omega \times \boldsymbol{R} \times \boldsymbol{R}^n \times \mathcal{S}^n$, where $\mathcal{S}^n$ is the linear space of real symmetric $n \times n$ matrices. Suppose a point $\gamma = \gamma(\mathbf{x}, u, v, w)$ lies in Γ, where $\mathbf{x} \in \Omega$, $u \in \boldsymbol{R}$, $v \in \boldsymbol{R}^n$ and $w \in \mathcal{S}^n$, then equation (2.4) is quasilinear

when F is an affine function of the w variables; otherwise it is fully-nonlinear. Hence the general second-order equation is elliptic if the matrix $[F_{ij}(\gamma)]$ given by

$$F_{ij}(\gamma) = \frac{\partial F(\gamma)}{\partial w_{ij}}$$

is positive definite.

It is a simple matter to map any quadrilateral domain to the unit square by an affine transformation (perhaps as part of transforming the problem to dimensionless form); because of this and the fact that ellipticity, linearity and quasilinearity are preserved in such as transformation, we can restrict ourselves to considering PDE's on the unit square. Further, we generally consider only *Dirichlet* boundary value problems, where function values are prescribed on the boundary of the domain. Our problem is therefore to find the solution $u = u(x, y)$ to the BVP

$$\begin{aligned} \mathcal{A}u &= f \quad \text{in} \quad \Omega = [0,1] \times [0,1] \\ u &= g \quad \text{on} \quad \partial\Omega \end{aligned} \tag{2.5}$$

where $u \in C^{2m}(\Omega)$, $f \in C^0(\Omega)$, $\mathcal{A} : C^{2m}(\Omega) \to C^0(\Omega)$ is an elliptic operator, and $g \in C^{2m}(\partial\Omega)$.

2.2 Discretisation

Equation (2.5) is of course the continuous problem. To enable a numerical solution to this problem, we now proceed with the standard finite-difference discretisation, whereby a uniform grid of horizontal spacing h and vertical spacing k is superimposed on the domain, producing a subspace Ω_h. (We will be using the subscript h as a generic discretisation parameter.) Defining $M = 1/h$ and $N = 1/k$, we construct discrete functions in the following manner:

$$f_{ij} = f(ih, jk) \qquad \text{for} \quad i = 0, 1, \ldots, M; \quad j = 0, 1, \ldots, N.$$

The standard finite-difference formulation replaces continuous variables by their values restricted to the grid points of

$$\Omega_h = \big\{(ih, jk) : i = 0, 1, \ldots, M;\ j = 0, 1, \ldots, N\big\},$$

and replaces continuous derivatives by central difference expressions obtained from truncated Taylor series.

We will use the notation

u	for the continuous exact solution,
u_h	for the discretised exact solution: $u\|_{\Omega_h}$,
v_h	for the exact solution of the discrete problem, and
$\tilde{v}_h$	for an approximate solution of the discrete problem.

We are now required to solve an $(M+1)(N+1) \times (M+1)(N+1)$ system of algebraic equations. The discrete problem is therefore to find the solution v_h to

$$\begin{aligned} \mathcal{A}_h v_h &= f_h \quad \text{in} \quad \Omega_h \\ v_h &= g_h \quad \text{on} \quad \partial\Omega_h \end{aligned} \tag{2.6}$$

where $v_h, f_h \in \mathcal{S}_h$ (an appropriate space of grid functions on Ω_h), $\mathcal{A}_h : \mathcal{S}_h \to \mathcal{S}_h$ is a discrete elliptic operator, and $g_h \in \mathcal{G}_h$ (an appropriate space of grid functions on $\partial\Omega_h$).

Since the PDE's considered here lie on the unit square, it is usual to use an *isotropic* grid ($M = N$ and $h = k$), unless the PDE itself has some preferred direction, as does an equation with boundary-layer properties, for example.

Taylor's theorem and the associated Taylor series are one of the most important tools in numerical analysis. It allows the approximation of a continuous function $f(x)$ by n^{th}-order polynomials, also giving an estimate of the *truncation* error thus produced.

Theorem 2.1 (Taylor) *If $f(x) \in C^{n+1}[a,b]$ and $x, x_0 \in [a,b]$, then*

$$f(x) = f(x_0) + \sum_{k=1}^{n} \frac{(x-x_0)^k}{k!} f^{(k)}(x_0) + \frac{(x-x_0)^{n+1}}{(n+1)!} f^{(n+1)}(\xi)$$

for some $\xi \in [x_0, x]$ if $x_0 < x$, or $\xi \in [x, x_0]$ if $x < x_0$.

An elementary application of Taylor's theorem leads to the following approximations for the first-order partial derivatives in the x-direction, along with the indicated truncation error:

$$u_x\Big|_{ij} \equiv \frac{\partial u}{\partial x}\Big|_{ij} = \begin{cases} M(u_{i+1,j} - u_{ij}) + O(h) & \text{(forward difference)} \\ \frac{1}{2}M(u_{i+1,j} - u_{i-1,j}) + O(h^2) & \text{(central difference)} \\ M(u_{ij} - u_{i-1,j}) + O(h) & \text{(backward difference)} \end{cases}$$

The following is a list of the most compact central finite-difference approximations in two dimensions up to second-order derivatives.

$$\begin{aligned} u_x\Big|_{ij} &= \tfrac{1}{2}M\,(u_{i+1,j} - u_{i-1,j}) + O(h^2) \\ u_y\Big|_{ij} &= \tfrac{1}{2}N\,(u_{i,j+1} - u_{i,j-1}) + O(k^2) \\ u_{xx}\Big|_{ij} &= M^2\,(u_{i+1,j} - 2u_{ij} + u_{i-1,j}) + O(h^2) \\ u_{yy}\Big|_{ij} &= N^2\,(u_{i,j+1} - 2u_{ij} + u_{i,j-1}) + O(k^2) \\ u_{xy}\Big|_{ij} &= \tfrac{1}{4}MN\,(u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}) + O(h^2+k^2) \end{aligned}$$

In addition, one can readily apply Taylor's theorem to obtain more accurate (higher-order) finite-difference approximations.

During the process of discretisation, central differences are typically used to replace the continuous derivatives, rather than forward or backward differences,

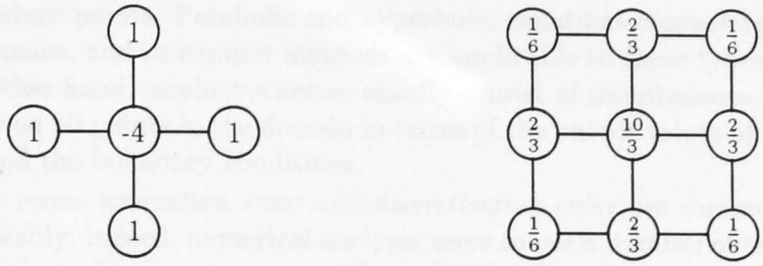


Figure 2.1: Two common computational molecules for Δ_h on isotropic grids.

unless the characteristics of the problem suggest otherwise. (The use of forward or backward differences is often desirable in solving fluid-flow problems; in that field, these processes are commonly called *upwinding* and *downwinding* respectively.)

A common method of defining finite-difference operators is to draw so-called computational *molecules* (also known as *stars* or *stencils*). Figure 2.1 shows the standard five-point and nine-point computational molecules for the discrete Laplacian operator on an isotropic grid. The five-point molecule has truncation error $O(h^2)$, while the nine-point star is accurate to $O(h^4)$.

Application of the isotropic five-point star to Poisson's equation in two dimensions gives us the discrete problem

$$N^2\,(v_{i+1,j} - 2v_{ij} + v_{i-1,j}) \;+\; N^2\,(v_{i,j+1} - 2v_{ij} + v_{i,j-1}) \;=\; f_{ij}$$

$$\text{for} \quad i,j = 1,2,\ldots,N-1$$

This represents the system of $N^2 \times N^2$ linear equations (with appropriate boundary equations)

$$v_{i+1,j} \;+\; v_{i-1,j} \;+\; v_{i,j+1} \;+\; v_{i,j-1} \;-\; 4v_{ij} \;=\; h^2 f_{ij} \tag{2.7}$$

which we write as $A_h v_h = f_h$.

Note that for $f \equiv 0$, the solutions to the finite-difference equations retain two important properties of the harmonic function u: the mean value property ($u(x,y)$ is equal to the arithmetic average of u at points in an arbitrary neighbourhood of (x,y)) and the maximum principle (extreme values of u occur on the boundary $\partial\Omega$).

An elliptic PDE will generally give rise to a matrix $\mathcal{A}_h$ which is *diagonally dominant*:

$$a_{ii} \;\geq\; \sum_{j=1,\,i\neq j}^{N^2} |a_{ij}| \qquad \text{with strict inequality for some } i.$$

This property is a consequence of the discretisation process for elliptic equations.

Finite-difference schemes are often divided into *explicit* and *implicit* categories. An explicit algorithm is a non-iterative "marching" process designed to obtain the solution at some current set of points in the domain (usually those corresponding to a particular time-step) in terms of the known preceding points and

the boundary points. Parabolic and hyperbolic equations characteristically have open domains, and so explicit methods are applicable to these types of problem. On the other hand, implicit schemes usually consist of simultaneous iterative calculations on all points in the domain in terms of the known values at the previous iterate and the boundary conditions.

The terms *truncation error* and *discretisation error* are commonly used interchangeably; indeed, numerical analysts seem to use a number of slightly different definitions of truncation error. Wasow [103] defines the discretisation error to be the combination of truncation errors arising from the Taylor series approximation of the continuous functions and from the finite-difference approximation of the boundary conditions. However, it is clear that the latter contributes nothing in the case of Dirichlet boundary conditions.

In addition, computed solutions always introduce *round-off error*, since calculations can only proceed to a finite number of binary or decimal places. Discretisation error is proportional to the interval size h, whereas round-off error is generally inversely proportional to h. For this reason we cannot assert that decreasing the grid size always increases the accuracy.

Finally, there are three terms which arise in the general investigation of numerical algorithms which should be explained (see for example [83, 94]): *convergence*, *consistency* and *stability*. A solution to a finite-difference equation which approximates a given PDE is said to be convergent if, at each grid point, the finite difference solution approaches the true solution of the PDE as the grid spacing approaches zero:

$$v_h \to u \quad \text{as} \quad h \to 0,$$

in other words, the discretisation error $u - v_h$ vanishes in the limit.

A finite-difference equation is said to be consistent with a PDE if, at each grid point, the discrete equation itself becomes identical to the PDE as the grid spacing approaches zero. The truncation error at some grid point can be defined as the difference between the discrete equation with the exact values u_h substituted for v_h, and the PDE; therefore consistency means that the truncation error vanishes in the limit.

The third important feature of a finite-difference method is the stability of the discrete equation, and of the algorithm which attempts to solve it; that is, the growth or decay of errors introduced by values previously calculated. Given a direct method or a converging iterative method, the numerical solution to the finite-difference equations is not v_h but $\tilde{v}_h$, since no computer has infinite-precision arithmetic. A finite-difference system is said to be stable if the cumulative effect of all round-off errors is negligible. To be more precise, let δ be the maximum round-off error committed during some numerical procedure, then this procedure is stable if the cumulative departure of this solution from the error-free solution tends to zero as $\delta \to 0$ and is bounded by some multiple of h^{-1} as $h \to 0$.

The following theorem provides a link between these three fundamental concepts. It concerns so-called *well-posed* problems, meaning problems which have a unique solution which depends continuously on the initial data.

Theorem 2.2 (Lax) *Given a well-posed linear initial value problem and a consistent finite-difference approximation to it, stability is the necessary and sufficient condition for convergence.*

Although restricted to linear problems, the Lax Equivalence theorem is important as it is often easier to prove stability and consistency than to show convergence.

2.3 Relaxation

In any iterative solution process, one begins with an initial approximation, then repeatedly refines the approximation according to some rule. The iteration obviously should converge to the true solution, but to be considered useful this convergence must also be rapid. This is especially true in the multigrid context, as only a small number of relaxation iterations are ever performed on the same equation.

Consider the system of n linear equations

$$\mathcal{A}u = f \qquad \text{with} \quad u, f \in \mathcal{X} \tag{2.8}$$

where $\mathcal{X}$ is an n-dimensional vector space. We desire some iterative process to (numerically) solve this system by generating a sequence of converging approximations $v^0 \to v^1 \to v^2 \to \ldots$. The iteration is to be generated by some linear mapping ψ:

$$v^{k+1} = \psi(\mathcal{A}, f, v^k).$$

We have assumed that ψ is independent of k; such iterations are said to be *stationary*. Since ψ is linear by assumption, we can write the iteration in the form

$$v^{k+1} = \mathcal{M}v^k + \mathcal{N}f \tag{2.9}$$

in which case $\mathcal{M}$ is called the *iteration matrix*. The iteration is completely described by $\mathcal{M}$, as we now demonstrate. Since this iteration must have the solution of the system (2.8) as a fixed point, we have

$$u = \mathcal{M}u + \mathcal{N}f \qquad \forall \quad f \in \mathcal{X}$$

so that

$$I = \mathcal{M} + \mathcal{N}\mathcal{A}$$

where I is the $n \times n$ identity matrix, and hence the iteration process is

$$v^{k+1} = \mathcal{M}v^k + (I - \mathcal{M})\mathcal{A}^{-1}f$$

provided $\mathcal{A}$ is non-singular. (If $\mathcal{A}$ is singular then this equation has an appropriate interpretation in terms of the Moore-Penrose pseudo-inverse $\mathcal{A}^+$.)

An explicit formula for the k^{th} iterate is

$$v^k = \mathcal{M}^k v^0 + \sum_{\kappa=0}^{k-1} \mathcal{M}^\kappa \mathcal{N}f$$

and so we see that the error in the k^{th} iterate is

$$v^k - u = \mathcal{M}^k(v^0 - u), \tag{2.10}$$

in other words the iteration matrix is the amplification matrix of the error [51].

The *spectral radius* of a matrix $\mathcal{M}$ is defined by $\rho(\mathcal{M}) = \max|\lambda_i|$, where λ_i is an eigenvalue of $\mathcal{M}$. Since

$$\lim_{k\to\infty} \mathcal{M}^k = 0 \qquad \text{iff} \quad \rho(\mathcal{M}) < 1,$$

we have the following important result:

Theorem 2.3 *The iteration* (2.9) *converges for every initial guess iff* $\rho(\mathcal{M}) < 1$.

Iterative methods fall into two classes: *point iterative* and *block iterative.* Point iterative processes use explicit components of the previous approximation to update the next iterate, while block iterative methods at each stage require the solution of several linear systems.

Most of the well-known point iterative methods are based on a partition of the matrix A of the form

$$A = D - L - U$$

where D is the diagonal of A, and L and U are the negatives of the lower and upper triangular parts of A, respectively. (We decompose A this way because, due to the mean value property, elliptic finite-difference equations generally give rise to a matrix A with diagonal entries of opposite sign to off-diagonal entries.) We assume that the diagonal elements of A are non-zero, so that $(D-L)^{-1}$ exists.

We introduce definitions of the *algebraic error*

$$e = u - v \tag{2.11}$$

and the *residual* (or *residue*, a measure of how well v satisfies the PDE)

$$r = f - \mathcal{A}v, \tag{2.12}$$

from which we obtain the crucial relation

$$\mathcal{A}e = r \tag{2.13}$$

which is called the *residual equation.* (Note that some authors use the *defect*, defined to be the negative of the residual.) Combining these definitions with equation (2.9), we find for this general iteration that

$$v^{k+1} = v^k + \mathcal{N}r^k$$

which corresponds to the relation

$$u = v + \mathcal{A}^{-1}r.$$

Hence an effective iteration will be constructed when $\mathcal{N} = (I - \mathcal{M})\mathcal{A}^{-1}$ is a close approximation to $\mathcal{A}^{-1}$ in some sense.

The method of *Jacobi relaxation* is derived simply by solving the finite-difference equation in terms of v_{ij}. In our standard example of Poisson's equation discretised by a five-point star (equation (2.7)) we find that the $(k+1)^{\text{th}}$ iterate at the (i,j) grid point is given by

$$v_{ij}^{k+1} = \frac{1}{4}\left(v_{i+1,j}^{k} + v_{i-1,j}^{k} + v_{i,j+1}^{k} + v_{i,j-1}^{k} - h^2 f_{ij}\right)$$

which we will write as

$$v_{ij}^{k+1} = \frac{1}{4}\Big(\sum_{nn} v_{ij}^{k} - h^2 f_{ij}\Big), \tag{2.14}$$

the sum over nn signifying the nearest (orthogonal) neighbours. Note that the order in which the grid points v_{ij} are updated during one relaxation "sweep" is irrelevant.

The term *relaxation* historically arose through the idea of updating the value of v at a particular grid point by relaxing the "residual forces" arising because current values of v at neighbouring grid points do not correctly satisfy the PDE.

The matrix A is called *reducible* if there exists a permutation matrix P such that

$$PAP^T = PAP^{-1} = \begin{bmatrix} A_1 & 0 \\ A_2 & A_3 \end{bmatrix}.$$

This means that some values of u are independent of some boundary conditions. Well-posed linear elliptic boundary value problems almost always lead to irreducible matrices [4]. Collatz [28] proved the following important theorem:

Theorem 2.4 (Collatz) *If $\mathcal{A}$ is diagonally dominant and irreducible, then Jacobi relaxation converges.*

An important generalisation of an iteration scheme such as Jacobi relaxation incorporates the idea of *weighted* or *damped* relaxation. This technique is also called successive over-relaxation (SOR). It is essentially a way to accelerate convergence by extrapolating the changes in the previous iterates to provide a superior new iterate v^{k+1}. In general, the method is given by

$$v^{k+1} = \omega\,\overline{v}^{k+1} + (1-\omega)\,v^{k}$$

where the weighting factor ω is generally in the range $[0,2]$, and $\overline{v}^{k+1}$ is now the iterate obtained in the manner described above, for example by equation (2.14). Clearly if $\omega = 1$, then the method reduces to the standard relaxation. Hence we see that in the weighted relaxation method, the new value of v is extrapolated from the standard iterate and the previous value.

The matrix representation of Jacobi relaxation is

$$v^{k+1} = D^{-1}(L+U)v^{k} + D^{-1}f,$$

which corresponds to solving the ij^{th} finite-difference equation in terms of v_{ij}. Similarly for weighted Jacobi relaxation we have:

$$v^{k+1} = [(1-\omega)I + \omega D^{-1}(L+U)]\,v^k + \omega D^{-1} f. \tag{2.15}$$

Gauss-Seidel relaxation incorporates a slight modification of the Jacobi method: the updated grid values are used as soon as they are calculated, rather than waiting until the next iteration; consequently we now require a systematic ordering of the grid points. Such orderings are numerous; for example, lexicographic (row-wise or column-wise ascending or descending), symmetric (ascending then descending), and red-black (all odd points then all even points).

Suppose in two dimensions we have some arbitrary, but fixed, ordering of interior grid points, indicated by v_κ, for $\kappa = 1, 2, \ldots, (M-1)(N-1)$. To obtain the κ^{th} component of the next iterate of the Gauss-Seidel relaxation, v_κ^{k+1}, we solve the equation

$$\sum_{\lambda=1}^{\kappa-1} A_{\kappa\lambda} v_\lambda^{k+1} + A_{\kappa\kappa} v_\kappa + \sum_{\lambda=\kappa+1}^{(M-1)(N-1)} A_{\kappa\lambda} v_\lambda^k = f_\kappa \tag{2.16}$$

in terms of v_κ. The Gauss-Seidel iteration is more conveniently written in matrix form:

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

or equivalently as

$$v^{k+1} = v^k - (D-L)^{-1}(Av^k - f).$$

Generalising to ω-weighted relaxation, we find

$$v^{k+1} = v^k - \omega(D-\omega L)^{-1}(Av^k - f),$$

or in a form which shows the iteration matrix $\mathcal{M}$ explicitly [101]:

$$v^{k+1} = (I - \omega D^{-1}L)^{-1}[(1-\omega)I + \omega D^{-1}U]\,v^k + \omega(I-\omega D^{-1}L)^{-1} D^{-1} f.$$

Red-black Gauss-Seidel relaxation is based on a division of the interior grid points v_{ij} into two equal classes:

red	for (i,j) = (odd, odd) or (even, even)
black	for (i,j) = (odd, even) or (even, odd) .

Figure 2.2 depicts the chequerboard pattern which this classification produces. (This scheme can be generalised to n-colour relaxation by classifying grid points according to the rule $(i+j) \bmod n$.) Note that this labelling of the grid points allows any five-point operator to independently update all grid points within each colour group.

For our example of the five-point star on Poisson's equation, ascending row-wise lexicographic Gauss-Seidel relaxation is given by

$$v_{ij}^{k+1} = \frac{1}{4}(v_{i+1,j}^k + v_{i-1,j}^{k+1} + v_{i,j+1}^k + v_{i,j-1}^{k+1} - h^2 f_{ij}),$$

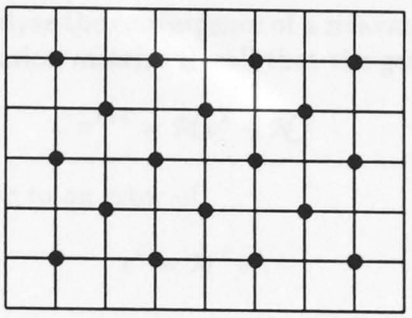


Figure 2.2: A 9×7 grid, where "red" grid points are indicated with a (•).

where the updating of v_{ij} proceeds in the order

$$v_{0,0},\ v_{1,0},\ \ldots,\ v_{M,0};\quad v_{0,1},\ v_{1,1},\ \ldots,\ v_{M,1};\quad \ldots;\quad v_{0,N},\ v_{1,N},\ \ldots,\ v_{M,N}.$$

We can de-emphasise the ordering role of k in this equation by simply writing

$$v_{ij} \leftarrow \frac{1}{4}\Big(\sum_{nn} v_{ij} - h^2 f_{ij}\Big),$$

(where the arrow indicates replacement) for some general ordering of the grid points.

Thus far we have discussed only point iterations, whereby a single grid point is updated at each step of the process, and the value of v_{ij}^{k+1} is determined by an explicit formula. *Block* iterative processes (also known as group iterative or implicit iterative methods) are a natural extension whereby groups of grid points are updated simultaneously at each step. The process requires the solution of simultaneous equations, and typically increases the convergence rate, while increasing the complexity of the relaxation algorithm.

There are a large number of possible ways of dividing the domain into blocks. We shall only consider *line relaxation*, in which individual rows or columns of grid points are simultaneously updated. Nevertheless, this still creates many different possible strategies for such relaxation: by row or column; with ascending, descending, symmetric (ascending then descending) or zebra (the block analogue of red-black) updates; and alternating direction implicit (ADI) methods which swap between row and column updates. In addition, we have our previous possibilities of ω-weighted Jacobi or Gauss-Seidel relaxation.

For our example of the five-point star on Poisson's equation in two dimensions, ascending row-wise Jacobi line relaxation is given by

$$v_{ij}^{k+1} = \frac{1}{4}\,(v_{i+1,j}^{k+1} + v_{i-1,j}^{k+1} + v_{i,j+1}^{k} + v_{i,j-1}^{k} - h^2 f_{ij}). \tag{2.17}$$

Thus we need to solve a system of $M-1$ linear equations in $M-1$ unknowns (the grid row v_{ij}, for $i = 1, 2, \ldots, M-1$). Given the use of a five-point star as a discretisation operator, we see that this system will be tridiagonal. Various efficient direct methods exist to solve such a system.

We can precisely analyse the convergence of a relaxation process by studying the properties of the iteration matrix. Recall that the general linear iteration

$$v^{k+1} = \mathcal{M}v^k + \mathcal{N}f$$

after k iterations gave rise to an error of

$$e^k = \mathcal{M}^k e^0,$$

and so the iteration converges unconditionally iff the spectral radius $\rho(\mathcal{M}) < 1$. Thus it is the eigenvalues and eigenvectors of $\mathcal{M}$ which determine the convergence properties.

For the sake of clarity, we restrict ourselves to one dimension for the moment. Using equation (2.15), we find the iteration matrix for weighted Jacobi relaxation for the one-dimensional Poisson equation discretised by the five-point star to be

$$\mathcal{M} = I - \tfrac{1}{2}\omega h^2 A,$$

hence relaxation is given by

$$v^{k+1} = v^k - \tfrac{1}{2}\omega h^2(Av^k - f).$$

The eigenvalues of $A = N^2\,\text{TRIDIAG}[-1, 2, -1]$ are

$$\lambda_j(A) = 4N^2 \sin^2\left(\tfrac{j\pi h}{2}\right) \qquad \text{for} \quad j = 1, 2, \ldots, N-1$$

and so the eigenvalues of $\mathcal{M}$ are

$$\begin{aligned}\lambda_j(\mathcal{M}) &= 1 - 2\omega \sin^2\left(\tfrac{j\pi h}{2}\right) \\ &= 1 - \omega(1 - \cos j\pi h)\end{aligned}$$

(see Figure 2.3). In particular we note that $\omega \in (0, 1]$ gives $|\lambda_j(\mathcal{M})| < 1$, and therefore a convergent Jacobi iteration. Weighted relaxation with $\omega \in (0, 1)$ is called *under-relaxation*, whereas relaxation with $\omega > 1$ is referred to as *over-relaxation*. We also observe that the convergence factor for Jacobi relaxation is $1 - O(h^2)$ irrespective of ω.

Since the eigenvectors of $\mathcal{A}$, w_j say, form an orthogonal basis, the error in the initial guess can be expressed in terms of an eigenvector expansion:

$$e^0 = u - v^0 = \sum_{j=1}^{N-1} c_j\, w_j$$

for some constants c_j, hence after k weighted Jacobi relaxations we have

$$e^k = \sum_{j=1}^{N-1} c_j\, \mathcal{M}^k w_j = \sum_{j=1}^{N-1} c_j\, \lambda_j^k(\mathcal{M})\, w_j.$$

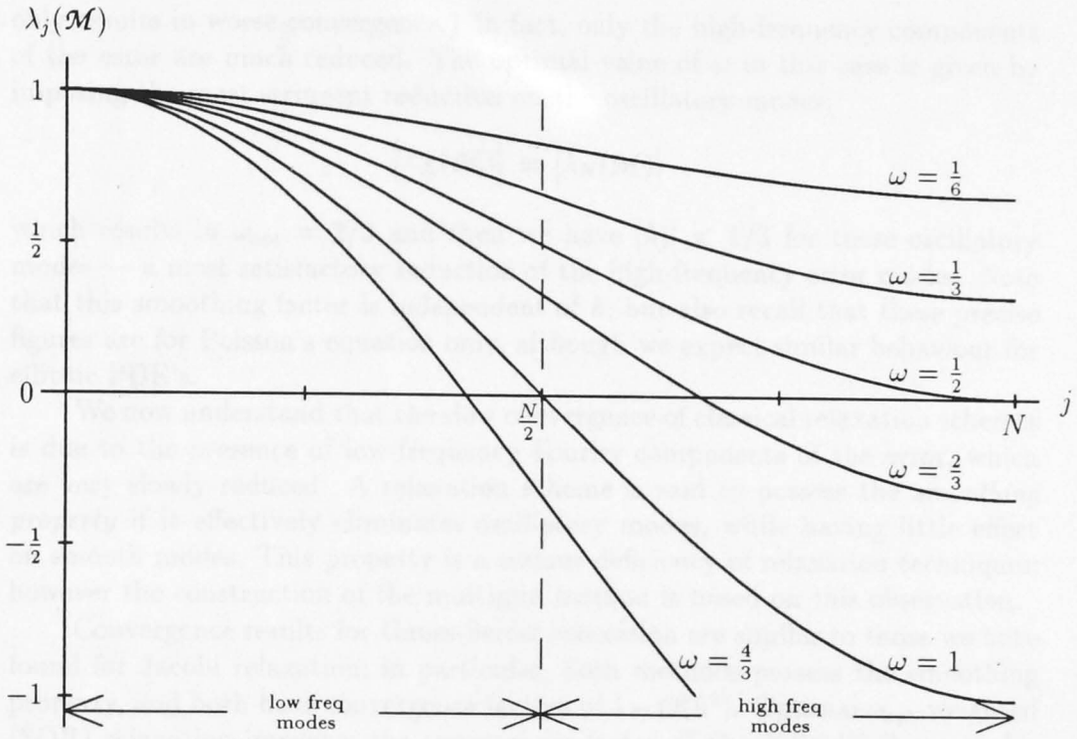


Figure 2.3: Graph of the eigenvalues of the 1-D weighted Jacobi iteration matrix for Poisson's equation, $\lambda_j(\mathcal{M}) = 1 - 2\omega \sin^2(j\pi h/2)$, for various values of ω, plotted for large $N = 1/h$.

In other words, the j^{th} mode of the initial error is reduced by the factor $\lambda_j^k(\mathcal{M})$ after k iterations. Moreover, since $|\lambda_j^k| \ll 1$ for high frequencies (when ω is not close to 0 or 1), the error e^k is much smoother than the original error e^0. This fact will be important when we later examine the multigrid process. We also see that Jacobi relaxation is *mode-invariant*: when applied to a linear combination of modes, the iteration can only alter their amplitudes, not convert some modes into others. The introduction of elementary Fourier analysis will allow us to better understand this behaviour.

One-dimensional *Fourier modes* are vectors of the form

$$v_j = \sin(j\kappa\pi h) \qquad \text{for} \quad j = 0, 1, \ldots, N$$

where the wavenumber $\kappa = 1, 2, \ldots, N-1$ gives the number of half-sine waves on the unit interval $[0, 1]$. We call modes in the lower half of the frequency spectrum ($\kappa = 1, 2, \ldots, N/2 - 1$) *smooth* modes, while high-frequency modes ($\kappa = N/2, N/2 + 1, \ldots, N-1$) are called *oscillatory* modes (see Figure 2.3).

Since the eigenvalue corresponding to the smoothest mode is

$$\lambda_1 = 1 - \tfrac{1}{2}\omega\pi^2 h^2 + O(h^4)$$

we see that there is no value of ω which will satisfactorily reduce the smooth components of the error. (Moreover, attempting to increase accuracy by decreasing h

only results in worse convergence.) In fact, only the high-frequency components of the error are much reduced. The optimal value of ω in this case is given by imposing the most stringent reduction on the oscillatory modes:

$$\left|\lambda_{\frac{N}{2}}(\mathcal{M})\right| = \left|\lambda_N(\mathcal{M})\right|$$

which results in $\omega_{\text{opt}} = 2/3$ and then we have $|\lambda_j| < 1/3$ for these oscillatory modes — a most satisfactory reduction of the high-frequency error modes. Note that this smoothing factor is independent of h; but also recall that these precise figures are for Poisson's equation only, although we expect similar behaviour for elliptic PDE's.

We now understand that the slow convergence of classical relaxation schemes is due to the presence of low-frequency Fourier components of the error, which are very slowly reduced. A relaxation scheme is said to possess the *smoothing property* if it effectively eliminates oscillatory modes, while having little effect on smooth modes. This property is a serious deficiency of relaxation techniques; however the construction of the multigrid method is based on this observation.

Convergence results for Gauss-Seidel relaxation are similar to those we have found for Jacobi relaxation; in particular, both methods possess the smoothing property, and both have convergence factors of $1 - O(h^2)$. Optimal ω_{opt}-weighted (SOR) relaxation improves the convergence factor of Gauss-Seidel (but not Jacobi) relaxation to $1 - O(h)$. These comments also apply to Jacobi and Gauss-Seidel line relaxation.

Weighted Gauss-Seidel relaxation converges for $0 < \omega \leq 2$ (see Ames [4]). Over-relaxation, with $\omega \in (1, 2]$, results in a better convergence rate than ordinary Gauss-Seidel relaxation. However under-relaxation, with $\omega \in (0, 1)$, improves the smoothing rate (even though this adversely affects the overall convergence rate), and is more often used in connection with multigrid.

In this same vein, we may remark that before the emergence of multigrid, Gauss-Seidel relaxation was favoured over Jacobi relaxation, due to its superior convergence properties. In fact under many circumstances, optimal Gauss-Seidel relaxation is exactly twice as fast as Jacobi's method [4]. However, Jacobi relaxation may have superior smoothing properties, and so is quite popular in the multigrid field.

We have examined some of the simpler relaxation methods. Many more sophisticated iteration schemes exist (see for example [4, 5, 80]), however these are not required for our purposes. The multigrid method is a construction of a powerful solution process from a set of simple tools, as we shall now see.

Chapter 3

Elements of Linear Multigrid

To iterate is human; to recurse, divine.

— Anonymous

Following on from our discussion of relaxation in the previous chapter, we now consider the other fundamental elements of the multigrid method: intergrid transfer, coarse-grid correction and nested iteration. We examine the two-grid multigrid scheme (for linear equations), which leads us to the full multi-grid process. To clarify the presentation, some of the material in this chapter is based upon one-dimensional problems; Section 3.3 extends the coverage to multidimensional problems. We then look at the large multigrid parameter space, and consider an automatic initial guess algorithm. Finally, we discuss the multigrid implementation for conventional serial computers.

3.1 Two-Grid Multigrid

In Chapter 1 we described how Southwell [91] used coarser grids to improve the speed of his hand-calculated relaxation. The heuristic reasons for the success of this modification are that an iteration scheme is improved by a more accurate initial guess, and that convergence factors are slightly better on coarser grids, since these behave like $1 - O(h^2)$ or $1 - O(h)$. We have employed the Fourier or *spectral* viewpoint of relaxation to show that many of the standard iterations possess the smoothing property; that is, the iteration process efficiently reduces the high-frequency components of the error, while acting very slowly on the low-frequency components. Since these frequencies are measured with respect to local grid scales, it becomes clear that once we have smoothed the current approximation on the fine grid Ω_h, we can begin to tackle the smooth modes which remain by transferring the problem to the coarse grid Ω_{2h}, whereby those smooth modes appear more oscillatory, and so can be effectively damped by coarse-grid relaxation. We shall now consider this idea more carefully.

Recall that (one-dimensional) Fourier components with wave number κ are smooth modes if $1 \leq \kappa < N/2$, and oscillatory modes if $N/2 \leq \kappa \leq N-1$. Given

that the coarse grid Ω_{2h} $(j = 0, 1, \ldots, N/2)$ consists of the even-numbered points of the fine grid Ω_h $(j = 0, 1, \ldots, N)$, we see that the fine-grid smooth modes are transformed into more oscillatory modes on the coarse grid; this situation is illustrated in Figure 3.1(a). The special case when $\kappa = N/2$ on Ω_h gives rise to the zero vector $\kappa' = 0$ on the coarse grid (see Figure 3.1(b)). Fine-grid oscillatory modes with $N/2 < \kappa < N$ are transformed into relatively smooth $\kappa' = (N - \kappa)$ coarse-grid modes — a misrepresentation phenomenon known as *aliasing* (see Figure 3.1(c)).

Suppose that for some problem we have obtained a relatively smooth approximate solution v^k by relaxation. The essential multigrid idea is to improve the approximation by calculating the residual $r^k = f - Av^k$ and solving the residual equation $Ae^k = r^k$ for e^k to obtain the correction $v^k + e^k = u$. These equations are useful provided that we can solve the residual equation exactly, which of course we cannot do in general. However, the residual equation is precisely of the same form as the original equation $Au = f$, and so we employ the same technique to approximately solve it: namely, we relax directly on the error on the coarse grid, using zero as our initial guess. It is important to note that e^k will be a smooth function, as indicated in the previous chapter, and can therefore be properly represented on a coarse grid.

Up to this point, we have two loosely-connected ideas. The first is called *nested iteration*, whereby we relax on the equation $Au = f$ on the coarse grid Ω_{2h} to obtain a good initial guess for relaxation on the fine grid Ω_h:

1. perform ν_{2h} relaxations on $|Au = f|_{2h}$ (with some initial guess v_{2h}^0) to obtain a new approximation v_{2h}
2. interpolate v_{2h} to the fine grid to obtain v_h
3. perform ν_h relaxations on $|Au = f|_h$ (with initial guess v_h) to obtain a new approximation v_h

The second idea is called *coarse-grid correction*:

1. perform ν_h relaxations on $|Au = f|_h$ (with some initial guess v_h^0) to obtain a new smooth v_h
2. compute the residual $r_h = f - Av_h$
3. restrict r_h to the coarse grid to obtain r_{2h}
4. perform ν_{2h} relaxations on $|Ae = r|_{2h}$ (with initial guess $e_{2h} = 0$) to obtain a new e_{2h} (or directly solve the coarse-grid equation: $e_{2h} = A_{2h}^{-1} r_{2h}$)
5. interpolate the approximate error e_{2h} to the fine grid to obtain e_h
6. correct the current approximation by $v_h \leftarrow v_h + e_h$

It will be apparent that we require intergrid transfer mechanisms to facilitate the coarse-to-fine-grid *interpolation* (also called prolongation), and the fine-to-coarse-grid *restriction* operations. We now turn our attention to these operations.

Consider the approximation of some linear equation $A_h v_h = f_h$ by the coarse-grid equation $A_{2h} v_{2h} = f_{2h}$. The finite-difference discretisation ensures that there is a natural interpretation of A_{2h}: in one dimension, it is simply the $N/2 \times N/2$

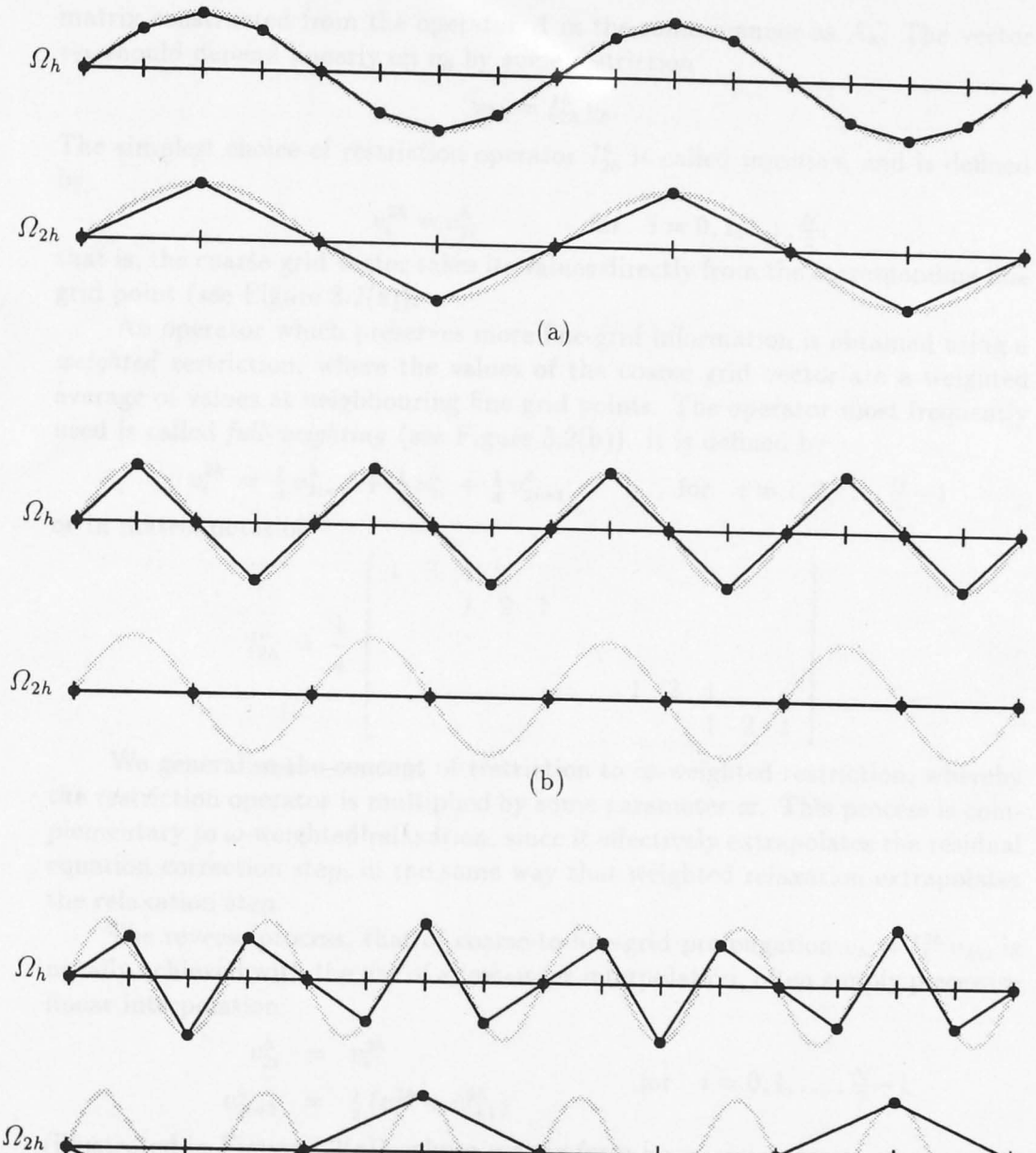


Figure 3.1: Fine-grid Fourier modes are classified according to how they are transformed onto a coarse grid: (a) smooth modes become more oscillatory, (b) $\kappa = N/2$ becomes the zero vector, and (c) oscillatory modes become smoother (aliasing). The cases $\kappa = 4,8$ and 12 are shown for $N = 16$ on a one-dimensional fine grid Ω_h, being projected onto the coarse grid with $N = 8$, where they become modes with $\kappa' = 4, 0$ and 4 respectively.

matrix constructed from the operator A in the same manner as A_h. The vector v_{2h} should depend linearly on v_h by some restriction

$$v_{2h} = I_{2h}^{h} v_h.$$

The simplest choice of restriction operator I_{2h}^{h} is called *injection*, and is defined by

$$v_i^{2h} = v_{2i}^{h} \qquad \text{for} \quad i = 0, 1, \ldots, \tfrac{N}{2}.$$

that is, the coarse-grid vector takes its values directly from the corresponding fine grid point (see Figure 3.2(a)).

An operator which preserves more fine-grid information is obtained using a *weighted* restriction, where the values of the coarse grid vector are a weighted average of values at neighbouring fine grid points. The operator most frequently used is called *full-weighting* (see Figure 3.2(b)). It is defined by

$$v_i^{2h} = \tfrac{1}{4} v_{2i-1}^{h} + \tfrac{1}{2} v_{2i}^{h} + \tfrac{1}{4} v_{2i+1}^{h} \qquad \text{for} \quad i = 1, 2, \ldots, \tfrac{N}{2} - 1$$

or in matrix notation

$$I_{2h}^{h} = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 & & & & & & \\ & & 1 & 2 & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & 2 & 1 \end{bmatrix}.$$

We generalise the concept of restriction to ϖ-weighted restriction, whereby the restriction operator is multiplied by some parameter ϖ. This process is complementary to ω-weighted relaxation, since it effectively extrapolates the residual equation correction step, in the same way that weighted relaxation extrapolates the relaxation step.

The reverse process, that of coarse-to-fine-grid prolongation $v_h = I_h^{2h} v_{2h}$, is usually achieved with the use of a low-order interpolation, often simply piecewise linear interpolation:

$$\begin{aligned} v_{2i}^{h} &= v_i^{2h} \\ v_{2i+1}^{h} &= \tfrac{1}{2}(v_i^{2h} + v_{i+1}^{2h}) \end{aligned} \qquad \text{for} \quad i = 0, 1, \ldots, \tfrac{N}{2} - 1$$

(illustrated in Figure 3.2(c)), whose matrix form is

$$I_{2h}^{h} = \frac{1}{2} \begin{bmatrix} 1 & & & & & \\ 2 & & & & & \\ 1 & 1 & & & & \\ & 2 & & & & \\ & 1 & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & 1 \\ & & & & & 2 \\ & & & & & 1 \end{bmatrix}.$$

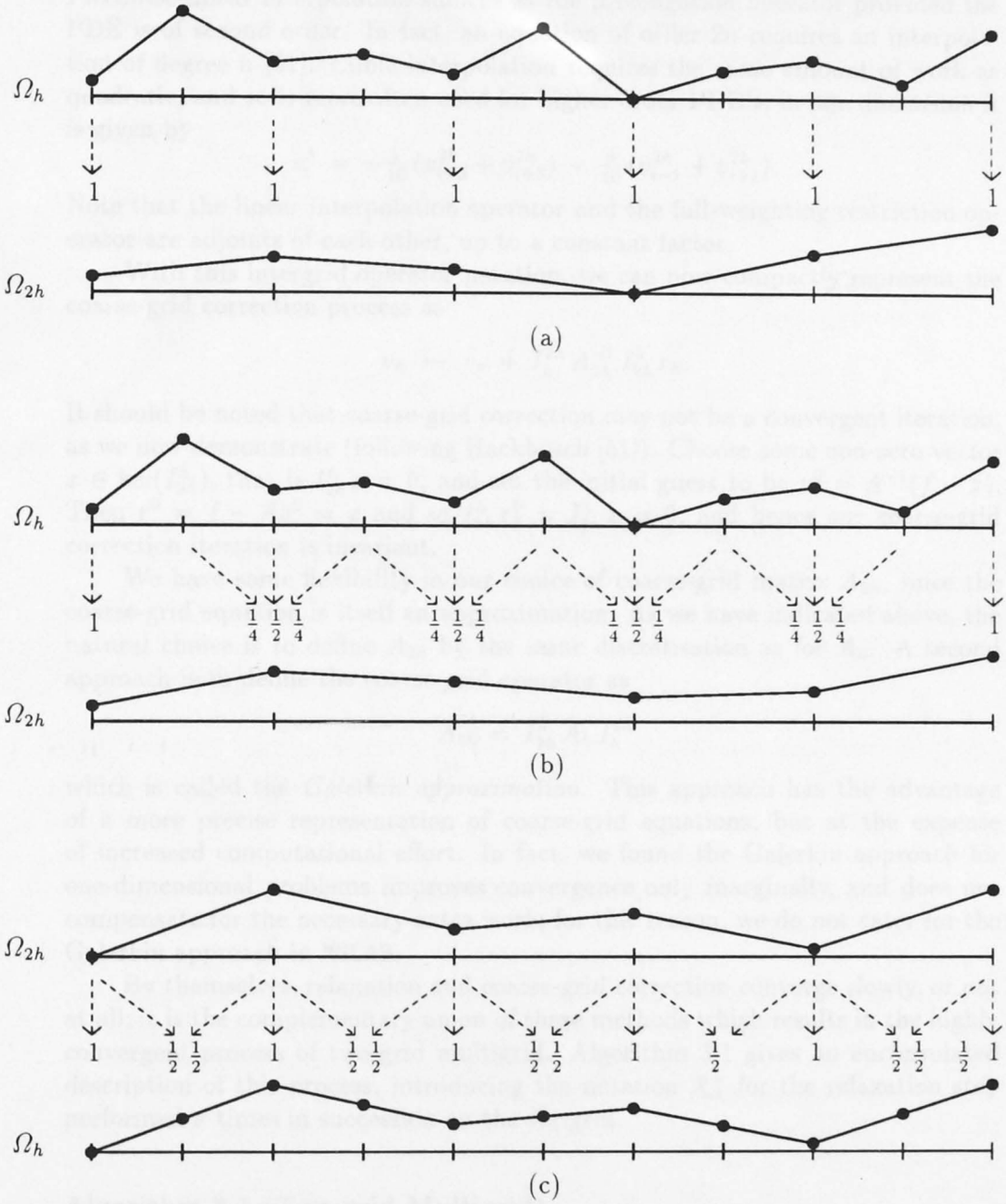


Figure 3.2: Intergrid transfer operations in one dimension: (a) injection restriction, (b) full-weighting restriction, and (c) linear interpolation. The numbers near each arrowhead indicate the relative weighting attached to that grid point for the transfer operation.

Piecewise linear interpolation suffices as the prolongation operator provided the PDE is of second order. In fact, an equation of order $2n$ requires an interpolation of degree n [51]. Cubic interpolation requires the same amount of work as quadratic, and so is more often used for higher-order PDE's; in one dimension it is given by

$$v_i^h = -\tfrac{1}{16}(v_{i-3}^{2h} + v_{i+3}^{2h}) + \tfrac{9}{16}(v_{i-1}^{2h} + v_{i+1}^{2h}).$$

Note that the linear interpolation operator and the full-weighting restriction operator are adjoints of each other, up to a constant factor.

With this intergrid operator notation, we can now compactly represent the coarse-grid correction process as

$$v_h \leftarrow v_h + I_h^{2h} A_{2h}^{-1} I_{2h}^h r_h.$$

It should be noted that coarse-grid correction may not be a convergent iteration, as we now demonstrate (following Hackbusch [51]). Choose some non-zero vector $x \in \ker(I_{2h}^h)$, that is $I_{2h}^h x = 0$, and set the initial guess to be $v^0 = A^{-1}(f - x)$. Then $r^0 = f - Av^0 = x$ and so $I_{2h}^h r_h^0 = I_{2h}^h x = 0$, and hence our coarse-grid correction iteration is invariant.

We have some flexibility in our choice of coarse-grid matrix A_{2h}, since the coarse-grid equation is itself an approximation. As we have indicated above, the natural choice is to define A_{2h} by the same discretisation as for A_h. A second approach is to define the coarse-grid operator as

$$A_{2h} = I_{2h}^h A_h I_h^{2h}$$

which is called the *Galerkin approximation*. This approach has the advantage of a more precise representation of coarse-grid equations, but at the expense of increased computational effort. In fact, we found the Galerkin approach for one-dimensional problems improves convergence only marginally, and does not compensate for the necessary extra work; for this reason, we do not cater for the Galerkin approach in `MGLAB`.

By themselves, relaxation and coarse-grid correction converge slowly, or not at all; it is the complementary union of these methods which results in the highly convergent process of two-grid multigrid. Algorithm 3.1 gives an encapsulated description of this process, introducing the notation $\mathcal{R}_h^\nu$ for the relaxation step performed ν times in succession on the Ω_h grid.

Algorithm 3.1 (Two-grid Multigrid)
An iteration for solving $A_h u_h = f_h$, given some initial guess v_h.

v_h	$\leftarrow$	$\mathcal{R}_h^{\nu_1}(v_h, f_h)$	pre-smoothing
r_h	$\leftarrow$	$f_h - A_h v_h$	residual calculation
r_{2h}	$\leftarrow$	$I_{2h}^h r_h$	residual restriction
e_{2h}	$\leftarrow$	$A_{2h}^{-1} r_{2h}$	coarse-grid equation solution
v_h	$\leftarrow$	$v_h + I_h^{2h} e_{2h}$	coarse-grid correction
v_h	$\leftarrow$	$\mathcal{R}_h^{\nu_2}(v_h, f_h)$	post-smoothing

Hackbusch [51] rigorously proves convergence of this iteration. We shall postpone discussion and analysis of convergence to the next section.

The final necessary component of our multigrid iteration is a measure of the closeness of the approximation v to the exact solution u. We use the following standard discrete norms for the error $e = u - v$:

$$\|e\|_2 = \sqrt{h \sum_{i=1}^{N} e_i^2} \quad \text{and} \quad \|e\|_\infty = \max_{1 \le i \le N} |e_i|.$$

Of course, the exact solution is frequently unavailable, in which case we measure how closely v satisfies the PDE, by using the above norms applied to the residue $r = f - Av$.

Many other measures may be defined, for example [1]

$$\epsilon = \frac{\|e\|}{\|u - \tilde{v}\|}$$

where $\tilde{v}$ (the "exact discrete solution") is the result of iterating many $V_L^{2,2,1}$-cycles (see below). Note that the denominator $\|u - \tilde{v}\|$ is a measure of the discretisation error.

3.2 Full Multigrid

The two-grid multigrid iteration offers computational benefits, but of course leaves open the question of how to solve the coarse-grid (residual) equation

$$A_{2h}\, e_{2h} = r_{2h}. \tag{3.1}$$

Although a direct solution is possible, and involves only $N/2$ unknowns, this is still impractical for real-world problems. However, since the coarse-grid equation is itself an approximation (to the fine-grid problem), we need only solve it approximately. Indeed, equation (3.1) is of the same form as our original linear equation, hence we can embed the same two-grid procedure to (approximately) solve our residual equation. This involves excursions to coarser and coarser grids Ω_{2h}, Ω_{4h}, Ω_{8h}, ...; the recursive process ceasing when the grid is so coarse (contains so few unknown grid values) that a direct solution is possible or that a few relaxations give the coarsest-grid solution to the required accuracy. Often the grid is coarsened until there is a *single* interior grid point. This ensures that the broadest possible range of Fourier components of the error can be attacked.

To allow a convenient description, we introduce a numbering of grid levels of the form

$$L = 1, 2, \ldots, \Lambda$$

where level $L = 1$ indicates the coarsest grid, and $L = \Lambda$ labels the finest grid (that is, $\Omega_\Lambda \equiv \Omega_h$). This means that the grid spacing on level L is

$$h_L = 2^{\Lambda - L}\, h$$

and the number of grid points on level L is

$$N_L = 2^{L-\Lambda} N.$$

Note that we only discuss the case where grid meshes are coarsened by a factor of two: $h_L/h_{L+1} = 2$; this is called *standard coarsening*. There are other possibilities (see Hackbusch [51]), but these are infrequently used.

The multigrid process described above visits the grids in the order

$$\Omega_\Lambda, \Omega_{\Lambda-1}, \ldots, \Omega_2, \Omega_1, \Omega_2, \ldots, \Omega_{\Lambda-1}, \Omega_\Lambda,$$

and so is called a *V-cycle* (see Figure 3.3(a)). Algorithm 3.2 presents a recursive description of the V-cycle iteration. We also give a non-recursive description of the process, since standard FORTRAN 77 does not permit recursion (see Algorithm 3.3).

We introduce the notation $V_\Lambda^{\nu_1,\nu_0,\nu_2}$-cycle to indicate a V-cycle of depth Λ levels, with ν_1 pre-relaxations, ν_0 coarsest-grid relaxations, and ν_2 post-relaxations.

A slight generalisation of the V-cycle is made by replacing the recursive call

V-cycle $(v, r, L-1)$

in Algorithm 3.2 by a sequence of recursive calls:

for i = 1 **to** γ **do**
  V-cycle $(v, r, L-1)$
endfor

Hence $\gamma = 1$ gives the V-cycle, while $\gamma = 2$ produces a scheme known as the *W-cycle* (see Figure 3.3(b)). In practice, no other values of γ are used.

So far we have only used the coarse-grid correction idea to construct a multigrid method. By embedding V-cycles in a nested iteration scheme, we arrive at the full multigrid V-cycle, which we call the *M-cycle*, also known as the FMV cycle (see Figure 3.3(c)). This removes the need for a reasonable fine-grid initial guess to commence the V-cycle; we instead begin the M-cycle with a coarsest-grid initial guess, the accuracy of which is practically irrelevant since the first step is to perform ν_0 coarsest-grid relaxations: $\mathcal{R}_1^{\nu_0}$. Algorithm 3.4 describes the process. We also give a non-recursive description (see Algorithm 3.5).

3.3 Multi-dimensional Multigrid

The material in the preceding sections was based on one-dimensional multigrid, so that it could be presented more clearly. We now indicate how these concepts are applied to two-dimensional problems, with obvious generalisations to higher dimensions.

We begin with the intergrid transfer operators: restriction and interpolation. Injection is again defined as the direct transfer of the value at fine-grid points to corresponding coarse-grid points:

$$v_{i,j}^{2h} = v_{2i,2j}^{h} \qquad \text{for} \quad i = 0, 1, \ldots, \tfrac{M}{2}; \quad j = 0, 1, \ldots, \tfrac{N}{2}.$$

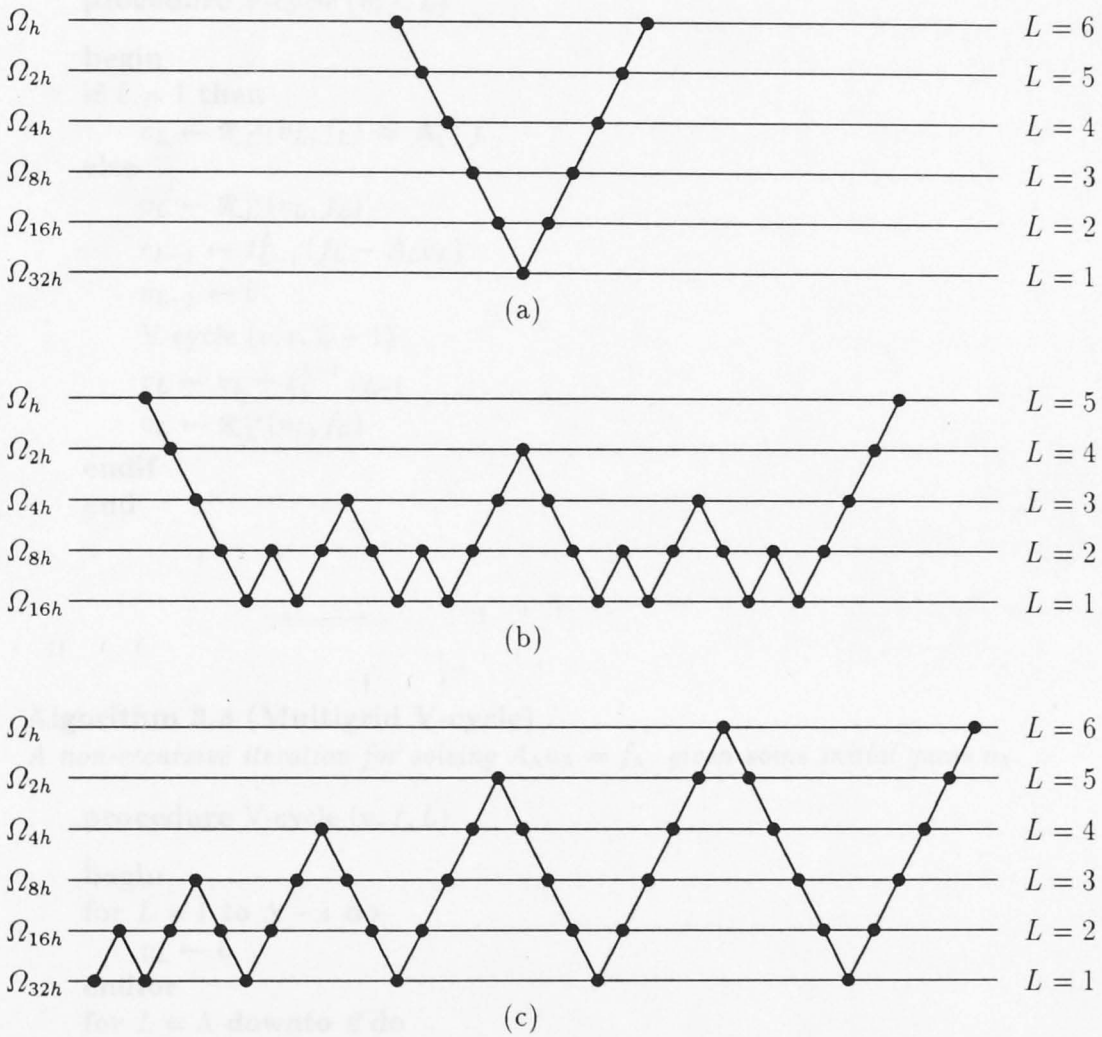


Figure 3.3: Diagram indicating the order in which grid levels are visited (reading from left to right) for the (a) V_6-cycle, (b) W_5-cycle, and (c) M_6-cycle.

Algorithm 3.2 (Multigrid V-cycle)
A recursive iteration for solving $A_\Lambda u_\Lambda = f_\Lambda$, given some initial guess v_Λ.

procedure V-cycle (v, f, L)

begin
if $L = 1$ **then**
$v_L \leftarrow \mathcal{R}_L^{\nu_0}(v_L, f_L) \approx A_1^{-1} f$
else
$v_L \leftarrow \mathcal{R}_L^{\nu_1}(v_L, f_L)$
$r_{L-1} \leftarrow I_{L-1}^L (f_L - A_L v_L)$
$v_{L-1} \leftarrow 0$
V-cycle $(v, r, L-1)$
$v_L \leftarrow v_L + I_L^{L-1} v_{L-1}$
$v_L \leftarrow \mathcal{R}_L^{\nu_2}(v_L, f_L)$
endif
end

Algorithm 3.3 (Multigrid V-cycle)
A non-recursive iteration for solving $A_\Lambda u_\Lambda = f_\Lambda$, given some initial guess v_Λ.

procedure V-cycle (v, f, L)

begin
for $L = 1$ **to** $\Lambda - 1$ **do**
$v_L \leftarrow 0$
endfor
for $L = \Lambda$ **downto** 2 **do**
$v_L \leftarrow \mathcal{R}_L^{\nu_1}(v_L, f_L)$
$f_{L-1} \leftarrow I_{L-1}^L (f_L - A_L v_L)$
endfor
$v_1 \leftarrow \mathcal{R}_1^{\nu_0}(v_1, f_1)$
for $L = 2$ **to** Λ **do**
$v_L \leftarrow v_L + I_L^{L-1} v_{L-1}$
$v_L \leftarrow \mathcal{R}_L^{\nu_2}(v_L, f_L)$
endfor
end

Algorithm 3.4 (Multigrid M-cycle)
A recursive iteration for solving $A_\Lambda u_\Lambda = f_\Lambda$, given some initial guess v_1.

procedure M-cycle (v, f, L)

begin
if $L \neq 1$ **then**
$f_{L-1} \leftarrow I_{L-1}^{L}\,(f_L - A_L v_L)$
$v_{L-1} \leftarrow 0$
$v_{L-1} \leftarrow$ M-cycle $(v, f, L-1)$
$v_L \leftarrow v_L + I_L^{L-1}\, v_{L-1}$
endif
$v_L \leftarrow$ V-cycle (v, f, L)
end

Algorithm 3.5 (Multigrid M-cycle)
A non-recursive iteration for solving $A_\Lambda u_\Lambda = f_\Lambda$, given some initial guess v_1.

procedure M-cycle (v, f, L)

begin
$v_1 \leftarrow \mathcal{R}_1^{\nu_0}(v_1, f_1)$
for $L = 2$ **to** Λ **do**
$v_L \leftarrow I_L^{L-1}\, v_{L-1}$
$v_L \leftarrow$ V-cycle (v, f, L)
endfor
end

Full-weighting restriction is given by

$$\begin{aligned} v^{2h}_{i,j} &= \tfrac{1}{4} v^h_{2i,2j} + \tfrac{1}{8}(v^h_{2i-1,2j} + v^h_{2i+1,2j} + v^h_{2i,2j-1} + v^h_{2i,2j+1}) \\ &\quad + \tfrac{1}{16}(v^h_{2i-1,2j-1} + v^h_{2i-1,2j+1} + v^h_{2i+1,2j-1} + v^h_{2i+1,2j+1}) \end{aligned}$$

$$\text{for} \quad i = 1, 2, \ldots, \tfrac{M}{2} - 1; \quad j = 1, 2, \ldots, \tfrac{N}{2} - 1$$

which we can write in stencil form as (*cf.* our earlier molecule diagrams)

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

The analogue of one-dimensional linear interpolation is called *bilinear* interpolation:

$$\begin{aligned} v^h_{2i,2j} &= v^{2h}_{i,j} \\ v^h_{2i+1,2j} &= \tfrac{1}{2}(v^{2h}_{i,j} + v^{2h}_{i+1,j}) \\ v^h_{2i,2j+1} &= \tfrac{1}{2}(v^{2h}_{i,j} + v^{2h}_{i,j+1}) \\ v^h_{2i+1,2j+1} &= \tfrac{1}{4}(v^{2h}_{i,j} + v^{2h}_{i+1,j} v^{2h}_{i,j+1} + v^{2h}_{i+1,j+1}) \end{aligned}$$

$$\text{for} \quad i = 1, 2, \ldots, \tfrac{M}{2} - 1; \quad j = 1, 2, \ldots, \tfrac{N}{2} - 1$$

which in stencil form is given by

$$\frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

We shall henceforth use linear interpolation as a generic term for interpolation in n dimensions using linear functions. Again note that full-weighting restriction and linear interpolation form a natural pair of intergrid operators, since I^h_{2h} and I^{2h}_h are then adjoint operators (up to a constant factor).

Cubic interpolation in two dimensions can be written in stencil form as

$$\frac{1}{256} \begin{bmatrix} 1 & 0 & -9 & -16 & -9 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -9 & 0 & 81 & 144 & 81 & 0 & -9 \\ -16 & 0 & 144 & 256 & 144 & 0 & -16 \\ -9 & 0 & 81 & 144 & 81 & 0 & -9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -9 & -16 & -9 & 0 & 1 \end{bmatrix}.$$

The coarse-grid PDE operator A_{2h} is naturally defined in two dimensions by the discretisation; the matrix is of size $MN \times MN$.

The two-dimensional accuracy measures for v_h are also analogous to those defined in one dimension; the discrete norms are given by

$$\|e\|_2 = \sqrt{hk \sum_{i=1}^{M} \sum_{j=1}^{N} e_{ij}^2} \qquad \text{and} \qquad \|e\|_\infty = \max_{\substack{1 \le i \le M \\ 1 \le j \le N}} |e_{ij}|.$$

We now consider the computational cost of relaxation for the various multigrid cycles. Relaxation is generally the most costly element of a multigrid algorithm (see Appendix B for an example cost-breakdown); indeed some authors ignore the cost of all other operations. In order to keep our remarks independent of the implementation, we choose the framework of *work units* (WU) used by Briggs [20] and others. A work unit is defined to be the cost of performing one relaxation sweep on the finest grid. We will assume that N_1 and M_1 (the number of grid points on the coarsest grid) are small, and that Λ is reasonably large so that

$$\sum_{L=1}^{\Lambda} 2^{-L} \approx 1.$$

Firstly consider one-dimensional multigrid; the cost of relaxation on level L is

$$W_L = 2^{L-\Lambda} \quad \text{WU}$$

therefore the total relaxation cost for a $V_\Lambda^{1,2,1}$-cycle is

$$W_V = 2\sum_{L=1}^{\Lambda} W_L \approx 4 \quad \text{WU}.$$

For a $V_L^{1,2,1}$-cycle, the cost is

$$W_{V(L)} \approx 4W_L = 2^{L-\Lambda+2} \quad \text{WU}$$

hence the total relaxation cost for an $M_\Lambda^{1,2,1}$-cycle is

$$W_M = \sum_{L=1}^{\Lambda} W_{V(L)} \approx 8 \quad \text{WU}.$$

For the purposes of calculating the cost of a $W_\Lambda^{1,2,1}$-cycle, we introduce the quantity η_L, the number of visits to level L during a W-cycle:

$$\eta_L = \begin{cases} 2^{\Lambda-2} & L = 1 \\ 3 \cdot 2^{\Lambda-L-1} & L = 2, 3, \ldots, \Lambda-1 \\ 2 & L = \Lambda \end{cases}$$

(see Figure 3.3), hence

$$W_W = \sum_{L=1}^{\Lambda} W_L \eta_L = \tfrac{1}{2}(3\Lambda - 1) \quad \text{WU}.$$

To generalise these results to d dimensions, we note that the basic cost of relaxation is now

$$W_L = 2^{d(L-\Lambda)} \quad \text{WU}$$

hence

$$W_V \approx \frac{2}{1-2^{-d}} \quad \text{WU}$$

$$W_M \approx \frac{2}{(1-2^{-d})^2} \quad \text{WU}$$

and

$$W_W = 2 + 2^{(1-\Lambda)(d-1)-1} + 3 \cdot 2^{\Lambda(1-d)-1} \sum_{L=2}^{\Lambda-1} 2^{L(d-1)} \quad \text{WU}.$$

The following table compares the relaxation costs for some typical cycles.

d	$V_\infty^{1,2,1}$	$M_\infty^{1,2,1}$	$W_7^{1,2,1}$
1	4.00	8.00	10.0
2	2.67	3.50	3.46
3	2.29	2.50	2.50

Formulae very similar to those given above apply to the cost of restriction, prolongation and residue calculation (each in terms of the cost on the finest grid). This is as much as we can say for the moment without considering the arithmetic cost of explicit finite-difference operators A_L, $\mathcal{R}$, I_{L-1}^L, I_L^{L-1}, and an explicit computer implementation for these.

3.4 Smoothing and Convergence Rates

In Chapter 2 we saw that the optimal smoothing rate for 1-D weighted Jacobi relaxation on the Poisson equation discretised by the five-point star occurs for $\omega_{\text{opt}} = 2/3$. We also introduced some terms describing the smoothing and convergence properties, which we now define more rigorously (and in two dimensions).

The *convergence factor* of a relaxation scheme is defined as [50]

$$\rho = \max_{1 \le i \le M} \max_{1 \le j \le N} |\lambda_{ij}(\mathcal{M})|$$

which is just the spectral radius of the iteration matrix $\mathcal{M}$. By writing the error of the k^{th} iterate in terms of a discrete Fourier expansion, we can see how relaxation affects the various frequency modes:

$$e_{ij}^k = \sum_{0 \le \theta \le \pi} \psi_\theta^k \, e^{\imath(i\theta_1 + j\theta_2)}$$

where $\imath^2 = -1$ and $\theta = (\theta_1, \theta_2)$. Thus the convergence factor is conveniently measured by

$$\rho = \max_{0 \le \theta \le \pi} \left| \frac{\psi_\theta^{k+1}}{\psi_\theta^k} \right|.$$

Strictly speaking, the error should be expressed as an expansion in the eigenbasis of $\mathcal{M}$; hence this equation is valid only for discrete operators whose eigenvectors are Fourier modes (such as those arising from Poisson's equation). However, this is the standard treatment since it is very convenient (and is exact) for the simple

linear equations, and because the true eigenvectors are often perturbations of the Fourier modes.

The *smoothing factor* of a relaxation scheme is simply the convergence factor over the domain of oscillatory modes:

$$\begin{aligned} \mu &= \max_{\frac{M}{2} \le i \le M} \max_{\frac{N}{2} \le j \le N} |\lambda_{ij}(\mathcal{M})| \\ &= \max_{\frac{\pi}{2} \le \theta \le \pi} \left| \frac{\psi_\theta^{k+1}}{\psi_\theta^k} \right|. \end{aligned}$$

The *smoothing rate* is defined as $1/|\log \mu|$.

We therefore have two mathematical tools at our disposal which allow us to predict the all-important smoothing rate of a relaxation: eigenvalue analysis and Fourier analysis. The analysis of Fourier components is called *local mode analysis*, and enables us to calculate the smoothing rate for a given difference and relaxation scheme. Note, however, that local mode analysis is not rigorous: it assumes periodic boundary conditions, and "freezes" the coefficients of a variable-coefficient PDE.

As a demonstration, we summarise the eigenvalue analysis for 1-D weighted Jacobi relaxation for the Poisson equation $\Delta u = f$ discretised by the usual five-point scheme (see also Section 2.3). We begin with the matrix partition (or *splitting*) of the PDE operator

$$Au = (D - L - U)\, u = f$$

which results in a weighted Jacobi iteration scheme of

$$v^{k+1} = \omega D^{-1}(L + U)\, v^k + (1 - \omega)\, v^k - \omega D^{-1} f$$

hence our iteration matrix is

$$\mathcal{M} = (1 - \omega)\, I + \omega D^{-1}(L + U).$$

The eigenvalues of $\mathcal{M}$ are

$$\lambda_j(\mathcal{M}) = 1 - 2\omega \sin^2 \frac{j\pi}{2N} \qquad \text{for} \quad j = 1, 2, \ldots, N-1.$$

The optimal weighting ω_{opt} is given by the condition

$$\left|\lambda_{\frac{N}{2}}\right| = \left|\lambda_N\right|$$

hence

$$\left(1 - 2\omega_{\text{opt}} \sin^2 \tfrac{\pi}{4}\right) = -\left(1 - 2\omega_{\text{opt}} \sin^2 \tfrac{\pi}{2}\right)$$

which gives us

$$\omega_{\text{opt}} = \tfrac{2}{3} \qquad \text{and} \qquad \mu_{\text{opt}} = \tfrac{1}{3}.$$

This compares favourably with a 1-D Gauss-Seidel relaxation smoothing factor of $\mu_{\text{opt}} = \frac{1}{2}$ (see Hackbusch [51]).

To contrast this technique, we proceed to analyse 2-D weighted Jacobi point relaxation for the Poisson equation using Fourier analysis. The relaxation is given by

$$v_{ij}^{k+1} = (1-\omega)\, v_{ij}^{k} + \tfrac{1}{4}\omega \left(\sum_{nn'} v_{i'j'}^{k} - h^2 f_{ij} \right)$$

and so the error in the $(k+1)^{\text{th}}$ iterate is

$$e_{ij}^{k+1} = (1-\omega)\, e_{ij}^{k} + \tfrac{1}{4}\omega \sum_{nn'} e_{i'j'}^{k}.$$

The Fourier representation of this is

$$\sum_{\theta} \psi_{\theta}^{k+1} e^{\imath(i\theta_1 + j\theta_2)}$$

$$= \sum_{\theta} \left[\psi_{\theta}^{k} + \tfrac{1}{4}\omega(\psi_{\theta}^{k} e^{-\imath\theta_1} + \psi_{\theta}^{k} e^{+\imath\theta_1} + \psi_{\theta}^{k} e^{-\imath\theta_2} + \psi_{\theta}^{k} e^{+\imath\theta_2} - 4\psi_{\theta}^{k}) \right] e^{\imath(i\theta_1 + j\theta_2)}$$

hence

$$\frac{\psi_{\theta}^{k+1}}{\psi_{\theta}^{k}} = 1 + \tfrac{1}{2}\omega(\cos\theta_1 + \cos\theta_2 - 2)$$

(see Figure 3.4) and so the smoothing factor is

$$\mu = \max\left(|1 - 2\omega|, |1 - \tfrac{\omega}{2}| \right).$$

We therefore obtain

$$\omega_{\text{opt}} = \tfrac{4}{5} \qquad \text{and} \qquad \mu_{\text{opt}} = \tfrac{3}{5}$$

with a general smoothing factor for weighted Jacobi relaxation of

$$\mu = \begin{cases} \left(1 - \frac{\omega}{2}\right)^{\nu} & \text{for} \quad 0 \le \omega \le \frac{4}{5} \\ (2\omega - 1)^{\nu} & \text{for} \quad \frac{4}{5} \le \omega < 1 \end{cases}.$$

We wish to emphasise that this result is exact because these Fourier modes are indeed the eigenbasis of the discrete 2-D Laplacian operator. In fact, the eigenvalues of $A = N^2\, \text{TRIDIAG}[D, T, D]$, where $T = \text{TRIDIAG}[-1, 4, -1]$ and $D = -I$, are

$$\lambda_{ij}(A) = 2N^2\,(2 - \cos i\pi h - \cos j\pi h) \qquad \text{for} \quad i, j = 1, 2, \ldots, N-1$$

and the iteration matrix is (*cf.* Section 2.3)

$$\mathcal{M} = I - \tfrac{1}{4}\omega h^2 A,$$

yielding precisely

$$\frac{\psi_{\theta}^{k+1}}{\psi_{\theta}^{k}} = \lambda_{ij}(\mathcal{M})$$

in this case.

For other relaxation schemes, we find smoothing factors of [51]

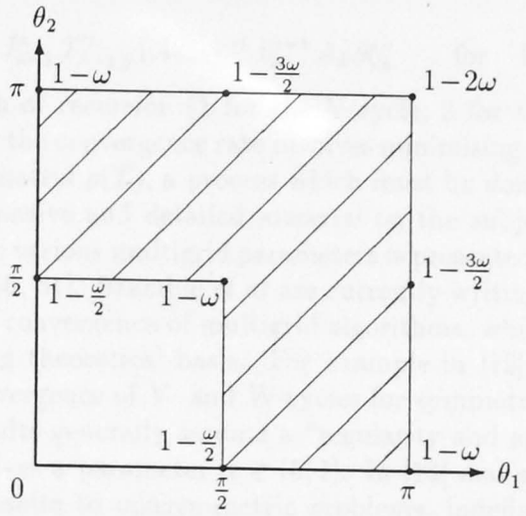


Figure 3.4: Fourier space diagram of the smoothing action of 2-D weighted Jacobi relaxation: $\psi_\theta^{k+1}/\psi_\theta^k = 1 + \frac{\omega}{2}(\cos\theta_1 + \cos\theta_2 - 2)$. The shaded region of the (θ_1, θ_2) plane contains the high-frequency (oscillatory) modes. Several representative points in that region are evaluated in terms of ω.

lexicographic pointwise Gauss-Seidel relaxation: $\mu = 2^{-\nu}$
lexicographic linewise Gauss-Seidel relaxation: $\mu = 5^{-\nu/2}$
symmetric linewise Gauss-Seidel relaxation: $\mu = 4^{-\nu}$.

We may also summarise the convergence factors as follows (see Ames [5] for further details):

$$\rho = \max|\lambda_{ij}| = \cos h \sim 1 - \tfrac{1}{2}h^2 \qquad \text{(Jacobi)}$$
$$\rho = \max|\lambda_{ij}| = \cos^2 h \sim 1 - h^2 \qquad \text{(Gauss-Seidel)}$$
$$\rho \sim 1 - 2h \qquad \text{(optimal Gauss-Seidel)}$$

The optimal weighting for Gauss-Seidel relaxation on Poisson's equation in 2-D discretised by the five-point star is [5]

$$\omega_{\text{opt}} = \frac{2}{1 + \pi h},$$

whereas for Jacobi relaxation it is

$$\omega_{\text{opt}} = \tfrac{2}{3}.$$

Matrix analysis provides a method of computing convergence of the entire multigrid scheme, although it is somewhat cumbersome. The two-grid process is described by

$$T = [I - I_h^{2h}(A_{2h})^{-1} I_{2h}^h A_h]\,\mathcal{R}^\nu, \tag{3.2}$$

and the multigrid cycle (without post-relaxation) from level k to level j $(j < k)$ is given by

$$T_{k,k-1} = [I - I_{k-1}^k (A_{k-1})^{-1} I_k^{k-1} A_k]\,\mathcal{R}_k^\nu \qquad \text{for} \quad 1 < k \le \Lambda,$$

$$T_{k,j} = T_{k,k-1} + I_{k-1}^{k} T_{k-1,j}^{\gamma} (A_{k-1})^{-1} I_{k}^{k-1} A_k \mathcal{R}_k^{\nu} \qquad \text{for} \quad 1 < j < k \leq \Lambda,$$

where γ is the depth of recursion (1 for the V-cycle, 2 for the W-cycle). The process of optimising the convergence rate involves minimising the spectral radius of the total process matrix $\rho(T)$, a process which must be done numerically.

Much more extensive and detailed material on the subject of convergence and its relation to the various multigrid parameters is presented in Brandt [15, 17] and Hackbusch [49, 50, 51]. Bramble *et al* are currently writing an important series of papers on the convergence of multigrid algorithms, which appear to place multigrid on a strong theoretical basis. For example in [12], results are given which guarantee convergence of V- and W-cycles for symmetric positive-definite problems. These results generally assume a "regularity and approximation" hypothesis, which involves a parameter $\alpha \in (0, 1]$. In [13] and subsequent papers, they extend these results to nonsymmetric problems, indefinite problems, *etc.* This coverage of convergence results may turn out to be a milestone in the development of multigrid methods.

3.5 Multigrid Parameter Space

We have drawn a picture of the multigrid process as a collection of neatly interrelated subprocesses, each having several free parameters. This results in a high-dimensional multigrid parameter space; in other words, there is a great deal of choice presented to the multigrid practitioner. The following is a partial list of these parameters:

- Type of cycle (V-cycle, W-cycle, M-cycle, ...)
- Number of cycles (fixed, determined adaptively, ...)
- Number of grid levels (Λ)
- Type of restriction I_{L-1}^{L} (injection, full-weighting, ...)
- Restriction weighting (ϖ)
- Type of interpolation I_{L}^{L-1} (piecewise linear, quadratic, cubic, ...)
- Number of relaxations (ν_1, ν_0, ν_2, determined adaptively, ...)
- Type of relaxation $\mathcal{R}$
  - Jacobi, Gauss-Seidel, ...
  - weighting (SOR) factor ω
  - pointwise, row-wise, column-wise, ...
  - red-black, lexicographic, symmetric, zebra, alternating, ...
- Size of coarsest grid (h_1)
- Type of grid coarsening h_L/h_{L+1} (standard coarsening, semi-coarsening, $\sqrt{2}$-coarsening, ...)
- Choice of coarse-grid matrix A_L ($L = 1, 2, \ldots, \Lambda - 1$) (natural, Galerkin approach, ...)

- Type of grid (uniform, isotropic, non-isotropic, fixed, adaptive, ...)
- Type of finite-difference discretisation (five-point, nine-point, ...)
- Type of error estimate (error, residual, two-norm, infinity-norm, ...)
- Type of initial guess (zero, user-defined, discrete-Laplacian, ...)
- Arithmetic precision (double, single, ...)
- Use of visualisation, immediate correction, debugging, graphic output, ...

We have various theoretical principles to guide us in the choice of multigrid parameters (such as "match the interpolation to the order of the PDE", "smooth sufficiently to reach truncation error on each level", and "red-black Gauss-Seidel relaxation is incompatible with injection"). Despite these guidelines, multigrid is far from being a fixed method, and some experimentation is desirable. As we have said, it was for this purpose that the software package `MGLAB` was written. It allows the multigrid practitioner to freely experiment with the parameter space for a given problem.

The above list contains three items not yet discussed: *immediate correction*, adaptive strategies and a *discrete-Laplacian* initial guess.

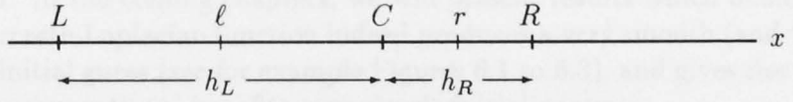
Immediate correction is a very useful tool for analysing (and debugging) multigrid algorithms. It is a process which may be added to the multigrid algorithm which makes a coarse-grid correction directly to the fine grid after each set of coarse-grid relaxations. This enables us to observe the immediate effect of each coarse-grid adjustment with respect to the fine-grid solution, a process which is otherwise not quantitatively apparent. Immediate correction is merely an aid to interpreting multigrid convergence, and has no effect itself on the convergence.

Strategies for the multigrid process may include the use of fixed or adaptive grids, fixed or adaptive values of ν (*ie.* relax until some tolerance condition is met), and fixed or adaptive cycle strategies (where switching from one level to another is controlled by some criterion).

3.6 Automatic Fine-grid Initial Guess

An interesting sub-problem is to find a method of generating an automatic initial guess v_h^0, satisfying the boundary conditions of the PDE. This is a useful option for V- and W-cycles, which begin on the finest grid Ω_h. As we expect elliptic boundary value problems to result in smooth solutions, a reasonable method is to generate the initial guess using a discrete Laplacian-type interpolation of the boundary data.

Firstly consider the following one-dimensional situation:



Writing v_L for $v(x=L)$ *etc*, the forward and backward finite-difference derivatives of the function v at the point C are

$$v'_{\text{fwd}}(C) = \frac{v_R - v_C}{h_R} \quad \text{and} \quad v'_{\text{bwd}}(C) = \frac{v_C - v_L}{h_L}.$$

Taking the mean of these, we find the central derivative to be

$$v'(C) = \frac{h_L(v_R - v_C) + h_R(v_C - v_L)}{2h_L h_R}. \tag{3.3}$$

We also know the central derivatives at the midpoints ℓ and r:

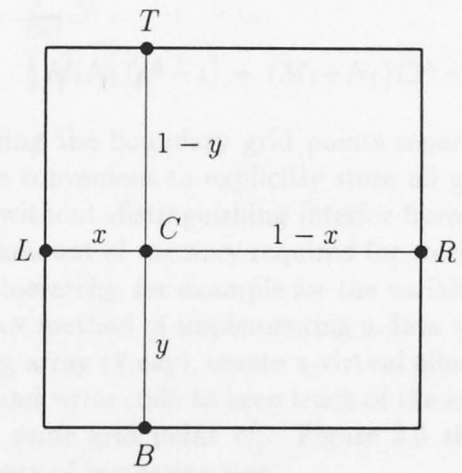
$$v'(\ell) = \frac{v_C - v_L}{h_L} \quad \text{and} \quad v'(r) = \frac{v_R - v_C}{h_R}.$$

Applying equation 3.3 to first derivatives, we obtain for the second derivative

$$\begin{aligned} v''(C) &= \frac{\frac{h_L}{2}[v'(r) - v'(C)] + \frac{h_R}{2}[v'(C) - v'(L)]}{2\,\frac{h_L}{2}\,\frac{h_R}{2}} \\ &= \frac{h_L^2(v_R - v_C) + h_L h_R(v_R - 2v_C + v_L) + h_R^2(v_L - v_C)}{2\,h_L^2\,h_R^2} \end{aligned}$$

which reduces to the usual second derivative when $h_L = h_R = h$.

In two dimensions, consider the following situation:



Applying our one-dimensional result to Laplace's equation $v_{xx} + v_{yy} = 0$ gives

$$v_C = \frac{y^2(1-y)^2[x\,v_R + (1-x)v_L] + x^2(1-x)^2[y\,v_T + (1-y)v_B]}{y^2(1-y)^2 + x^2(1-x)^2}$$

which relates the value of an interior point $v(x,y)$ to the boundary data, and is our desired result. Note that the coefficients of v_R,v_L,v_T,v_B sum to unity, as expected. In the coming chapters, we will present results which demonstrate that this discrete-Laplacian function indeed produces a very smooth (and visually satisfying) initial guess (see for example Figures 6.1 to 6.3), and gives rise to very significant computational benefits over simple initial guesses.

3.7 Multigrid Implementation

MGLAB was developed firstly in standard FORTRAN 77 so that the package could be run on the widest possible class of computers. Once it was complete, the library of routines was then ported to a Connection Machine, using the CM Fortran language, so that observations could be made about multigrid on a parallel processing system (see Chapter 8). This section is concerned with the implementation of MGLAB on a standard serial machine.

We will firstly consider memory requirements for the multigrid algorithm. Recall that M_1+1 and N_1+1 give the actual number of points on the coarsest grid ($L = 1$) in the x- and y-directions, respectively. Then standard coarsening of the grid defines M_L and N_L for all finer grids:

$$M_L = 2^{L-1}M_1 \qquad \text{and} \qquad N_L = 2^{L-1}N_1 \qquad \text{for} \qquad L = 2, 3, \ldots, \Lambda.$$

The total number of grid points on level L is

$$t_L = \left(2^{L-1}M_1 + 1\right)\left(2^{L-1}N_1 + 1\right),$$

and hence the total number of grid points on all levels is

$$\begin{aligned} \tau_\Lambda &= \sum_{L=1}^{\Lambda} t_L \\ &= \tfrac{1}{3}M_1N_1\left(4^\Lambda - 1\right) + \left(M_1+N_1\right)\left(2^\Lambda - 1\right) + \Lambda. \end{aligned} \tag{3.4}$$

Rather than storing the boundary grid points separately from the interior grid points, it is more convenient to explicitly store all grid-point values in the same data structure, without distinguishing interior from boundary grid points. Hence τ_Λ is the total amount of memory required for storing one full representation of the multi-grid hierarchy, for example for the variable v_L ($L = 1, 2, \ldots, \Lambda$). The standard FORTRAN method of implementing a data structure of this kind is to declare a single long array (V say), create a virtual allocation of grid variables $v_1, v_2, \ldots, v_\Lambda$ from it, and write code to keep track of the correspondence between an element V(I) and some grid point v_{ij}^L. Figure 3.5 shows how we assemble virtual blocks of memory of increasing size

$$M_1 \times N_1,\ M_2 \times N_2,\ \ldots,\ M_\Lambda \times N_\Lambda$$

in this way.

Since FORTRAN indices are numbered from one upwards, the index into the single array V of the grid point v_{00}^L (the "start" of level L) is

$$\begin{aligned} s_L &= 1 + \sum_{k=1}^{L-1} M_kN_k \\ &= \tfrac{1}{3}M_1N_1\left(4^{L-1} - 1\right) + \left(M_1+N_1\right)\left(2^{L-1} - 1\right) + L. \end{aligned}$$

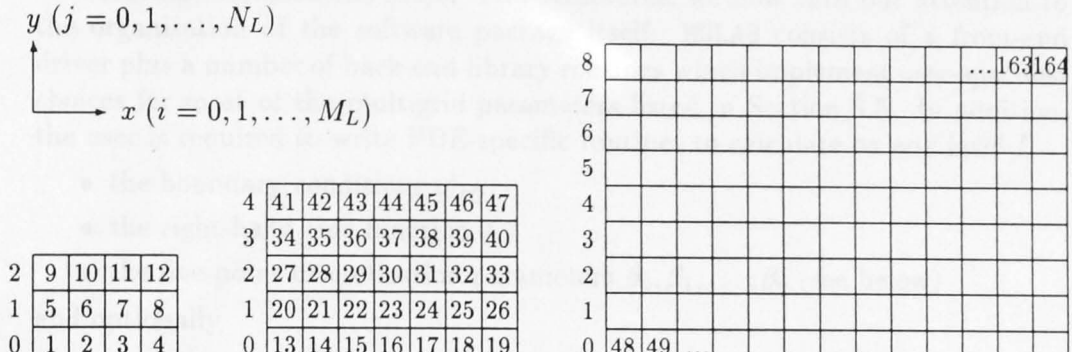


Figure 3.5: Diagram of the first three levels of a multi-grid hierarchy with $M_1 = 3$ and $N_1 = 2$ laid out with respect to a single long FORTRAN array. Each square corresponds to a grid point, and is labelled with its index $\texttt{MAP}(L, i, j)$.

One way to achieve the index book-keeping is to use a FORTRAN statement function

$$\texttt{MAP}(L, i, j) = s_L + (N_L{+}1) * i + j.$$

This is efficient due to the combination of using look-up tables (LUT's) for s_L and the statement function, which is compiled to in-line code on almost all machines.

We need to allocate sufficient memory to store a full multi-grid representation for the variables v and f, and a single Λ-grid for the variables r and e. Thus the total memory required for the major data structures is

$$T_\Lambda = 2\,(\tau_\Lambda + t_\Lambda) \quad \text{variables.}$$

Each variable is of the nominated precision; we regard double precision (8-byte variables) to be the standard choice. Moreover, a standard choice for the coarsest grid is $M_1 = N_1 = 2$ (see Section 3.2). Using equation (3.4), the following table gives the number of real variables required for a single multi-grid hierarchy of Λ levels, τ_Λ, and the total memory requirement in megabytes of MGLAB, T_Λ, assuming double-precision variables.

Λ	5	6	7	8	9	10
τ_Λ	1493	5718	22359	88408	351577	1402202
T_Λ	0.04	0.15	0.60	2.36	9.38	37.43

To obtain solutions of fine resolution, we see that a substantial amount of memory is required. For this reason alone, it is desirable to utilise a computer with a large amount of main memory, such as the Connection Machine.

Having discussed the major data structures, we now turn our attention to the organisation of the software package itself. MGLAB consists of a front-end driver plus a number of back-end library routines which implement user-specified choices for most of the multigrid parameters listed in Section 3.5. In addition, the user is required to write PDE-specific routines to calculate on any level L

- the boundary conditions $v|_{\partial\Omega}$
- the right-hand side function f
- the five-point discretisation parameters $\beta_0, \beta_1, \ldots, \beta_4$ (see below)

and optionally

- the action of the PDE operator Av
- the initial guess v^0
- the exact solution u
- a pointwise relaxation based on A.

Recall from Section 2.3 that the method of ascending row-wise Jacobi line relaxation for Poisson's equation discretised by the five-point star is given by

$$v_{ij}^{k+1} = \tfrac{1}{4}\left(v_{i+1,j}^{k+1} + v_{i-1,j}^{k+1} + v_{i,j+1}^{k} + v_{i,j-1}^{k} - h^2 f_{ij}\right),$$

that is, v is the solution of some tridiagonal system $Bv = \phi$. We have implemented line relaxation using the LINPACK [34] tridiagonal solver. For efficiency reasons, the process is decomposed into two parts: firstly pre-factorisation of the coefficient matrix B (using the routine xGBFA), and secondly repeated solution of the system for some ϕ (using xGBSL), where x = D for double precision and S for single precision. To simplify this discussion, we shall consider only constant coefficient PDE's. Suppose we wish to solve a BVP of the form

$$au_{xx} + cu_{yy} + du_x + eu_y + gu = f$$

with the Dirichlet boundary conditions

$$u(0,y) = x_0, \qquad u(1,y) = x_1,$$
$$u(x,0) = y_0, \qquad u(x,1) = y_1.$$

By employing the standard five-point finite differences, we find that the discretised equation is

$$\beta_0 v_{ij} + \beta_1 v_{i-1,j} + \beta_2 v_{i+1,j} + \beta_3 v_{i,j-1} + \beta_4 v_{i,j+1} = f_{ij}$$

where

$$\begin{aligned}
\beta_0 &= g - 2\left(aM^2 + cN^2\right) \\
\beta_1 &= M\left(aM - \tfrac{1}{2}d\right) \\
\beta_2 &= M\left(aM + \tfrac{1}{2}d\right) \\
\beta_3 &= N\left(cN - \tfrac{1}{2}e\right) \\
\beta_4 &= N\left(cN + \tfrac{1}{2}e\right).
\end{aligned}$$

Thus for ascending row-wise line relaxation we need to successively update each row j by solving the tridiagonal system

$$\beta_1 v_{i-1,j} + \beta_0 v_{ij} + \beta_2 v_{i+1,j} = f_{ij} - \beta_3 v_{i,j-1} - \beta_4 v_{i,j+1} \qquad \text{for} \quad i = 1, 2, \ldots, M-1$$

in other words the $(M-1) \times (M-1)$ system

$$\text{TRIDIAG}[\beta_1, \beta_0, \beta_2]\, v_j = \phi,$$

where the boundary conditions are incorporated into the column vector ϕ as follows

$$\begin{aligned} \phi_1 &= f_{ij} - \beta_3\, v_{1,j-1} - \beta_4\, v_{1,j+1} - \beta_1\, x_0. \\ \phi_{M-1} &= f_{ij} - \beta_3\, v_{M-1,j-1} - \beta_4\, v_{M-1,j+1} - \beta_2\, x_1. \end{aligned}$$

As already stated, the constant matrix $B = \text{TRIDIAG}[\beta_1, \beta_0, \beta_2]$ is pre-factored using the routine xGBFA, then relaxation proceeds by forming ϕ for each row then solving $Bv_j = \phi$ by means of xGBSL.

Note that the five-point discretisation parameters $\beta_0, \beta_1, \ldots, \beta_4$ are also employed by MGLAB to enable automatic pointwise relaxation

$$v_{ij} = \frac{1}{\beta_0} \left[\beta_1 v_{i-1,j} + \beta_2 v_{i+1,j} + \beta_3 v_{i,j-1} + \beta_4 v_{i,j+1} - \frac{f_{ij}}{N^2} \right]$$

and automatic residual calculation

$$r_{ij} = f_{ij} - \left[\beta_1 v_{i-1,j} + \beta_2 v_{i+1,j} + \beta_3 v_{i,j-1} + \beta_4 v_{i,j+1} - \beta_0 v_{ij} \right]$$

of constant-coefficient PDE's on isotropic grids. Where this is not appropriate, the user is required to write a routine to explicitly calculate the action of the PDE operator Av and another routine to perform relaxation.

The following is a list of subroutines found in a typical MGLAB front-end (those marked with an asterisk are essential):

<main program>*	Allocate memory
INIT_PDE_PARAMS*	Define PDE parameters and relaxation coefficients β_i
CALC_BDY_CONDS*	Calculate $v_L\|_{\partial\Omega}$
CALC_RHS_FUNCTION_F*	Calculate f_L
CALC_RESIDUE_R	Calculate $f_L - A_L v_L$
CALC_INITIAL_GUESS_V	Calculate v_L^0
CALC_EXACT_SOLN_U	Calculate $u\|_{\Omega_L}$
USER_JACOBI_RELAX	Perform pointwise weighted Jacobi relaxation $\mathcal{R}_L^\nu$
USER_ASC_GAUSS_SEIDEL_RELAX	Perform pointwise weighted ascending G-S relaxation
USER_R_B_GAUSS_SEIDEL_RELAX	Perform pointwise weighted red-black G-S relaxation

The following subroutines are to be found in the serial MGLAB library:

INITIALISE	Initialise data structures and parameters
INIT_CONSTANTS	Initialise constants
READ_USER_OPTIONS	Query user for multigrid parameter choices
INIT_LEVEL_LUT	Initialise look-up tables
VALIDATE_LMAX	Ensure sufficient memory is available for Λ levels
FACTORISE_LINE_RELAX_MATRIX	Pre-factorise tridiagonal matrix B
PRINT_HEADING	Indicate which PDE is being solved
SOLVE_PDE	Perform the chosen multigrid process
MULTIGRID_V_CYCLE	Perform $1 \times M_L^{\nu_1,\nu_0,\nu_2}$-cycle
V_CYCLE	Perform $1 \times V_L^{\nu_1,\nu_0,\nu_2}$-cycle
PERFORM_RELAXATION	Perform ω-weighted relaxation $\mathcal{R}_L^{\nu}$:
JACOBI_RELAXATION	pointwise Jacobi
ASCENDING_GAUSS_SEIDEL_RELAX	ascending pointwise Gauss-Seidel
RED_BLACK_GAUSS_SEIDEL_RELAX	red-black pointwise Gauss-Seidel
HORIZ_JACOBI_LINE_RELAX	row-wise Jacobi
VERT_JACOBI_LINE_RELAX	column-wise Jacobi
HORIZ_ASC_GS_LINE_RELAX	ascending row-wise Gauss-Seidel
PERFORM_RESTRICTION	Perform ϖ-weighted restriction I_{L-1}^L:
INJECTION_RESTRICTION	injection
WEIGHTED_RESTRICTION	full-weighting
IMMEDIATE_CORRECTION	Output result of $v_\Lambda + I_\Lambda^L v_L$
COARSE_GRID_CORRECTION	Perform $v_L \leftarrow v_L + I_L^{L-1} v_{L-1}$
BILINEAR_INTERPOLATION	Perform interpolation I_L^{L-1}
CALC_RESULT	Calculate $\|e\|$ and $\|r\|$ as appropriate
CALC_ERROR	Calculate $e = u - v$
PRINT_VF	Output all elements of v_L or f_L
PRINT_R	Output all elements of r_L
FUNCTION INFINITY_NORM	Calculate $\|x\|_\infty$
FUNCTION TWO_NORM	Calculate $\|x\|_2$
FUNCTION LOG_10	Calculate $\log_{10} x$

The following is a list of multigrid parameter options which are available to the user of the serial version of MGLAB. (We use the terminology "P-cycle" to mean pure relaxation on a fine grid.)

- cycle type (P, V, W, or M)
- number of grid levels (Λ)
- number of grid points on the coarsest grid (M_1, N_1)
- number of cycles (n)
- number of pre-relaxations (ν_1)

- number of relaxations on the coarsest grid (ν_0)
- number of post-relaxations (ν_2)
- relaxation class (based on $Av = f$ (user-defined), based on β_i (automatic), or based on $\pm\Delta v = f$)
- relaxation method (pointwise or linewise)
- relaxation type (Jacobi, ascending Gauss-Seidel, or red-black Gauss-Seidel) if pointwise method
- relaxation type (row Jacobi, column Jacobi, or ascending row Gauss-Seidel) if linewise method
- relaxation weighting (SOR) factor (ω)
- restriction type (injection or full-weighting)
- restriction weighting factor (ϖ)
- norm type (two-norm, infinity-norm, or both)
- when to print out intermediate calculations (never, once per cycle, or after each calculation)
- whether or not to use immediate correction

In addition, each package is available in single or double precision.

The inclusion of ϖ-weighted restriction in the package allows us to investigate half-injection, in particular. It is well-known that the combination of injection restriction and red-black Gauss-Seidel relaxation yields poor convergence [20, 93]. This is due to decoupling of the red and black modes, and is partially remedied by employing half-injection.

One will observe the absence of some parameter choices mentioned in Section 3.5, such as higher-order interpolation and further methods of relaxation (symmetric, zebra, *etc*). These absences are simply the result of time constraints, not programming difficulties.

A complete listing of the front-end program implementing Problem MG10 is given in Appendix A, while a partial listing of the `MGLAB` package appears at Appendix C. (These listings are of the CM Fortran versions, rather than the standard FORTRAN 77 versions, because these are more compact and much more readable.)

It should be noted that `MGLAB` is not highly optimised for speed of execution nor for memory requirements. This is because of the experimental research philosophy of the laboratory-type software, also performance has sometimes been sacrificed for the sake of flexibility. Nevertheless, its performance is reasonable; a typical problem is solved by $1 \times M_8^{2,2,2}$-cycle in less than ten CPU seconds on a typical serial machine, such as a Sun-4. The following chapter considers the numerical results of some model linear problems in detail.

Chapter 4

Model Linear Problems

What is shown by example, men think they may justly do.

— Cicero, *Ad Atticum*

What we have to learn to do, we learn by doing.

— Aristotle, *Ethica Nicomachea*

In this chapter, we discuss the numerical solution of several model linear boundary value problems using `MGLAB`. The results give us insights into the multigrid process and, conversely, permit us to check the validity of our algorithm.

4.1 Statement of Model Linear Problems

Five representative linear problems were created to enable testing of and experimentation with the software package `MGLAB`, our implementation of a laboratory-type environment for solving elliptic boundary value problems. The elliptic two-dimensional second-order linear problems to be solved are Poisson's equation, the convection-diffusion equation, an inhomogeneous Helmholtz equation, the anisotropic diffusion (boundary-layer) equation, and a variable-coefficient PDE. All have Dirichlet boundary conditions on the unit square, as discussed in Chapter 2. A statement of these model linear problems appears on the following page. Where no initial guess v^0 is specified, we shall be relying on the automatic discrete-Laplacian initial guess. These problems are fairly representative of the types of linear elliptic boundary value problems which researchers deal with. Figures 4.1 and 4.2 show the solutions of these problems for the following data (which we call the standard data set):

Problem MG12	$\sigma = 1, \quad \tau = 2, \quad \alpha = 6.0, \quad \beta = 10.0$
Problem MG14	$\sigma = 1, \quad \tau = 4$
Problem MG16	$\varepsilon^2 = 4.0$

Problem MG10 (Poisson's equation)

$$\Delta u = 2y^2(1-y^2)(1-6x^2) + 2x^2(1-x^2)(1-6y^2) \qquad \text{in } \Omega$$

$$u = 0 \qquad \text{on } \partial\Omega$$

$$v^0 = 0$$

$$u = x^2y^2(1-x^2)(1-y^2)$$

Problem MG12 (inhomogeneous Helmholtz equation)

$$-\Delta u + \beta\pi^2 u = \alpha\pi^2 \sin\sigma\pi x \sin\tau\pi y \qquad \text{in } \Omega; \quad \sigma,\tau \in \mathbb{Z}; \quad \alpha,\beta \in \mathbb{R}$$

$$u = 0 \qquad \text{on } \partial\Omega$$

$$v^0 = 0$$

$$u = \frac{\alpha}{\beta + \sigma^2 + \tau^2} \sin\sigma\pi x \sin\tau\pi y \qquad (\beta + \sigma^2 + \tau^2 \neq 0)$$

Problem MG14 (convection-diffusion equation)

$$-\Delta u + \sigma\pi u_x + \tau\pi u_y = \sigma^2 e^{-X} \sin Y + \tau^2 \sinh X (\sin Y + \cos Y) \qquad \text{in } \Omega;$$

$$\text{where} \quad X = \sigma\pi x, \; Y = \tau\pi y; \qquad \sigma,\tau \in \mathbb{Z};$$

$$u(0,y) = 0 \qquad u(1,y) = \pi^{-2} \sinh\sigma\pi \sin Y$$
$$u(x,0) = 0 \qquad u(x,1) = 0$$

$$u = \pi^{-2} \sinh X \sin Y$$

Problem MG16 (anisotropic diffusion equation)

$$u_{xx} + \varepsilon^2 u_{yy} = 0 \qquad \text{in } \Omega; \qquad \varepsilon \in \mathbb{R}, \quad \varepsilon \neq 0$$

$$u(0,y) = \sin\pi y \qquad u(1,y) = 0$$
$$u(x,0) = 0 \qquad u(x,1) = 0$$

$$v^0 = (1-x)\sin\pi y$$

$$u = \frac{1 - e^{2\varepsilon\pi(1-x)}}{1 - e^{2\varepsilon\pi}} e^{\varepsilon\pi x} \sin\pi y = \frac{\sinh\varepsilon\pi(1-x)}{\sinh\varepsilon\pi} \sin\pi y$$

Problem MG18 (variable-coefficient equation)

$$(1+y^2)\,u_{xx} + (1+x^2)\,u_{yy} - (x^4+y^4)\,u = (x^2+y^2)\,e^{xy} \qquad \text{in } \Omega$$

$$u(0,y) = 1 \qquad u(1,y) = e^y$$
$$u(x,0) = 1 \qquad u(x,1) = e^x$$

$$u = e^{xy}$$

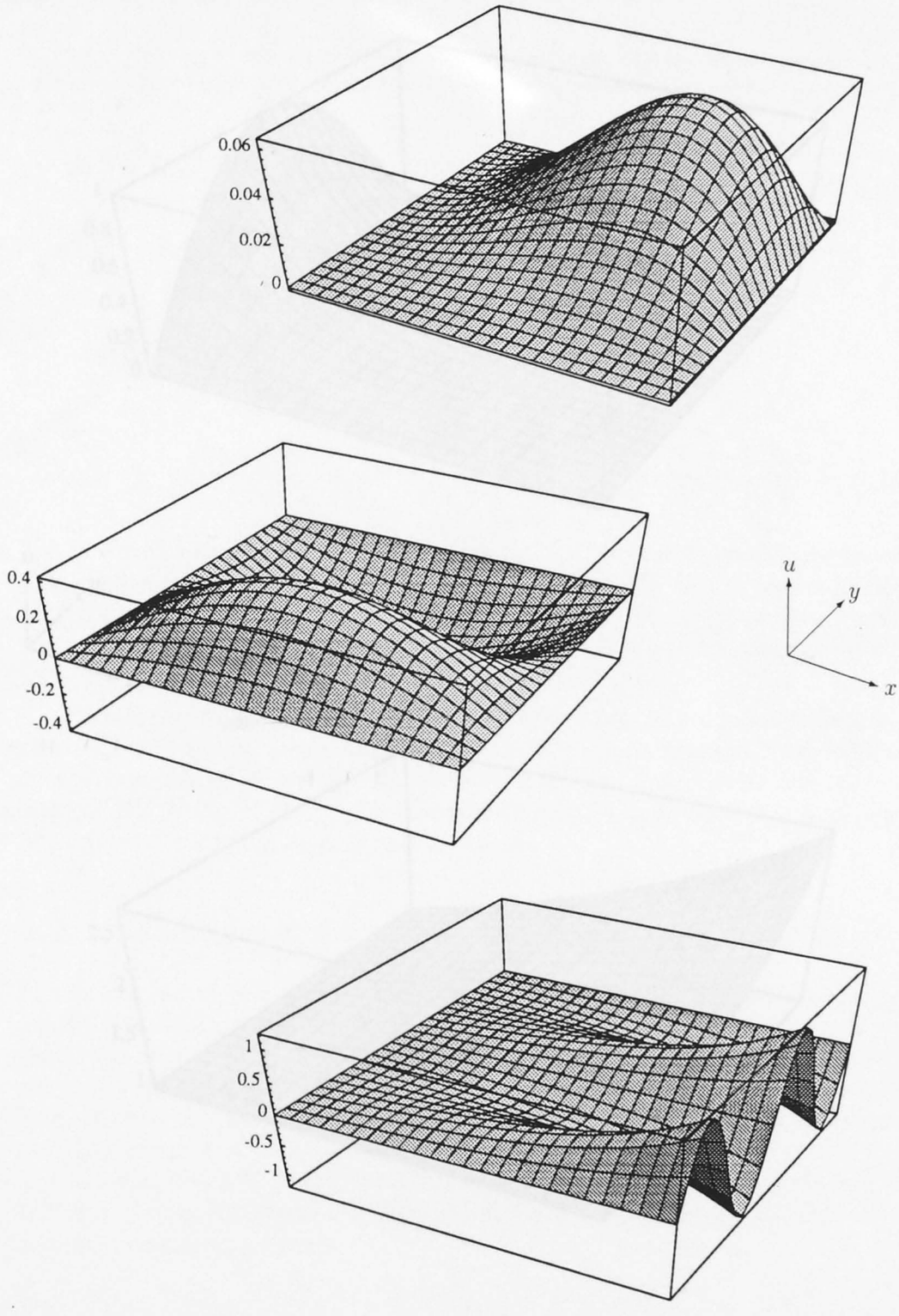


Figure 4.1: Solutions to the model linear problems MG10, MG12 and MG14 (top, centre and bottom, respectively) for the standard data set.

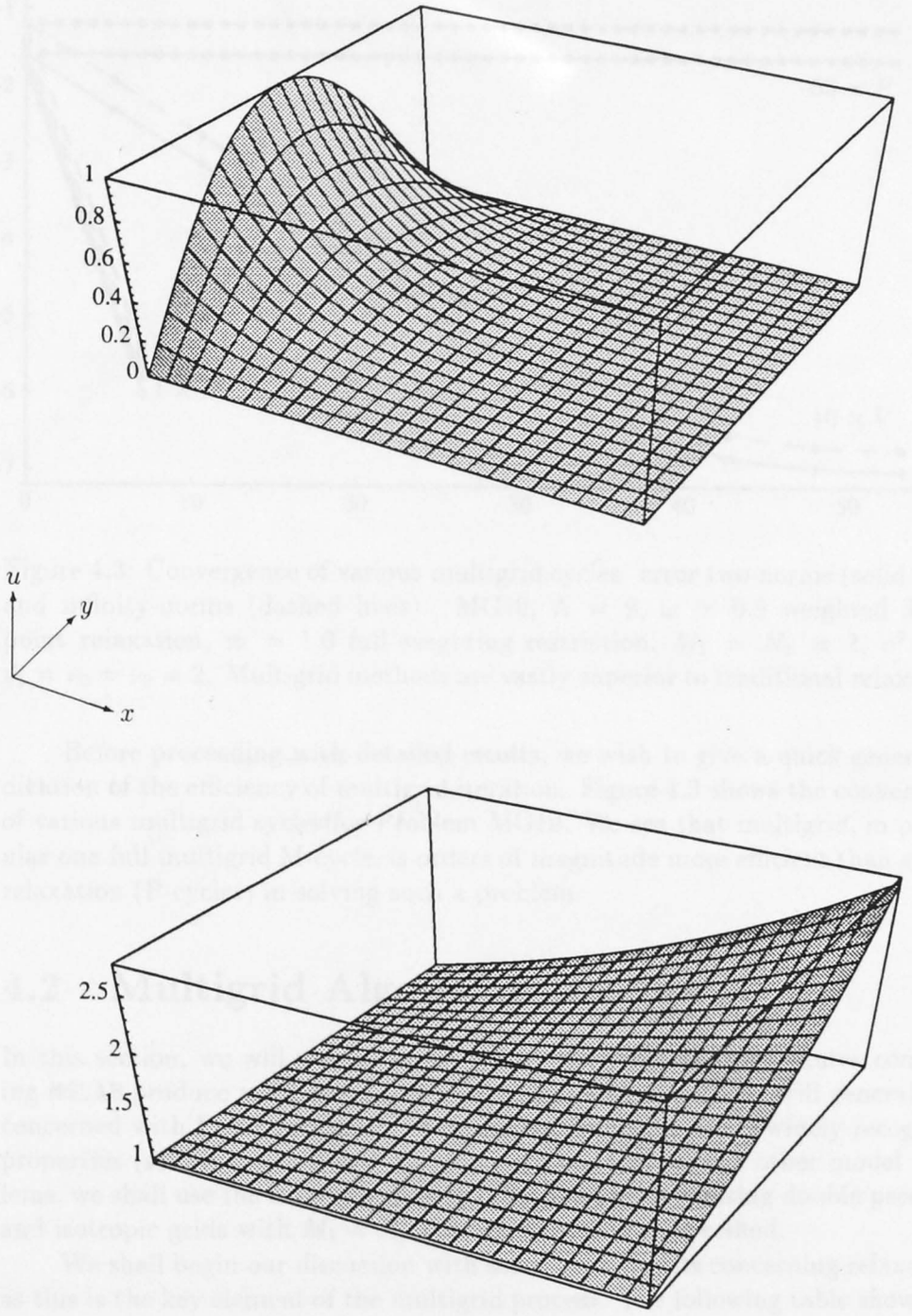


Figure 4.2: Solutions to the model linear problems MG16 and MG18 (top and bottom, respectively) for the standard data set.

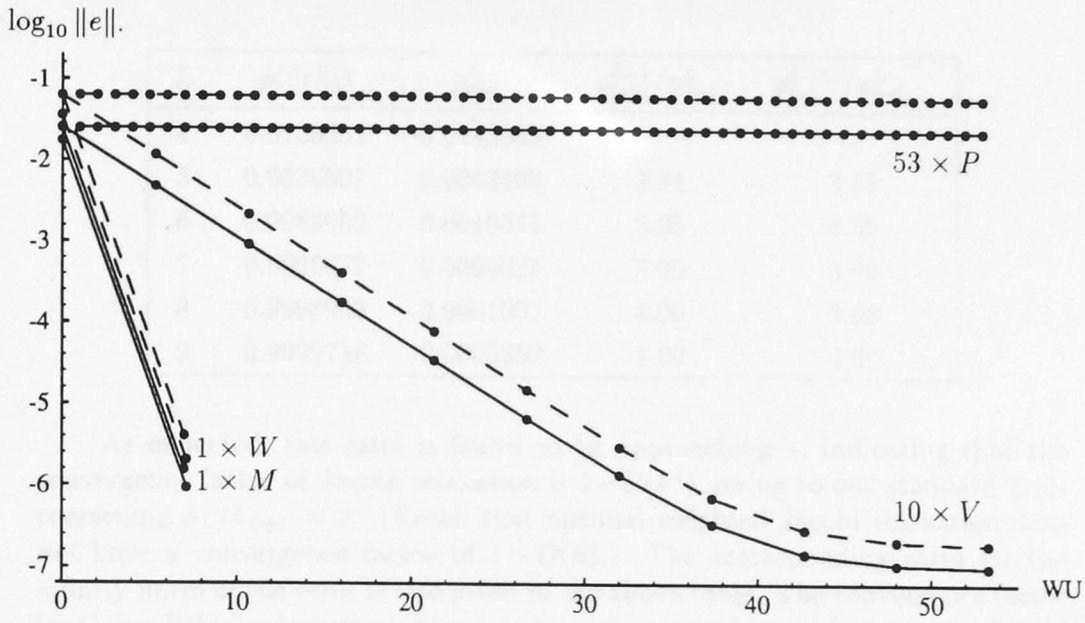


Figure 4.3: Convergence of various multigrid cycles: error two-norms (solid lines) and infinity-norms (dashed lines). MG10, $\Lambda = 9$, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $M_1 = N_1 = 2$, $v^0 = 0$, $\nu_1 = \nu_0 = \nu_2 = 2$. Multigrid methods are vastly superior to traditional relaxation.

Before proceeding with detailed results, we wish to give a quick general indication of the efficiency of multigrid iteration. Figure 4.3 shows the convergence of various multigrid cycles for Problem MG10. We see that multigrid, in particular one full multigrid M-cycle, is orders of magnitude more efficient than simple relaxation (P-cycles) in solving such a problem.

4.2 Multigrid Algorithm Results

In this section, we will present data which verify that the procedures comprising **MGLAB** produce valid numerical results. For this reason, we will generally be concerned with Problem MG10 (Poisson's equation), as it has widely-recognised properties (see Chapters 2 and 3). When we do examine our other model problems, we shall use the standard data set. Results are given using double precision and isotropic grids with $M_1 = N_1 = 2$ unless otherwise specified.

We shall begin our discussion with some experiments concerning relaxation, as this is the key element of the multigrid process. The following table shows the measured convergence factor

$$\hat{\rho} = \frac{\|e^{k+1}\|}{\|e^k\|}$$

for the two-norm of the error averaged over $4 \times P_L$-cycles of 0.8-weighted Jacobi relaxation on Problem MG10 with an initial guess of $v^0 = 0$. The quantity $\varrho = 1 - \hat{\rho}$ is also shown, together with its ratio with respect to ϱ on level $L-1$.

L	$\hat{\rho}(\|\|e\|\|_2)$	$\varrho^L_{\|\|e\|\|_2}$	$\varrho^{L-1}_{\|\|e\|\|_2}/\varrho^L_{\|\|e\|\|_2}$	$\varrho^{L-1}_{\|\|e\|\|_\infty}/\varrho^L_{\|\|e\|\|_\infty}$
4	0.9756352	0.0243648	-	-
5	0.9936601	0.0063400	3.84	3.57
6	0.9983959	0.0016041	3.95	4.30
7	0.9995977	0.0004024	3.99	3.96
8	0.9998993	0.0001007	4.00	3.98
9	0.9999748	0.0000252	4.00	4.00

As expected, this ratio is found to be approaching 4, indicating that the convergence factor of Jacobi relaxation is $1-O(h^2)$, owing to our standard grid-coarsening $h_L/h_{L+1} = 2$. (Recall that optimal-weighted Jacobi relaxation does *not* have a convergence factor of $1-O(h)$.) The corresponding ratio for the infinity-norm of the error is also given in the above table. The convergence factor for Gauss-Seidel relaxation is found to have the same characteristics as for Jacobi relaxation.

The following table gives similar figures to those above for optimal-weighted (ascending) Gauss-Seidel relaxation. Recall from Section 3.4 that the optimal weighting for Gauss-Seidel relaxation on Poisson's equation discretised by the five-point star is $\omega_{\text{opt}} = 2/(1+\pi h)$.

L	ω_{opt}	$\varrho^L_{\|\|e\|\|_2}$	$\varrho^{L-1}_{\|\|e\|\|_2}/\varrho^L_{\|\|e\|\|_2}$	$\varrho^L_{\|\|e\|\|_\infty}$	$\varrho^{L-1}_{\|\|e\|\|_\infty}/\varrho^L_{\|\|e\|\|_\infty}$
4	1.67175	0.1268433	-	0.1331408	-
5	1.82120	0.0673781	1.88	0.0706676	1.88
6	1.90642	0.0351858	1.91	0.0365719	1.93
7	1.95209	0.0180439	1.95	0.0186633	1.96
8	1.97575	0.0091428	1.97	0.0094359	1.98
9	1.98780	0.0046025	1.99	0.0047435	1.99

As expected, this ratio is found to be approaching 2, indicating that the convergence factor of optimal SOR Gauss-Seidel relaxation is indeed $1-O(h)$.

Next we shall confirm that the theoretical smoothing rates are achieved in practice. For this purpose, we have created Problem MG0 as follows:

Problem MG0 (Laplace's equation)

$$\Delta u = 0 \qquad \text{in } \Omega$$

$$u = 0 \qquad \text{on } \partial\Omega$$

$$v^0 = \sin(k\pi x)\sin(k\pi y) \qquad k \in \mathbb{Z}$$

$$u = 0$$

Notice that the initial guess is a single Fourier k-mode ($k = 0, 1, \ldots, N$) in each dimension, and that we will have $e \equiv r$ and $\|\cdot\|_2 \equiv \|\cdot\|_\infty$. Figure 4.4 (on page 60) shows the measured convergence factor $\hat{\rho}$ of various ω-weighted Jacobi relaxations for each initial guess. Recall from Section 3.4 that the convergence factor for weighted Jacobi relaxation on the two-dimensional Poisson equation discretised by the five-point star is

$$\rho = \left| \frac{\psi_\theta^{k+1}}{\psi_\theta^k} \right| = \left| 1 + \tfrac{1}{2}\omega(\cos\theta_1 + \cos\theta_2 - 2) \right|$$

As expected, each set of measured points follows the graph of

$$\rho = \left| 1 + \omega(\cos k\pi h - 1) \right|.$$

In other words, we have verified the convergence factor for the points along the line $\theta_1 = \theta_2$ in the Fourier space diagram of Figure 3.4. To verify all points (θ_1, θ_2), we would require initial guesses of

$$v^0 = \sin(j\pi x)\sin(k\pi y) \qquad \text{for} \quad j = 0, 1, \ldots, M; \quad k = 0, 1, \ldots, N.$$

Figure 4.5 shows the convergence of weighted Jacobi relaxation for various ω. We see that, as expected, the optimal smoothing weight $\omega_{\text{opt}} = 0.8$ is not the optimal convergence weight. The eigenvalues of the weighted Jacobi iteration matrix for the 2-D Poisson equation are $\lambda(\mathcal{M}) = 1 + \frac{1}{2}\omega(\cos\theta_1 + \cos\theta_2 - 2)$, hence the eigenvalue corresponding to the highest frequency mode is $\lambda \approx 1 - 2\omega$. Thus after sufficiently many relaxations with $\omega > 1$, we see the appearance of divergent high-frequency modes.

Figure 4.6 shows the convergence of 0.8-weighted Jacobi relaxation on grids at various levels L, each profile being scaled to the same work units. It is clear that relaxation can be very inefficient on fine grids; this is because non-oscillatory functions appear very smooth on fine grids.

Considering these results, we have validated our relaxation algorithms to our satisfaction. Let us now turn to results of the multigrid iteration itself.

Figure 4.7 shows the convergence of $1 \times V_9^{2,2,2}$-cycle for Problem MG10. We use immediate correction (see Section 3.5) to observe the convergence behaviour inside the cycle; we call this the *micro-structure*. The abscissa has units of WU, hence the points indicate the relative amount of (relaxation) computation, which initially is large (due to fine-grid relaxation), becomes very small (working on the coarsest grid) and becomes large again. Concentrating on the $\|e\|_\cdot$ profile, we see the importance of coarse-grid correction: convergence is poor until we begin the second half of the V-cycle.

Figure 4.8 is the analogue of Figure 4.6 for V-cycles for various Λ. We have performed consecutive V-cycles so that each trial solution has approximately the same cost in work units. As before, we see that the efficiency of iteration is high on coarse grids, hence these are not only crucial for multigrid, but are computationally cheap. We also observe that the profiles for $\Lambda = 2, 4, 6$ reach a limiting value of $\|e\|_2$: this is the discretisation error.

Figure 4.9 clarifies the position with respect to discretisation error. We have plotted the limiting value of the error infinity-norm after many V_Λ-cycles for successive Λ, and for each model problem (using the standard data). We find that each profile has a slope of

$$m = -0.6025 \pm 0.0008$$

and hence $10^m = 0.2497 \pm 0.0005 \simeq \frac{1}{4}$. Since we have used standard coarsening $h_{L+1}/h_L = \frac{1}{2}$, our experiment confirms that the discretisation error is $O(h^2)$.

Figure 4.10 shows the convergence of $21 \times V_\Lambda^{2,2,2}$-cycles for various Λ. It is immediately apparent that V-cycle convergence is independent of h, one of the startling features of multigrid iteration. Eventually the profiles reach a limiting value of residue norm, due to round-off error.

Figure 4.11 shows the convergence of $10 \times V_\Lambda^{2,2,2}$-cycles for various multi-grid hierarchies. These are as follows:

Λ	$M_1 \times N_1$	$M_2 \times N_2$	$M_3 \times N_3$	$M_4 \times N_4$	$M_5 \times N_5$	$M_6 \times N_6$	$M_7 \times N_7$
3	33×33	65×65	129×129	-	-	-	-
4	17×17	33×33	65×65	129×129	-	-	-
5	9×9	17×17	33×33	65×65	129×129	-	-
6	5×5	9×9	17×17	33×33	65×65	129×129	-
7	3×3	5×5	9×9	17×17	33×33	65×65	129×129

These are chosen so that each hierarchy results in a 129×129 fine grid. The graph shows that the fastest convergence is attained using a hierarchy with the coarsest possible grid. This is because a 3×3 grid permits the lowest frequency error modes to be reduced with relaxation.

It is interesting to note that increasing ν for the $\Lambda = 3$ hierarchy does not improve convergence. The following experiment demonstrates this; we have increased ν from two to five, and we have reduced the number of V-cycles to compensate for the increased relaxation cost.

Method	$\|e\|_\infty$	$\|r\|_\infty$	WU
$10 \times V_3^{2,2,2}$-cycle	0.0509124	2.0306385	51.25
$4 \times V_3^{5,5,5}$-cycle	0.0509118	2.0278172	51.25

There is very little computational gain in increasing ν for the $\Lambda = 3$ hierarchy; the coarsest possible grids are required for good convergence.

Figure 4.12 shows the convergence of $10 \times V_8^{\nu_1,\nu_0,\nu_2}$-cycles for various combinations of (ν_1, ν_0, ν_2). We see that larger ν gives greater accuracy, but naturally at a certain computational cost. An exception to this is the case of $(1, 2, 2)$ which costs the same as $(2, 2, 1)$, yet gives a significantly better solution. The ν combination with the best performance is a matter of judgement, however $(2, 2, 2)$ seems a reasonable choice.

Lastly, let us consider results for the full multigrid M-cycle. Figure 4.13 shows the micro-structure of one M-cycle for Problem MG10. The graph reflects the fact that the M-cycle is composed of successively larger V-cycles, resulting in an increasingly accurate solution. Looking at the error norm profile, we see that (after the first few V-cycles) the accuracy is consistently improved by more than half a decimal digit for each successive V-cycle. We can see this more clearly from the following table of figures. Naturally, each succeeding V-cycle is computationally more expensive than the previous one (by a factor of approximately four), hence convergence will appear to slow with respect to computation time.

L	$\log_{10}\|e\|_2$	$\hat{\rho}(\|e\|_2)$	$\log_{10}\|r\|_2$	$\hat{\rho}(\|r\|_2)$
1	−1.76	-	−0.73	-
2	−2.49	0.186	−1.39	0.218
3	−2.85	0.429	−1.31	1.212
4	−3.30	0.357	−1.41	0.779
5	−3.80	0.316	−1.56	0.715
6	−4.33	0.293	−1.71	0.708
7	−4.89	0.279	−1.86	0.709
8	−5.46	0.270	−2.01	0.711
9	−6.03	0.265	−2.15	0.715

Figure 4.14 shows the convergence of $1 \times M_8^{2,2,2}$-cycle for each of the model linear problems, using the discrete-Laplacian initial guess described in Section 3.6. The profiles have very similar characteristics; however less accurate solutions were obtained for the two anisotropic problems, as would be expected.

A close inspection of the graph will show that the MG12 profile does not have a value plotted for zero work units. This is because the initial guess on the coarsest grid was fortuitously identical to the exact solution.

Figure 4.15 compares the convergence of V-cycles for single- and double-precision arithmetic. We see that each norm approaches some limiting value (for the residue norms in double precision, this value is about 10^{-12}) which is determined by a combination of discretisation and round-off errors. For residue norms in this case, this value is roughly four or five orders of magnitude greater than arithmetic precision; while there is relatively little difference in the limiting value of the error norms. We observe that virtually no discrepancy appears between the single- and double-precision results until about the fourth V-cycle. This leads us to conjecture that an M-cycle, which does not repeat a V-cycle on the same level, will give results almost independent of precision in certain circumstances. The following table indicates that this is true for a smooth BVP; it shows the result of $1 \times M_9^{2,2,2}$-cycle on Problem MG10 with the usual multigrid parameters for single and double precision.

Let us now return our attention to Figure 4.3, which shows the accuracy of three types of multigrid iteration. The sequence of V-cycles evidently converges to the numerical solution of the discrete problem; that is, the solution to the linear

	$\|e\|_2$	$\|e\|_\infty$	$\|r\|_2$	$\|r\|_\infty$
SP	0.00000092	0.00000206	0.00703635	0.16377530
DP	0.00000088	0.00000191	0.00724267	0.16387057

system $Av = f$ is eventually computed to within round-off error (dictated by arithmetic precision). Further evidence of this is that the residuals do approach the level of round-off error (Figure 4.10). Since the system $Av = f$ is itself an approximation to the continuous problem, discretisation errors (of $O(h^2)$) prevent the solution from approaching an accuracy comparable to round-off error. We notice that neither the W- nor M-cycle attain the numerical solution of the discrete algebraic problem that the $10 \times V$-cycles do; this is because insufficient smoothing was performed in this case. To remedy this, we would increase the number of relaxations ν.

Finally, Figure 4.16 shows the CPU execution times for `MGLAB` on some of the many different serial machines available at the Australian National University. These times are averaged over at least four separate trials, and execution took place during periods of minimal activity. Some details of the hardware and compilation are as follows:

Name	Make and Model	Memory (Mb)	Compilation command
`phys4`	VAX 3100	16	`fortran/nocheck/optimize`
`csc1`	VAX 8700	32	`fortran/nocheck/optimize`
`romeo`	Sun 4/390	32	`f77 -O3` or `f77 -O4`
`huxley`	Sun 4/690 (x4)	32	`f77 -O3` or `f77 -O4`
`vulcan`	Sequent S27 (x8)	16	`fortran -O3`
`gauss`	Apollo DN10000	16	`f77 -O -A cpu,a88k`
`vp`	Fujitsu VP2200	256	`frt -Wv,-p2200 -Ne -Os`

Most of these types of computers are familiar to researchers, except perhaps the Fujitsu VP2200: a vector-processing supercomputer. We will not describe its hardware nor architecture because we are treating it merely as a (very large and fast) scalar machine. Indeed, the vector processor was virtually idle during the execution of `MGLAB` because the statement function `MAP` (see Section 3.7) could not be vectorised by the compiler. The times for the VP2200 were obtained during March 1992 when the cycle time of the machine was 4.0 nanoseconds; very recently this was reduced to 3.2 nanoseconds.

We can see from Figure 4.16 that (even with virtually no vectorisation) the VP2200 performs multigrid very rapidly, while the execution time for standard serial machines can be significant.

We have examined the basic results produced by `MGLAB`, and are satisfied that they are consistent with the multigrid concepts presented in the previous chapters. We have not made an attempt to present exhaustive results for all possible multigrid parameters.

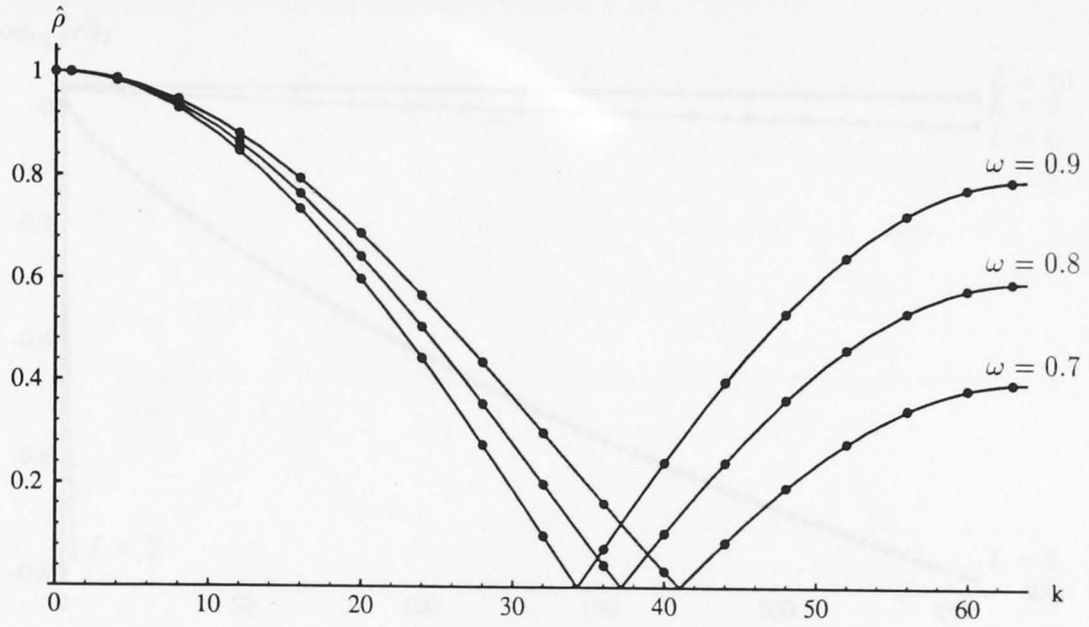


Figure 4.4: Convergence factor of ω-weighted Jacobi point relaxation for various ω on Laplace's equation in 2-D with the initial guesses $v^0 = \sin(k\pi x)\sin(k\pi y)$ (Fourier modes) $k = 0, 1, \ldots, 64$ on a 65×65 grid. The optimal smoothing factor (over *all* high-frequency Fourier modes) $\mu_{\text{opt}} = 0.6$ occurs for $\omega = 0.8$.

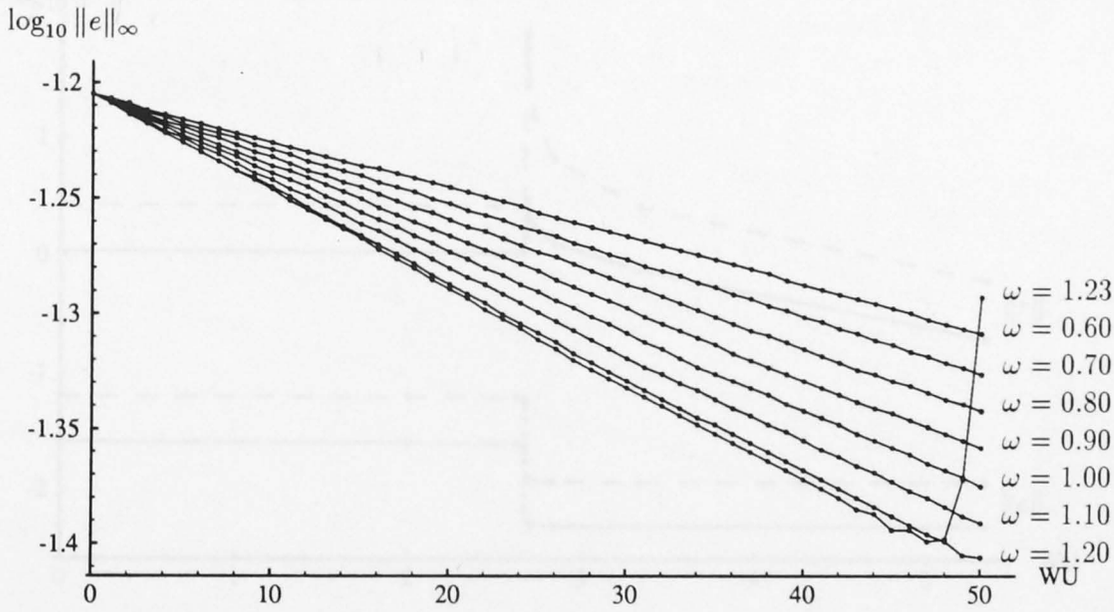


Figure 4.5: Convergence of $50 \times P_5$-cycles using weighted Jacobi point relaxation for various ω. MG10, $v^0 = 0$. Optimal convergence does not imply optimal smoothing.

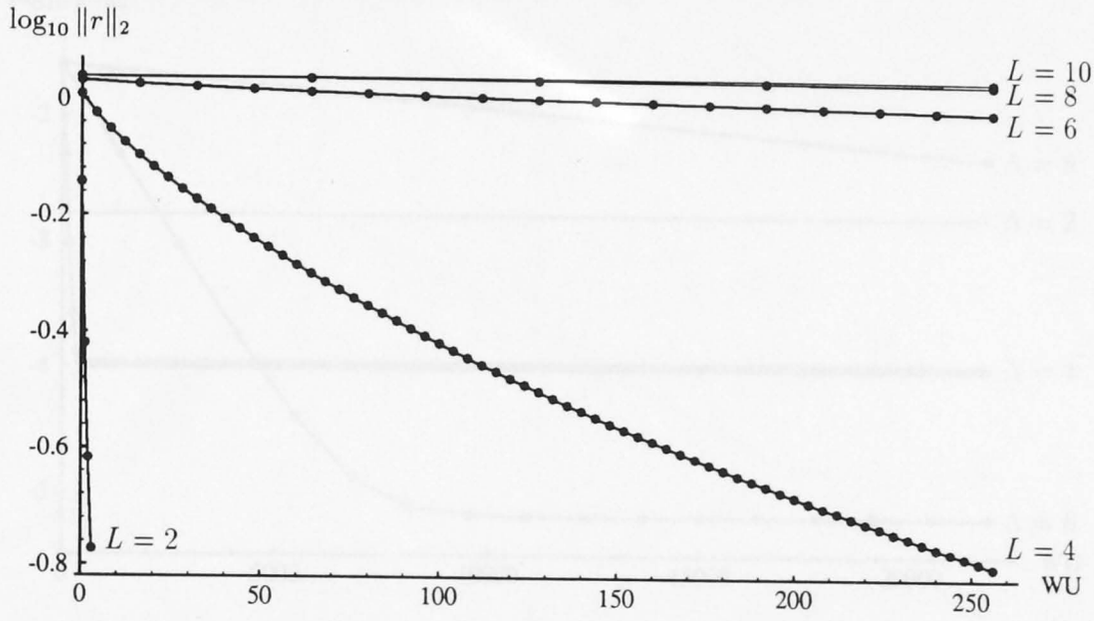


Figure 4.6: Convergence of various P_L-cycles, each performing approximately the same amount of work. MG10, $\omega = 0.8$ weighted Jacobi point relaxation, $v^0 = 0$, work units (WU) for each L are scaled to those for $L = 2$. For smooth v, relaxation on fine grids is very inefficient.

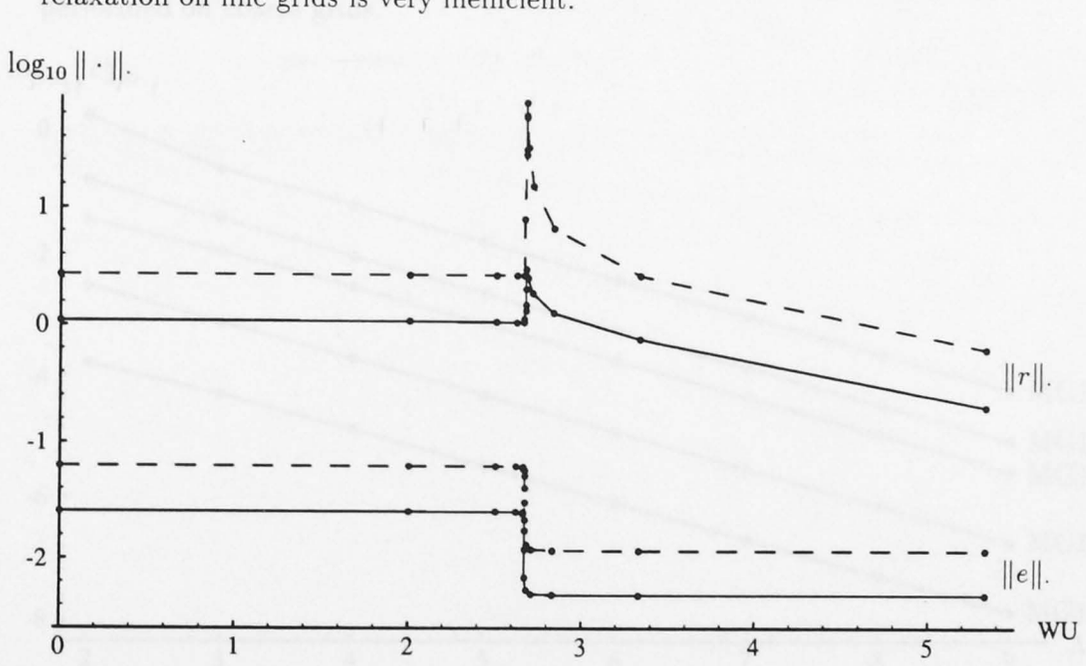


Figure 4.7: Micro-structure of V-cycle convergence as measured by two-norms (solid lines) and infinity-norms (dashed lines). MG10, $1 \times V_8^{2,2,2}$-cycle, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $v^0 = 0$, immediate correction. Coarse-grid correction is essential for rapid convergence.

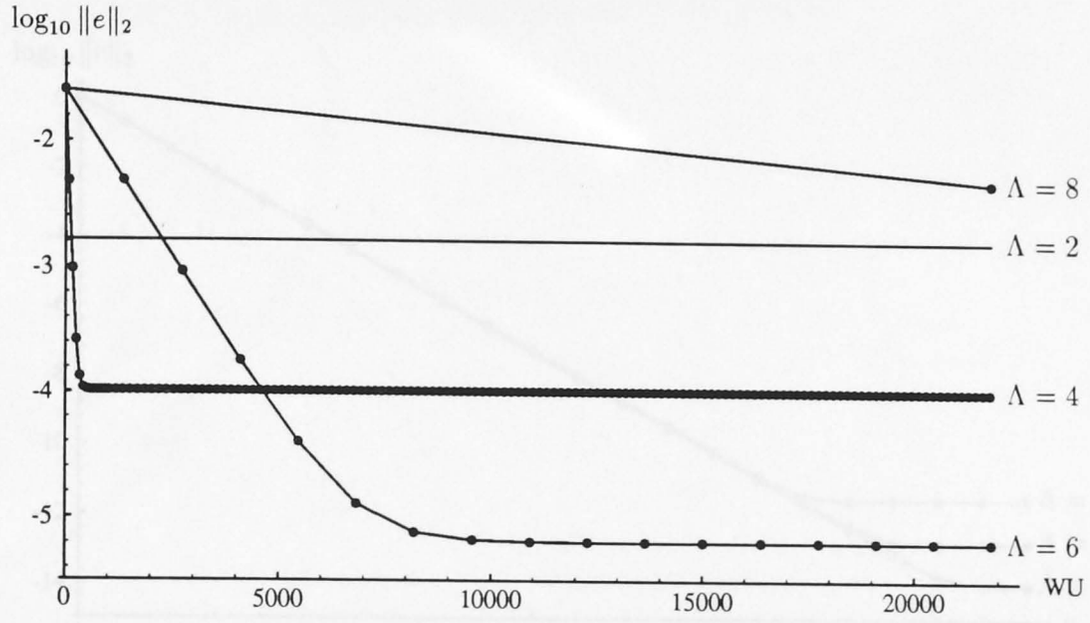


Figure 4.8: Convergence of $n \times V_\Lambda^{2,2,2}$-cycles for various Λ, each performing approximately the same amount of work. MG10, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $v^0 = 0$, work units (WU) for each Λ are scaled to those for $\Lambda = 2$. As much computation as possible should be performed on coarse grids.

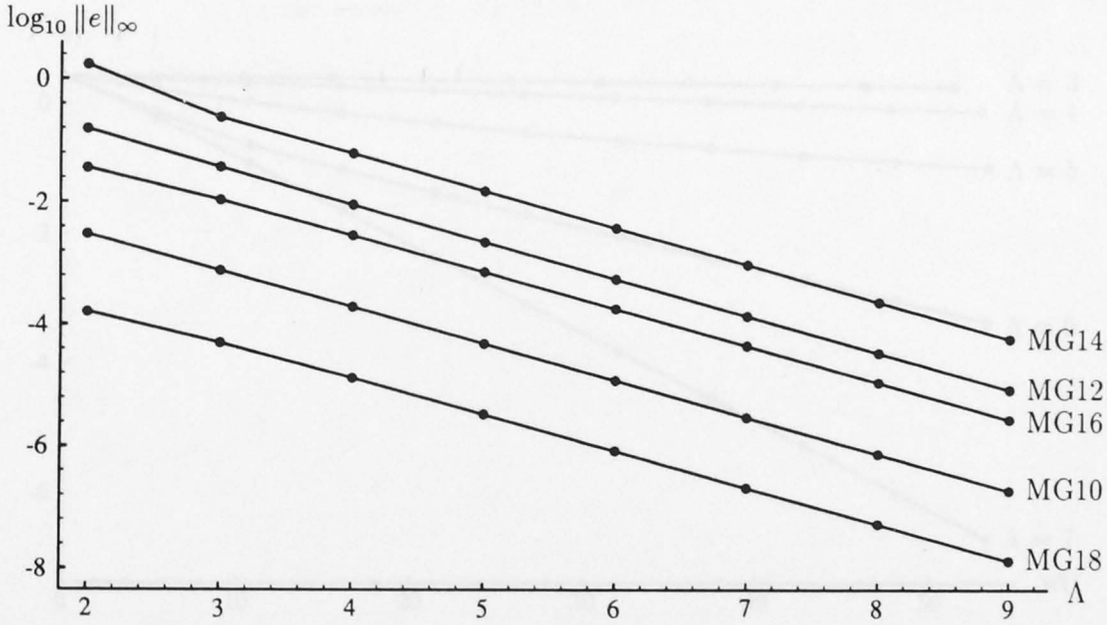


Figure 4.9: Discretisation errors as measured by the error norm of $\lim_{n\to\infty} n \times V_\Lambda^{2,2,2}$-cycles for various Λ. $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction. The discretisation error is $O(h^2)$.

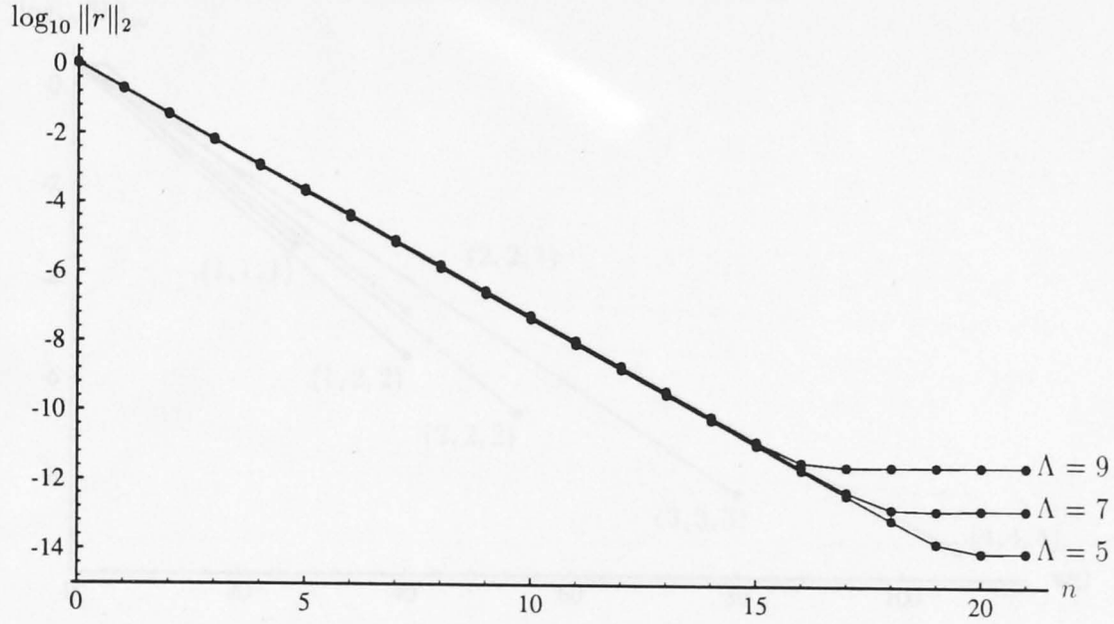


Figure 4.10: Convergence of $21 \times V_\Lambda^{2,2,2}$-cycles for various Λ. MG10, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $v^0 = 0$. Convergence is independent of h.

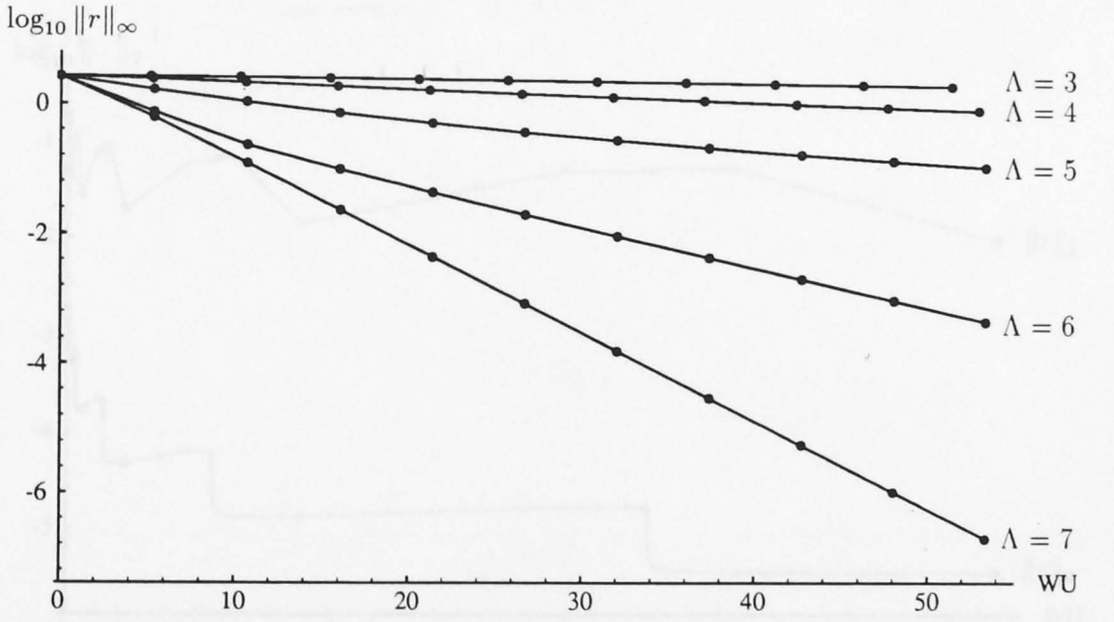


Figure 4.11: Convergence of $10 \times V_\Lambda^{2,2,2}$-cycles for various multi-grid hierarchies, each resulting in identical finest grids of size 129×129. MG10, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $v^0 = 0$. Rapid convergence requires the inclusion of the coarsest possible grids.

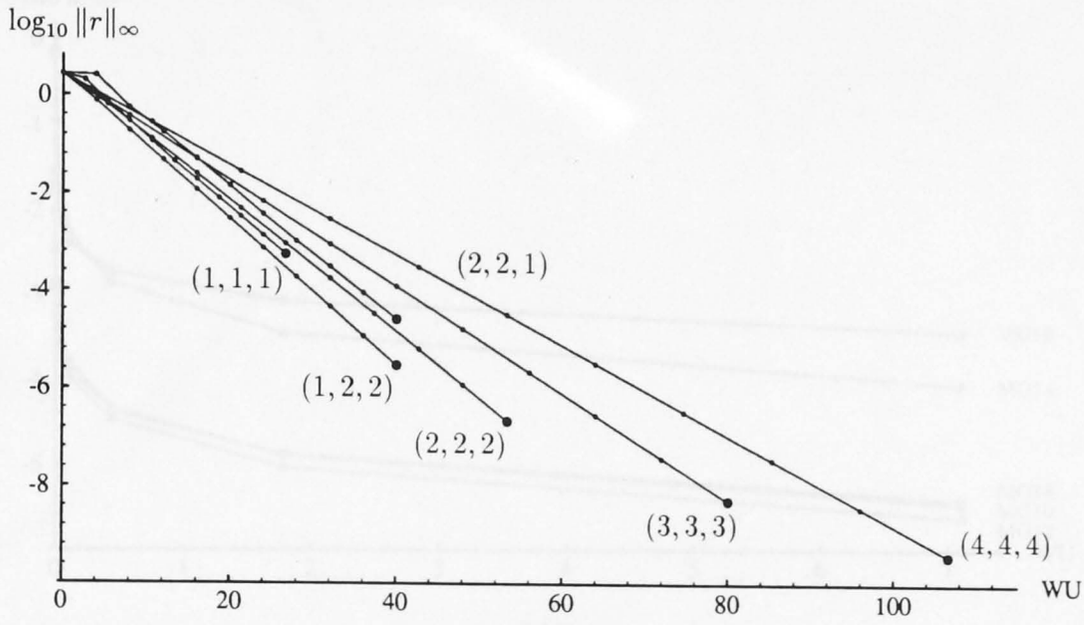


Figure 4.12: Convergence of $10 \times V_8^{\nu_1,\nu_0,\nu_2}$-cycles for various combinations of (ν_1, ν_0, ν_2). MG10, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $v^0 = 0$.

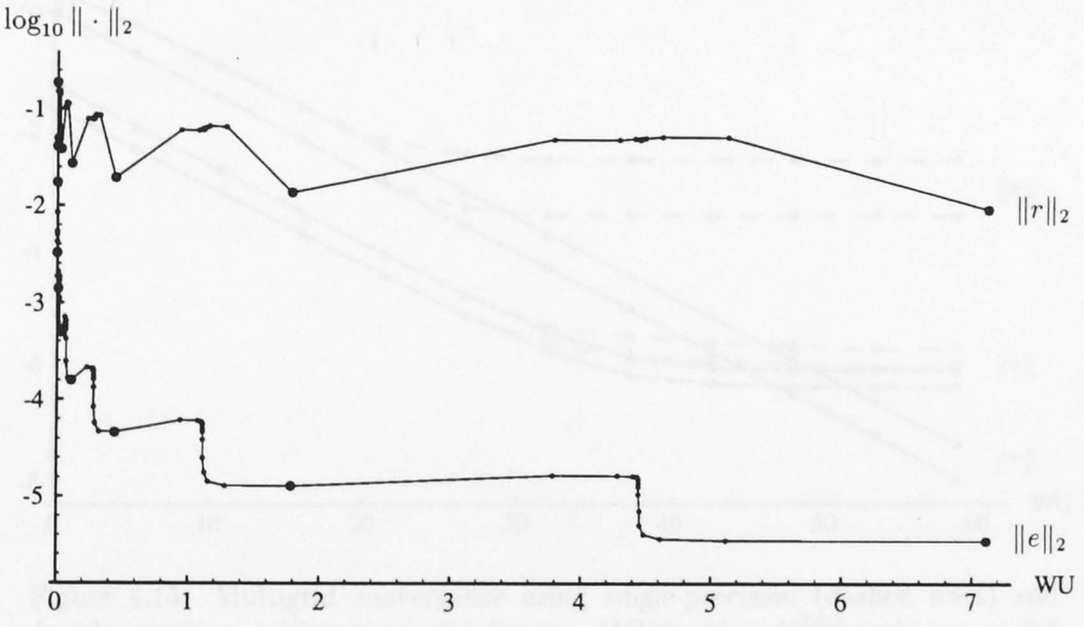


Figure 4.13: Micro-structure of M-cycle convergence. Larger circles mark the juncture of successive V-cycles. MG10, $1 \times M_8^{2,2,2}$-cycle, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $v^0 = 0$, immediate correction.

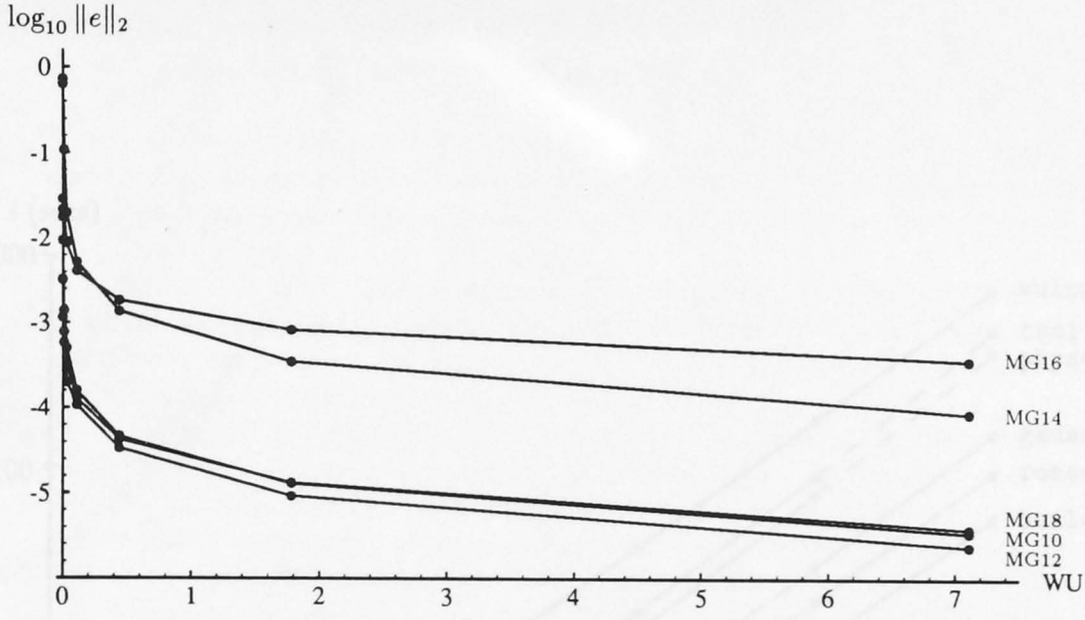


Figure 4.14: Convergence of $1 \times M_8^{2,2,2}$-cycle for the model linear problems. $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, discrete-Laplacian initial guess, immediate correction. Multigrid iterations for each model problem have similar characteristics.

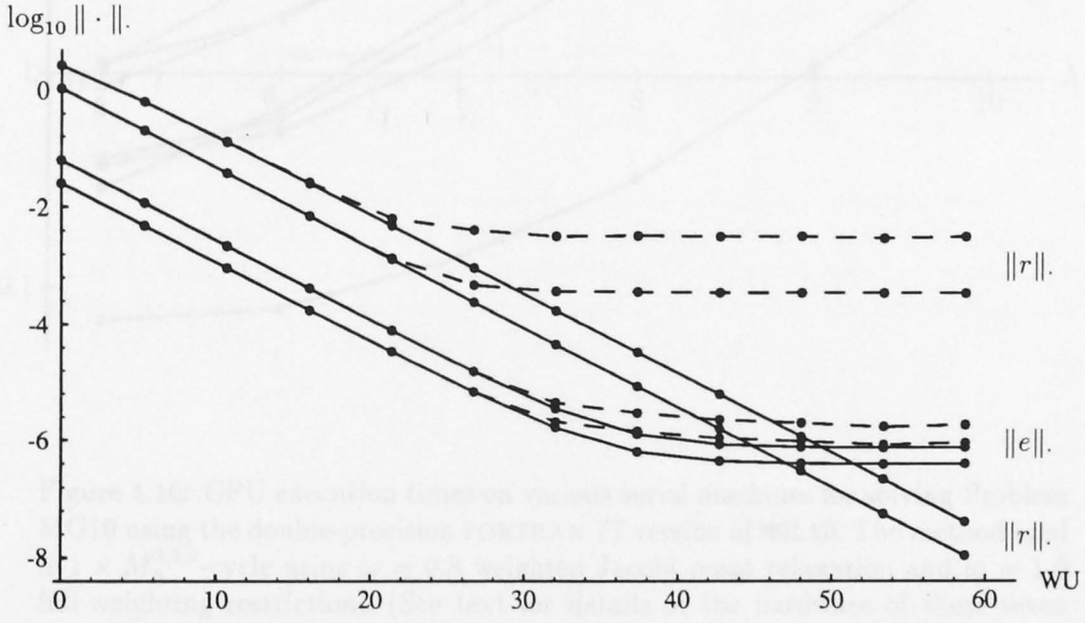


Figure 4.15: Multigrid convergence using single-precision (dashed lines) and double-precision arithmetic (solid lines). MG10, $11 \times V_8^{2,2,2}$-cycle, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $v^0 = 0$, immediate correction. Accuracy of solutions is almost independent of arithmetic precision.

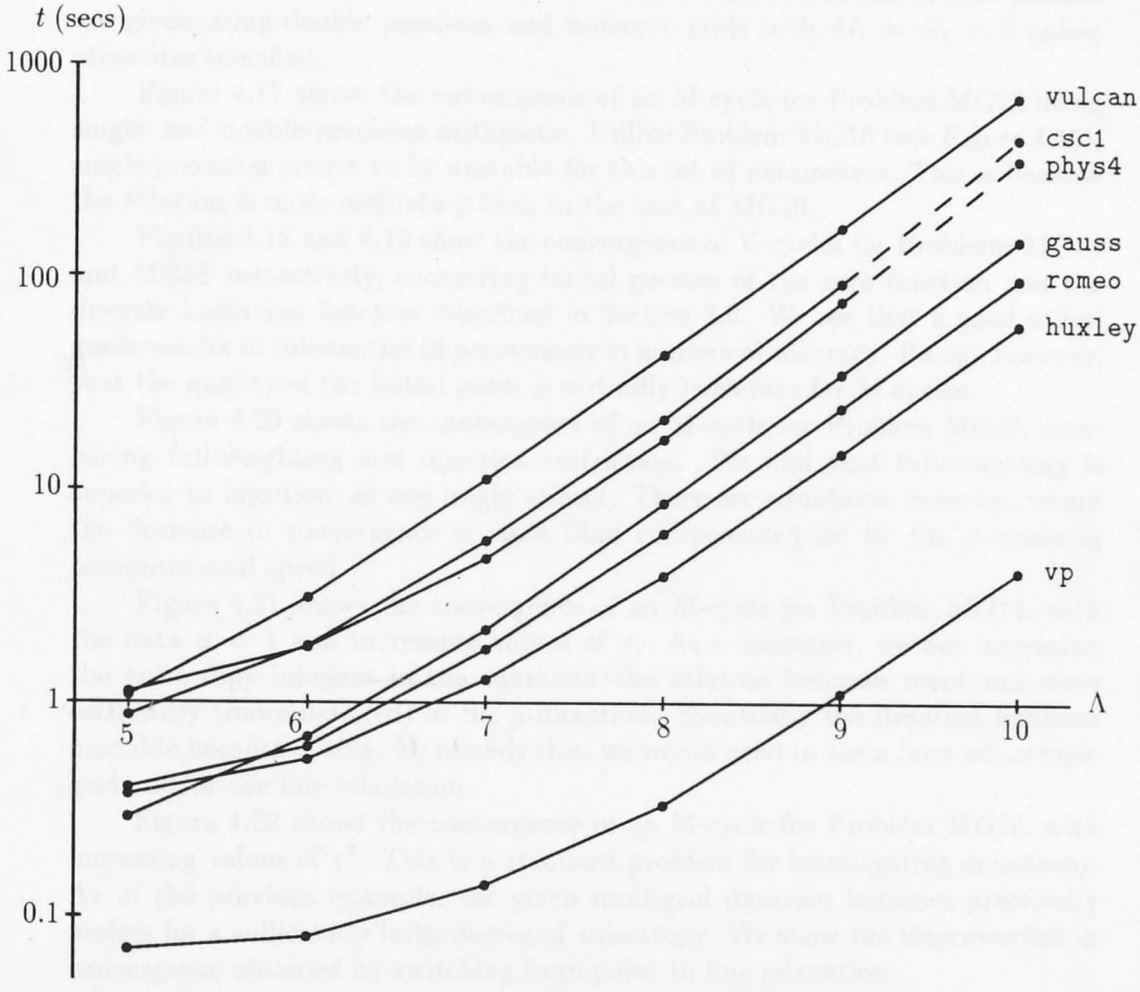


Figure 4.16: CPU execution times on various serial machines for solving Problem MG10 using the double-precision FORTRAN 77 version of MGLAB. The method used is $1 \times M_{\Lambda}^{2,2,2}$-cycle using $\omega = 0.8$ weighted Jacobi point relaxation and $\varpi = 1.0$ full-weighting restriction. (See text for details of the hardware of these seven machines.) Each dashed line is an estimated result.

4.3 Model Problem Results

In this section, we will accept our multigrid implementation as valid and concentrate on numerical results for Problems MG12, MG14, MG16 and MG18. Results are given using double precision and isotropic grids with $M_1 = N_1 = 2$ unless otherwise specified.

Figure 4.17 shows the convergence of an M-cycle for Problem MG12 using single- and double-precision arithmetic. Unlike Problem MG10 (see Figure 4.15), single precision proves to be unstable for this set of parameters. This is because the solution is more oscillatory than in the case of MG10.

Figures 4.18 and 4.19 show the convergence of V-cycles for Problems MG16 and MG18 respectively, comparing initial guesses of the zero function and the discrete-Laplacian function described in Section 3.6. We see that a good initial guess results in substantial improvements in numerical accuracy. Recall, however, that the quality of the initial guess is virtually irrelevant for M-cycles.

Figure 4.20 shows the convergence of an M-cycle for Problem MG18, comparing full-weighting and injection restriction. We find that full-weighting is superior to injection, as one might expect. There are situations, however, where the decrease in convergence is more than compensated for by the increase in computational speed.

Figure 4.21 shows the convergence of an M-cycle for Problem MG14, with the data $\sigma = 1$ and increasing values of τ. As τ increases, we are increasing the anisotropy inherent in the equation: the solution becomes more and more oscillatory (more detailed) in the y-direction. Eventually the iteration becomes unstable because of this. To remedy this, we would need to use a finer anisotropic grid and/or use line relaxation.

Figure 4.22 shows the convergence of an M-cycle for Problem MG16, with increasing values of ε^2. This is a standard problem for investigating anisotropy. As in the previous example, the given multigrid iteration becomes practically useless for a sufficiently large degree of anisotropy. We show the improvement in convergence obtained by switching from point to line relaxation.

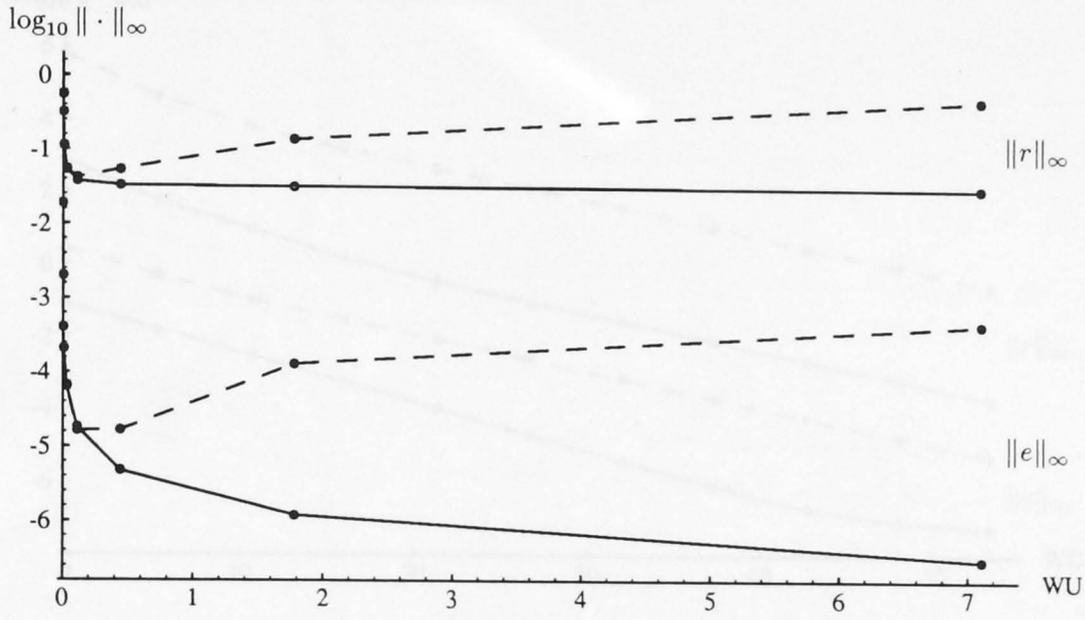


Figure 4.17: Convergence of $1 \times M_{10}^{2,2,2}$-cycle using single-precision (dashed lines) and double-precision arithmetic (solid lines). MG12, standard data, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, $v^0 = 0$. Single precision is unstable in this case.

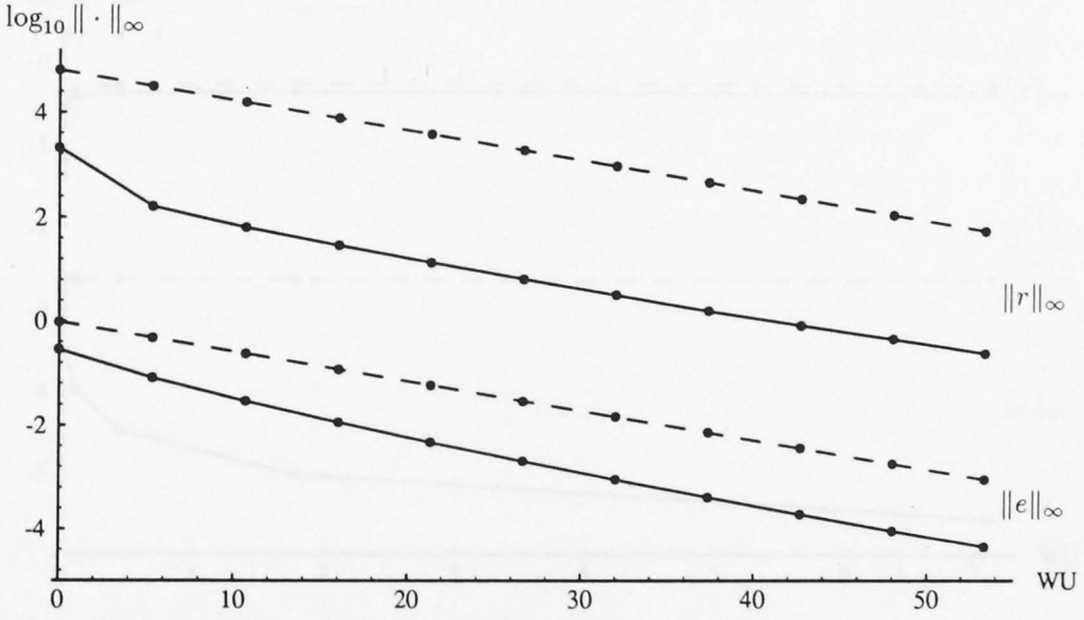


Figure 4.18: Convergence of $10 \times V_8^{2,2,2}$-cycles for initial guesses of $v^0 = 0$ (dashed lines) and the discrete-Laplacian function (solid lines). MG16, standard data, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction. A smooth initial guess has significant advantages.

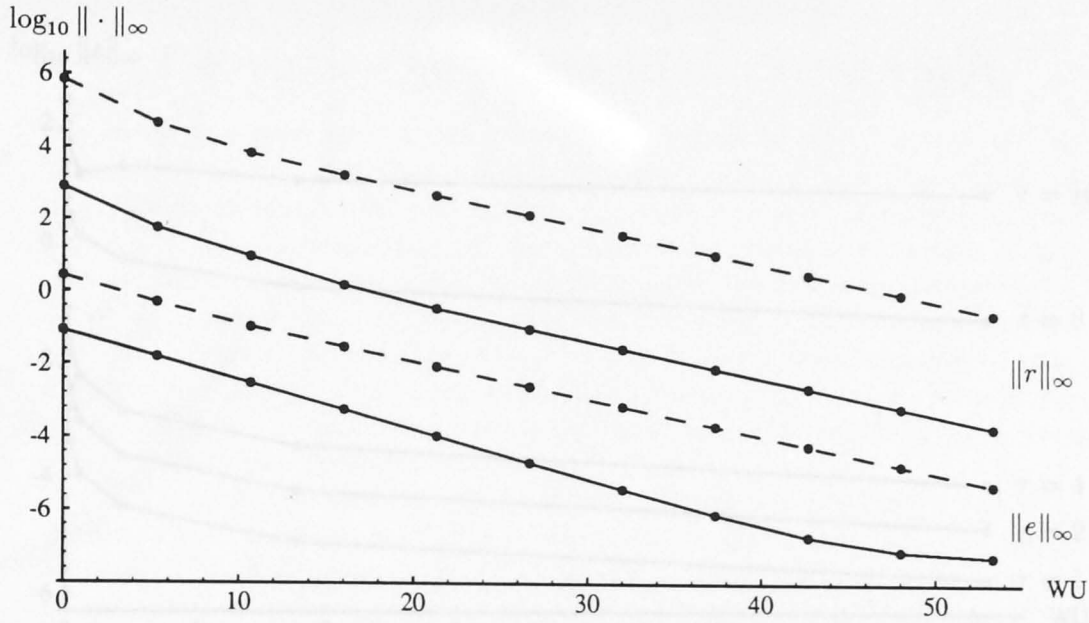


Figure 4.19: Convergence of $10 \times V_8^{2,2,2}$-cycles for initial guesses of $v^0 = 0$ (dashed lines) and the discrete-Laplacian function (solid lines). MG18, standard data, $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction. A smooth initial guess has significant advantages.

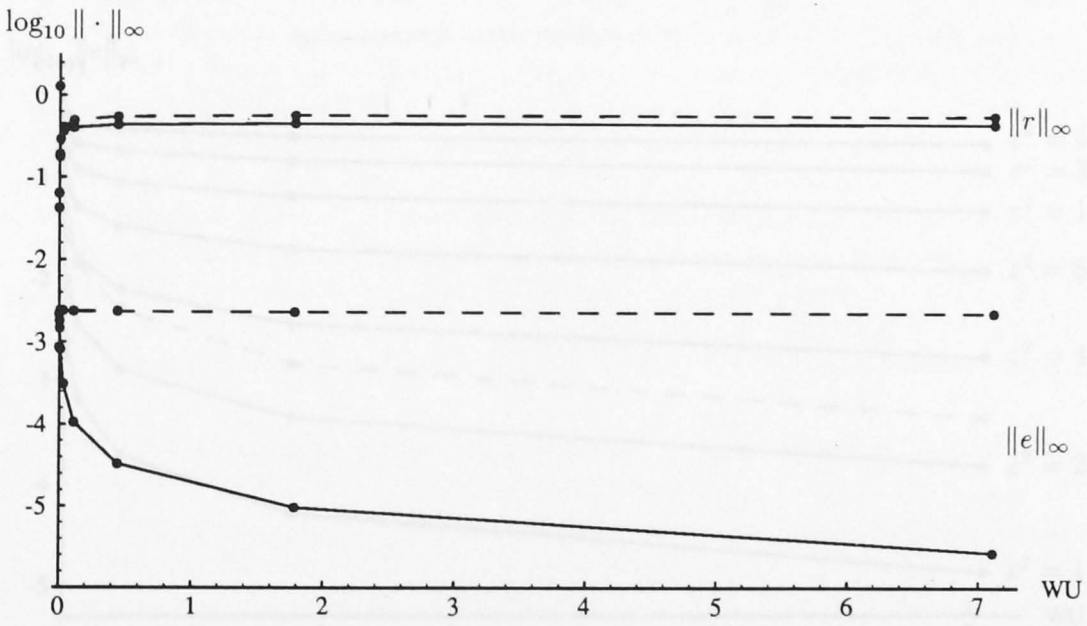


Figure 4.20: Convergence of $1 \times M_9^{2,2,2}$-cycle for Problem MG18, comparing injection restriction (dashed lines) and $\varpi = 1.0$ full-weighting (solid lines). $\omega = 0.8$ weighted Jacobi point relaxation, $v^0 = 0$. Full-weighting is generally superior to injection.

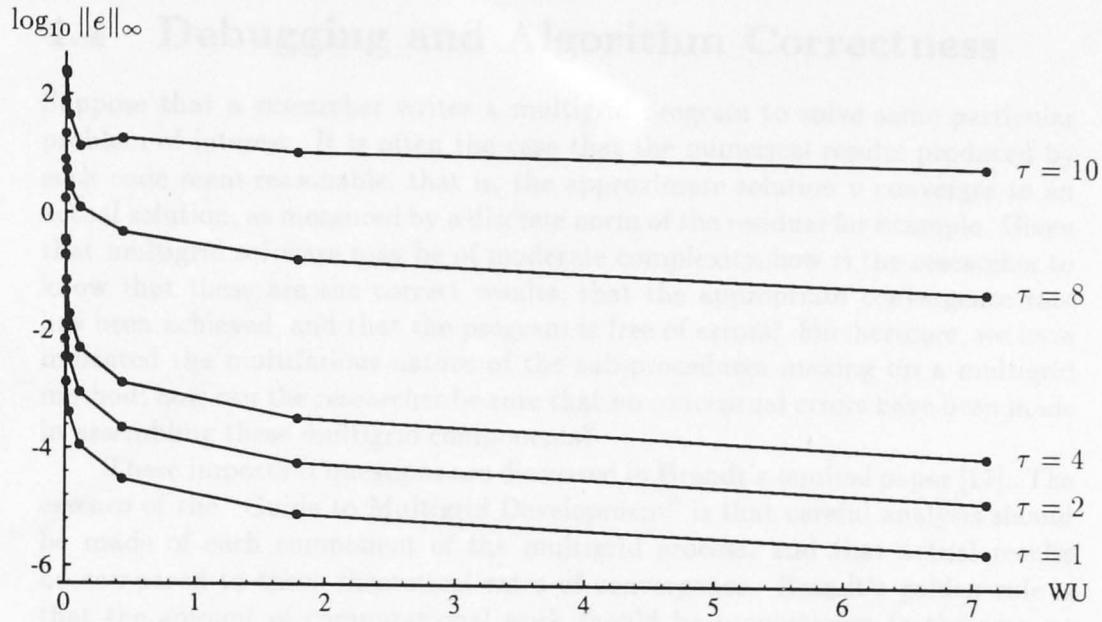


Figure 4.21: Convergence of $1 \times M_9^{2,2,2}$-cycle for Problem MG14 with $\sigma = 1$ and various values of τ. $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction. The iteration becomes more unstable as the degree of anisotropy increases.

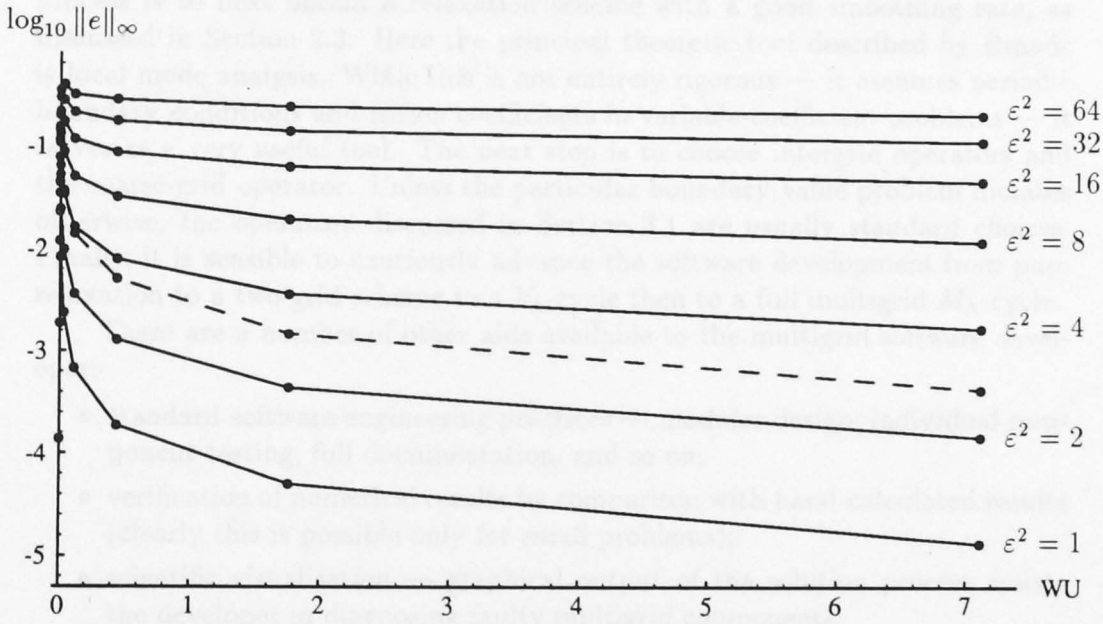


Figure 4.22: Convergence of $1 \times M_8^{2,2,2}$-cycle for Problem MG16 for various values of ε^2. $\omega = 0.8$ weighted Jacobi point relaxation, $\varpi = 1.0$ full-weighting restriction, user-defined initial guess. Also $\omega = 0.6$ weighted Jacobi column relaxation for $\varepsilon^2 = 64$ is shown (dashed line).

4.4 Debugging and Algorithm Correctness

Suppose that a researcher writes a multigrid program to solve some particular problem of interest. It is often the case that the numerical results produced by such code seem reasonable, that is, the approximate solution v converges to an actual solution, as measured by a discrete norm of the residual for example. Given that multigrid software may be of moderate complexity, how is the researcher to know that these are the correct results, that the appropriate convergence rate has been achieved, and that the program is free of errors? Furthermore, we have indicated the multifarious nature of the sub-procedures making up a multigrid method; how can the researcher be sure that no conceptual errors have been made in assembling these multigrid components?

These important questions are discussed in Brandt's seminal paper [17]. The essence of the "Guide to Multigrid Development" is that careful analysis should be made of each component of the multigrid process, and that actual results be compared to these theoretical rates of convergence. Brandt's golden rule is that the amount of computational work should be proportional to the amount of physical change in the computed system: that stalling numerical processes indicate there is a better way to achieve the same goal.

The process of writing a multigrid algorithm should begin with a stable discretisation scheme. For regular elliptic problems, simple differencing is stable in every respect [17]. Possibly the most crucial step in constructing a multigrid process is to next obtain a relaxation scheme with a good smoothing rate, as discussed in Section 2.3. Here the principal theoretic tool described by Brandt is local mode analysis. While this is not entirely rigorous — it assumes periodic boundary conditions and frozen coefficients in variable-coefficient problems — it serves as a very useful tool. The next step is to choose intergrid operators and the coarse-grid operator. Unless the particular boundary value problem dictates otherwise, the operators discussed in Section 3.1 are usually standard choices. Finally, it is sensible to cautiously advance the software development from pure relaxation to a two-grid scheme to a V_Λ-cycle then to a full multigrid M_Λ-cycle.

There are a number of other aids available to the multigrid software developer:

- standard software engineering practices — modular design, individual component testing, full documentation, and so on;
- verification of numerical results by comparison with hand-calculated results (clearly this is possible only for small problems);
- scientific visualisation — graphical output of the solution process assists the developer in diagnosing faulty multigrid components;
- the immediate-correction process — an excellent tool for diagnosing software faults: the point of failure can traced to a particular multigrid element;
- reference to and comparison with published software and results; and
- cross-checking of results from dual independent platforms — in our case, the CM Fortran and standard FORTRAN 77 codes for `MGLAB` are quite different.

4.5 Comparison with Published Results

A desirable method of verifying software is to compare its results for some given problem against those produced by publicly available software, or published results. The difficulty with attempting to directly compare multigrid results this way is that usually not all of the multigrid parameters are fully specified in a paper, or that the reference software package does not have exactly the same choice of parameters.

Nevertheless, we have at our disposal two simple multigrid programs which have been explicitly published: `MGOOD` by Foerster and Witsch (listed in the appendix of [93] and described in [40]), and `FMV` by Briggs and McCormick [77]. Both programs are designed to be illustrative of the multigrid process, and hence are elementary. Each is designed to solve Poisson's equation on the unit square with Dirichlet boundary conditions, corresponding to our Problem MG10. They both use the usual five-point discretisation, together with red-black Gauss-Seidel relaxation, half-injection and bilinear interpolation. `FMV` performs $1 \times M_\Lambda^{2,1,1}$-cycle, while `MGOOD` is a little more flexible, allowing $n \times V_\Lambda^{\nu_1,1,\nu_2}$- W- and M-cycles. It was found that `MGLAB` gave the same results as these two programs, and hence our confidence in the correctness of `MGLAB` is increased.

Turning now to published multigrid results, we have found two sources which seem to permit direct comparison of results, although there is often some uncertainty with some multigrid parameters, as we have mentioned. The first set of results appear in Briggs [20], where he solves Problem MG10 using red-black Gauss-Seidel relaxation with (a) $6 \times V_\Lambda^{1,1,1}$-cycles ($\Lambda = 4, 5, 6$), (b) $1 \times M_6^{1,1,1}$-cycle, and (c) $1 \times M_6^{2,1,1}$-cycle. Again, we have reproduced the results with `MGLAB`.

Next, we shall attempt to compare our multigrid results with those of Adams [1] who has published results for the package `MUDPACK`; his Example 4 is identical to our Problem MG16, with the notation $\epsilon = 1/\varepsilon^2$. This is the model anisotropic diffusion equation, in which a boundary layer forms along $x = 0$ as $\varepsilon \to \infty$. Adams states that he used $1 \times M_7^{2,1}$-cycle with line y relaxation to solve this problem. We have assumed that the following parameters were used:

- $M_1 = N_1 = 2$,
- standard five-point discretisation: $\beta_0 = 2(\varepsilon^2+1)$, $\beta_1 = \beta_2 = 1$, $\beta_3 = \beta_4 = \varepsilon^2$,
- ω-weighted Jacobi column-wise relaxation,
- ϖ-full-weighting restriction,
- standard discrete two-norms,
- an initial guess of $v^0 = 0$, and
- double precision calculations.

In addition, we set $\nu_0 = 5$ as an approximation to solving directly on the coarsest grid, and we found the optimal values of ω and ϖ by numerical experiment. The results of `MUDPACK` and `MGLAB` are presented in Table 4.1. Given the number of uncertain parameters, the comparison seems favourable.

ε^2	ω_{opt}	ϖ_{opt}	$L=5$	$L=6$	$L=7$
1	0.69	1.15	−3.93	−4.52	−5.12
			−3.94	−4.54	−5.14
10	0.64	1.13	−3.36	−4.28	−5.05
			−3.5	−4.2	−4.8
100	0.64	1.20	−2.53	−3.28	−4.21
			−3.8*	−3.5	−4.1
1000	0.76	1.54	−2.25	−2.58	−3.31
			−2.4	−2.7	−3.3
10000	1.35	10.5	−2.15	−2.50	−2.74
			−2.96*	−2.72	−2.74

Table 4.1: Comparison of results of $\log_{10}\|e\|_2$ obtained by MGLAB (first row of figures in each block) and MUDPACK (second row of figures) in solving Problem MG16 for various values of ε. The MGLAB method used is $1 \times M_L^{2,5,1}$-cycle with column-wise ω_{opt} weighted Jacobi line relaxation, ϖ_{opt} full-weighting restriction and zero initial guess.

A further source of discrepancy may be that Adams actually used the measure $\|v-u_0\|_2$, where u_0 is the injection of the limiting value of numerous $V_{L-1}^{2,1}$-cycles, since the exact solution u was not available to him. Also note that the entries in Table 4.1 marked with an asterisk seem questionable, as the norm appears to *increase* as L increases (as we proceed deeper into the M-cycle).

Note that a simple multigrid approach to solving this anisotropic problem ($\varepsilon \gg 1$) is not particularly efficient. Such specialised problems require more refined techniques; see for example [51, 53, 61].

Chapter 5

Nonlinear Multigrid

> *The assumption of linearity underlies, as a fundamental postulate, a considerable domain of mathematics. Therefore the mathematical tools available to the natural scientist are essentially linear. However Nature, with scant regard for the desires of the mathematician, is essentially nonlinear ...*
>
> — W. F. Ames [4]

In this chapter, we discuss multigrid methods for solving nonlinear partial differential equations. We will concentrate on the full approximation scheme (FAS) proposed by Brandt [15] in 1977. We will find that the FAS cycle requires only minor modifications to the linear multigrid algorithm already presented.

5.1 Full Approximation Scheme

The most interesting boundary value problems are often nonlinear in nature. Examples of nonlinear physical problems include chaotic and turbulent systems, which exhibit complex behaviour, not to mention more classical problems such as elasticity, diffusion and vibration. In order to simplify the discussion, so far we have dealt only with linear equations, however multigrid is also ideally suited to solving nonlinear problems.

Our notation is unchanged; we require the solution u to the nonlinear boundary value problem

$$\begin{aligned} \mathcal{A}u &= f \quad \text{in} \quad \Omega = [0,1] \times [0,1] \\ u &= g \quad \text{on} \quad \partial\Omega \end{aligned} \tag{5.1}$$

where $\mathcal{A}$ is a nonlinear elliptic operator. We will sometimes write $\mathcal{A}u = f$ in the form $\mathcal{F} \equiv \mathcal{A}u - f = 0$. This problem is well-posed if and only if the Jacobian of $\mathcal{F}$, $J = \partial\mathcal{F}/\partial v$, evaluated at u is non-singular [51].

One approach to solving problem (5.1) is to combine a linearisation process (such as Newton's method) with the linear multigrid iteration, which transforms the nonlinear problem into a sequence of linear ones. While this indirect method requires a relatively simple addition to the linear multigrid process, it is somewhat

restricted in its application, so we shall instead develop an intrinsically nonlinear multigrid iteration.

Before proceeding, we define F_i, the *components* of the operator $\mathcal{F}$. Suppose $\mathcal{F} : \mathbf{R}^m \to \mathbf{R}^n$,

$$\mathcal{F}(x_1, x_2, \ldots, x_m) = (y_1, y_2, \ldots, y_n)$$

say, then we can split this action of $\mathcal{F}$ into its components

$$F_i(x_1, x_2, \ldots, x_m) = y_i \qquad \text{for} \quad i = 1, 2, \ldots, n.$$

The first element of the multigrid scheme, the relaxation process, is again required to have suitable smoothing properties. Linear relaxation methods usually have several analogues for nonlinear problems (see for example Ortega and Rheinboldt [84]). Recall that linear 2-D Gauss-Seidel relaxation is given by (*cf.* equation 2.16)

$$v_\kappa^{k+1} = \frac{1}{A_{\kappa\kappa}} \left(f_\kappa - \sum_{\lambda=1}^{\kappa-1} A_{\kappa\lambda}\, v_\lambda^{k+1} - \sum_{\lambda=\kappa+1}^{\tau} A_{\kappa\lambda}\, v_\lambda^{k} \right) \qquad \text{for} \quad \kappa = 1, 2, \ldots, \tau$$

where $\tau = (M-1)(N-1)$. If we interpret this iteration in terms of solving the κ^{th} equation of the system for v_κ^{k+1}, while keeping the other $\tau - 1$ variables fixed, then we can extend this idea to nonlinear equations; that is, if $\mathcal{F}$ has components $F_1, \ldots, F_\tau$, then the basic step of the nonlinear Gauss-Seidel iteration is to replace v_κ^k by the solution v_κ of the κ^{th} equation

$$F_\kappa(v_1^{k+1}, \ldots, v_{\kappa-1}^{k+1}, v_\kappa, v_{\kappa+1}^{k}, \ldots, v_\tau^{k}) = 0. \tag{5.2}$$

Thus one complete relaxation iteration $v^{k+1} \leftarrow v^k$ involves the successive solution of these τ nonlinear equations, $\kappa = 1, 2, \ldots, \tau$. Naturally, we may generalise this iteration by introducing a weighting parameter ω:

$$v_\kappa^{k+1} = v_\kappa^k + \omega(v_\kappa - v_\kappa^k).$$

In a completely analogous fashion, we can derive the nonlinear Jacobi iteration from the linear iteration

$$v_\kappa^{k+1} = \frac{1}{A_{\kappa\kappa}} \left(f_\kappa - \sum_{\substack{\lambda=1 \\ \lambda \neq \kappa}}^{\tau} A_{\kappa\lambda}\, v_\lambda^{k} \right) \qquad \text{for} \quad \kappa = 1, 2, \ldots, \tau$$

which is equivalent to solving the κ^{th} equation for v_κ while all other v_λ are held at their values from the previous iterate v_λ^k. Therefore the nonlinear Jacobi iteration updates v_κ^k with the solution v_κ of the equations

$$F_\kappa(v_1^{k}, \ldots, v_{\kappa-1}^{k}, v_\kappa, v_{\kappa+1}^{k}, \ldots, v_\tau^{k}) = 0 \qquad \text{for} \quad \kappa = 1, 2, \ldots, \tau. \tag{5.3}$$

Even assuming that the nonlinear equations (5.2) and (5.3) have unique solutions, there remains the difficulty of solving them analytically. In general

this is not possible, therefore a one-dimensional iterative method is applied, such as Newton's method, the secant method, or Steffensen iteration (see [84] or [102]). For example, the one-step Jacobi-Newton nonlinear iteration is defined by

$$v_\kappa^{k+1} = v_\kappa^k - \frac{F_\kappa(v^k)}{\partial_\kappa F_\kappa(v^k)} \qquad \text{for} \quad \kappa = 1, 2, \ldots, \tau.$$

Unfortunately, nonlinear relaxation cannot guarantee simultaneous reduction of all smooth error modes.

Let us now turn our attention to the second element of the multigrid method: the coarse-grid problem. The exact coarse-grid correction e is given by

$$\mathcal{A}(v-e) = f.$$

Combining this with the residual equation $v = \mathcal{A}^{-1}(f-r)$ and the definition $v' = \mathcal{A}^{-1}f$ gives us [51]

$$\begin{aligned} e &= \mathcal{A}^{-1}(f-r) - \mathcal{A}^{-1}f \\ &= -J^{-1}(v')r + O(\|r\|^2) \qquad \text{by Taylor's theorem} \end{aligned}$$

since

$$\left[\mathcal{A}^{-1}f\right]' = J^{-1}(\mathcal{A}^{-1}f).$$

Given a coarse-grid approximation v_{L-1}, we define

$$f_{L-1} = \mathcal{A}_{L-1}(v_{L-1}) \qquad \text{and} \qquad r_{L-1} = -f_{L-1} - \varpi I_{L-1}^L r_L.$$

In the usual two-grid fashion, we assume that the coarse-grid equation can be solved exactly: $e_{L-1} = \mathcal{A}_{L-1}^{-1}(r_{L-1})$, hence a Taylor expansion applied to

$$\begin{aligned} \tilde{v}_L &= \tfrac{1}{\varpi} I_L^{L-1}(v_{L-1} - e_{L-1}) \\ &= \tfrac{1}{\varpi} I_L^{L-1}\left[\mathcal{A}_{L-1}^{-1}(f_{L-1}) - \mathcal{A}_{L-1}^{-1}(-f_{L-1} - \varpi I_{L-1}^L r_L)\right] \end{aligned}$$

gives us [51]

$$\tilde{v}_L = I_L^{L-1} J_{L-1}^{-1}(v_{L-1}) I_{L-1}^L r_L + O\left(\varpi \|I_{L-1}^L r_L\|^2\right).$$

We conclude from these two Taylor expansions that

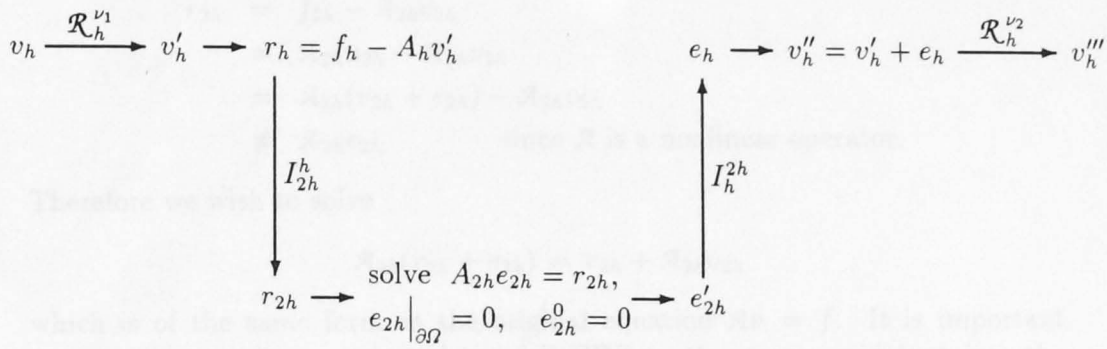
$$e_L \approx \mathcal{A}_L^{-1} r_L \qquad \text{and} \qquad \tilde{v}_L \approx I_L^{L-1} \mathcal{A}_{L-1}^{-1} I_{L-1}^L r_L$$

and so we expect $e_L \approx \tilde{v}$ when the residue is smooth, just as in the linear case.

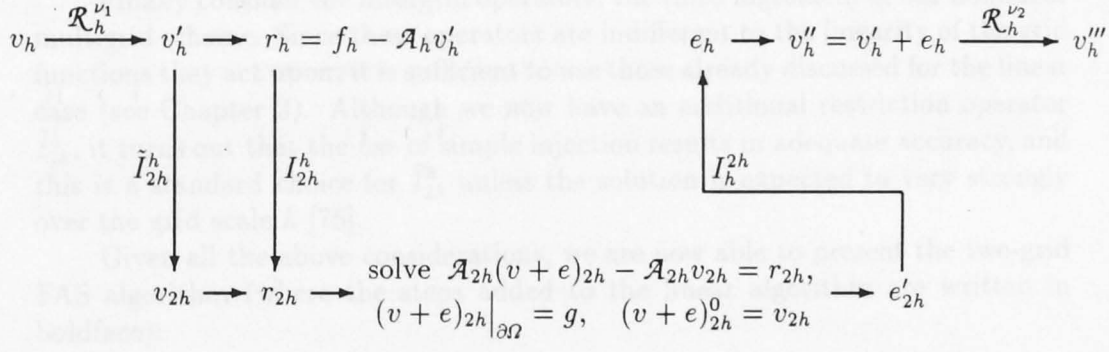
The most commonly used nonlinear multigrid scheme is the *full approximation scheme* (FAS) introduced by Brandt [15], which uses $\varpi = 1$ and

$$v_{L-1} = \tilde{I}_{L-1}^L v_L,$$

where $\tilde{I}_{L-1}^L$ may differ from the restriction I_{L-1}^L. In this direct nonlinear approach to multigrid, no global linearisation is explicitly carried out. A diagram of the FAS two-grid process appears at Figure 5.1.



(a) Two-grid linear V-cycle



(b) Two-grid nonlinear V-cycle

Figure 5.1: Pictorial representation of the (a) linear, and (b) nonlinear (FAS) two-grid V-cycle schemes (after Stüben and Trottenberg [93]). The linear cycle consists of pre-relaxation, residual calculation, residual restriction, solution of the coarse-grid residual equation, error interpolation, error correction and post-relaxation. The full approximation scheme adds a v_h'-restriction step, $\tilde{I}_{2h}^h$, and requires the solution of a full version of the coarse-grid equation. Multigrid schemes recursively solve the coarse-grid equation, which is of the same form as the original problem: solve $\mathcal{A}v = f$, where $v|_{\partial\Omega} = g$, with initial guess $v^0 = v_0$.

The derivation of the coarse-grid equation for the nonlinear method proceeds as follows:

$$\begin{aligned} r_{2h} &= f_{2h} - \mathcal{A}_{2h} v_{2h} \\ &= \mathcal{A}_{2h} u_{2h} - \mathcal{A}_{2h} v_{2h} \\ &= \mathcal{A}_{2h}(v_{2h} + e_{2h}) - \mathcal{A}_{2h} v_{2h} \\ &\neq \mathcal{A}_{2h} e_{2h} \qquad \text{since } \mathcal{A} \text{ is a nonlinear operator.} \end{aligned}$$

Therefore we wish to solve

$$\mathcal{A}_{2h}(v_{2h} + e_{2h}) = r_{2h} + \mathcal{A}_{2h} v_{2h}$$

which is of the same form as the original equation $\mathcal{A}v = f$. It is important to note that we have projected the full PDE to the coarse grid (not just the residual equation), hence we solve the FAS coarse-grid equation with the actual boundary conditions g of the problem, rather than with zero boundary conditions as in the linear case. Another notable point is that while we solve for the "full approximation" $(v+e)$ on the coarse grid (rather than just the correction e as in the linear case), it is the correction e that is transferred back to the fine grid. This is important since only residual and correction quantities are smoothed by relaxation and so can be properly represented on coarser grids.

Finally consider the intergrid operators, the third ingredient of our nonlinear multigrid scheme. Since these operators are indifferent to the linearity of the grid functions they act upon, it is sufficient to use those already discussed for the linear case (see Chapter 3). Although we now have an additional restriction operator $\tilde{I}_{2h}^{h}$, it turns out that the use of simple injection results in adequate accuracy, and this is a standard choice for $\tilde{I}_{2h}^{h}$ unless the solution is expected to vary strongly over the grid scale h [75].

Given all the above considerations, we are now able to present the two-grid FAS algorithm (where the steps added to the linear algorithm are written in boldface):

Algorithm 5.1 (Two-grid Full Approximation Scheme)
An iteration for solving $\mathcal{A}_h u_h = f_h$, given some initial guess v_h.

$v_h \leftarrow$	$\mathcal{R}_h^{\nu_1}(v_h, f_h)$	pre-smoothing
$r_h \leftarrow$	$f_h - \mathcal{A}_h v_h$	residual calculation
$r_{2h} \leftarrow$	$I_{2h}^{h} r_h$	residual restriction
$v_{2h} \leftarrow$	$\tilde{I}_{2h}^{h} v_h$	**approximate solution restriction**
$e_{2h} \leftarrow$	$\mathcal{A}_{2h}^{-1}(r_{2h} + \mathcal{A}_{2h} v_{2h}) - v_{2h}$	**coarse-grid equation solution**
$v_h \leftarrow$	$v_h + I_h^{2h} e_{2h}$	coarse-grid correction
$v_h \leftarrow$	$\mathcal{R}_h^{\nu_2}(v_h, f_h)$	post-smoothing

This naturally extends to the multigrid FAS algorithm, where (as before) $\gamma = 1$ gives the V-cycle, while $\gamma = 2$ produces the W-cycle:

Algorithm 5.2 (Multigrid Full Approximation Scheme V- or W-cycle)
A recursive iteration for solving $\mathcal{A}_\Lambda u_\Lambda = f_\Lambda$, given some initial guess v_Λ.

procedure FAS-VW-cycle (v, f, L)

begin
if $L = 1$ **then**
$v_L \leftarrow \mathcal{R}_L^{\nu_0}(v_L, f_L)$
else
$v_L \leftarrow \mathcal{R}_L^{\nu_1}(v_L, f_L)$
$r_{L-1} \leftarrow I_{L-1}^L (f_L - \mathcal{A}_L v_L)$
$\tilde{v}_{L-1} \leftarrow \tilde{I}_{L-1}^L v_L$
$v_{L-1} \leftarrow \tilde{v}_{L-1}$
$f_{L-1} \leftarrow r_{L-1} + \mathcal{A}_{L-1}\tilde{v}_{L-1}$
for $i = 1$ **to** γ **do**
FAS-VW-cycle $(v, f, L-1)$
endfor
$e_{L-1} \leftarrow v_{L-1} - \tilde{v}_{L-1}$
$v_L \leftarrow v_L + I_L^{L-1} e_{L-1}$
$v_L \leftarrow \mathcal{R}_L^{\nu_2}(v_L, f_L)$
endif
end

Just as in the linear case, the M-cycle is recursively constructed from a sequence of increasingly larger V-cycles (or possibly W-cycles, although this is not the usual choice):

Algorithm 5.3 (Multigrid Full Approximation Scheme M-cycle)
A recursive iteration for solving $\mathcal{A}_\Lambda u_\Lambda = f_\Lambda$, given some initial guess v_1.

procedure FAS-M-cycle (v, f, L)

begin
if $L \neq 1$ **then**
$f_{L-1} \leftarrow I_{L-1}^L (f_L - \mathcal{A}_L v_L)$
$v_{L-1} \leftarrow 0$
$v_{L-1} \leftarrow$ FAS-M-cycle $(v, f, L-1)$
$v_L \leftarrow v_L + I_L^{L-1} v_{L-1}$
endif
$v_L \leftarrow$ FAS-VW-cycle (v, f, L)
end

We note that these algorithms are generalisations of the linear multigrid methods in the sense that they produce identical results for a linear boundary value problem.

Convergence analysis of nonlinear multigrid is a difficult area. There are two aspects of our theoretical knowledge which lag behind our practical experience: it is difficult to prove convergence without imposing fairly strict conditions on the boundary value problem, and the theoretical convergence bounds are not particularly sharp. Numerical experiments indicate that the class of problems suitable for multigrid is far wider than we can currently prove, moreover that the actual numerical convergence rates generally far exceed any theoretical estimate.

Hackbusch [50, 51] analyses the convergence of the nonlinear multigrid iteration in some detail. His main result is that these iterations behave asymptotically in the same manner as the corresponding linear processes.

Stüben and Trottenberg [93] show that, under reasonable assumptions, the FAS M-cycle has the following two properties:

1. an approximation u_h to the exact solution of the discrete problem $\tilde{u}_h$ can be computed with an error $\|u_h - \tilde{u}_h\|$ which is smaller than the discretisation error $\|u - \tilde{u}_h\|$, and
2. the number of arithmetic operations required to do so is proportional to $\mathcal{N}_h$, the number of grid points on Ω_h, with a small constant of proportionality (depending on γ, h_L/h_{L+1}, the type of cycle, *etc*).

We can confirm this second property by extending the discussion of computational costs of multigrid in Section 3.3 as follows [93]. Suppose w_{L+1}^L denotes the computational work involved in one $(L, L+1)$ two-grid cycle (excluding the work required to solve the coarse-grid equation), and let w_1 be the work performed on the coarsest grid Ω_1. Then the work required for a complete multigrid (V- or W-) cycle is

$$w_\Lambda = \sum_{L=1}^{\Lambda} \gamma^{\Lambda-L} w_L^{L-1} + \gamma^{\Lambda-1} w_1.$$

Ignoring boundary effects, we have in d dimensions

$$\mathcal{N}_L = 2^d \mathcal{N}_{L-1}.$$

Let the computational work per grid point for a two-grid cycle be bounded by some constant C

$$w_L^{L-1} < C\mathcal{N}_L$$

then we obtain (in 2-D)

$$w_\Lambda < \begin{cases} \frac{4}{3} C\mathcal{N}_\Lambda & \text{for } \gamma = 1 \quad \text{(V-cycle)} \\ 2C\mathcal{N}_\Lambda & \text{for } \gamma = 2 \quad \text{(W-cycle)} \end{cases}$$

This estimate of w_Λ, together with the fact that the multigrid convergence rate is independent of h, shows that the multigrid iteration is asymptotically optimal; that is, multigrid solves an algebraic system of n unknowns to the level of truncation error in $O(n)$ operations.

5.2 FAS Implementation

To simplify the design of MGLAB, we chose to forego the general nonlinear relaxation methods discussed above in favour of single-step Newton relaxation on the nonlinear equation; this idea is discussed in Stuben and Trottenberg [93]. This is sufficient, since (as always) we only require the approximate solution to the (approximate) coarse-grid equation — it is fruitless to attempt to calculate the exact solution of an approximate equation.

If our discretised PDE is represented in the form $F(v_{ij}) = 0$, then we generate a relaxation scheme from the Newton iteration

$$v_{ij}^{k+1} = v_{ij}^{k} - \frac{F(v_{ij}^{k})}{F'(v_{ij}^{k})} \qquad \text{for} \quad i = 0, 1, \ldots, M; \quad j = 0, 1, \ldots, N. \tag{5.4}$$

The results of this relaxation method are most acceptable, as we demonstrate in the next chapter.

A sufficient condition for the convergence of Newton's iteration is [8]

$$\left| \frac{F(v_{ij})F''(v_{ij})}{[F'(v_{ij})]^2} \right| < 1 \tag{5.5}$$

which must be satisfied for each PDE treated in this way.

It is fortunate that the linear multigrid algorithm can be easily modified to perform the nonlinear full approximation scheme. The back-end library routines in MGLAB require only two minor changes. Firstly, each routine which performs some coarse-grid calculation with the boundary conditions is modified to use the actual boundary conditions of the problem, rather than zero boundary conditions as in the linear method.

Secondly, subroutine V_CYCLE is changed (see Algorithm 5.1) to

- set the initial coarse-grid guess to $\tilde{I}_{2h}^{h} v_{L-1}$,
- add $\mathcal{A}_L v_L$ to the right-hand side functional, and
- subtract the full approximation v_L from the coarse-grid error correction term.

We also require the user to write a PDE-specific relaxation routine (a Newton-step method as described above, for example) for the front-end driver program.

This gives us a fully nonlinear multigrid method, which we shall apply to several interesting nonlinear problems in the next two chapters.

Chapter 6

Model Nonlinear Problems

How can it be that mathematics, a product of human thought independent of experience, is so admirably adapted to the objects of reality?

— Albert Einstein

In this chapter, we discuss the numerical solution of several model nonlinear boundary value problems using MGLAB. This will be analogous to the discussion of the linear model problems in Chapter 4. These results will help to confirm that we have a valid nonlinear multigrid implementation.

6.1 Statement of Model Nonlinear Problems

Four nonlinear boundary value problems were selected to test MGLAB and to explore the effect of various multigrid parameters on the solution process. A statement of these model nonlinear problems appears on the following page.

These problems cover a certain range of nonlinear behaviour. Problem MG20 is related to an equation believed to govern the (free) boundary of phase transition in a solid/liquid interface model. In fact, the parabolic problem $u_t = \mathcal{F}(u)$ coupled with a suitable thermal equation is a model for crystalisation and dendrite growth (see Caginalp and Fife [21]).

Problem MG22 is the static porous medium equation. It models the density of an ideal gas (with ratio of specific heats γ) which has reached an equilibrium flow through a homogeneous porous medium. This equation also arises in models of plasma physics. For the equation to represent a physical system, we must have $u \geq 0$ and $\gamma > 0$. The PDE is elliptic except at any point where $u = 0$. For more details on our present knowledge of this difficult equation, see Aronson [7].

Problem MG24 is a scalar version of the Navier-Stokes equations, where ν is the kinematic viscosity (it is also an ellipticity parameter). This problem has rather arbitrary boundary conditions, hence the exact solution is unknown.

Finally, Problem MG26 is perhaps the most interesting model quasilinear elliptic equation, as it encompasses the behaviour of soap films (minimal surfaces,

Problem MG20 (solidification equation)

$$\Delta u + 2\gamma^2(1+\beta^2)(u-u^3) = 0 \qquad \text{in } \Omega; \qquad \alpha, \beta, \gamma \in \boldsymbol{R}$$

$$u(0,y) = \tanh[\gamma(\alpha-y)] \qquad u(1,y) = \tanh[\gamma(\alpha-\beta-y)]$$
$$u(x,0) = \tanh[\gamma(\alpha-\beta x)] \qquad u(x,1) = \tanh[\gamma(\alpha-\beta x-1)]$$

$$v^0 = 0 \qquad \text{in } \Omega$$

$$u = \tanh[\gamma(\alpha-\beta x-y)]$$

Problem MG22 (static porous medium equation)

$$\Delta(u^\gamma) = 0 \qquad \text{in } \Omega; \qquad \gamma \in \boldsymbol{R}, \quad \gamma \neq 0$$

$$u(0,y) = \left[\frac{\sin\varepsilon\,\cosh(\mu y+\varepsilon)}{\pi^2}\right]^{1/\gamma} \qquad u(1,y) = \left[\frac{\sin\varepsilon\,\cosh(\mu y+\varepsilon)}{\pi^2}\right]^{1/\gamma}$$
$$u(x,0) = \left[\frac{\sin(\mu x+\varepsilon)\,\cosh\varepsilon}{\pi^2}\right]^{1/\gamma} \qquad u(x,1) = \left[\frac{\sin(\mu x+\varepsilon)\,\cosh(\pi-\varepsilon)}{\pi^2}\right]^{1/\gamma}$$

where $\mu = \pi - 2\varepsilon; \quad \varepsilon \in \boldsymbol{R}$

$$v^0 = \left[\frac{\sin\varepsilon\,\cosh\varepsilon}{\pi^2}\right]^{1/\gamma}$$

$$u = \left[\frac{\sin(\mu x+\varepsilon)\,\cosh(\mu y+\varepsilon)}{\pi^2}\right]^{1/\gamma}$$

Problem MG24 (pseudo Navier-Stokes equation)

$$\nu\Delta u - u(u_x+u_y) = 0 \qquad \text{in } \Omega; \quad \nu \in \boldsymbol{R}, \quad \nu \neq 0$$

$$u(0,y) = \sin\pi y \qquad u(1,y) = \sin\tfrac{5}{2}\pi y$$
$$u(x,0) = 0 \qquad u(x,1) = x$$

$$v^0 = 0 \qquad \text{in } \Omega$$

u is unknown

Problem MG26 (prescribed mean curvature equation)

$$\frac{u_{xx}(1+{u_y}^2) - 2u_xu_yu_{xy} + u_{yy}(1+{u_x}^2)}{(1+{u_x}^2+{u_y}^2)^{3/2}} = H(x,y) \qquad \text{in } \Omega$$

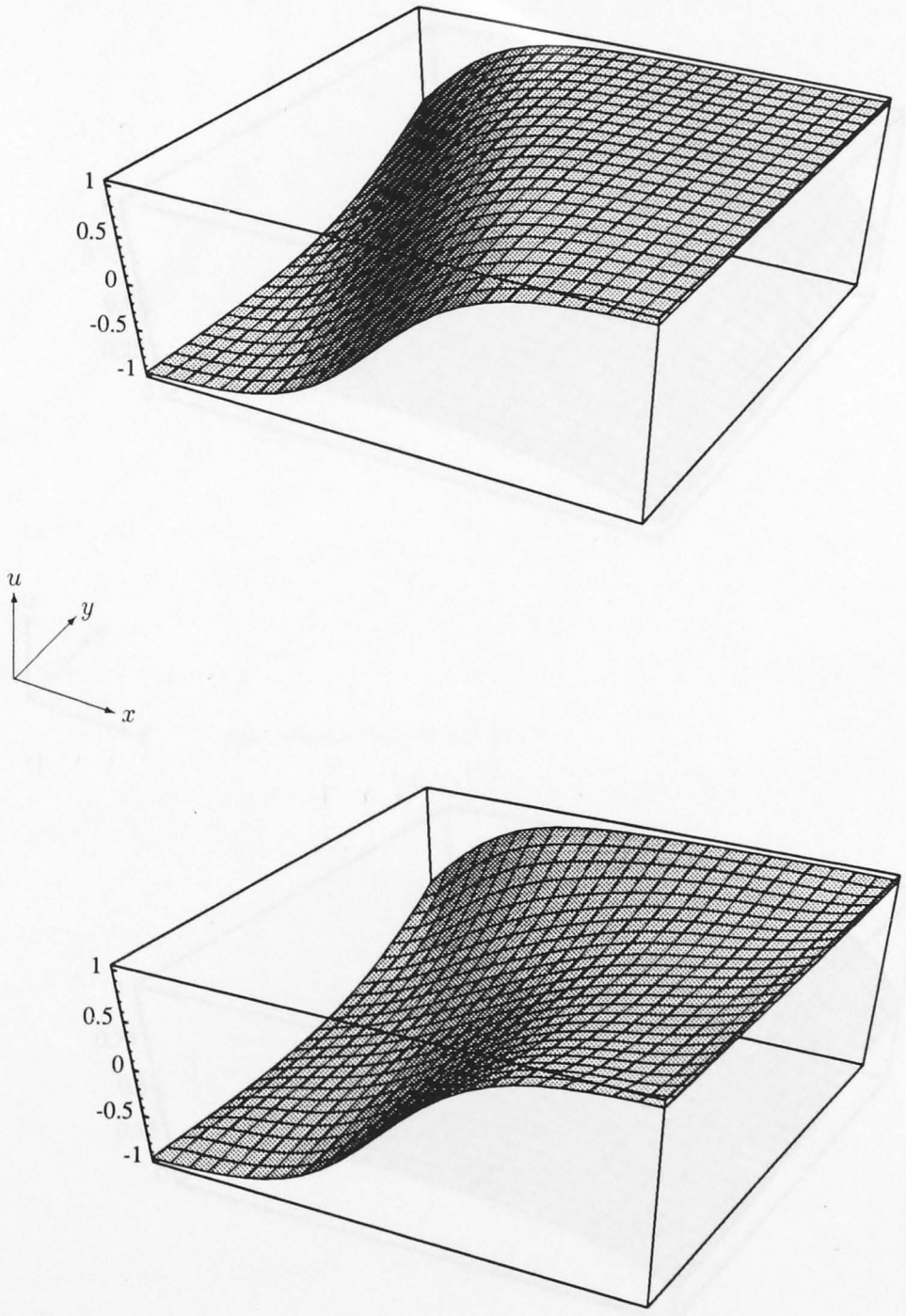


Figure 6.1: Solution (top) and discrete-Laplacian initial guess (bottom) for Problem MG20 using the standard data set (except $\gamma = -2.5$).

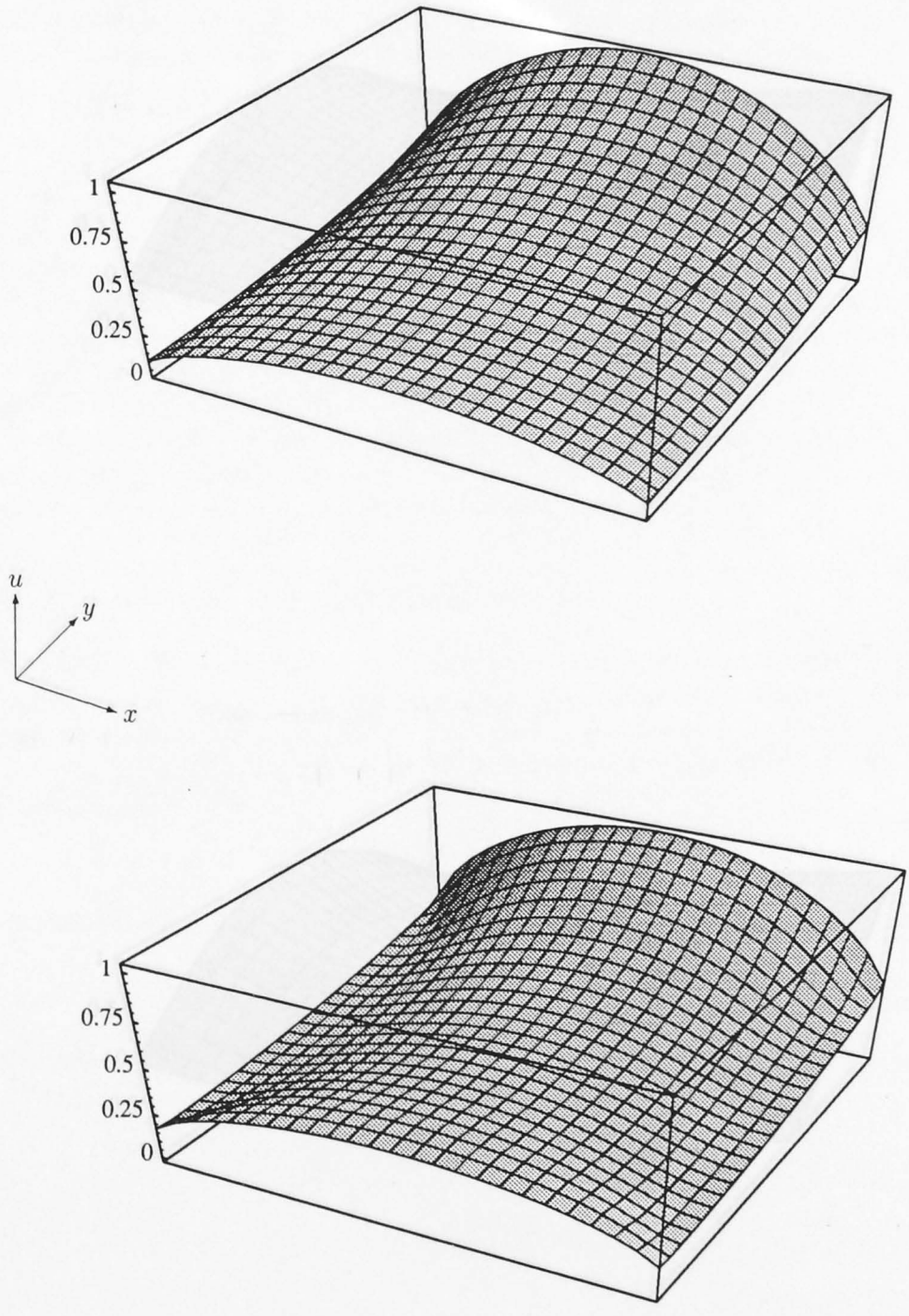


Figure 6.2: Solution (top) and discrete-Laplacian initial guess (bottom) for Problem MG22 using the standard data set.

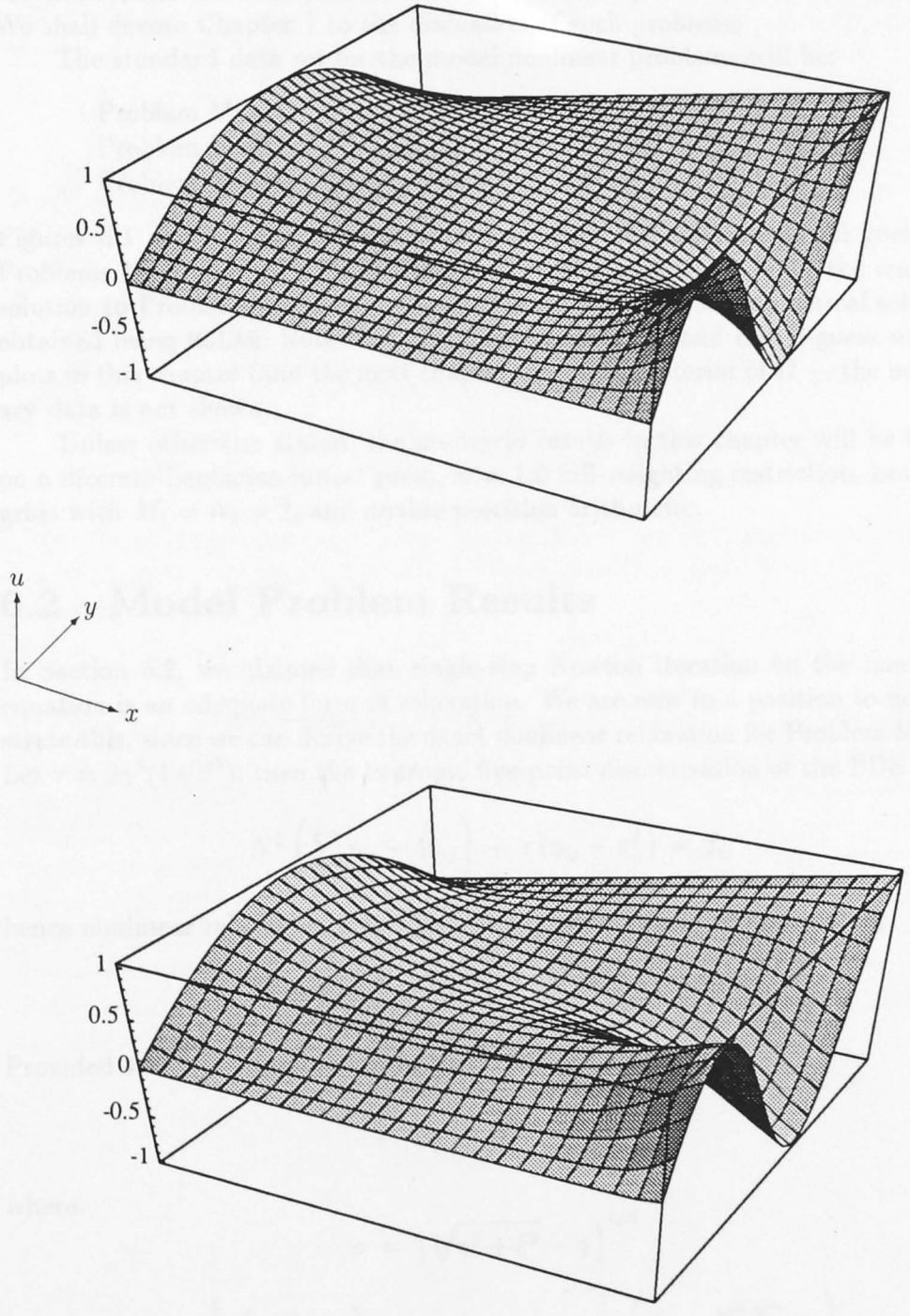


Figure 6.3: Numerical solution (top) and discrete-Laplacian initial guess (bottom) for Problem MG24 using the standard data set.

that is, surfaces of zero mean curvature), soap bubbles (surfaces of constant mean curvature), and more complex structures (surfaces of prescribed mean curvature). We shall devote Chapter 7 to the discussion of such problems.

The standard data set for the model nonlinear problems will be:

Problem MG20	$\alpha = 1.0, \quad \beta = 2.0, \quad \gamma = -1.0$
Problem MG22	$\gamma = 2.0, \quad \varepsilon = 0.1$
Problem MG24	$\nu = 0.8$

Figures 6.1 and 6.2 show the solution and discrete-Laplacian initial guess for Problems MG20 and MG22 for these particular parameters. Since the analytic solution to Problem MG24 is unknown, Figure 6.3 shows the numerical solution obtained using `MGLAB`. Note that all numerical-solution and initial-guess surface plots in this chapter (and the next chapter) are of the interior of Ω — the boundary data is not shown.

Unless otherwise stated, the multigrid results in this chapter will be based on a discrete-Laplacian initial guess, $\varpi = 1.0$ full-weighting restriction, isotropic grids with $M_1 = N_1 = 2$, and double-precision arithmetic.

6.2 Model Problem Results

In Section 5.2, we claimed that single-step Newton iteration on the nonlinear equation is an adequate form of relaxation. We are now in a position to demonstrate this, since we can derive the exact nonlinear relaxation for Problem MG20. Let $\tau = 2\gamma^2(1+\beta^2)$, then the isotropic five-point discretisation of the PDE is

$$N^2 \left(\sum_{nn} v_{ij} - 4v_{ij} \right) + \tau \left(v_{ij} - v_{ij}^3 \right) = f_{ij}$$

hence nonlinear relaxation is based on the solution of the cubic equation

$$\tau v_{ij}^3 + (4N^2 - \tau) v_{ij} + f_{ij} - N^2 \sum_{nn} v_{ij} = 0.$$

Provided $4N^2 > \tau$, the solution to this equation is

$$v_{ij} = \sigma - \frac{\xi}{\sigma}$$

where

$$\sigma = \left(\sqrt{\eta^2 + \xi^3} - \eta \right)^{1/3}$$

$$\xi = \frac{1}{3\tau} \left(4N^2 - \tau \right) \qquad \text{and} \qquad \eta = \frac{1}{2\tau} \left(f_{ij} - N^2 \sum_{nn} v_{ij} \right).$$

On the other hand, the single Newton step is based on

$$\tilde{v}_{ij} = v_{ij} - \frac{F(v_{ij})}{F'(v_{ij})}$$

where

$$F(v_{ij}) = v_{ij}^3 + 3\xi v_{ij} + 2\eta$$

hence

$$\tilde{v}_{ij} = \frac{2\,(v_{ij}^3 - \eta)}{3\,(v_{ij}^2 + \xi)}.$$

In the following experiment, we perform $1 \times V_8^{2,2,2}$-cycle of $\omega = 1.0$ weighted Jacobi relaxation on Problem MG20 with the standard data set, comparing the results of a Newton step and the exact nonlinear relaxation (thus we guarantee that $4N^2 \geq 16 > \tau = 10.0$).

Method	$\|e\|_2$	$\|e\|_\infty$	$\|r\|_2$	$\|r\|_\infty$
Initial Guess	0.088609	0.174754	5.871917	220.52840
Exact	0.012512	0.024418	0.729692	14.575176
Newton	0.012495	0.024382	0.729485	14.575177

This clearly shows that single-step Newton iteration is very satisfactory. A similar experiment can be performed on Problem MG22, since the exact nonlinear relaxation is

$$v_{ij} = \left[\frac{1}{4}\left(\sum_{nn} v_{ij}^{\gamma} - \frac{f_{ij}}{N^2}\right)\right]^{1/\gamma}.$$

We next wish to experimentally determine the optimal weighting for relaxation for Problem MG20. Figure 6.4 (on page 92) shows the convergence of $11 \times V_8^{2,2,2}$-cycles for Problem MG20 using ω-weighted Jacobi point relaxation. We find that the multigrid iteration is unstable for $\omega > 1.0$, presumably because over-relaxation excites some oscillatory modes. While the residue norm for $\omega = 0.8$ is linear and descends most steeply after eleven V-cycles, we chose $\omega_{\text{opt}} = 1.0$ since both residue and error norms converge fastest of all the asymptotically stable weights after one V-cycle. In fact it does turn out that $\omega = 1.0$ results in approximately optimal residue convergence over $1 \times M_8^{2,2,2}$-cycle, as we see from the following figures.

ω	$\|e\|_\infty$	$\|r\|_\infty$
0.8	0.00000804	0.22085107
0.9	0.00000527	0.14771894
1.0	0.00000398	0.12505166
1.1	0.00000341	0.21971002

Figure 6.5 shows the micro-structure of one V-cycle for Problem MG20. The characteristics of the error profiles are similar to those of the linear multigrid scheme (*cf.* Figure4.7). The residue profiles are slightly different, however, reflecting the fact that the FAS scheme maintains an approximation of the full equation on each grid level, rather than just the residual equation.

Figure 6.6 shows the micro-structure of one M-cycle for Problem MG20. The M-cycle has the same overall behaviour as in the linear multigrid scheme (*cf.* Figure4.13); this is to be expected since the M-cycle is composed of single V-cycles.

Figure 6.7 (analogous to Figure 4.9) examines the discretisation error of the FAS scheme for Problems MG20 and MG22. We have again plotted the limiting value of the error infinity-norm after many V_Λ-cycles for successive Λ. We find that each profile has a slope of

$$m = -0.602 \pm 0.001$$

and hence $10^m = 0.250 \pm 0.001$. This experiment confirms that the discretisation error is $O(h^2)$ for nonlinear problems.

Figure 6.8 (analogous to Figure 4.10) shows the convergence of $14 \times V_\Lambda^{2,2,2}$-cycles for Problem MG20 for various Λ. We find that FAS V-cycle convergence is independent of h.

We believe that the above results are consistent with a valid nonlinear multigrid scheme; let us now experiment with the model problem parameters.

The following table shows the result of $1 \times M_8^{2,2,2}$-cycle using $\omega = 1.0$ Jacobi relaxation on Problem MG20. We set $\alpha = 1.0$, $\beta = 2.0$ and vary γ.

γ	τ	$\log_{10}\|e\|_\infty$	$\log_{10}\|r\|_\infty$
-0.1	0.1	-7.694	-3.217
-0.5	2.5	-6.004	-1.505
-1.0	10.0	-5.400	-0.903
-1.5	22.5	-5.172	-0.551
-2.0	40.0	-3.119	-0.302
-2.2	48.4	-3.151	-0.219
-2.3	52.9	-1.293	0.136
-2.4	57.6	-0.712	0.672
-2.5	62.5	-0.376	1.080
-2.6	67.6	0.518	2.922

We see that this particular multigrid iteration becomes unstable for $\tau \gtrsim 60$. This is because increasing τ makes the interface in the solution increasingly sharp; in fact in the limit as $\tau \to \infty$, this becomes a step function. In the xy–plane, the edge of the interface has the equation $y = \alpha - \beta x$; anisotropic grids are therefore of little use to us because the interface does not necessarily lie parallel to either coordinate axis. To numerically investigate the behaviour of this problem for large τ, we would need to increase the grid resolution (by utilising larger Λ's); however we note that because the solution is slowly-varying everywhere except at the interface, this is a wasteful scheme, and a more appropriate one would be to use adaptive gridding.

Let us now turn our attention to the porous medium equation. The following table shows the result of $1 \times M_7^{2,2,2}$-cycle for Problem MG22 using ω-weighted Jacobi point relaxation.

ω	$\log_{10}\|e\|_\infty$	$\log_{10}\|r\|_\infty$
0.8	−3.550	0.442
0.9	−3.694	0.280
1.0	−3.817	0.143
1.1	−3.909	0.249

We choose $\omega = 1.0$ since it approximately minimises the resultant residue norm.

Since the porous medium equation degenerates when $u = 0$, we have constructed Problem MG22 with a parameter ε so that the domain of the sine function is restricted to $[\varepsilon, \pi-\varepsilon]$ rather than $[0, \pi]$, thus ensuring $u > 0$ for $0 < \varepsilon < \frac{\pi}{2}$. In the following experiment, we have performed $1 \times M_7^{2,2,2}$-cycle with $\omega = 1.0$, $\gamma = 2.0$ and various values of ε.

ε	$\log_{10}\|e\|_\infty$	$\log_{10}\|r\|_\infty$
0.1	−3.817	0.143
0.01	−2.892	0.945
0.001	−2.572	1.229
0.0	−2.447	1.355

We find that convergence does indeed deteriorate as the boundary data approaches zero, but the iteration remains stable.

Next let us fix ε to be 0.1 and vary γ. We know that for large γ, the solution tends towards $u = 1$, while for small positive γ, the solution tends to $u = 0$ everywhere except for a spike at $x = \frac{1}{2}$, $y = 1$:

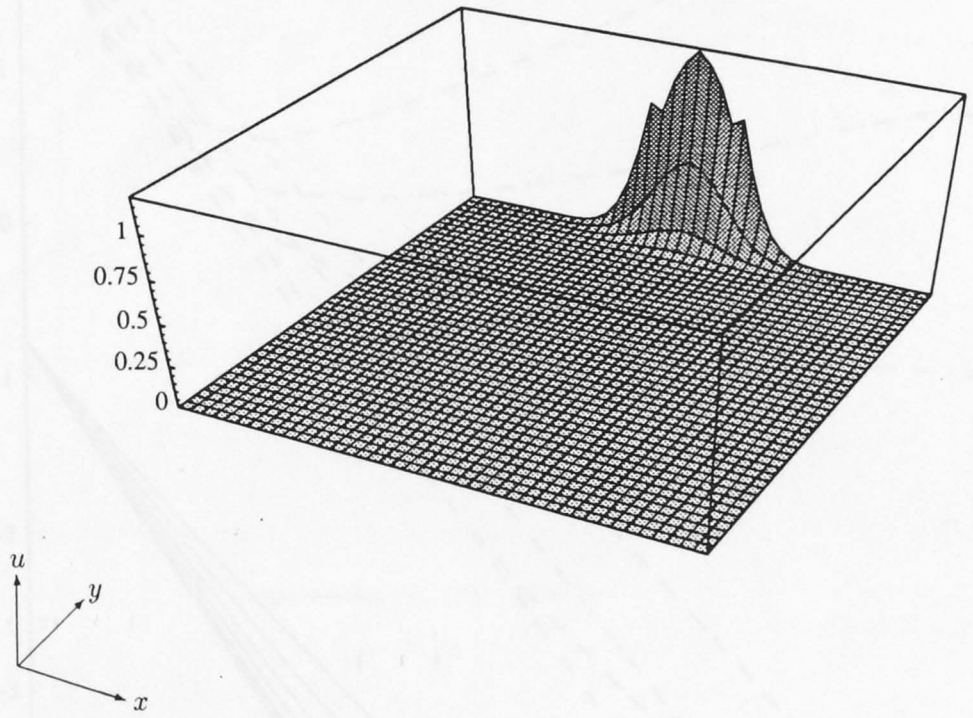
$$u \to \delta(\tfrac{1}{2}, 1) \qquad \text{as} \quad \gamma \to 0^+.$$

We therefore expect the performance of a multigrid iteration to deteriorate as $\gamma \to 0$. Figure 6.9 shows the result of one M-cycle for various γ; we do indeed find that this iteration becomes ineffective for $\gamma \lesssim 0.1$. We note that the best solution is obtained when $\gamma = 1$, as we would expect, since this corresponds to Laplace's equation.

MGLAB allows us to investigate the behaviour of multigrid in more detail; for example, the surface plot on the following page shows the resultant numerical error e for $\gamma = 0.05$ from the above experiment. It is clear that the greatest error occurs where the solution is rapidly-changing, as we expect.

Finally, let us briefly look at the pseudo-Navier-Stokes equation. Table 6.1 shows the result of $1 \times M_7^{2,2,2}$-cycle for Problem MG24 using ω-weighted Jacobi point relaxation. In this case, $\omega = 0.9$ appears to be an appropriate choice for the Jacobi relaxation weight.

Figure 6.10 shows the resulting residue norms after one M-cycle with various values of ν. The iteration becomes unstable as $\nu \to 0$ because the PDE becomes less elliptic (the equation is singular for $\nu = 0$).



ω	$\log_{10} \|r\|_2$	$\log_{10} \|r\|_\infty$
0.7	−0.63	0.63
0.8	−0.85	0.43
0.9	−1.02	0.30
1.0	−0.95	0.37
1.1	−0.64	0.61

Table 6.1: Result of $1 \times M_7^{2,2,2}$-cycle for Problem MG24 using ω-weighted Jacobi point relaxation.

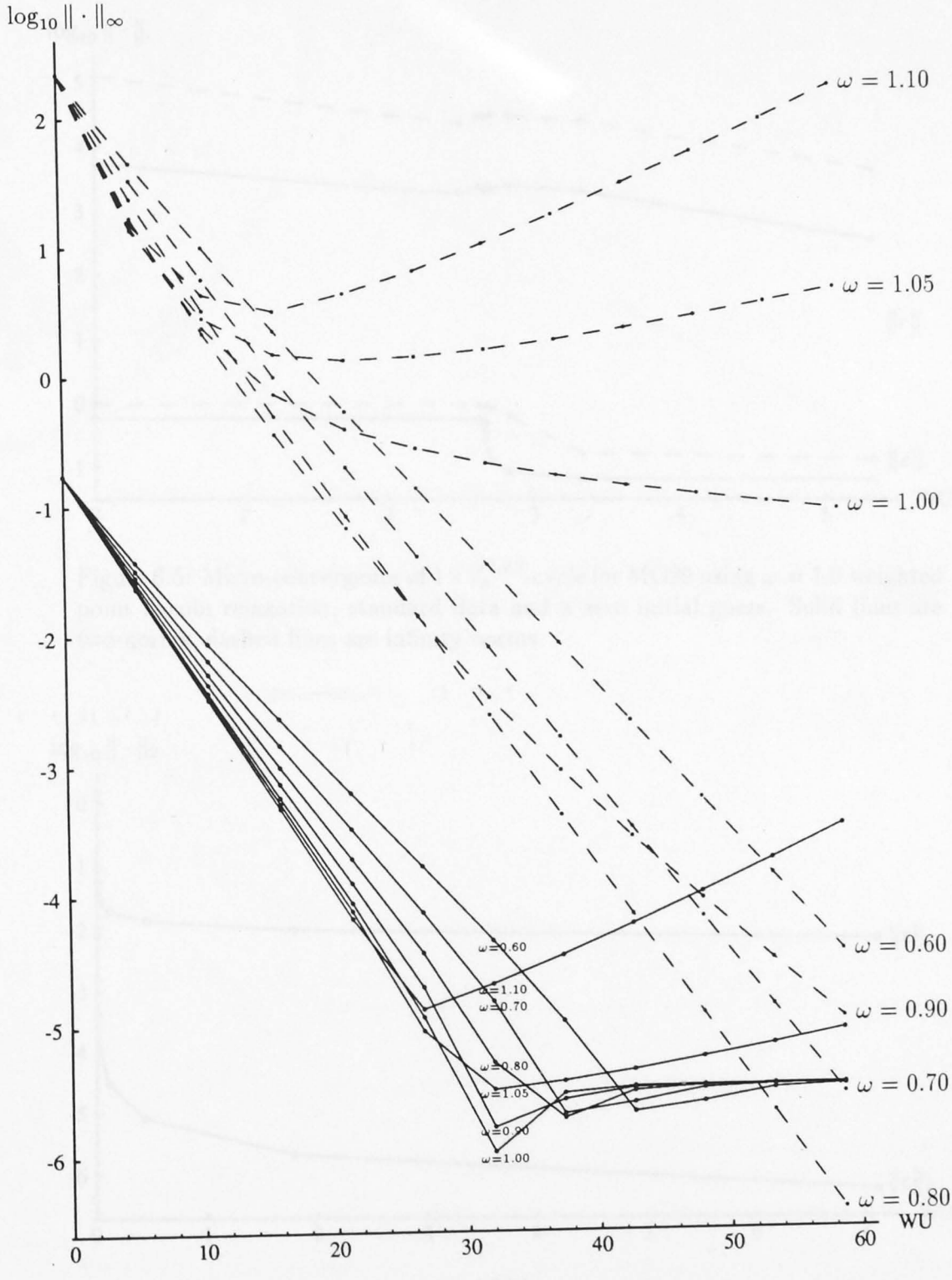


Figure 6.4: Convergence of $11 \times V_8^{2,2,2}$-cycles for Problem MG20 using various ω-weighted Jacobi point relaxation. Solid lines are error norms, dashed lines are residue norms.

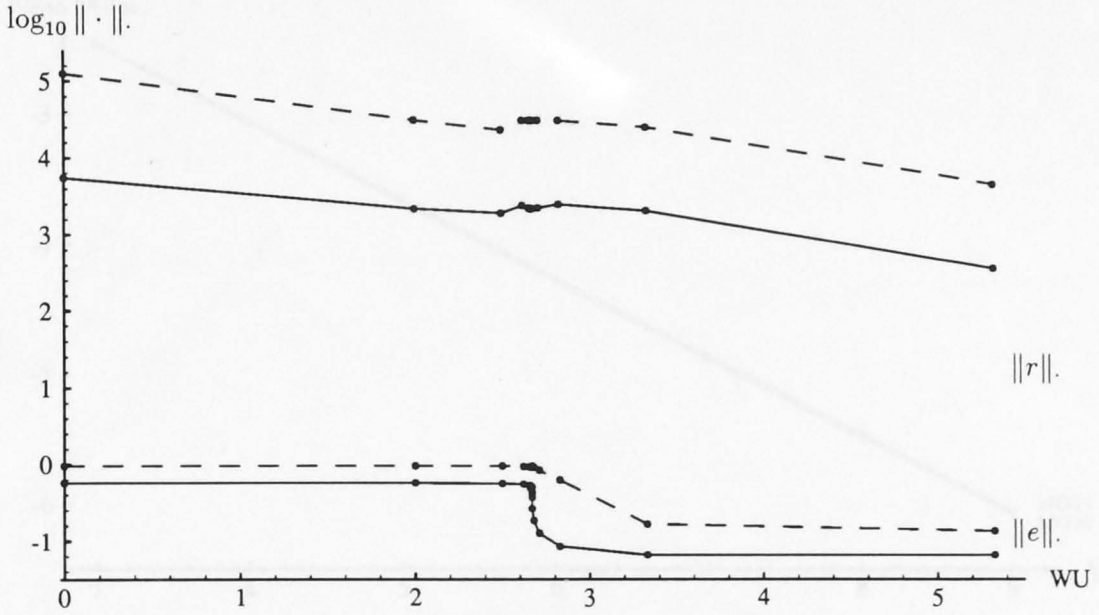


Figure 6.5: Micro-convergence of $1 \times V_8^{2,2,2}$-cycle for MG20 using $\omega = 1.0$ weighted point Jacobi relaxation, standard data and a zero initial guess. Solid lines are two-norms, dashed lines are infinity-norms.

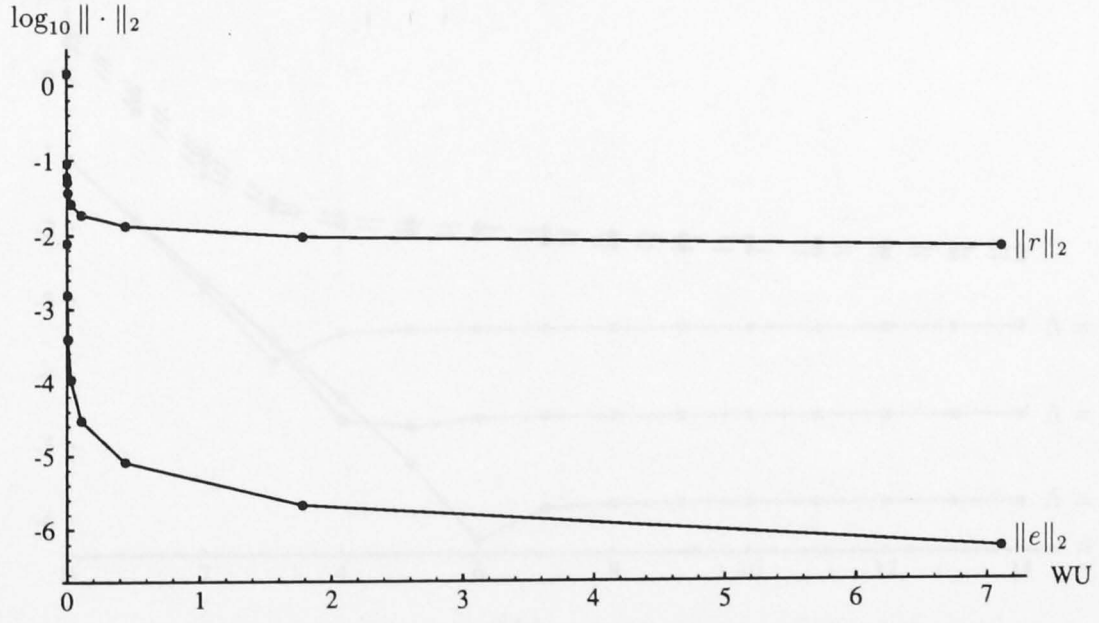


Figure 6.6: Micro-convergence of $1 \times M_9^{2,2,2}$-cycle for MG20 using $\omega = 1.0$ weighted point Jacobi relaxation and standard data.

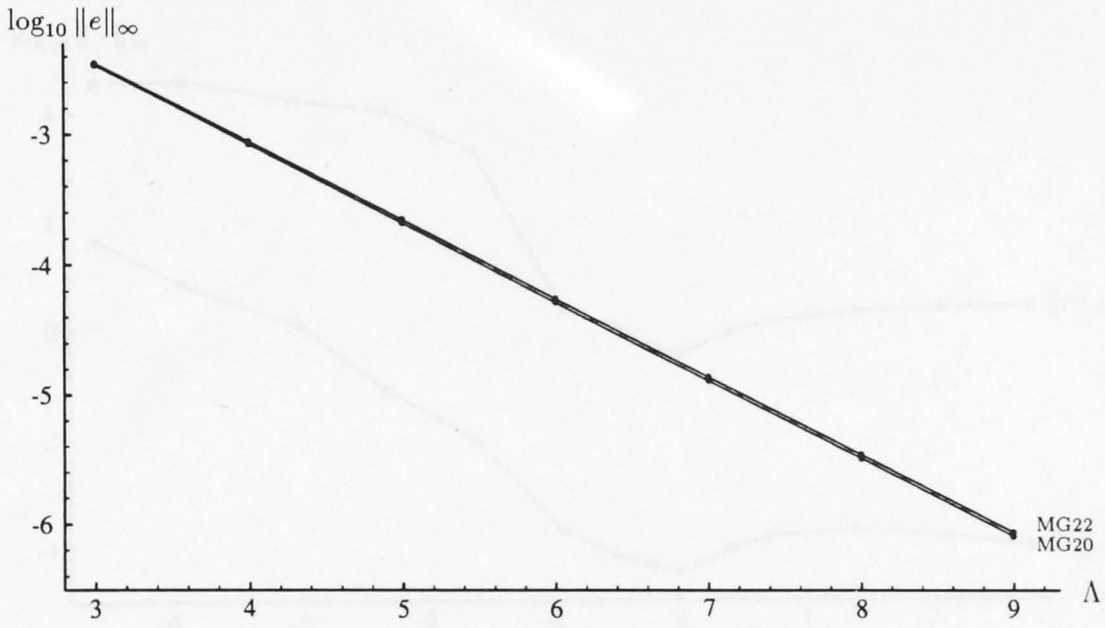


Figure 6.7: Discretisation errors as measured by the error norm of $\lim_{n\to\infty} n \times V_\Lambda^{2,2,2}$-cycles for various Λ using $\omega = 1.0$ weighted Jacobi point relaxation. The discretisation error is $O(h^2)$.

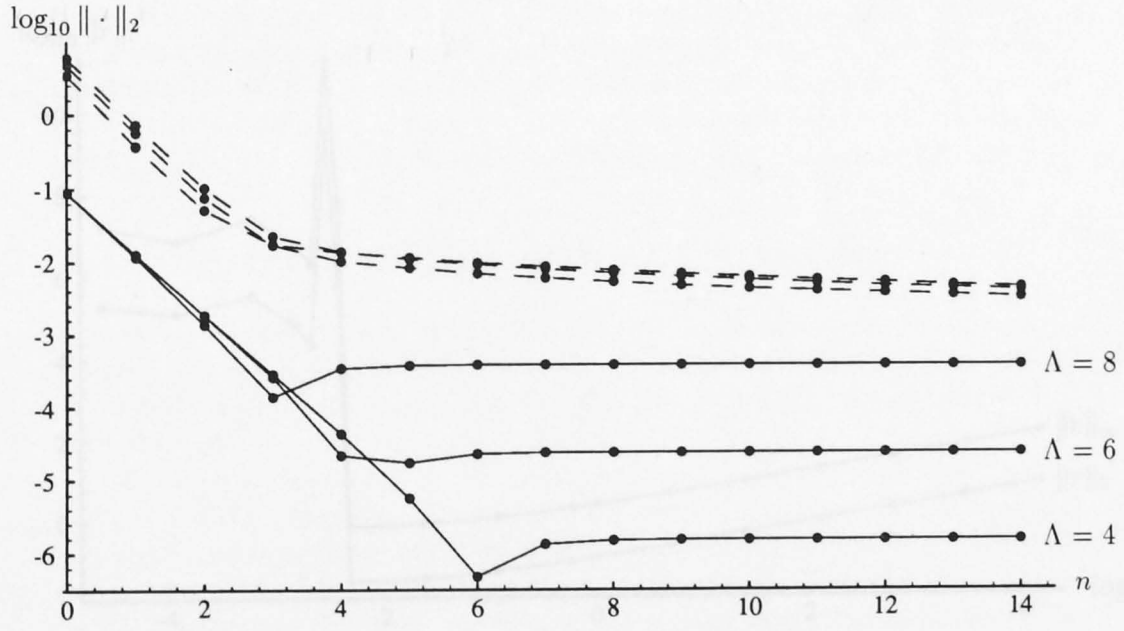


Figure 6.8: Convergence of $14 \times V_\Lambda^{2,2,2}$-cycles for various Λ. Solid lines are error norms, dashed lines are residue norms. MG20, $\omega = 1.0$ weighted Jacobi point relaxation. Convergence is independent of h.

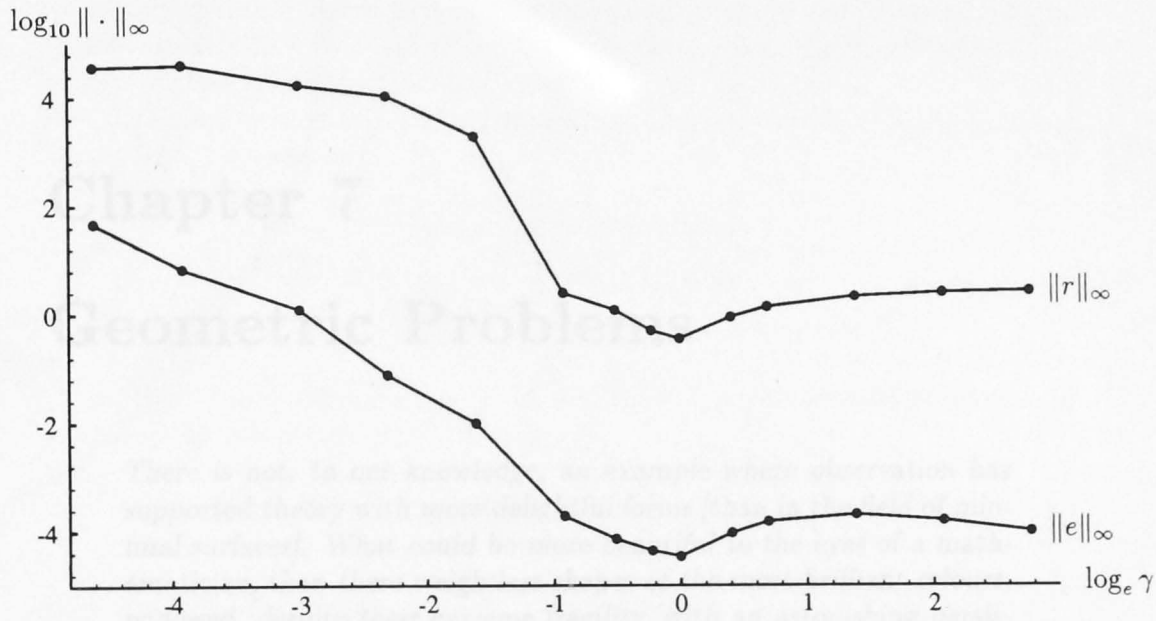


Figure 6.9: Convergence of $1 \times M_7^{2,2,2}$-cycle for various γ for Problem MG22 using $\omega = 1.0$ weighted Jacobi point relaxation. This multigrid iteration becomes ineffective for $\gamma \lesssim 0.1$.

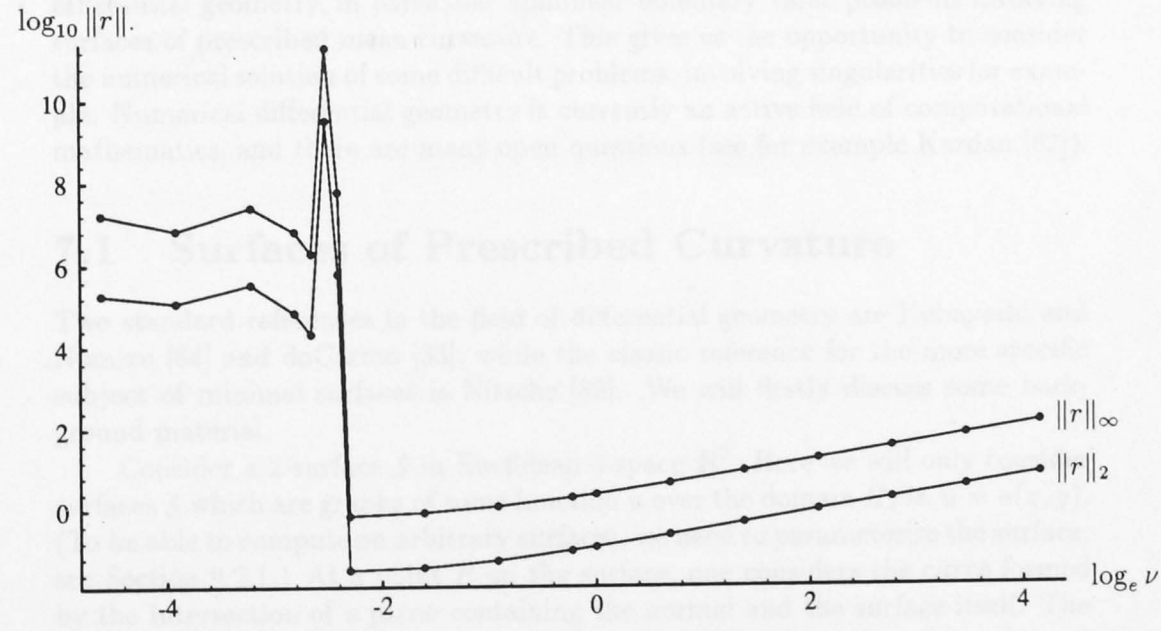


Figure 6.10: Convergence of $1 \times M_7^{2,2,2}$-cycle for various ν for Problem MG24 using $\omega = 0.9$ weighted Jacobi point relaxation. This multigrid iteration becomes unstable for $\nu \lesssim 0.1$.

Chapter 7

Geometric Problems

> *There is not, to our knowledge, an example where observation has supported theory with more delightful forms [than in the field of minimal surfaces]. What could be more beautiful to the eyes of a mathematician, than these weightless shapes of the most brilliant colours, endowed, despite their extreme fragility, with an astonishing persistence?*
>
> — G. Van der Mensbrugghe, as quoted in Nitsche [82]

In this chapter, we examine some interesting problems which arise in the field of differential geometry, in particular nonlinear boundary value problems involving surfaces of prescribed mean curvature. This gives us the opportunity to consider the numerical solution of some difficult problems, involving singularities for example. Numerical differential geometry is currently an active field of computational mathematics, and there are many open questions (see for example Kazdan [62]).

7.1 Surfaces of Prescribed Curvature

Two standard references in the field of differential geometry are Kobayashi and Nomizu [64] and doCarmo [33]; while the classic reference for the more specific subject of minimal surfaces is Nitsche [82]. We will firstly discuss some background material.

Consider a 2-surface $\mathcal{S}$ in Euclidean 3-space $\boldsymbol{R}^3$. Here we will only consider surfaces $\mathcal{S}$ which are graphs of some function u over the domain Ω, *ie.* $u = u(x, y)$. (To be able to compute on arbitrary surfaces, we need to parameterise the surface; see Section 9.2.1.) At a point P on the surface, one considers the curve formed by the intersection of a plane containing the normal and the surface itself. The *normal curvature* at P in the direction defined by the plane is then the reciprocal of the radius of the osculating circle, that is, the circle which matches the curve infinitesimally around P. One chooses either direction of the normal to fix the sign of the curvature. The principal curvatures k_1 and k_2 at P are defined to be the maximum and minimum of the normal curvatures at P, respectively (see [33]

for more details). These principal curvatures are always in directions orthogonal to each other, unless the normal curvature is the same in all directions (as on a sphere) — this is called an *umbilic point.*

There are various types of curvature; each is defined in terms of the principal curvatures. For example:

$$\begin{aligned} H &= k_1 + k_2 && \text{mean curvature} \\ K &= k_1\, k_2 && \text{Gauss curvature} \\ N &= k_1^{-1} + k_2^{-1} && \text{harmonic mean curvature.} \end{aligned}$$

(Note that some authors write $H = \frac{1}{n}\sum_{i=1}^{n} k_i$.) These lead to the following equations for prescribed mean curvature

$$\frac{u_{xx}(1+u_y{}^2) - 2u_x u_y u_{xy} + u_{yy}(1+u_x{}^2)}{(1+u_x{}^2+u_y{}^2)^{3/2}} = H(x,y), \tag{7.1}$$

prescribed Gauss curvature

$$\frac{u_{xx}u_{yy} - u_{xy}{}^2}{(1+u_x{}^2+u_y{}^2)^2} = K(x,y), \tag{7.2}$$

and prescribed harmonic mean curvature

$$\frac{[\,u_{xx}(1+u_y{}^2) - 2u_x u_y u_{xy} + u_{yy}(1+u_x{}^2)\,]\,(1+u_x{}^2+u_y{}^2)^{1/2}}{u_{xx}u_{yy} - u_{xy}{}^2} = N(x,y)$$

for a surface $u = u(x,y)$.

We mention in passing that a more compact expression for the prescribed mean curvature equation is

$$\operatorname{div}\left(\frac{\nabla u}{\sqrt{1+|\nabla u|^2}}\right) = H(x,y)$$

and similarly for the prescribed Gauss curvature equation:

$$\det D^2 u = K(x,y)\left(1+|Du|^2\right)^2.$$

We note that the left-hand side is the determinant of the Hessian of u, and therefore the equation is of the Monge-Ampère type. We also observe that $N = H/K$.

Let us investigate the ellipticity of these equations. The quantity $|\alpha^{ij}|$ (see Section 2.1) for the mean curvature equation is

$$(1+u_x{}^2+u_y{}^2)^{-3/2} \begin{vmatrix} 1+u_y{}^2 & -u_x u_y \\ -u_y u_x & 1+u_x{}^2 \end{vmatrix} = (1+u_x{}^2+u_y{}^2)^{-1/2}$$

which is evidently positive, irrespective of the nature of the solution u, hence equation (7.1) is elliptic everywhere. For the Gauss curvature equation, we have

$$|\alpha^{ij}| = (1 + {u_x}^2 + {u_y}^2)^{-2} \begin{vmatrix} u_{yy} & -u_{xy} \\ -u_{yx} & u_{xx} \end{vmatrix} = K(x,y)$$

and so equation (7.2) is elliptic if $K > 0$ in Ω.

Now let us consider the suitability of these equations for numerical solution using **MGLAB**. The standard isotropic discretisation for the residual of equation (7.1) is

$$\begin{aligned} r_{ij} = f_{ij} - N^2 \Big\{ & (\mathcal{L} - 2\mathcal{C} + \mathcal{R})\big[1 + \tfrac{N^2}{4}(\mathcal{U} - \mathcal{D})^2\big] - \tfrac{N^2}{8}(\mathcal{R} - \mathcal{L})(\mathcal{U} - \mathcal{D})\,\mathcal{X} \\ & + (\mathcal{D} - 2\mathcal{C} + \mathcal{U})\big[1 + \tfrac{N^2}{4}(\mathcal{R} - \mathcal{L})^2\big]\Big\} \\ & \Big\{1 + \tfrac{N^2}{4}\big[(\mathcal{R} - \mathcal{L})^2 + (\mathcal{U} - \mathcal{D})^2\big]\Big\}^{-3/2} \end{aligned}$$

where we have written $\mathcal{C}$, $\mathcal{L}$, $\mathcal{R}$, $\mathcal{U}$ and $\mathcal{D}$ for v^k_{ij}, $v^k_{i-1,j}$, $v^k_{i+1,j}$, $v^k_{i,j+1}$ and $v^k_{i,j-1}$ respectively, and where $\mathcal{X} = (v^k_{i-1,j-1} + v^k_{i+1,j+1} - v^k_{i-1,j+1} - v^k_{i+1,j-1})$, therefore the basic step of Newton iteration (see equation (5.4)) is

$$v^{k+1}_{ij} = \mathcal{C} - \frac{f_{ij}\Big\{1 + \frac{N^2}{4}\big[(\mathcal{R} - \mathcal{L})^2 + (\mathcal{U} - \mathcal{D})^2\big]\Big\}^{3/2} - N^2\,\Psi}{N^2\Big\{4 + \frac{N^2}{2}\big[(\mathcal{R} - \mathcal{L})^2 + (\mathcal{U} - \mathcal{D})^2\big]\Big\}},$$

where

$$\begin{aligned} \Psi = & (\mathcal{L} - 2\mathcal{C} + \mathcal{R})\big[1 + \tfrac{N^2}{4}(\mathcal{U} - \mathcal{D})^2\big] - \tfrac{N^2}{8}(\mathcal{R} - \mathcal{L})(\mathcal{U} - \mathcal{D})\,\mathcal{X} \\ & + (\mathcal{D} - 2\mathcal{C} + \mathcal{U})\big[1 + \tfrac{N^2}{4}(\mathcal{R} - \mathcal{L})^2\big]. \end{aligned}$$

We observe that $r''_{ij}(v_{ij}) = \partial^2 r_{ij}/\partial \mathcal{C}^2 = 0$, hence by equation (5.5), this Newton iteration scheme is guaranteed to converge.

On the other hand, the prescribed Gauss curvature equation leads to a discrete residual of

$$r_{ij} = f_{ij} - \frac{16(\mathcal{L} - 2\mathcal{C} + \mathcal{R})(\mathcal{D} - 2\mathcal{C} + \mathcal{U}) - \mathcal{X}^2}{\big[\frac{4}{N^2} + (\mathcal{R} - \mathcal{L})^2 + (\mathcal{U} - \mathcal{D})^2\big]^2}$$

and so the Newton convergence criterion in this case is

$$\left| \frac{16\,(\mathcal{L} - 2\mathcal{C} + \mathcal{R})(\mathcal{D} - 2\mathcal{C} + \mathcal{U}) + \mathcal{X}^2 - f_{ij}\big[\frac{4}{N^2} + (\mathcal{R} - \mathcal{L})^2 + (\mathcal{U} - \mathcal{D})^2\big]^2}{8\,(\mathcal{L} + \mathcal{R} + \mathcal{U} + \mathcal{D} - 4\mathcal{C})^2} \right| < 1.$$

There is essentially nothing that can be said *a priori* about this inequality. Moreover, the left-hand side is not well-behaved, as the denominator vanishes wherever the surface is locally flat. In any case, we find experimentally that this simple

approach to a relaxation scheme for Gauss curvature problems generally leads to divergent iterations. To be able to solve such problems, we would need to use other relaxation methods. It turns out that equations of the Monge-Ampère type are difficult to analyse and especially difficult to solve numerically. However, necessary conditions for the existence of unique solutions were found in 1983 [99]:

Theorem 7.1 (Trudinger and Urbas) *Let Ω be a uniformly convex domain in $\mathbf{R}^n$, and K a positive function in Ω. Then the classical Dirichlet problem*

$$\det D^2 u = K(x)\left(1+|Du|^2\right)^{\frac{n+2}{2}} \quad \text{in } \Omega; \qquad u = \phi \quad \text{on } \partial\Omega$$

has a unique convex solution for arbitrary ϕ iff

$$\int_\Omega K\,dx < \omega_n \qquad \text{and} \qquad K|_{\partial\Omega} = 0.$$

(The quantity ω_n is the volume of a unit ball of dimension n.) They go on to consider the Dirichlet problem for general Monge-Ampère equations:

$$\det D^2 u = f(x,u,Du) \quad \text{in } \Omega; \qquad u = \phi \quad \text{on } \partial\Omega.$$

In [65], Kuo and Trudinger prove the stability of a discretisation scheme of the Dirichlet problem for fully nonlinear uniformly elliptic second-order PDE's

$$f(x,u,Du,D^2u) = 0$$

in bounded domains $\Omega \in \mathbf{R}^n$.

This is as much as we shall say about the equations of prescribed Gauss and harmonic mean curvature. Let us therefore turn our attention back to the consideration of prescribed mean curvature, as we are able to investigate the numerical solution of such problems using the current version of `MGLAB`.

The existence theorem corresponding to Theorem 5.1 is due to Serrin [88] (part (a)) and to Giusti [45] (part (b)):

Theorem 7.2 (Serrin and Giusti) *Let Ω be a smooth bounded domain in $\mathbf{R}^n$, and H a smooth function in Ω.*
(a) Then the prescribed mean curvature Dirichlet problem

$$div\left(\frac{\nabla u}{\sqrt{1+|\nabla u|^2}}\right) = H(x,y) \quad \text{in } \Omega; \qquad u = \phi \quad \text{on } \partial\Omega$$

has a unique solution $u \in C^2(\Omega) \cap C^{0,1}(\overline{\Omega})$ for arbitrary $\phi \in C^2(\overline{\Omega})$ iff

$$\left|\int_\Omega H\,dx\right| < |\partial\Omega| \qquad \text{and} \qquad H(x) \le H_{n-1}(x) \qquad \forall \quad x \in \partial\Omega \tag{7.3}$$

where H_{n-1} is the $(n-1)$-dimensional mean curvature of $\partial\Omega$.
(b) The mean curvature equation has a solution in Ω iff $|\int_\Omega H\,dx| \le |\partial\Omega|$. In the extreme case of equality, the solution is unique up to a constant and is "vertical" on $\partial\Omega$.

This existence theorem provides us with a sharp test for the numerical solution of a parameterised boundary value problem (see Section 7.4).

7.2 Minimal Surfaces

Let us now consider the minimal surface problem, where we seek a surface whose graph $u = u(x, y)$ everywhere has zero mean curvature:

$$u_{xx}(1 + u_y{}^2) - 2u_x u_y u_{xy} + u_{yy}(1 + u_x{}^2) = 0. \tag{7.4}$$

This equation models the shape of a soap film within wire boundaries where the surface tension of the film is the dominant force, and hence the soap film forms a surface locally of least surface area. The problem of determining a minimal surface with some given closed space curve boundary is called Plateau's problem, after the Belgian mathematician who investigated it around 1873. In fact, the history of minimal surfaces began with Lagrange more than a century earlier [82].

There are various ways to approach the minimal surface problem: by using tools from the calculus of variations, PDE theory, and functional analysis. In 1937, Courant gave an existence proof by reducing Plateau's problem to Dirichlet's problem [82].

There are a number of families of classical solutions to the minimal surface equation; prototypes of some of these families are as follows:

$$\begin{aligned}
u(x,y) &= Ax + By + C && \text{(plane)} \\
u(x,y) &= \sqrt{\cosh^2 x - y^2} && \text{(horizontal catenoid)} \\
u(x,y) &= \cosh^{-1}\sqrt{x^2 + y^2} && \text{(vertical catenoid)} \\
u(x,y) &= \tan^{-1}(y/x) && \text{(helicoid)} \\
u(x,y) &= \log\left|\tfrac{\sin x}{\sin y}\right| && \text{(Scherk's surface)} \\
u(x,y) &= \sin^{-1}(\sinh x \sinh y) && \text{(Scherk's fifth surface)}
\end{aligned}$$

In addition, there are known solutions which are given implicitly, or in terms of inverses of elliptic integrals, or as power series solutions, for example Enneper's surface.

Four minimal surface problems were selected, all defined by equation (7.4) with the boundary conditions specified on the following page. Problems MG30 and MG32 have simple straight-line boundaries, yet the surface u in each case is defined by inverses of elliptic integrals. For MG30, it is found that (see Nitsche [82] section 5.2)

$$\mathcal{F}(y)\,\mathcal{F}(u(x,y)) + \mathcal{F}(u(x,y))\,\mathcal{F}(x) + \mathcal{F}(x)\,\mathcal{F}(y) = -1$$

where $\mathcal{F}$ is the inverse of the elliptic integral ξ given by

$$\xi(s) = \int_0^s \frac{2\,d\sigma}{\sqrt{3 + 10\sigma^2 + 3\sigma^4}}.$$

In the case of Problem MG32, the surface is defined by $\mathcal{E}(x)\,\mathcal{E}(y) = \mathcal{E}(u(x,y))$ where $\mathcal{E}$ is the inverse of ζ given by $\zeta(t) = \int_0^t (1 + \tau^2 + \tau^4)^{-1/2} d\tau$.

Problem MG30 ("batwing")

$$\begin{aligned} u(0,y) &= y & u(1,y) &= 1-y \\ u(x,0) &= x & u(x,1) &= 1-x \end{aligned}$$

u is unknown in closed form

u has ruled solutions $u = x = \frac{1}{2}$ and $u = y = \frac{1}{2}$

Problem MG32 ("step")

$$\begin{aligned} u(0,y) &= 0 & u(1,y) &= 1 \\ u(x,0) &= 0 & u(x,1) &= 1 \end{aligned}$$

u is unknown in closed form

u is singular at $x = 0;\ y = 1$ and $x = 1;\ y = 0$

u has ruled solutions $x = y;\ u = \frac{x+y}{2}$,

$x = 1 - y;\ u = \frac{1}{2}$ and $x = \frac{1}{2};\ u = y$

Problem MG34 (helicoid)

$$\begin{aligned} u(0,y) &= \tfrac{\pi}{2} & u(1,y) &= \tan^{-1} y \\ u(x,0) &= 0 & u(x,1) &= \tan^{-1} \tfrac{1}{x} \end{aligned}$$

$$u = \tan^{-1} \frac{y}{x}$$

u is singular at $x = 0;\ y = 0$

Problem MG36 (Scherk's surface)

$$\begin{aligned} u(0,y) &= \log\left[\tfrac{\sin\varepsilon}{\sin(y+\varepsilon)}\right] & u(1,y) &= \log\left[\tfrac{\sin(1+\varepsilon)}{\sin(y+\varepsilon)}\right] \\ u(x,0) &= \log\left[\tfrac{\sin(x+\varepsilon)}{\sin\varepsilon}\right] & u(x,1) &= \log\left[\tfrac{\sin(x+\varepsilon)}{\sin(1+\varepsilon)}\right] \end{aligned}$$

where $0 < \varepsilon < \pi - 1$

$$u = \log\left[\frac{\sin(x+\varepsilon)}{\sin(y+\varepsilon)}\right]$$

u is singular as $\varepsilon \to 0$

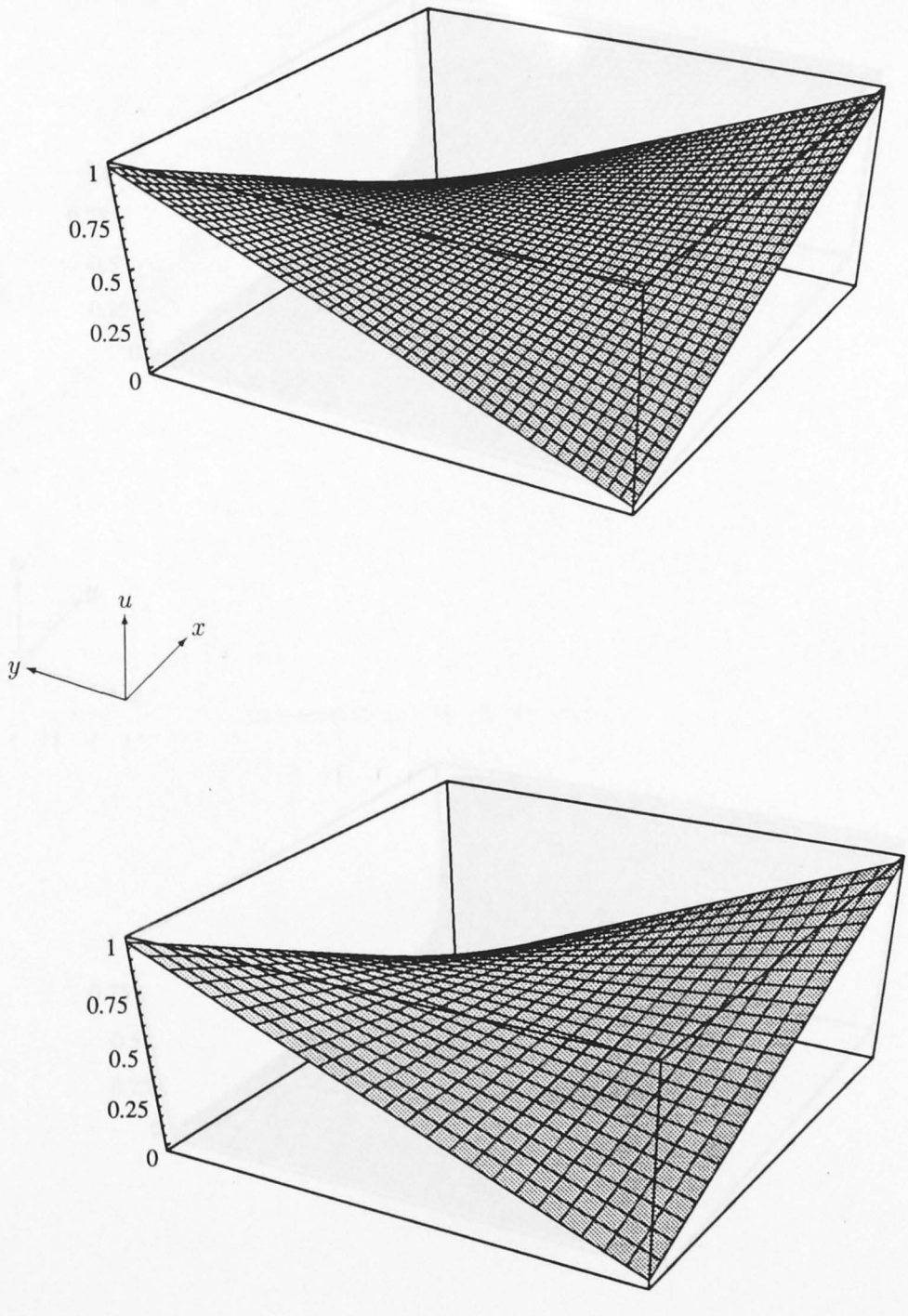


Figure 7.1: Numerical solution (top) and discrete-Laplacian initial guess (bottom) of Problem MG30.

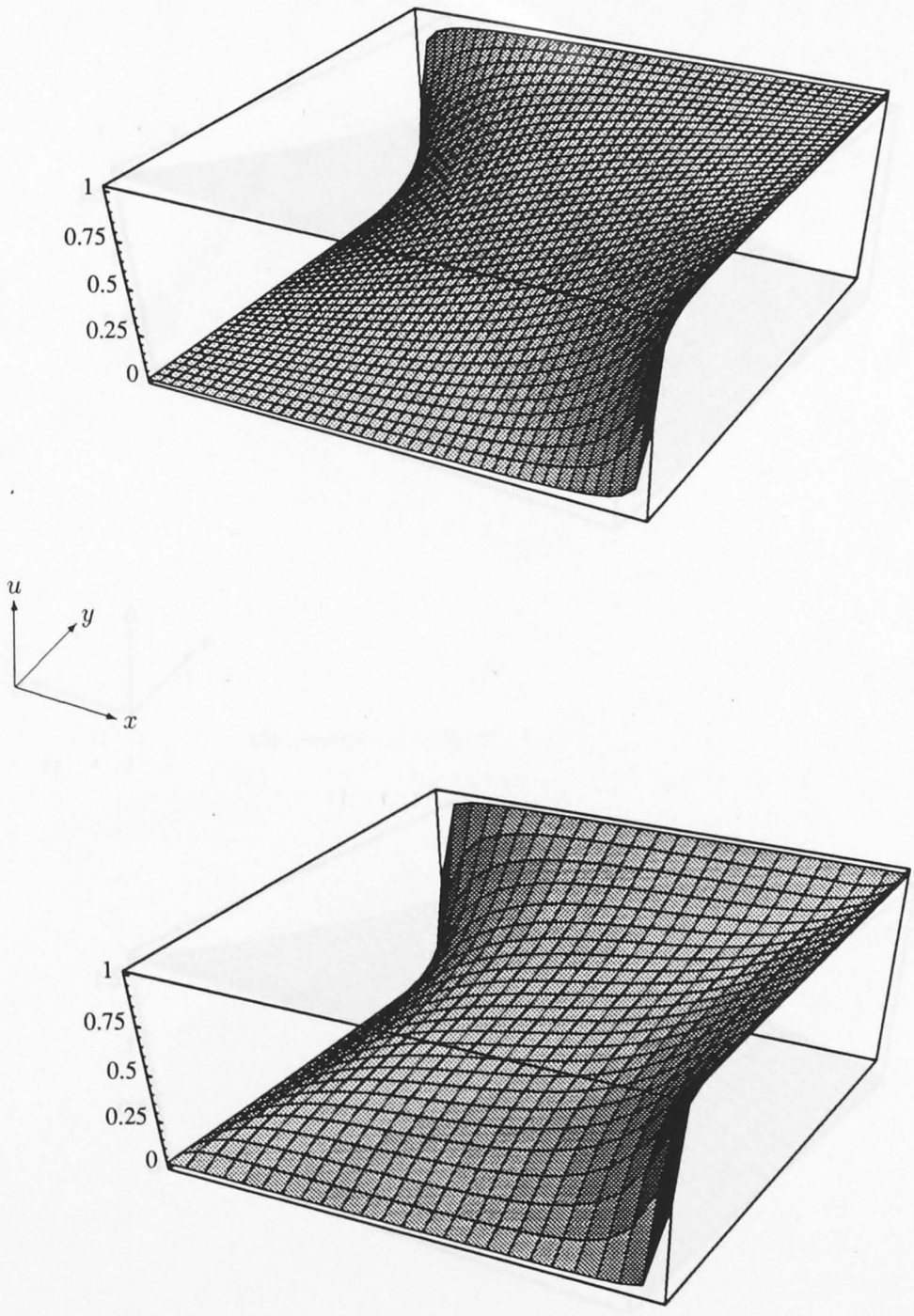


Figure 7.2: Numerical solution (top) and discrete-Laplacian initial guess (bottom) of Problem MG32.

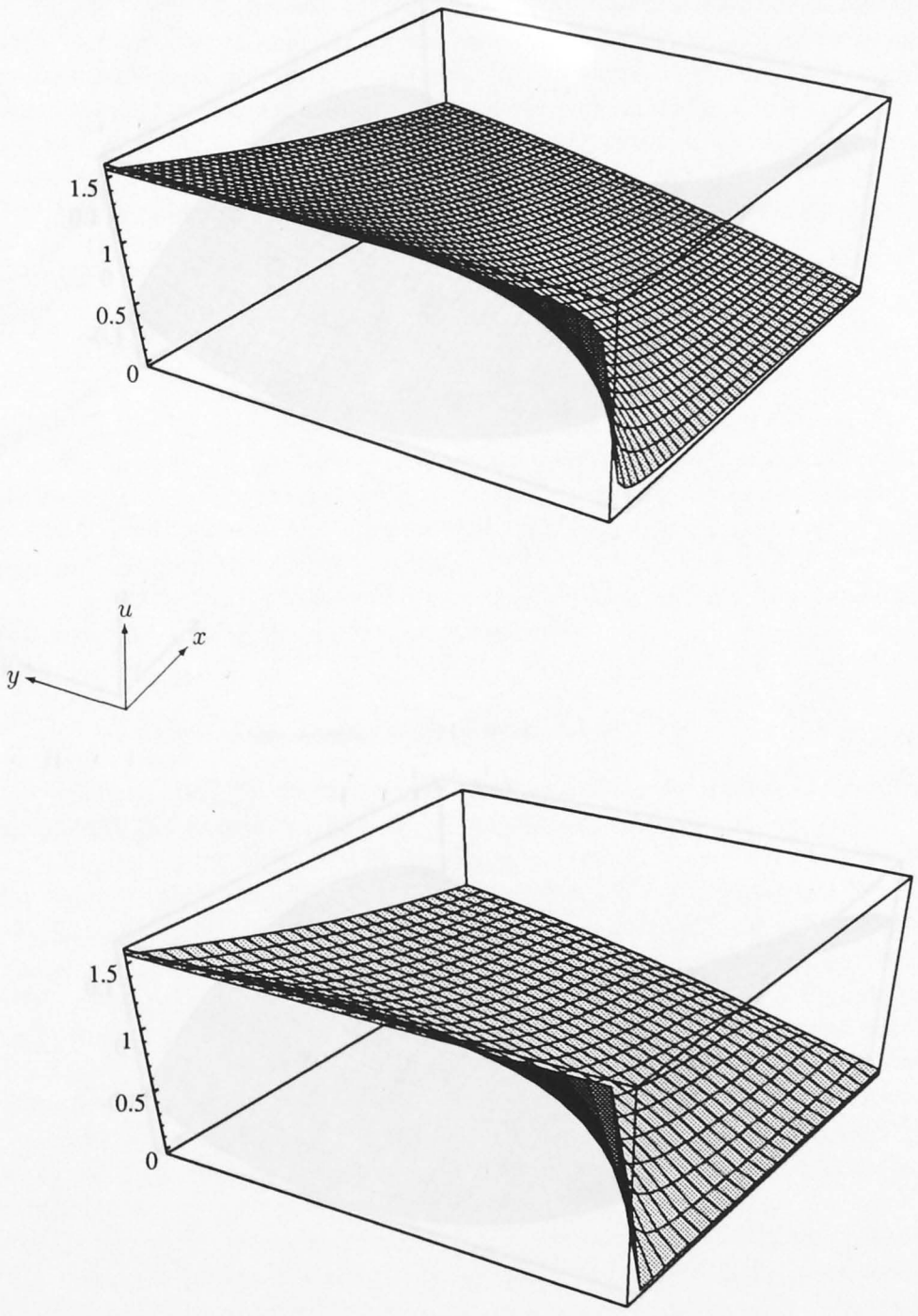


Figure 7.3: Numerical solution (top) and discrete-Laplacian initial guess (bottom) of Problem MG34.

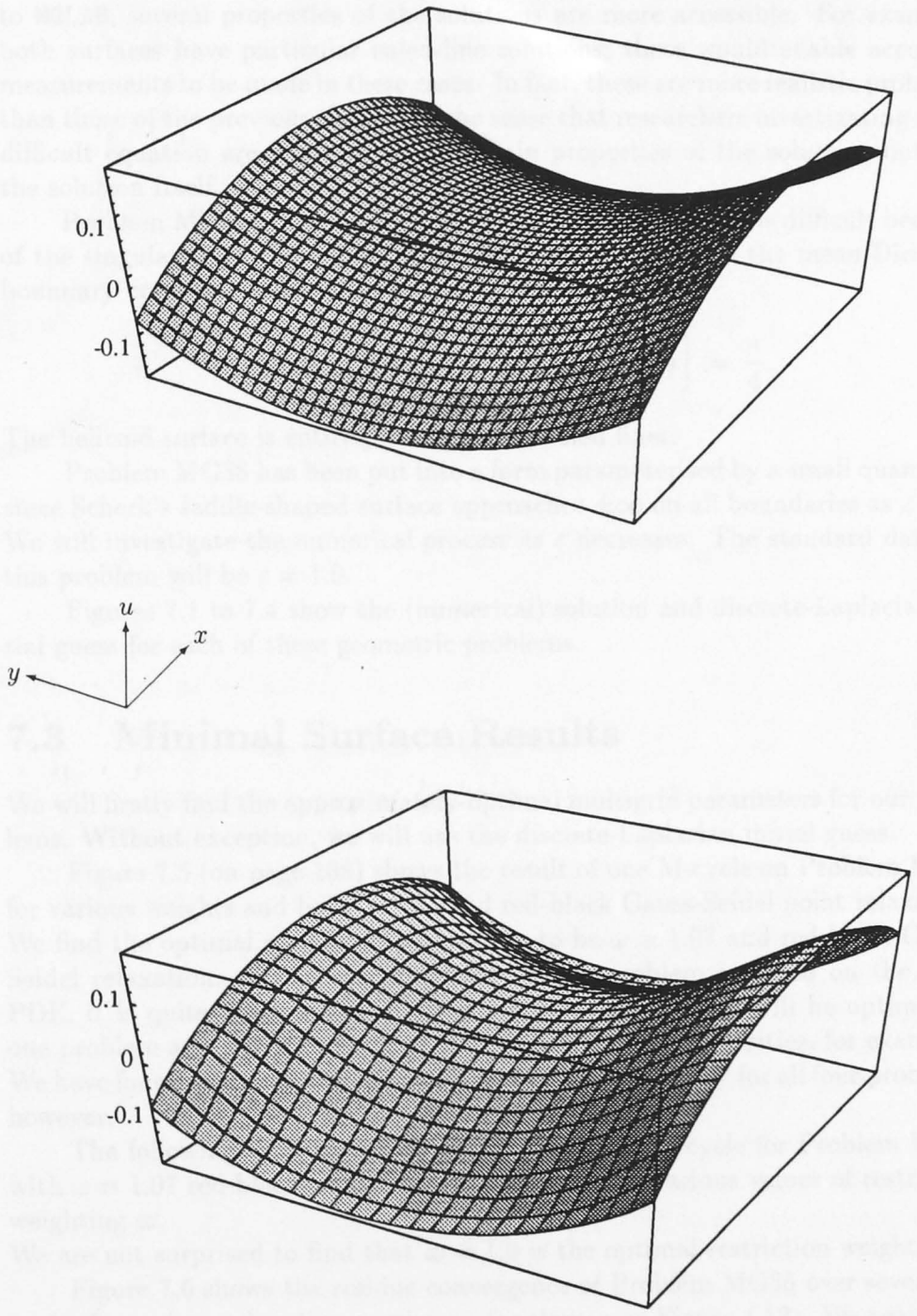


Figure 7.4: Numerical solution (top) and discrete-Laplacian initial guess (bottom) of Problem MG36 ($\varepsilon = 1.0$).

While the exact solutions of Problems MG30 and MG32 are unavailable to MGLAB, several properties of the solutions are more accessible. For example, both surfaces have particular ruled-line solutions; these would enable accuracy measurements to be made in these cases. In fact, these are more realistic problems than those of the previous chapter in the sense that researchers investigating some difficult equation are likely to know certain properties of the solution, but not the solution itself.

Problem MG34 involves the helicoid minimal surface and is difficult because of the singularity at the origin. Note that we have imposed the mean Dirichlet boundary condition at the origin

$$u(0,0) = \frac{1}{2}\left[\lim_{x\to 0} u(x,0) + \lim_{y\to 0} u(0,y)\right] = \frac{\pi}{4}.$$

The helicoid surface is entirely composed of ruled lines.

Problem MG36 has been put into a form parameterised by a small quantity ε since Scherk's saddle-shaped surface approaches $\pm\infty$ on all boundaries as $\varepsilon \to 0$. We will investigate the numerical process as ε decreases. The standard data for this problem will be $\varepsilon = 1.0$.

Figures 7.1 to 7.4 show the (numerical) solution and discrete-Laplacian initial guess for each of these geometric problems.

7.3 Minimal Surface Results

We will firstly find the approximately-optimal multigrid parameters for our problems. Without exception, we will use the discrete-Laplacian initial guess.

Figure 7.5 (on page 108) shows the result of one M-cycle on Problem MG36 for various weights and both Jacobi and red-black Gauss-Seidel point relaxation. We find the optimal relaxation parameters to be $\omega \simeq 1.07$ and red-black Gauss-Seidel relaxation. While each minimal surface problem is based on the same PDE, it is quite possible that certain multigrid parameters will be optimal for one problem and not another (due to the presence of singularities, for example). We have found these relaxation parameters to be satisfactory for all four problems, however.

The following table shows the result of $1 \times M_7^{2,2,2}$-cycle for Problem MG36 with $\omega = 1.07$ red-black Gauss-Seidel relaxation and various values of restriction weighting ϖ.

We are not surprised to find that $\varpi = 1.0$ is the optimal restriction weighting.

Figure 7.6 shows the residue convergence of Problem MG36 over several V-cycles for various relaxation numbers ν (analogous to Figure 4.12). We again find $\nu = (2,2,2)$ to be a satisfactory choice.

We therefore fix our multigrid parameters (for all minimal surface problems) to be

- isotropic grids $N_1 = M_1 = 2$
- discrete-Laplacian initial guess

ϖ	$\log_{10}\|\|e\|\|_2$	$\log_{10}\|\|e\|\|_\infty$	$\log_{10}\|\|r\|\|_2$	$\log_{10}\|\|r\|\|_\infty$
0.90	−5.92	−5.57	−3.22	−2.24
0.95	−6.40	−6.04	−3.40	−2.35
1.00	−7.33	−6.82	−3.51	−2.50
1.05	−6.59	−6.19	−3.43	−2.46
1.10	−6.44	−6.02	−3.30	−2.34

- two relaxation sweeps $\nu_1 = \nu_2 = \nu_0 = 2$
- $\omega = 1.07$ weighted red-black Gauss-Seidel point relaxation
- $\varpi = 1.0$ full-weighted restriction

Using these parameters results in a satisfactory convergence rate for Problems MG30 and MG36 using $1 \times M_8^{2,2,2}$-cycle:

	MG30		MG36			
L	$\log_{10}\|\|r\|\|_\infty$	$\hat{\rho}(\|\|r\|\|_\infty)$	$\log_{10}\|\|e\|\|_\infty$	$\hat{\rho}(\|\|e\|\|_\infty)$	$\log_{10}\|\|r\|\|_\infty$	$\hat{\rho}(\|\|r\|\|_\infty)$
1	$-\infty$	-	$-\infty$	-	$-\infty$	-
2	−4.88	-	−4.61	-	−4.31	-
3	−2.26	420.3	−4.94	0.468	−2.64	46.13
4	−1.99	1.885	−5.20	0.554	−2.53	1.308
5	−2.25	0.550	−5.73	0.299	−2.50	1.065
6	−2.55	0.500	−6.33	0.248	−2.50	1.008
7	−2.69	0.721	−6.82	0.325	−2.50	0.991
8	−2.76	0.846	−7.35	0.294	−2.50	0.992

Figure 7.7 compares the convergence of Problem MG32 for one M-cycle and twelve V-cycles. We find that a single full multigrid cycle does not perform as well as usual, because errors are not reduced sufficiently (*ie.* to the magnitude of discretisation errors) on each level. We find that we require very large values of ν to ensure v is sufficiently smooth on each level. It is interesting to see that $1 \times M^{18,18,18}$-cycle still achieves a better result than $12 \times V^{2,2,2}$-cycles. We may compare this with the results from well-behaved problems (see Figure 4.3 for example), where one M-cycle is much more efficient than many V-cycles. An alternative strategy in cases such as Problem MG32 would be to perform one M-cycle followed by several V- or W-cycles.

Problem MG34 yields almost identical results to those shown in Figure 7.7; we therefore conjecture that it is the singularities in Problems MG32 and MG34 which cause them to behave in this way. Figures 7.2 and 7.3 shows how the numerical solution "pulls away" from the infinitely-steep points of u.

Finally, Figure 7.8 shows the result of one M-cycle for Problem MG36 for various ε. As expected, the accuracy of the solution diminishes as $\varepsilon \to 0$, in this case it does so smoothly.

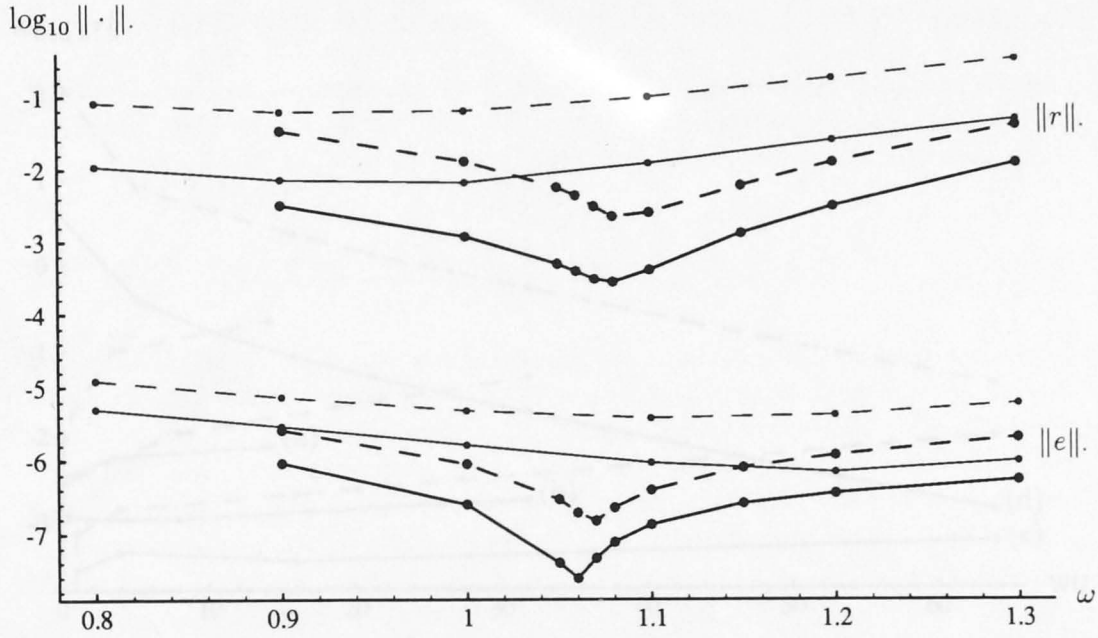


Figure 7.5: Result of $1 \times M_7^{2,2,2}$-cycle on Problem MG36 for Jacobi (thin lines) and red-black Gauss-Seidel (thick lines) relaxation of various weights. Solid lines are two-norms, dashed lines are infinity-norms.

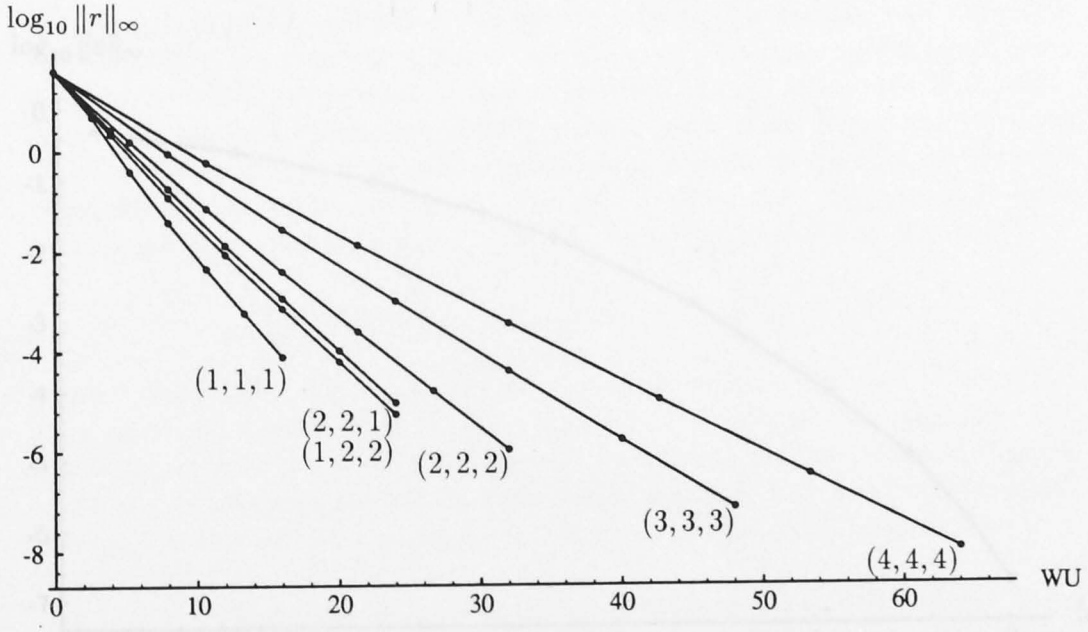


Figure 7.6: Convergence of $6 \times V_7^{\nu_1,\nu_0,\nu_2}$-cycles for various combinations of (ν_1, ν_0, ν_2). MG36, $\omega = 1.07$ weighted Jacobi point relaxation.

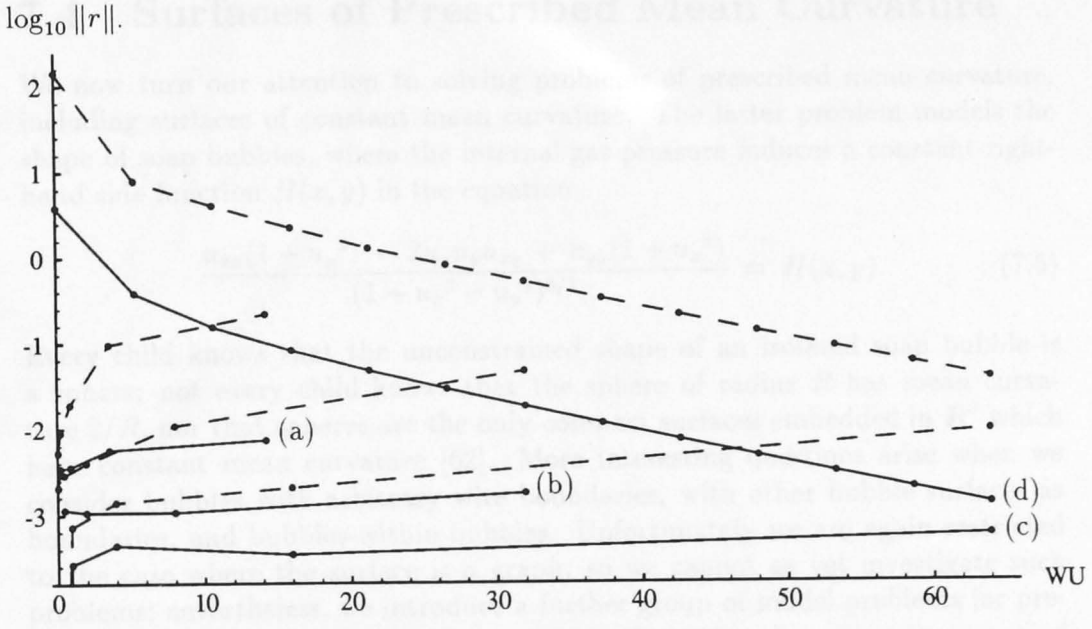


Figure 7.7: Convergence of (a) $1 \times M_7^{4,4,4}$-cycle, (b) $1 \times M_7^{9,9,9}$-cycle, (c) $1 \times M_7^{18,18,18}$-cycle, and (d) $12 \times V_7^{2,2,2}$-cycles for Problem MG32. Solid lines are two-norms, dashed lines are infinity-norms.

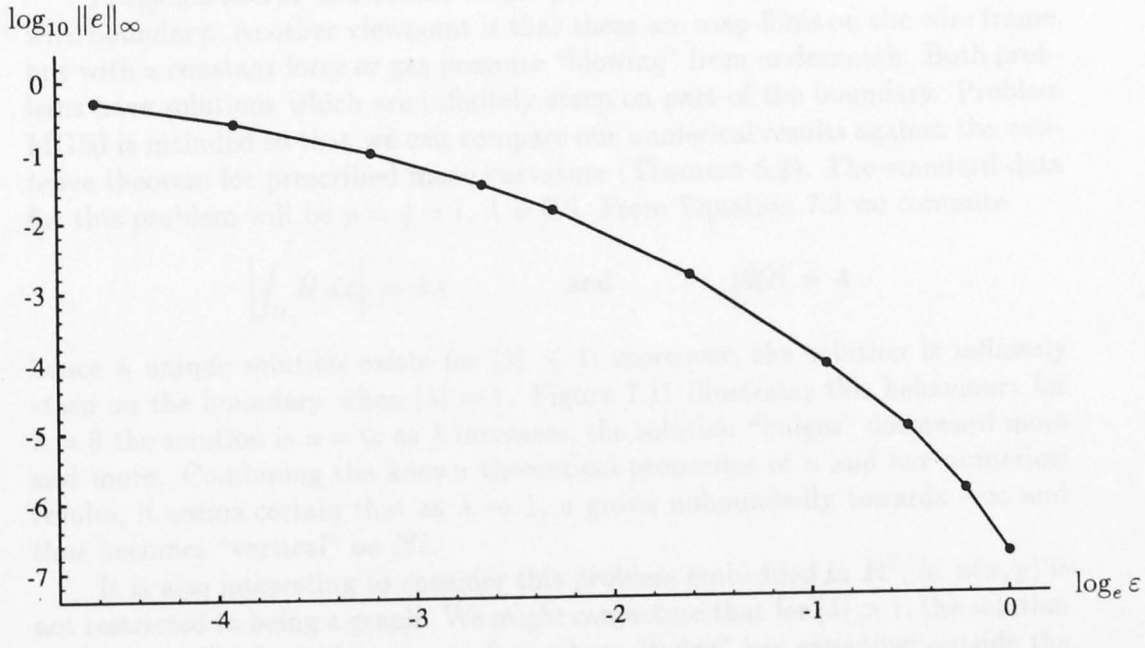


Figure 7.8: Convergence of $1 \times M_7^{2,2,2}$-cycle for various ε for Problem MG36.

7.4 Surfaces of Prescribed Mean Curvature

We now turn our attention to solving problems of prescribed mean curvature, including surfaces of constant mean curvature. The latter problem models the shape of soap bubbles, where the internal gas pressure induces a constant right-hand side function $H(x,y)$ in the equation

$$\frac{u_{xx}(1+u_y{}^2) - 2u_x u_y u_{xy} + u_{yy}(1+u_x{}^2)}{(1+u_x{}^2+u_y{}^2)^{3/2}} = H(x,y). \qquad (7.5)$$

Every child knows that the unconstrained shape of an isolated soap bubble is a sphere; not every child knows that the sphere of radius R has mean curvature $2/R$, nor that spheres are the only compact surfaces embedded in $\boldsymbol{R}^3$ which have constant mean curvature [62]. More interesting questions arise when we consider bubbles with arbitrary wire boundaries, with other bubble surfaces as boundaries, and bubbles-within-bubbles. Unfortunately we are again restricted to the case where the surface is a graph, so we cannot as yet investigate such problems; nevertheless, we introduce a further group of model problems for prescribed mean curvature to test the versatility of `MGLAB` and to compare numerical results with the existence theorems already discussed. These BVP's are based on equation (7.5) with the prescribed curvatures $H(x,y)$ and boundary conditions specified on the following page.

Figures 7.9 and 7.10 show the numerical solution and discrete-Laplacian initial guess for Problems MG40 and MG42. Figure 7.11 shows the numerical solution for Problem MG50 for three representative values of λ.

Problems MG40 and MG42 model part of a bubble surface with the given wire boundary. Another viewpoint is that these are soap films on the wire frame, but with a constant force or gas pressure "blowing" from underneath. Both problems have solutions which are infinitely steep on part of the boundary. Problem MG50 is included so that we can compare our numerical results against the existence theorem for prescribed mean curvature (Theorem 5.2). The standard data for this problem will be $p = q = 1$, $\lambda = 0.5$. From Equation 7.3 we compute

$$\left| \int_\Omega H \, dx \right| = 4\lambda \qquad \text{and} \qquad |\partial\Omega| = 4$$

hence a unique solution exists for $|\lambda| \leq 1$; moreover, the solution is infinitely steep on the boundary when $|\lambda| = 1$. Figure 7.11 illustrates this behaviour: for $\lambda = 0$ the solution is $u = 0$; as λ increases, the solution "bulges" downward more and more. Combining the known theoretical properties of u and our numerical results, it seems certain that as $\lambda \to 1$, u grows unboundedly towards $-\infty$ and thus becomes "vertical" on $\partial\Omega$.

It is also interesting to consider this problem embedded in $\boldsymbol{R}^3$, *ie.* $u(x,y)$ is not restricted to being a graph. We might conjecture that for $|\lambda| > 1$, the solution to the generalised problem is a surface whose "bulge" has expanded outside the unit square, so that it has roughly the shape of a sphere of diameter $D > 1$ with a square-shaped hole at the top/bottom.

Problem MG40 (quarter cylinder)

$$H(x,y) = 1$$

$$u(0,y) = \sqrt{y(2-y)} \qquad u(1,y) = \sqrt{y(2-y)}$$
$$u(x,0) = 0 \qquad u(x,1) = 1$$

$$u = \sqrt{y(2-y)}$$

u is infinitely steep at $y = 0$

Problem MG42 (cut hemisphere)

$$H(x,y) = 2\sqrt{2}$$

$$u(0,y) = \sqrt{y(1-y)} \qquad u(1,y) = \sqrt{y(1-y)}$$
$$u(x,0) = \sqrt{x(1-x)} \qquad u(x,1) = \sqrt{x(1-x)}$$

$$u = \sqrt{x(1-x)+y(1-y)}$$

u is infinitely steep at the four corners $(x,y) = \{(0,0),(0,1),(1,0),(1,1)\}$

Problem MG50 (prescribed mean curvature)

$$H(x,y) = \lambda pq\pi^2 \sin(p\pi x)\sin(q\pi y) \qquad p,q \in \mathbb{Z}^{\text{odd}}; \quad \lambda \in \boldsymbol{R}$$

$$u(0,y) = 0 \qquad u(1,y) = 0$$
$$u(x,0) = 0 \qquad u(x,1) = 0$$

u is unknown

u exists and is unique when $|\lambda| \leq 1$

u is infinitely steep on $\partial\Omega$ when $|\lambda| = 1$

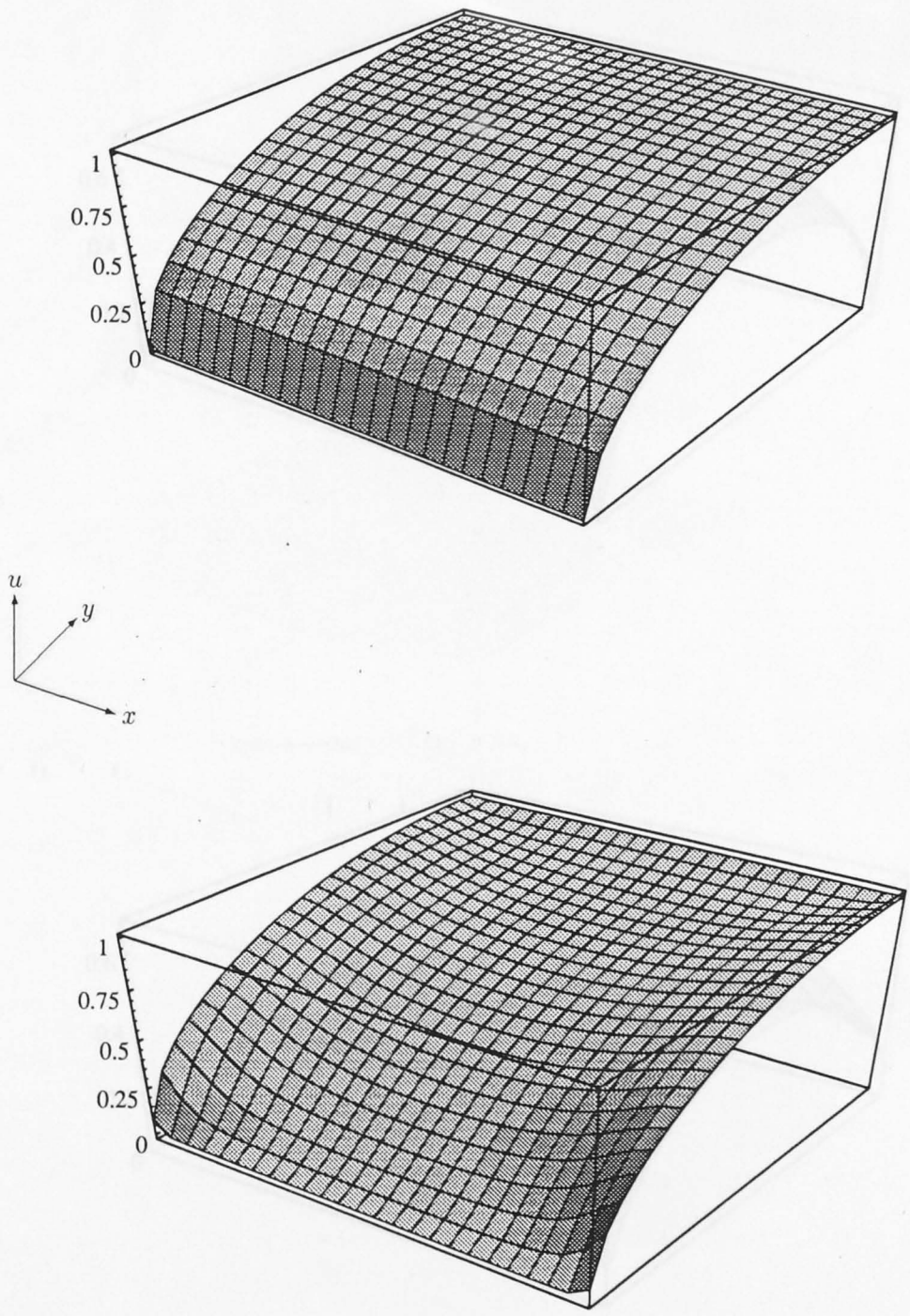


Figure 7.9: Solution (top) and discrete-Laplacian initial guess (bottom) for Problem MG40.

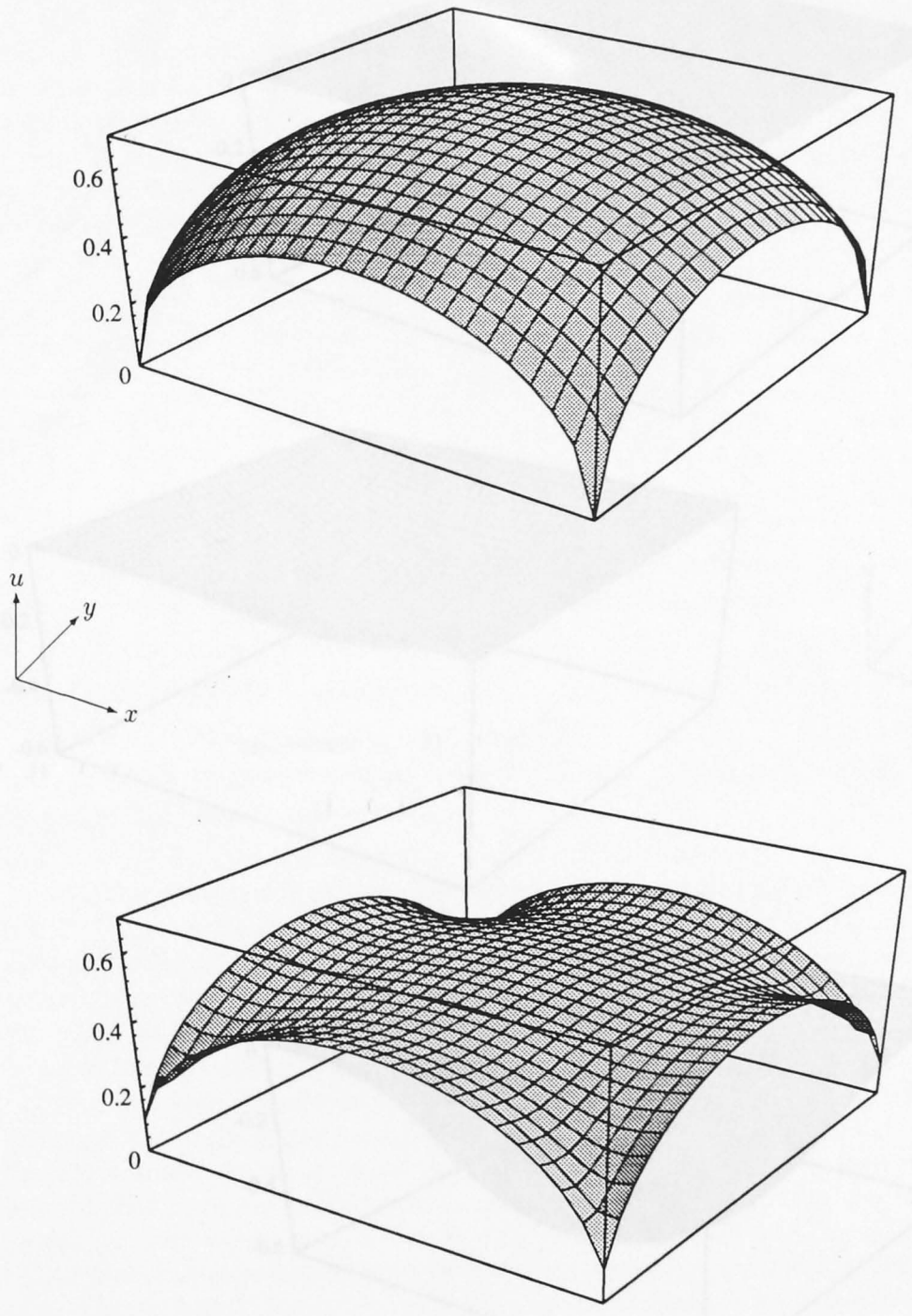


Figure 7.10: Solution (top) and discrete-Laplacian initial guess (bottom) for Problem MG42.

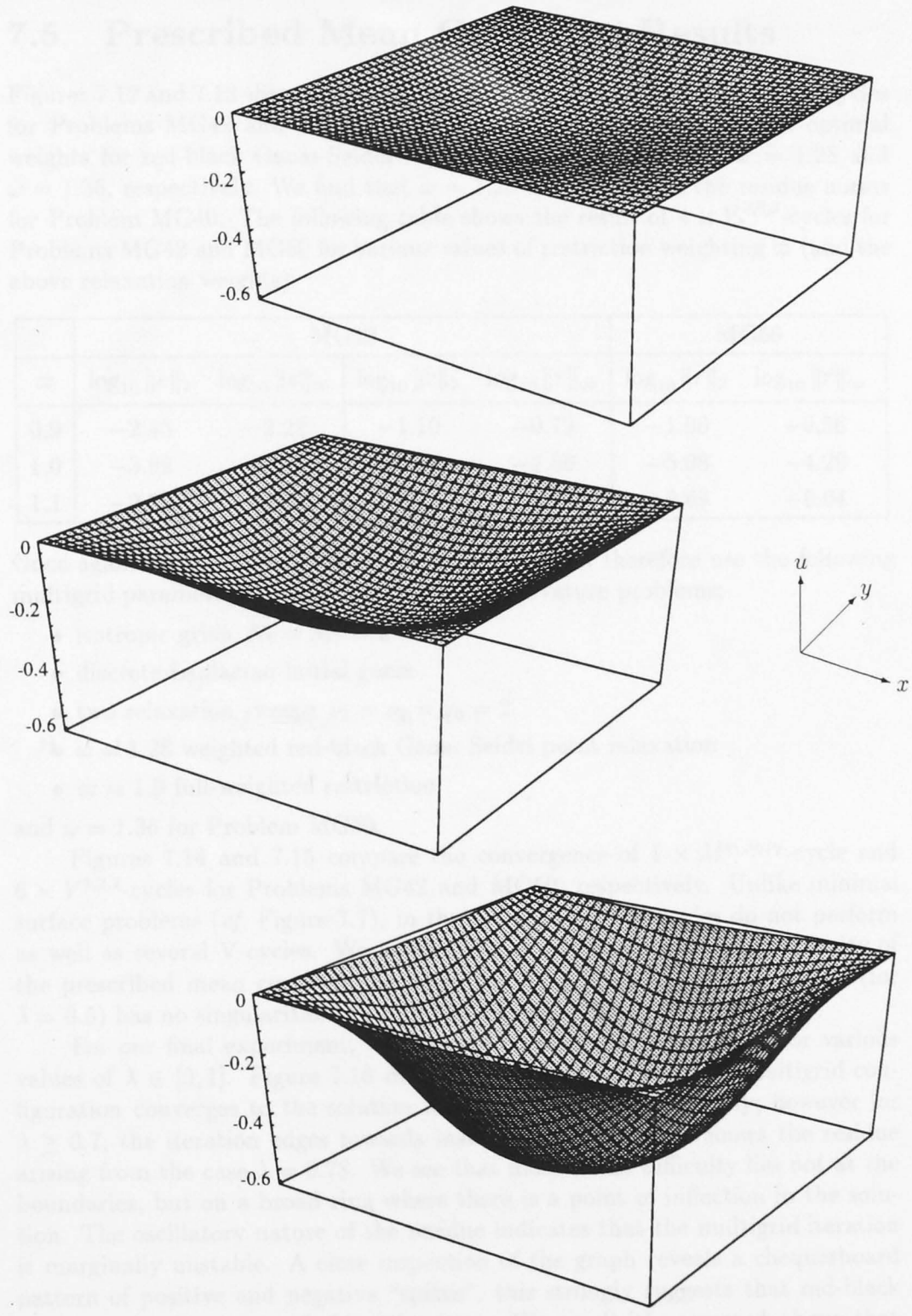


Figure 7.11: Numerical solution of Problem MG50 for $p = q = 1$ and $\lambda = 0.2$ (top), $\lambda = 0.5$ (centre), $\lambda = 0.7$ (bottom).

7.5 Prescribed Mean Curvature Results

Figures 7.12 and 7.13 show the result of ω-weighted relaxation for several V-cycles for Problems MG42 and MG50, respectively. Our conclusion is that optimal weights for red-black Gauss-Seidel relaxation are approximately $\omega = 1.28$ and $\omega = 1.36$, respectively. We find that $\omega = 1.28$ also minimises the residue norms for Problem MG40. The following table shows the result of $4 \times V_7^{2,2,2}$-cycles for Problems MG42 and MG50 for various values of restriction weighting ϖ (and the above relaxation weights).

	MG42				MG50	
ϖ	$\log_{10}\|\|e\|\|_2$	$\log_{10}\|\|e\|\|_\infty$	$\log_{10}\|\|r\|\|_2$	$\log_{10}\|\|r\|\|_\infty$	$\log_{10}\|\|r\|\|_2$	$\log_{10}\|\|r\|\|_\infty$
0.9	−2.43	−2.22	−1.10	−0.79	−1.08	−0.56
1.0	−3.92	−2.23	−3.93	−2.80	−5.08	−4.29
1.1	−2.27	−2.02	−0.85	−0.42	−0.69	−0.04

Once again we find $\varpi = 1.0$ to be optimal. We will therefore use the following multigrid parameters for our constant mean curvature problems:

- isotropic grids $N_1 = M_1 = 2$
- discrete-Laplacian initial guess
- two relaxation sweeps $\nu_1 = \nu_2 = \nu_0 = 2$
- $\omega = 1.28$ weighted red-black Gauss-Seidel point relaxation
- $\varpi = 1.0$ full-weighted restriction

and $\omega = 1.36$ for Problem MG50.

Figures 7.14 and 7.15 compare the convergence of $1 \times M^{\nu_1,\nu_0\nu_2}$-cycle and $6 \times V^{2,2,2}$-cycles for Problems MG42 and MG50, respectively. Unlike minimal surface problems (*cf.* Figure 7.7), in these cases single M-cycles do not perform as well as several V-cycles. We believe this is due merely to the complexity of the prescribed mean curvature equation for $H \neq 0$, since Problem MG50 (for $\lambda = 0.5$) has no singularities or other obvious difficulties.

For our final experiment, we attempt to solve Problem MG50 for various values of $\lambda \in [0, 1]$. Figure 7.16 demonstrates that the specified multigrid configuration converges to the solution for $\lambda \in [0, 0.7]$ approximately; however for $\lambda \gtrsim 0.7$, the iteration edges towards instability. Figure 7.17 shows the residue arising from the case $\lambda = 0.78$. We see that most of the difficulty lies not at the boundaries, but on a broad ring where there is a point of inflection in the solution. The oscillatory nature of the residue indicates that the multigrid iteration is marginally unstable. A close inspection of the graph reveals a chequerboard pattern of positive and negative "spikes"; this strongly suggests that red-black relaxation is the cause of this behaviour. We implicitly assumed above that ω_{opt} did not vary with λ; in fact, in the $\lambda \simeq 0.8$ regime, we now find $\omega = 0.8$ weighted Jacobi relaxation to be approximately optimal, but only slightly superior to red-black Gauss-Seidel (see Figure 7.16). Figures 7.18 and 7.19 show the

approximate solution v and residue r after $6 \times V_7^{5,5,5}$-cycles of $\omega = 0.8$ weighted Jacobi relaxation for $\lambda = 0.8$. The Jacobi residue has the same "ripple" pattern as the Gauss-Seidel residue; while it is somewhat smoother, we can see instabilities (high-frequency errors) arising around the edge of the ring. All this goes to show that the breakdown of our multigrid iteration is due to an unstable relaxation method for $\lambda \gtrsim 0.8$.

The instability we have found has the same characteristics as the instability which arises when attempting to solve the heat equation $u_t = \Delta u$ with too large a time-step. In that case also, the instability germinates at the inflection points.

We have discussed this problem in considerable detail because we feel it demonstrates the synergistic way in which numerical and theoretical mathematics can assist each other in the investigation of many interesting as yet unsolved problems.

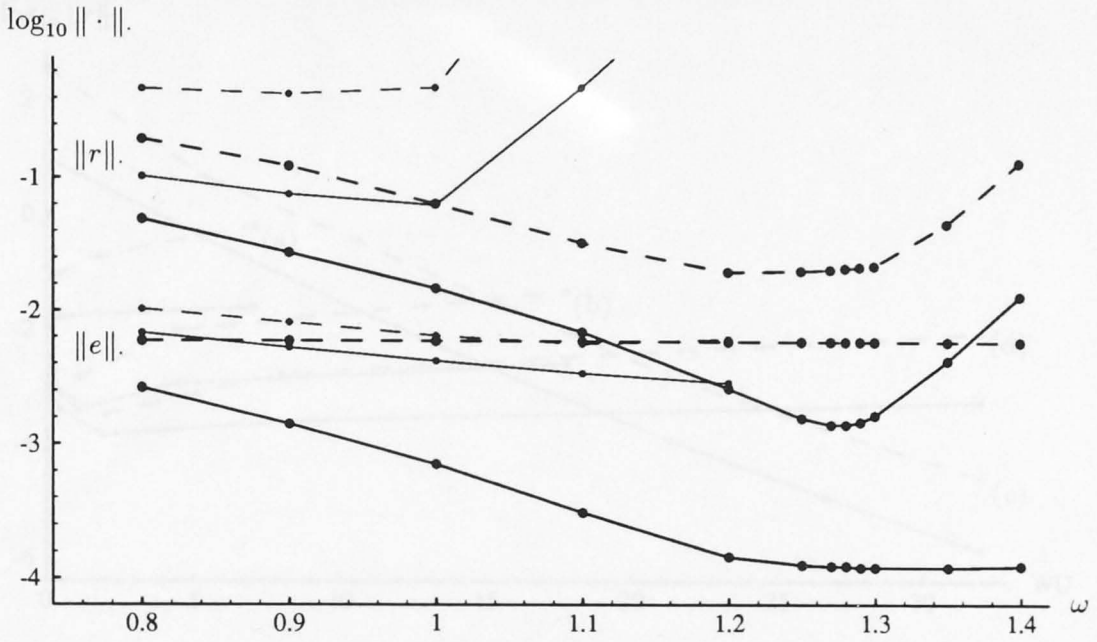


Figure 7.12: Result of $3 \times V_7^{2,2,2}$-cycles on Problem MG42 for Jacobi (thin lines) and red-black Gauss-Seidel (thick lines) relaxation of various weights. Solid lines are two-norms, dashed lines are infinity-norms.

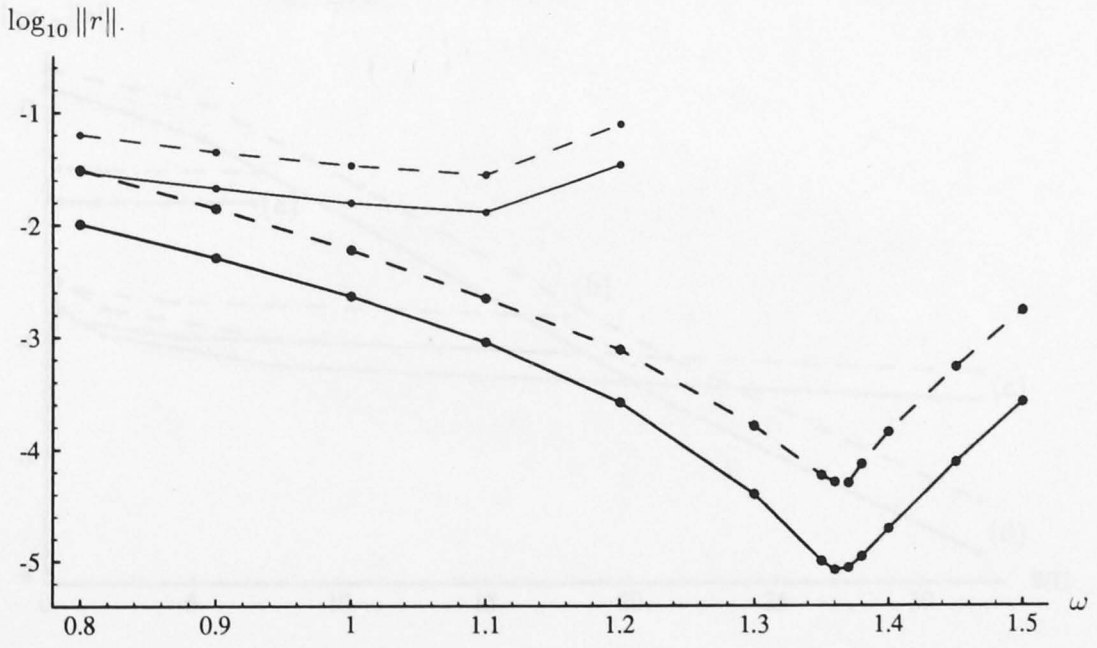


Figure 7.13: Result of $4 \times V_7^{2,2,2}$-cycles on Problem MG50 for Jacobi (thin lines) and red-black Gauss-Seidel (thick lines) relaxation of various weights. Solid lines are two-norms, dashed lines are infinity-norms.

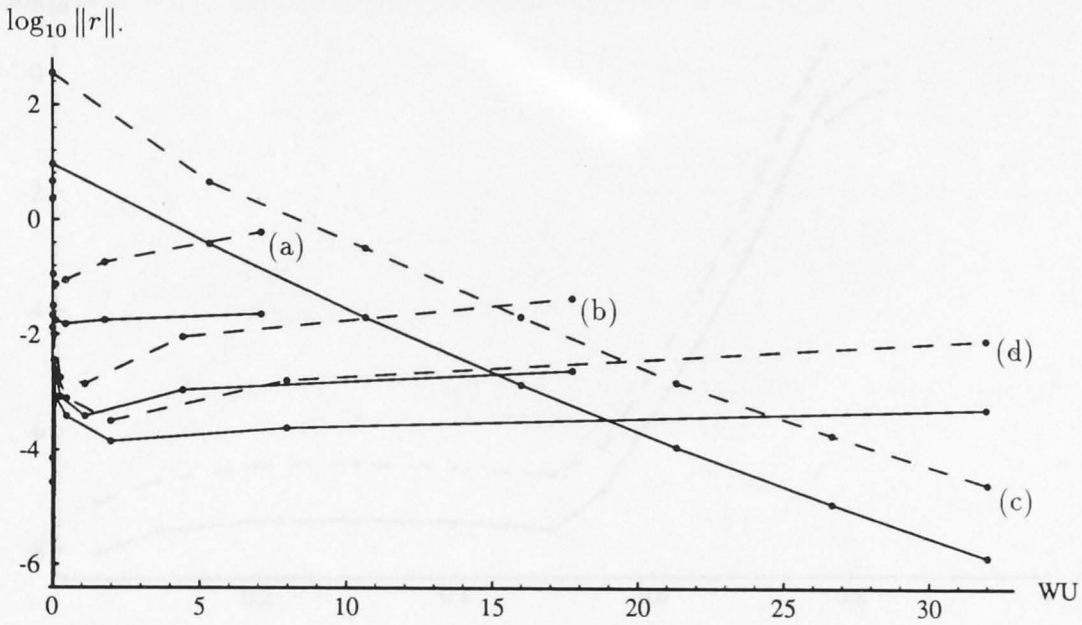


Figure 7.14: Convergence of (a) $1 \times M_7^{2,2,2}$-cycle, (b) $1 \times M_7^{5,5,5}$-cycle, (c) $1 \times M_7^{9,9,9}$-cycle, and (d) $6 \times V_7^{2,2,2}$-cycles for Problem MG42. Solid lines are two-norms, dashed lines are infinity-norms.

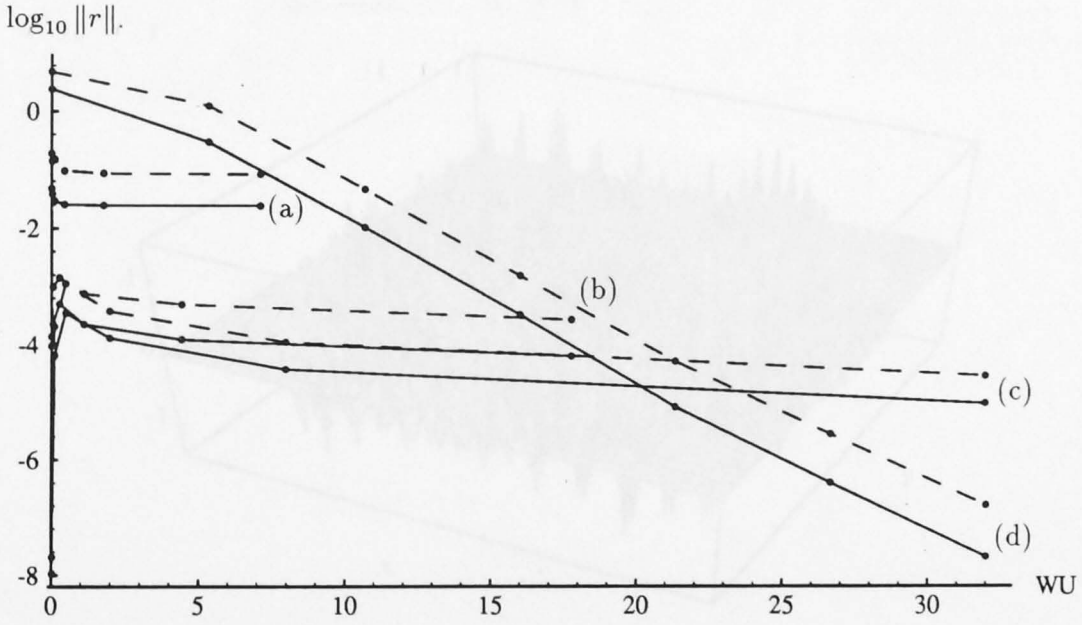


Figure 7.15: Convergence of (a) $1 \times M_7^{2,2,2}$-cycle, (b) $1 \times M_7^{5,5,5}$-cycle, (c) $1 \times M_7^{9,9,9}$-cycle, and (d) $6 \times V_7^{2,2,2}$-cycles for Problem MG50. Solid lines are two-norms, dashed lines are infinity-norms.

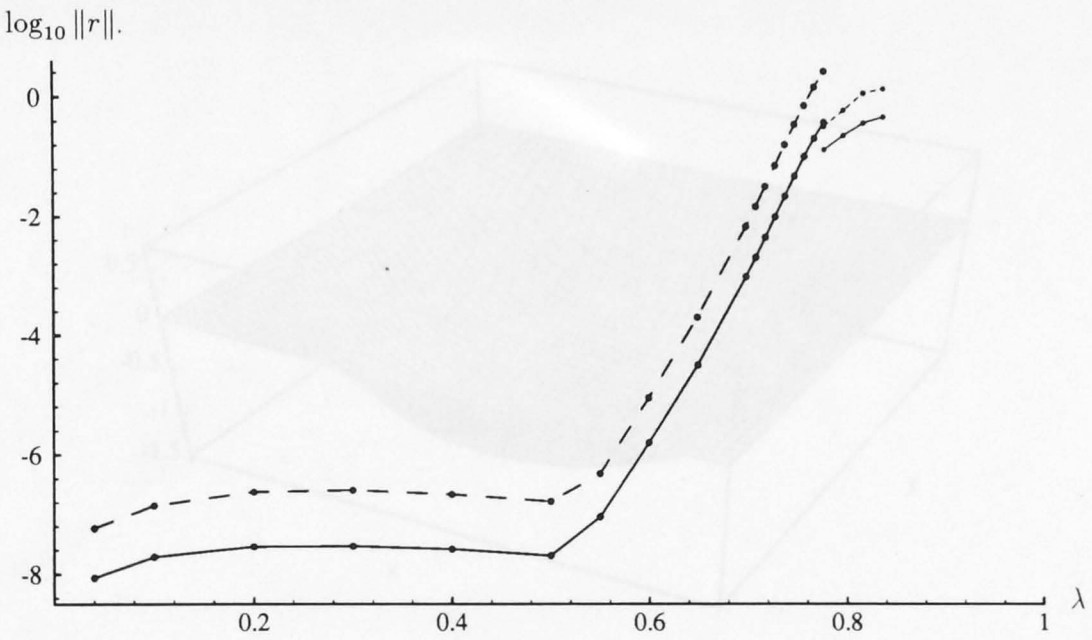


Figure 7.16: Convergence of $6 \times V_7^{2,2,2}$-cycles for Problem MG50 for $\omega = 1.36$ weighted red-black Gauss-Seidel relaxation for various λ. Also shown is $6 \times V_7^{5,5,5}$-cycles of $\omega = 0.8$ weighted Jacobi relaxation (thin lines, see text for details). Solid lines are two-norms, dashed lines are infinity-norms.

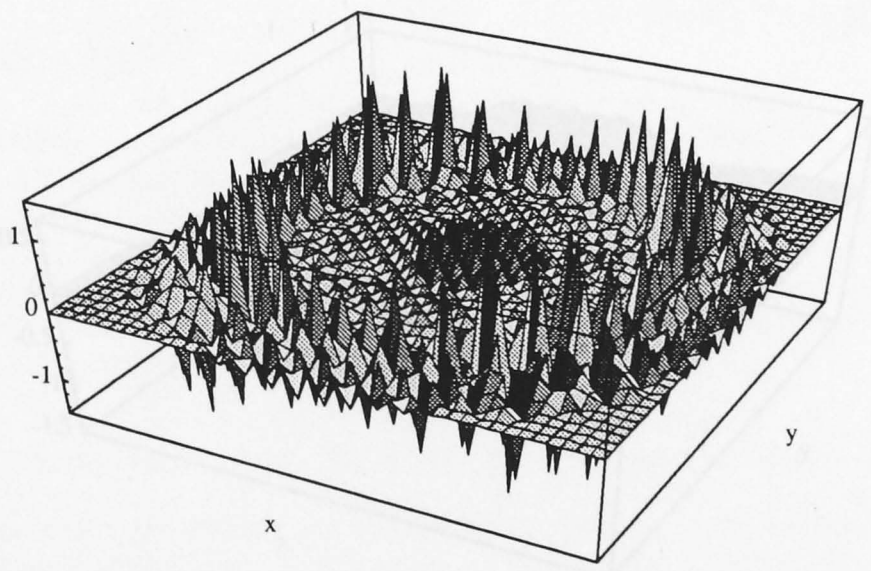


Figure 7.17: Residue r after $6 \times V_7^{2,2,2}$-cycles of $\omega = 1.36$ weighted red-black Gauss-Seidel point relaxation on Problem MG50 with $\lambda = 0.78$.

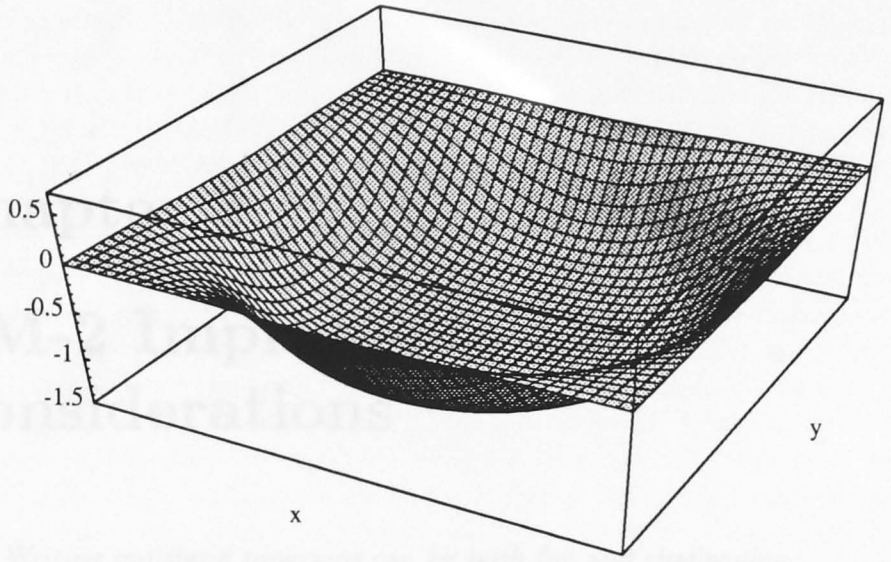


Figure 7.18: Approximate solution v after $6 \times V_7^{5,5,5}$-cycles of $\omega = 0.8$ weighted Jacobi point relaxation on Problem MG50 with $\lambda = 0.8$.

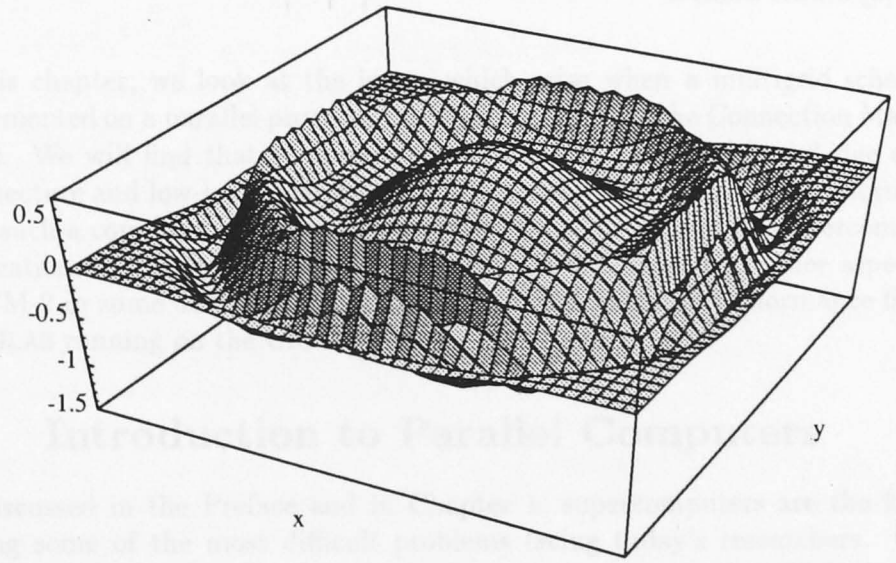


Figure 7.19: Residue r after $6 \times V_7^{5,5,5}$-cycles of $\omega = 0.8$ weighted Jacobi point relaxation on Problem MG50 with $\lambda = 0.8$.

Chapter 8

CM-2 Implementation Considerations

Writing multigrid programs can be both fun and challenging.

— William Briggs [20]

I still have my first computer program. It factored numbers into primes. You would dial a ten-digit number into the console, and it would punch the factors on cards. The program initially was about 70 instructions long, and as I recall, by the time I finished it I had removed more than 100 errors out of those 70 lines of code.

— Donald Knuth [3]

In this chapter, we look at the issues which arise when a multigrid scheme is implemented on a parallel processing system, in this case the Connection Machine CM-2. We will find that the programmer requires a detailed knowledge of the architecture and low-level operations in order to extract the optimal performance from such a computer — a statement which is true in general for supercomputer applications; accordingly, we will consider the architecture and other aspects of the CM-2 in some detail. We will also present and interpret performance figures for `MGLAB` running on the CM-2.

8.1 Introduction to Parallel Computers

As discussed in the Preface and in Chapter 1, supercomputers are the key to solving some of the most difficult problems facing today's researchers. Often such problems are simulations of some physical phenomenon for which there are a large number of degrees of freedom: most likely three spatial and one temporal dimension, together with a possibly high-dimensional parameter space. In many of these cases, such as aerodynamic design and oil reservoir simulation, advances in research are limited by available computer resources rather than by lack of understanding of the underlying physics.

The advanced scientific computation community is eagerly awaiting the development of a teraflop machine (one capable of 10^{12} floating-point operations per second), a goal which will soon be realised. It is expected that this development will foster major advances in many areas of research, as it corresponds to roughly an order of magnitude increase in spatial resolution for three-dimensional simulations.

Supercomputers may be defined as the class of computers which, at any given point in history, perform extremely fast computation (relative to the mainstream). Conventional supercomputers, such as the Cray series, are essentially based on serial technology, and suffer from physical limitations such as the need to dissipate heat in a small volume, the speed of light and input/output bottlenecks. In the past few years, the performance of these machines has been improved by the use of vector pipelines and further developments in VLSI technology, however it seems unlikely that computer manufacturers can in the future sustain their order-of-magnitude improvements in performance every few years. It is widely believed that parallel computers provide the only hope for achieving teraflop performance in the foreseeable future. Furthermore, massively-parallel systems provide a very economical dollar cost per megaflop, due to their economies of scale. For these reasons, a great deal of effort has been recently invested into research on parallel machines.

Following McBryan *et al* [75], we briefly describe a taxonomy of computers. The three criteria are:

1. SIMD or MIMD,
2. shared or distributed memory,
3. scalar or vector-pipelined floating-point units,

which distinguishes eight classes of machine.

SIMD (single instruction, multiple data) means that every processor synchronously executes the same instruction, but on different data. MIMD (multiple instruction, multiple data) means that each asynchronous independent processor may execute a different instruction stream on different data. In this way, a conventional serial scalar computer is classified as an SISD machine.

In shared (global) memory machines, each processor has direct access to all memory, whereas processors in distributed (local) memory machines have *direct* access only to their own private memory.

Pipelined FPU's utilise special hardware to allow, in effect, several subprocedures of an arithmetic operation to be performed simultaneously. Thus when many similar operations are required, such as adding two long vectors $a_i + b_i$, high performance can be achieved.

The following list gives some examples of the eight classes thus formed:

1. scalar (SISD) computers: PC, Macintosh, Sun, VAX
2. vector computers: IBM 3090, Cray 1, Fujitsu VP-100 series
3. scalar shared memory multiprocessors: Sequent, Concurrent, Butterfly
4. vector shared memory multiprocessors: Cray Y-MP, ETA-10, VP-2600

5. scalar array processors: ICL DAP, CM-1, MasPar
6. pipelined array processors: CM-2
7. scalar distributed memory multiprocessors: NCUBE, iPSC/1
8. pipelined distributed memory multiprocessors: SUPRENUM, iPSC/2, CM-5

It should be pointed out that this simple classification scheme is not rigorous: there are certain computers which do not easily fit into these categories.

When innovative computer hardware is introduced, one of the main problems encountered by computational researchers is how to efficiently redesign the existing numerical algorithms, or how to invent completely new algorithms, so as to take best advantage of the special capabilities of the new machine. Parallel computers of one form or another are beginning to dominate the supercomputer market, and it is evident that this trend will continue for some time. Numerical methods for solving boundary value problems are well established for conventional serial machines, whereas the field of parallel algorithms is still undergoing rapid development. Moreover, there exist a large number of different parallel machines and a large body of experimental algorithms which run on them. With such a wide variety of architectures and systems, one of the challenges is to develop efficient and semi-portable parallel codes.

Here we shall consider the implementation of our multigrid package MGLAB on the Connection Machine CM-2, using the CM Fortran language (see [96]). This language is an implementation of FORTRAN 77 by Thinking Machines Corporation, designed to provide high-level parallel constructs for the programmer which run efficiently on the CM-2, whilst retaining a good deal of portability. CM Fortran includes the array-handling facilities of Fortran 90, plus extra array facilities included in early versions of the Fortran 90 draft standard, but removed in the final ANSI standard. On the other hand, CM Fortran does not support those features of FORTRAN 77 which depend on the order in which data is stored in CM memory, as one might expect. CM Fortran is recognised as being an important parallel programming language in its own right; indeed, Sun and IBM have committed themselves to supporting CM Fortran [57].

The Connection Machine is one of the few examples of a parallel machine on which a serial FORTRAN 77 programmer can easily understand the execution model, which is based on simple principles, and immediately start writing or porting programs in CM Fortran.

We have demonstrated that multigrid is among the most effective methods for solving elliptic partial differential equations on serial computers. Moreover, it is known that multigrid can be effectively mapped to a hypercube so as to maintain its optimal properties. Specifically, this is true for massively-parallel systems and for small systems where there are many grid points per processor (see Chan and Tuminaro [24]). However, we shall see that there are some interesting problems in implementing a parallel multigrid algorithm. At this stage, there is no definitive parallel multigrid method, however the problem is being attacked on both the computer science and numerical analysis fronts.

8.2 Overview of the CM-2 System

We have emphasised that an efficient program must be carefully designed to suit the hardware upon which it executes. Here we describe relevant aspects of the architecture and operation of the Connection Machine CM-2, a massively-parallel SIMD supercomputer currently sited at the Australian National University, and operated by the Parallel Computing Research Facility within the Centre for Information Science Research, and the ANU Supercomputer Facility. For further background or technical information, the reader is referred to Hillis [56] and the CM-2 Technical Summary [95].

The design philosophy of the CM-2 is that of a data-parallel computing system. Data parallel computing associates one processor with each data element, and so exploits the natural parallelism that is inherent in grid-based finite-difference problems, as well as many other computationally intensive types of problem. An SIMD architecture is well-suited to such problems, where the same operation is applied to different data at each grid point. Not only does the execution time decrease by perhaps orders of magnitude, but the programming task is often simplified.

The CM-2 installed at the ANU at the end of 1990 is configured with 16K single-bit processors, each with 256K bits of memory (the equivalent of 4K double-precision variables), which share 512 Weitek 32-bit (single precision) floating-point units. It has twin Sun-4 front-end hosts; all programs are developed and executed on a front-end as usual, except that when a parallel instruction is encountered, this instruction is broadcast to all processors in the CM via a sequencer and an instruction broadcast bus. Hence the CM may be viewed simply as an external array processor, designed to speed-up computations on large data structures. All serial data remains on the front-end. Masks (which define a *context*) allow selected processors to participate in or ignore certain parallel instructions.

When parallel data structures total more than the available number of physical processors, the system software operates in *virtual processor* mode, whereby some larger number ($2^n \times 16$K) of data processors are made available to the user, each with correspondingly reduced memory. Each physical processor then simulates the appropriate number of virtual processors. This operation is transparent to the programmer, and provides a very convenient programming feature. The virtual processor (VP) ratio is defined to be

VP ratio = (number of virtual processors) / (number of physical processors).

Inter-processor communication is implemented by means of a binary hypercube network with a total bandwidth of about 65 megabytes per second (on a 16K machine). Arbitrary communication patterns on the Connection Machine are supported by wormhole routing. The general router can be thought of as allowing every processor to send a message to any other processor, with all messages being sent and delivered at the same times. Nearest-neighbour communication in multidimensional rectangular grids, called NEWS communication, is particularly efficient since it is implemented with special hardware; indeed it is approximately

15 times faster than general router communication. We will discuss hypercube communication in much greater detail in Section 8.5.

There are three mechanisms for transferring data between the front-end and the thousands of processors: (a) broadcasting (a single front-end datum is sent to all the processors at once), (b) global combining (the front-end receives the sum, maximum, logical OR *etc.* of some parallel variable), and (c) the scalar memory bus (whereby the front-end can read or write a single datum to an individual processor).

The input/output system supports high-speed parallel transfer between processors and a bank of disk drives (called a DataVault) at 25 megabytes per second, and between processors and a graphics display device (called a frame buffer) at 40 megabytes per second, allowing CM data to be examined graphically in real-time. These I/O operations proceed independently from the sequencer and front-end.

Connection Machine software includes parallel versions of FORTRAN, C and Lisp. Each language includes extensions to support data parallelism, but otherwise is entirely based upon the relevant draft standard. No synchronisation instructions are required, due to the SIMD architecture. In each case one declares parallel variables, which are automatically allocated on the hypercube. The compilers produce code that is a mixture of front-end assembler code and Paris, the assembler language of the CM-2. (In Section 8.8 we will describe a slicewise model not currently available on the Connection Machine at the ANU; under this execution model, the assembler language produced is called PEAC.)

The CM system can support multiple users, and may be configured to run several programs simultaneously under timesharing.

This overview gives us enough information to now consider a simple multigrid implementation on the CM-2. Later, we will look more closely at the CM architecture when we discuss high-performance algorithms.

8.3 Simple Multigrid Implementation

We have stated that there is great interest in the design of parallel algorithms for mathematical computations. The goal of such research is to find algorithms which efficiently exploit the parallelism on a given machine architecture. This task must take into account the inter-processor communication overhead, an expense which is absent in conventional serial algorithms. Much of our discussion will be aimed at determining efficient multigrid communications on the Connection Machine. The hierarchy of grids in multigrid algorithms presents a special challenge in minimising the communications overhead.

Given that our goal is to port an existing serial version of `MGLAB` to the CM-2, the most important consideration is the layout of the parallel data structures. Due to the hypercube nature of the CM, all array dimensions are expanded to the next highest power of two. This is somewhat unfortunate, as our serial version of `MGLAB` incorporates explicit boundary conditions, and so uses grids of dimension $(2^{\Lambda}+1) \times (2^{\Lambda}+1)$. These figures assume that there is precisely one interior grid

point on the coarsest grid (*ie.* $M_1 = N_1 = 2$), a restriction which we must impose for multigrid efficiency reasons (see Section 3.2). In the CM version of MGLAB we therefore choose to incorporate boundary conditions implicitly, a scheme which uses grids of dimension $(2^\Lambda - 1) \times (2^\Lambda - 1)$, while insisting that $M_1 = N_1 = 2$.

For the moment, let us view the thousands of CM processors as being arranged in a two-dimensional array $2^\Lambda \times 2^\Lambda$, with the last row and column not being used in our multigrid algorithm. Then the simplest layout of the parallel variables v, f, r and so on, is to layer each level L within the memory of a processor (see Figure 8.1). Hence an appropriate declaration is

```
      DOUBLE PRECISION v(lmax,nmax,nmax)
CMF$  LAYOUT v(:SERIAL,:NEWS,:NEWS)
```

where `nmax = 2**lmax-1` $= 2^\Lambda - 1$.

The advantages of this approach are that

- a multigrid algorithm is simple to implement (obvious data structures), and
- intergrid transfers can be optimally efficient (injection is intra-processor).

The disadvantages of this approach are that on coarse grids

- many processors are idle (parallel efficiency decreases), and
- nearest grid neighbours are far apart (stencil communication costs increase).

Recall that the multigrid iteration is essentially some combination of the grid operators $\mathcal{A}_L$, $\mathcal{R}_L$, I_L^{L-1} and I_{L-1}^L, each of which can be represented as a stencil, involving some mixture of computation and communication on the grid. Since the CM architecture favours nearest-neighbour orthogonal (NEWS) communication, we prefer to use highly compact five-point operators, though sometimes we are forced to use nine-point stencils. Given our CM implementation strategy, we see that nearest-neighbour communication on coarser grids corresponds to power-of-two communication — an important fact which will explained shortly.

Our Dirichlet boundary conditions will be stored on single $(2^\Lambda - 1) \times (2^\Lambda - 1)$ grids, so as to ensure intra-processor availability of boundary conditions data for each and every grid point. An appropriate declaration for the "left" ($x = 0$) boundary condition is therefore

```
      DOUBLE PRECISION left__bc(nmax,nmax)
CMF$  LAYOUT left__bc(:NEWS,:NEWS)
```

A very convenient way to implicitly treat the boundary conditions is to use the EOSHIFT statement of CM Fortran: for example,

```
      DOUBLE PRECISION left(nmax,nmax)
CMF$  LAYOUT left(:NEWS,:NEWS)

      left = EOSHIFT(v(L,:,:), DIM = 1, SHIFT = -step(L),
                     BOUNDARY = left__bc(1,:))
```

where `step(L) = 2**(lmax-L)` $= 2^{\Lambda - L}$. Recall that coarse grids for linear PDE's have zero boundary conditions:

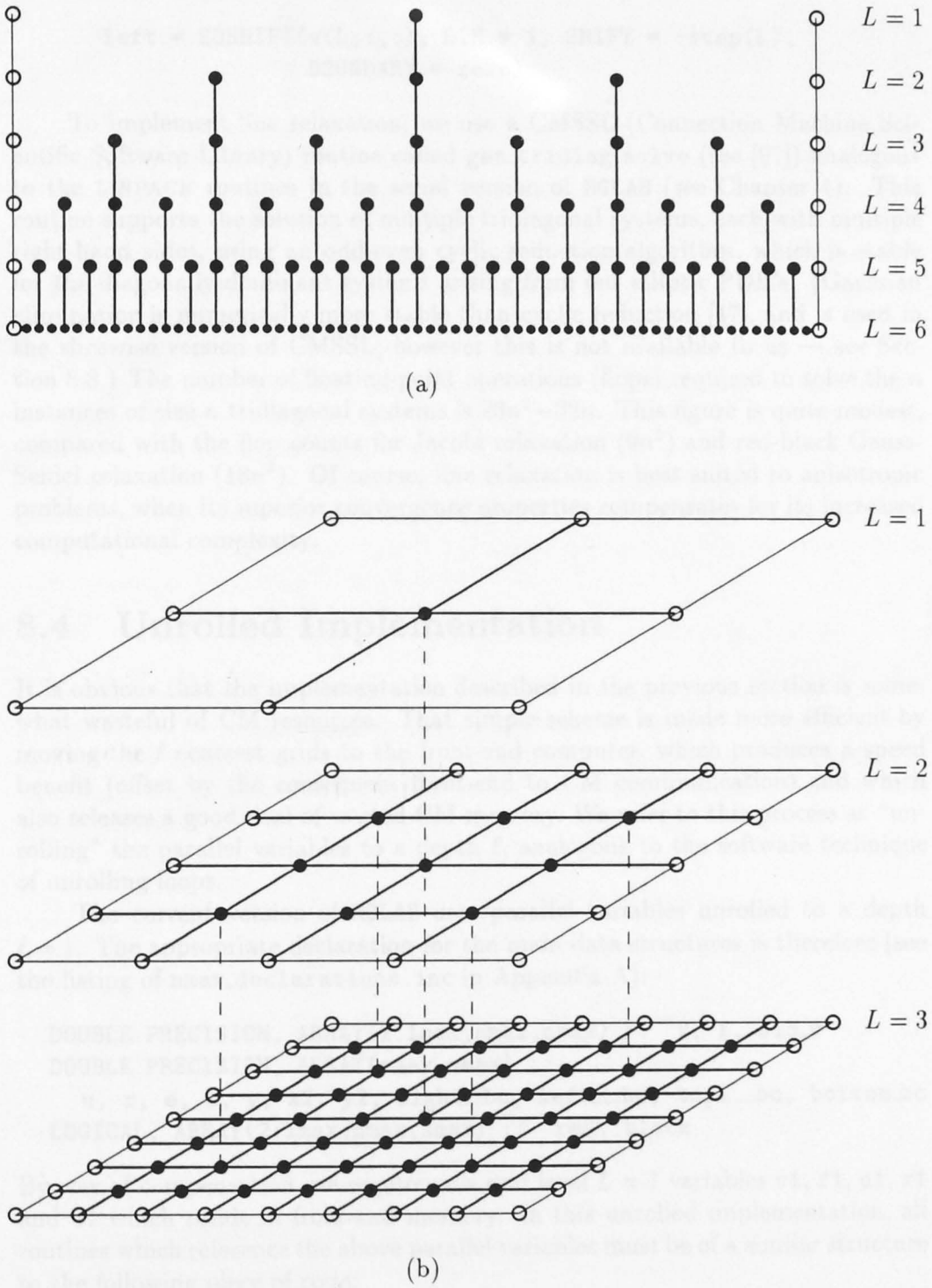


Figure 8.1: Pictorial representation of the (a) one-dimensional, and (b) two-dimensional pyramidal hierarchy of multi-grids for $\Lambda = 6$ and 3 respectively. Grid points depicted above one another are stored within the same processor. Boundary grid points are marked with open circles because they are handled implicitly in the CM Fortran version of `MGLAB`.

```
left = EOSHIFT(v(L,:,:), DIM = 1, SHIFT = -step(L),
               BOUNDARY = zero)
```

To implement line relaxation, we use a CMSSL (Connection Machine Scientific Software Library) routine called `gen_tridiag_solve` (see [97]) analogous to the LINPACK routines in the serial version of `MGLAB` (see Chapter 4). This routine supports the solution of multiple tridiagonal systems, each with multiple right-hand sides, using an odd-even cyclic reduction algorithm, which is stable for the diagonally-dominant systems arising from our elliptic PDE's. (Gaussian elimination is numerically more stable than cyclic reduction [47], and is used in the slicewise version of CMSSL; however this is not available to us — see Section 8.8.) The number of floating-point operations (flops) required to solve the n instances of size n tridiagonal systems is $23n^2 - 32n$. This figure is quite modest, compared with the flop counts for Jacobi relaxation ($9n^2$) and red-black Gauss-Seidel relaxation ($18n^2$). Of course, line relaxation is best suited to anisotropic problems, when its superior convergence properties compensates for its increased computational complexity.

8.4 Unrolled Implementation

It is obvious that the implementation described in the previous section is somewhat wasteful of CM resources. That simple scheme is made more efficient by moving the ℓ coarsest grids to the front-end computer, which produces a speed benefit (offset by the consequent front-end to CM communication) and which also releases a good deal of wasted CM memory. We refer to this process as "unrolling" the parallel variables to a depth ℓ, analogous to the software technique of unrolling loops.

The current version of `MGLAB` uses parallel variables unrolled to a depth $\ell = 1$. The appropriate declaration for the main data structures is therefore (see the listing of `nmax_declarations.inc` in Appendix A):

```
DOUBLE PRECISION, ARRAY(2:lmax,nmax,nmax) ::  v, f, old_v
DOUBLE PRECISION, ARRAY(nmax,nmax) ::
  u, r, e, x, y, x2, y2, right_bc, left__bc, top___bc, bottom_bc
LOGICAL, ARRAY(2:lmax,nmax,nmax) ::  red, black
```

By way of compensation, we require new grid level $L = 1$ variables `v1`, `f1`, `u1`, `r1` and `e1` which reside in front-end memory. In this unrolled implementation, all routines which reference the above parallel variables must be of a similar structure to the following piece of code:

```
IF (L == 1) THEN
   v1 = zero
ELSE
   v(L,:,:)  = zero
ENDIF
```

Routines such as `bilinear_interpolate_v` listed in Appendix C indicate the increase in programming complexity required for $\ell = 1$ unrolling. In the ideal situation, we would unroll to a depth of ℓ_{opt} levels, determined experimentally to be the point at which execution times no longer decrease (due to the increasing amount of front-end to CM communication and the slower serial calculations for large arrays on the front-end). This programming task was not attempted. However, we estimate the optimal depth to be $\ell_{\text{opt}} \approx 3$. This implementation would then consist of a serial multigrid algorithm on the front-end for levels $L = 1, 2, \ldots, \ell_{\text{opt}}$ and a parallel algorithm on the CM for levels $L = \ell_{\text{opt}}+1, \ldots, \Lambda$. In Section 8.7 we will show that a significant speed-up is obtained by unrolling to a depth of $\ell = 1$.

Analogous to the discussion in Section 3.7, we now consider the memory requirements for the implementation of `MGLAB` just described ($\ell = 1$) on the CM-2. As we have mentioned, the CM at the ANU is configured with 16K processors, each with access to 256K bits of RAM, equivalent to 4K double-precision variables, giving a total capacity of $2^{26} \approx 67$ million double-precision variables. Since logicals are stored in single bits on the CM, the amount of memory required for allocation of global data is

$$\Sigma_L = \left(3\tfrac{1}{32}\, L_{max} + 11\right) N_{max}^2 \qquad \text{double-precision variables,}$$

where L_{max} and N_{max} are respectively `lmax` $-\,1$ and `nmax` rounded up to the next power of two. Serial dimensions of arrays are treated similarly to NEWS dimensions in that allocation sizes are rounded up to a power of two, even though this appears to be unnecessary. We therefore find the following memory requirements for multigrid on L levels:

L	L_{max}	N_{max}	Σ_L
4	4	16	5920
5	4	32	23680
6	8	64	144384
7	8	128	577536
8	8	256	2310144
9	8	512	9240576
10	16	1024	62390272
11	16	2048	249561088

We see that we have sufficient memory to solve to a depth of $\Lambda = 10$ levels, giving a satisfactory linear resolution of at least 0.001 in our solution u.

The above figures relate strictly to allocation of the main CM data structures. (`MGLAB` dynamically allocates exactly the amount of required CM memory.) When we attempt to execute such a program, we find that the CM requires a certain amount of memory for house-keeping and, more significantly, temporary variables introduced by the compiler. In the fieldwise model (see Section 8.8), these temporary variables consume a great deal of memory. For example, NEWS

communication generates temporary copies of the target array, and any operations on logical variables (stored in 1 bit) create 32-bit temporary copies of the entire array variable.

For these reasons, the standard CM version of MGLAB exhausts the memory capacity of the ANU's 16K machine when attempting to solve to a depth of $\Lambda = 10$ levels. However, we have been lavish in our use of memory, and we have therefore built a "cut-down" version of MGLAB which differs in three respects from the original:

1. red-black relaxation is not permitted (the parallel variables `red` and `black` can be deleted),
2. immediate correction is not permitted (`old_v` can be deleted), and
3. the statement

   ```
   v(2:lmax-1,:,:) = zero
   ```

   in subroutine V_CYCLE is changed to

   ```
   DO l = 2, lmax-1
      v(l,:,:) = zero
   ENDDO
   ```

   (reducing the size of compiler temporaries).

Using this small version of MGLAB, we may proceed to $\Lambda = 10$ levels. (Except that even these memory requirements exceed the capacity of 8K processors running fieldwise double-precision multigrid.) We note that to obtain a resolution corresponding to $\Lambda = 11$, one would need to totally redesign the multigrid implementation, since even a single parallel variable of the form

```
      DOUBLE PRECISION v(2:lmax,nmax,nmax)
CMF$  LAYOUT v(:SERIAL,:NEWS,:NEWS)
```

would then occupy 67108864 double-precision variables, exactly the capacity of the ANU's Connection Machine.

8.5 Hypercubes and Gray Codes

Let us now return to the hypercube viewpoint of the CM-2. We shall look at the relationship between grids and hypercube structures, and the effect on multigrid communication.

A hypercube of dimension n (an n-cube) is an undirected graph consisting of 2^n nodes that are labelled by the n-bit binary numbers $0, 1, \ldots, 2^n - 1$ (see Figure 8.2). Two nodes are directly connected by an edge (corresponding to a wire) if and only if their labels differ by one bit in one position of the binary code. The m^{th} bit of the n-bit binary number corresponds to the m^{th} dimension of the hypercube. The diameter of an n-cube is n, that is, the traversal of at most n edges is required to reach any node from any other node. One appealing feature of the hypercube is its isotropic homogeneity; unlike many other ensemble

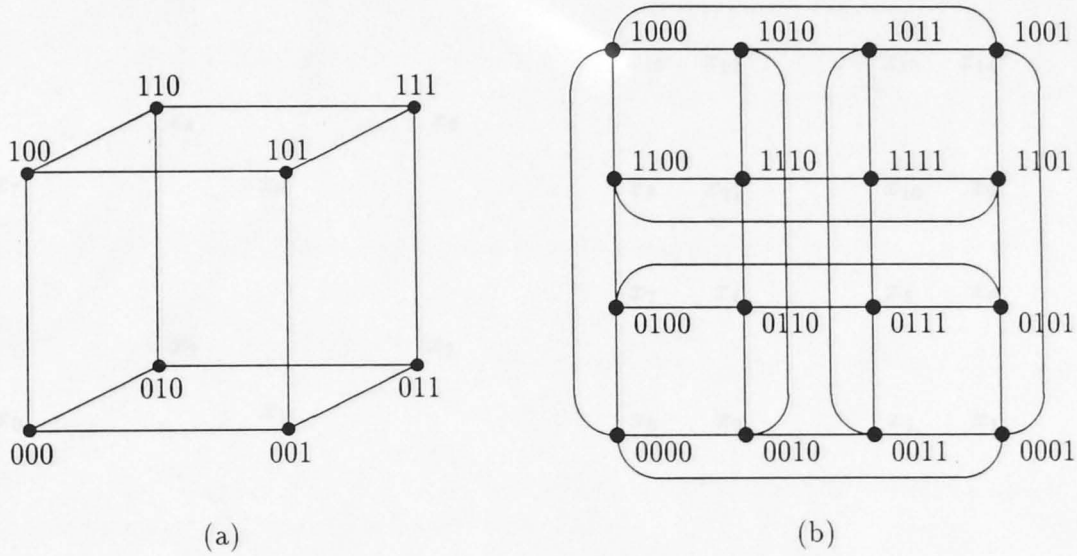


Figure 8.2: (a) Three-dimensional pictorial representation of a 3-cube, and (b) two-dimensional representation of a 4-cube. In both cases, wires which appear to cross in fact do not physically touch.

architectures, no node plays a privileged role. We can now picture the CM-2 situated at the ANU as having its 16K processors connected in the form of a 14-dimensional hypercube.

With the widespread availability of architectures based on the binary hypercube topology, there is growing interest in the relationship between this and other topologies, such as linear arrays, trees, rings and multidimensional grids. For instance, the question of algorithm portability across architectures reduces to the problem of embedding certain graphs into some target graph, in this case a hypercube. Such embeddings of linear arrays, rings, trees, pyramids and grids are well-known to computer scientists [67]. Indeed, a class of binary codes known as binary reflected Gray codes (BRGC's) provide simple algorithms for embedding linear arrays, rings and in particular, multidimensional grids into a hypercube. In other words, the Gray code assigns a processor to each grid point in a manner which ensures certain beneficial properties. These codes and their properties have been extensively studied [85].

BRGC's and other issues relating to multigrid on hypercube systems are discussed in, for example, Chan *et al* [23], Chan and Saad [24] and Briggs *et al* [19]. See also the very comprehensive list of references in [19].

A BRGC can be recursively defined as follows. Let $G_k = \{g_0, g_1, \ldots, g_{2^k-1}\}$ be a k-bit Gray code; let $\overline{G}_k$ denote the sequence obtained from G_k by reversing its order; and let $0G_k$ denote the sequence obtained from G_k by prefixing a zero to each element of the sequence (and similarly for $1G_k$). Then a BRGC of order

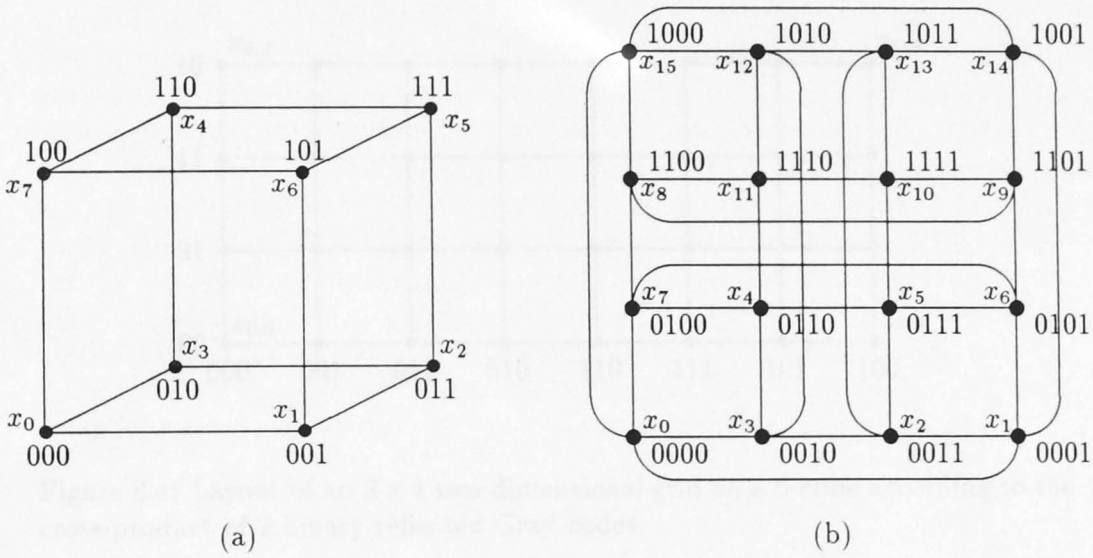


Figure 8.3: Layout of (a) 8 one-dimensional grid points on a 3-cube, and (b) 16 one-dimensional grid points on a 4-cube, according to a binary reflected Gray code.

$k-1$ is defined by $G_1 = \{0, 1\}$ and

$$G_{k+1} = \{0G_k, 1\overline{G}_k\} \qquad \text{for} \quad k = 1, 2, \ldots$$

For example,

$$\begin{aligned} G_2 &= \{00, 01, 11, 10\} \\ G_3 &= \{000, 001, 011, 010, 110, 111, 101, 100\} \\ G_4 &= \{0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, \\ &\quad 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000\}. \end{aligned}$$

We see that the Gray code for successive integers (grid points) differs by only one bit. This property guarantees that communication between any two neighbouring grid points involves only nearest-neighbour processors. Thus we have defined a mapping of 2^n one-dimensional grid points to a hypercube of dimension n which preserves the proximity property; in the terminology of graph theory, we have found a Hamiltonian path on the hypercube. Figure 8.3 shows this layout of one-dimensional grid points on the 3- and 4-cubes depicted in Figure 8.2.

The embedding of higher-dimensional grids into a hypercube is accomplished by means of the cross-product of one-dimensional BRGC's, an operation which preserves the proximity property. Suppose we have an $m_1 \times m_2$ two-dimensional grid, where $m_k = 2^{p_k}$. Then the appropriate mapping onto an n-cube, where $n = p_1 + p_2$, is given by the cross-product $G_1 \otimes G_2$, where G_k is the BRGC of the m_k grid points in the k^{th} coordinate. The cross-product of Gray codes is

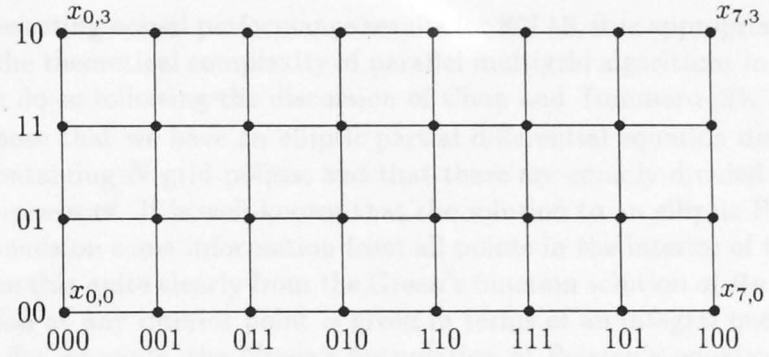


Figure 8.4: Layout of an 8×4 two-dimensional grid on a 5-cube according to the cross-product of 2 binary reflected Gray codes.

defined to be simply the concatenation of the individual bit sequences. This embedding obviously generalises to higher dimensions. As a two-dimensional example, consider the layout of an 8×4 grid on a 5-cube (see Figure 8.4). The binary node number of each grid point is obtained by concatenating its binary x-coordinate and its binary y-coordinate. (It is now clear why the Connection Machine expands all dimensions to a power of two.)

A more critical requirement for multigrid is that the proximity property be preserved on coarser grids. A remarkable property of the BRGC is that the distance between neighbouring grid points on all coarse grids is exactly two. (For a proof of this, see [60, 85].) This is because coarse-grid neighbours are always separated by 2^n fine-grid points (in our square array model of the CM discussed earlier). Thus the compact stencil operations of multigrid are using the communications system of the CM very efficiently, operating over distances of either one or two hypercube wires.

We mention that in 1986 Chan and Saad [22] developed an exchange algorithm which reduces the cost of coarse-grid communication to exactly one. The idea is to exchange the data of some nodes so as to ensure that the grid points of that level reside in physically neighbouring processors. However, this strategy is only effective when sufficiently many relaxations are performed on each level, in order to overcome the exchange overhead.

Finally, we mention that the concurrent multigrid algorithm of Gannon and Van Rosendale (see Chapter 9) can also be mapped with minimum communication overhead onto a hypercube by means of BRGC's.

To summarise, it is imperative for an algorithm to efficiently utilise the communication hardware, since communication can be more expensive than computation. We have seen that the marriage of power-of-two grid communication and BRGC-mapped hypercubes produces a natural and efficient communication scheme.

8.6 Complexity of Parallel Multigrid Solution

Before presenting actual performance results for MGLAB, it is appropriate to firstly consider the theoretical complexity of parallel multigrid algorithms in very broad terms (we do so following the discussion of Chan and Tuminaro [25, 100]).

Suppose that we have an elliptic partial differential equation discretised on a mesh containing N grid points, and that these are equally divided amongst P parallel processors. It is well-known that the solution to an elliptic PDE at each point depends on some information from all points in the interior of the domain. We can see this quite clearly from the Green's function solution of $\mathcal{A}u = f$, where the solution at any interior point is given in terms of an integral over the whole domain. For example, the Green's formulation of Poisson's equation with zero boundary data is

$$u(x,y) = \int_{\Omega} G(x,y,\xi,\eta)\, f(\xi,\eta)\, d\xi\, d\eta$$

where G is the Green's function. The global nature of discrete elliptic PDE's is characterised by the dense nature of the matrix $\mathcal{A}_h^{-1}$. We are therefore concerned with the optimal asymptotic time for collapsing information from N grid points to (any) single grid point. The best that can be done within a processor is $O(\frac{N}{P})$, since each point must contribute; and the optimal time for combining the resulting P pieces of information into one datum is $O(\log P)$ using a tree-visiting method. Hence a lower bound on the time for solving an elliptic PDE is

$$t = O\left(\tfrac{N}{P} + \log P\right).$$

If $P \approx N$ then this time is $O(\log N)$.

Crude lower bounds for the convergence of an iterative algorithm can similarly be obtained by determining the minimum number of iterations required to propagate information between any two grid points. Thus the necessary condition for a rapidly-convergent method is a "global" iteration operator. We now see why traditional explicit methods such as Jacobi iteration converge slowly: a single iteration involves updating the value at each grid point with respect to its immediate neighbours. Purely local methods such as these require many iterations to propagate information throughout the domain. Specifically, after k Jacobi iterations with a nine-point stencil, information will have been transmitted no further than k grid points away, and therefore a lower bound on the convergence rate of Jacobi's method on an $n \times n$ grid is $O(n)$. The true parallel multigrid algorithm, on the other hand, converges in $O(\log N)$ iterations. This is because the fine-to-coarse then coarse-to-fine intergrid transfers allow information to quickly propagate throughout the domain; in other words, multigrid has a global iteration operator.

On parallel processing systems, we see that a computational trade-off arises: local methods parallelise easily, but converge more slowly than global methods.

8.7 Timing Results

Let us firstly investigate the raw speed of the ANU's CM-2 by examining the fundamental times for typical communication and calculation in single and double precision. The times quoted here will be "CM-busy" times (from the CM Fortran timing facilities) obtained from 16K processors running at 7 megahertz in single-user batch mode. Programs were compiled with optimisation (`cmf -O`) and executed without any run-time checking. The CM software used was CM Fortran 1.1.3, CMSSL 2.2.1 and CMSS (CM system software) 6.1.1.

Since we are only concerned with power-of-two NEWS communication, we shall use the notation $s = 2^n$ and terminology "s-shift" for the `CSHIFT` operation of distance s. Figure 8.5 shows a graph of the CPU times for 10000 repetitions of various s-shifts on a 128×128 grid on 16K processors (and hence a VP ratio of one). This is compared with the ideal hypercube communication times of

$$\begin{array}{ll} 1 \text{ time unit} & \text{for} \quad n = 0 \\ 2 \text{ time units} & \text{for} \quad n = 1, 2, \ldots. \end{array}$$

We see that s-shifts for $n = 0, 1, 2$ are faster than model hypercube times due to efficient on-chip communication (see Section 8.8); nevertheless, we confirm that multidimensional arrays are mapped onto the CM using binary reflected Gray codes. We also note that double-precision s-shifts are about twice as slow as in single precision

$$t^{\text{DP}}_{\text{comm}} \,/\, t^{\text{SP}}_{\text{comm}} \approx 2$$

as expected. Figure 8.5 also indicates the corresponding CPU times on a 512×512 grid (a VP ratio of 16). The next section explains how each processor then contains a 4×4 contiguous block of grid points. In that case, significant amounts of off-chip communication do not occur until $n = 5$, compared with $n = 3$ for a unit VP ratio. This difference is a factor of $2^{5-3} = 4$ because the larger grid has 4×4 times the number of grid points. Also note that the ratio of communication times with respect to VP ratio are

$$t^{16}_{\text{comm}} \,/\, t^{1}_{\text{comm}} \approx 16$$

as expected.

We next consider fundamental times for calculation on the Connection Machine, as measured by 5000 repetitions of the array operation `C = C + A*B`, where each array is 128×128 or 256×256. We find the following times t_{calc}:

VP ratio	$t^{\text{SP}}_{\text{calc}}$	$t^{\text{DP}}_{\text{calc}}$	$\tilde{t}^{\text{SP}}_{\text{calc}}$	$\tilde{t}^{\text{DP}}_{\text{calc}}$
1	0.62	18.1	0.26	0.36
16	7.40	289	3.78	5.50

We see that a typical floating-point calculation is from 30 to 40 times slower in double precision than in single precision:

$$t^{\text{DP}}_{\text{calc}} \,/\, t^{\text{SP}}_{\text{calc}} \approx 35.$$

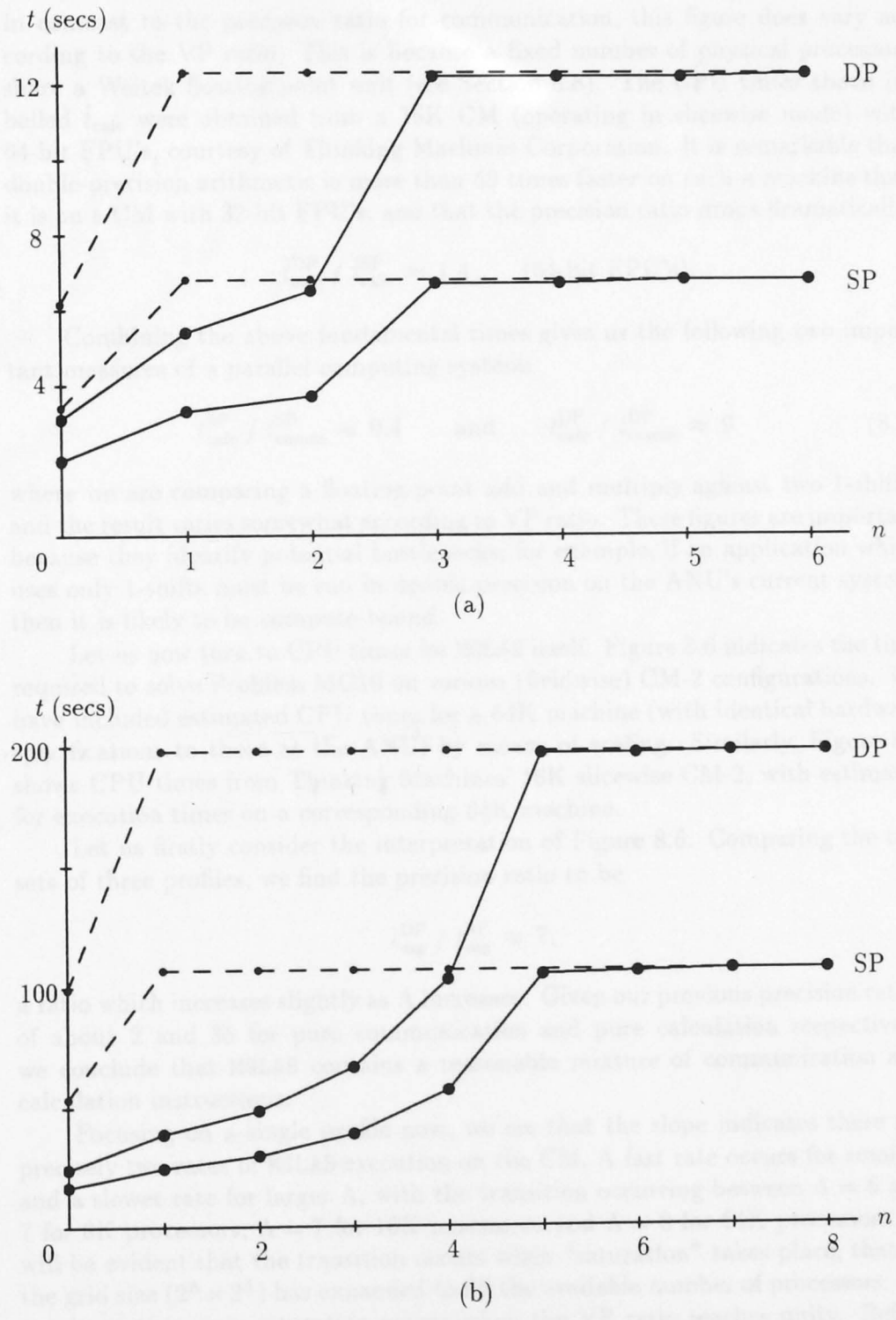


Figure 8.5: CPU execution times on a 16K-processor CM-2 for `CSHIFT`s of length $s = 2^n$ in single precision (SP) and double precision (DP) at a VP ratio of (a) one and (b) sixteen. Each dashed line represents the ideal hypercube times.

In contrast to the precision ratio for communication, this figure does vary according to the VP ratio. This is because a fixed number of physical processors share a Weitek floating-point unit (see Section 8.8). The CPU times above labelled $\tilde{t}_{\text{calc}}$ were obtained from a 16K CM (operating in slicewise mode) with 64-bit FPU's, courtesy of Thinking Machines Corporation. It is remarkable that double-precision arithmetic is more than 50 times faster on such a machine than it is on a CM with 32-bit FPU's, and that the precision ratio drops dramatically:

$$\tilde{t}^{\text{DP}}_{\text{calc}} \,/\, \tilde{t}^{\text{SP}}_{\text{calc}} \approx 1.4 \qquad \text{(64-bit FPU's)}.$$

Combining the above fundamental times gives us the following two important measures of a parallel computing system:

$$t^{\text{SP}}_{\text{calc}} \,/\, t^{\text{SP}}_{\text{comm}} \approx 0.4 \qquad \text{and} \qquad t^{\text{DP}}_{\text{calc}} \,/\, t^{\text{DP}}_{\text{comm}} \approx 9 \tag{8.1}$$

where we are comparing a floating-point add and multiply against two 1-shifts, and the result varies somewhat according to VP ratio. These figures are important because they identify potential bottlenecks; for example, if an application which uses only 1-shifts must be run in double precision on the ANU's current system, then it is likely to be compute-bound.

Let us now turn to CPU times for `MGLAB` itself. Figure 8.6 indicates the time required to solve Problem MG10 on various (fieldwise) CM-2 configurations. We have included estimated CPU times for a 64K machine (with identical hardware specifications to those at the ANU) by means of scaling. Similarly, Figure 8.7 shows CPU times from Thinking Machines' 16K slicewise CM-2, with estimates for execution times on a corresponding 64K machine.

Let us firstly consider the interpretation of Figure 8.6. Comparing the two sets of three profiles, we find the precision ratio to be

$$t^{\text{DP}}_{\text{mg}} \,/\, t^{\text{SP}}_{\text{mg}} \approx 7,$$

a ratio which increases slightly as Λ increases. Given our previous precision ratios of about 2 and 35 for pure communication and pure calculation respectively, we conclude that `MGLAB` contains a reasonable mixture of communication and calculation instructions.

Focusing on a single profile now, we see that the slope indicates there are precisely two rates of `MGLAB` execution on the CM. A fast rate occurs for small Λ and a slower rate for larger Λ, with the transition occurring between $\Lambda = 6$ and 7 for 8K processors, $\Lambda = 7$ for 16K processors and $\Lambda = 8$ for 64K processors. It will be evident that the transition occurs when "saturation" takes place, that is, the grid size $(2^{\Lambda} \times 2^{\Lambda})$ has expanded to fill the available number of processors. To put it another way, saturation occurs when the VP ratio reaches unity. Before saturation has occurred, M-cycle CPU time increases slowly with Λ; this is solely due to the increasing number of grid visits. Indeed, the CPU times for $50 \times P_{\Lambda}$-cycles shown in Table 8.1 for 16K demonstrate that there is virtually no increase in CPU time up to saturation, as expected.

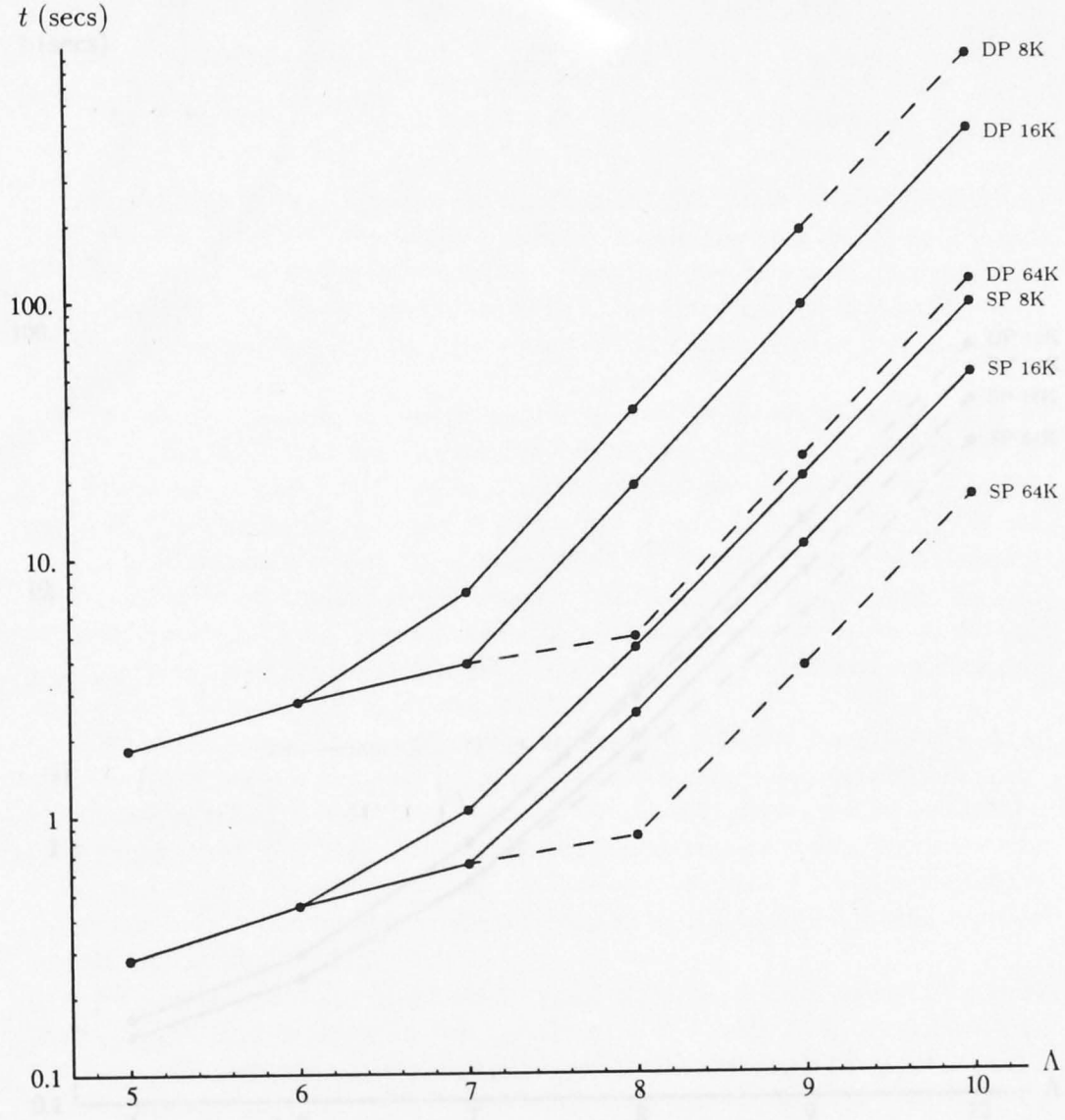


Figure 8.6: CPU execution times on various fieldwise Connection Machine CM-2 configurations for solving Problem MG10 using the CM Fortran version of `MGLAB`. (For $\Lambda = 10$, the "small" version is utilised.) The method used is $1 \times M_{\Lambda}^{2,2,2}$-cycle using 0.8-weighted Jacobi point relaxation, full-weight restriction and zero initial guess. Two sets of three configurations are shown: single precision (SP) and double precision (DP) for each of 8K, 16K and 64K processors. Each dashed line is an estimated result (see text for details).

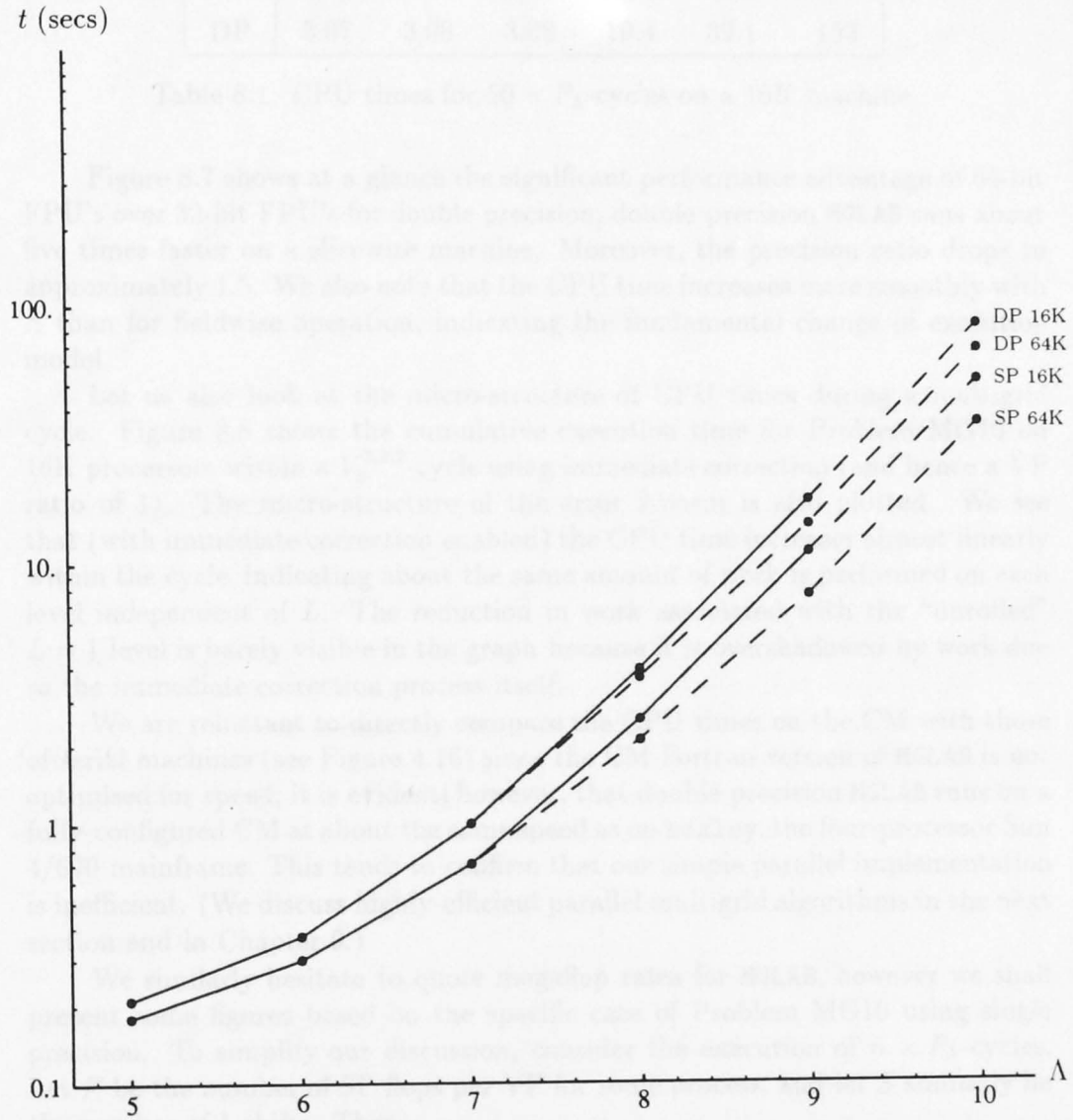


Figure 8.7: CPU execution times on various slicewise Connection Machine CM-2 configurations for solving Problem MG10 using the CM Fortran version of `MGLAB`. Other details are as for Figure 8.6. Two sets of two configurations are shown: single precision (SP) and double precision (DP) for 16K and 64K processors. Each dashed line is an estimated result (see text for details).

Λ	5	6	7	8	9	10
SP	0.65	0.66	0.68	1.11	2.63	8.31
DP	3.07	3.08	3.08	10.4	39.1	153

Table 8.1: CPU times for $50 \times P_\Lambda$-cycles on a 16K machine.

Figure 8.7 shows at a glance the significant performance advantage of 64-bit FPU's over 32-bit FPU's for double precision; double-precision `MGLAB` runs about five times faster on a slicewise machine. Moreover, the precision ratio drops to approximately 1.5. We also note that the CPU time increases more smoothly with Λ than for fieldwise operation, indicating the fundamental change of execution model.

Let us also look at the micro-structure of CPU times during a multigrid cycle. Figure 8.8 shows the cumulative execution time for Problem MG10 on 16K processors within a $V_8^{2,2,2}$-cycle using immediate correction (and hence a VP ratio of 1). The micro-structure of the error 2-norm is also plotted. We see that (with immediate correction enabled) the CPU time increases almost linearly within the cycle, indicating about the same amount of work is performed on each level independent of L. The reduction in work associated with the "unrolled" $L = 1$ level is barely visible in the graph because it is overshadowed by work due to the immediate correction process itself.

We are reluctant to directly compare the CPU times on the CM with those of serial machines (see Figure 4.16) since the CM Fortran version of `MGLAB` is not optimised for speed; it is evident, however, that double precision `MGLAB` runs on a fully-configured CM at about the same speed as on `huxley`, the four-processor Sun 4/690 mainframe. This tends to confirm that our simple parallel implementation is inefficient. (We discuss highly-efficient parallel multigrid algorithms in the next section and in Chapter 9.)

We similarly hesitate to quote megaflop rates for `MGLAB`, however we shall present some figures based on the specific case of Problem MG10 using single precision. To simplify our discussion, consider the execution of $n \times P_\Lambda$-cycles. Let F be the number of SP flops per VP for some process, and let S similarly be the number of 1-shifts. Then

$$\begin{aligned} F(\texttt{RELAX\_V}) &= 12 \\ F(\texttt{CALC\_RESULT}) &= F(\texttt{CALC\_RESIDUE\_R}) + 3 \\ F(\texttt{CALC\_RESIDUE\_R}) &= 11 \\ S(\texttt{CALC\_RESULT}) &= 4 \\ S(\texttt{CALC\_RESIDUE\_R}) &= 4 \end{aligned}$$

hence

$$\begin{aligned} F(n \times P_\Lambda\text{-cycles}) &= (n+1)\,F(\texttt{CALC\_RESULT}) + n\,F(\texttt{RELAX\_V}) \\ &= 26n + 14 \end{aligned}$$

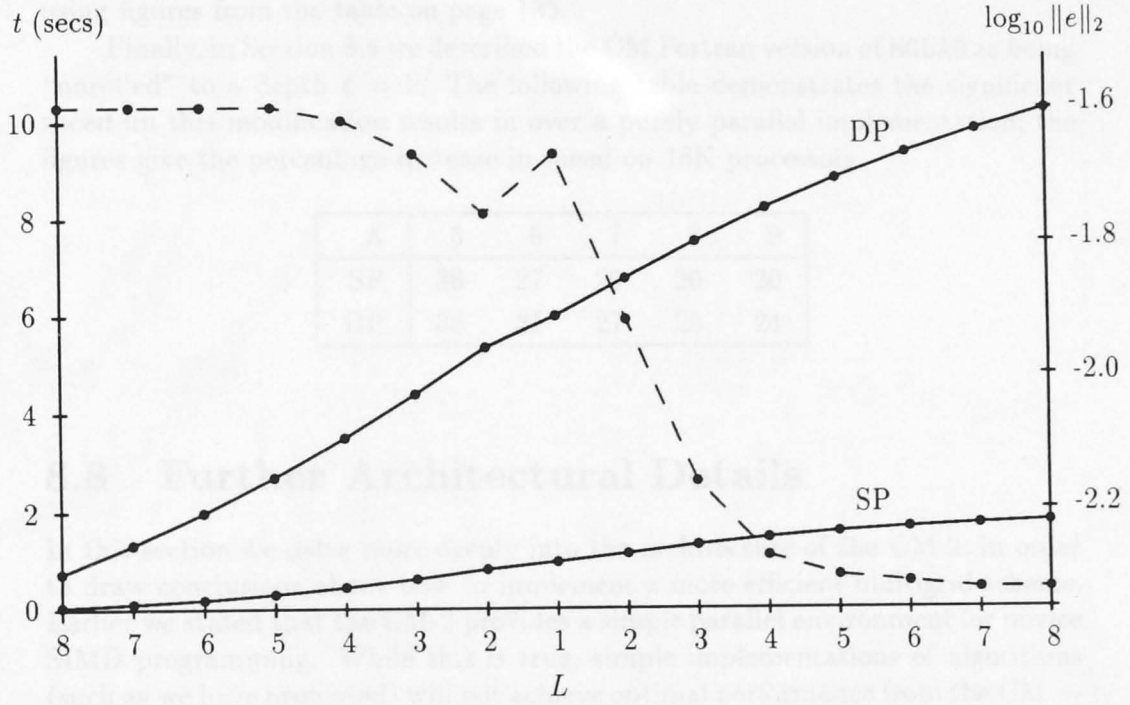


Figure 8.8: Cumulative CPU execution times within a $V_8^{2,2,2}$-cycle for Problem MG10 on 16K processors in single precision (SP) and double precision (DP). The method used is 0.8-weighted Jacobi point relaxation and full-weight restriction with immediate correction and 2-norms. The abscissa indicates the active level within the V-cycle; values are taken after relaxation at that level. Also shown are the corresponding error norms (dashed line and right-hand axis.)

and similarly

$$S\,(n \times P_\Lambda\text{-cycles}) \;=\; 8n + 4.$$

Therefore the SP flop rate is given by

$$N \;=\; F\,(2^\Lambda - 1)^2\,/\,t,$$

a figure which ignores CSHIFT's and front-end calculations (as is customary). In fact, we already know from equation (8.1) that one 1-shift takes about the same time as 2.5 flops in single precision. Let the actual value be δ, then a flop rate which approximately incorporates the cost of communication is given by

$$\tilde{N} \;=\; (F + \delta S)\,(2^\Lambda - 1)^2\,/\,t.$$

Taking the figure $t = 8.31$ seconds for $50 \times P_{10}$-cycles from Table 8.1, we find

$$\tilde{N} \;=\; 279 \quad \text{SP megaflops.}$$

On a 16K slicewise machine, we can accurately estimate the performance to be at least

$$\frac{7.40}{3.78} \times \tilde{N} \;=\; 546 \quad \text{SP megaflops}$$

using figures from the table on page 135.

Finally, in Section 8.4 we described the CM Fortran version of `MGLAB` as being "unrolled" to a depth $\ell = 1$. The following table demonstrates the significant speed-up this modification results in over a purely parallel implementation; the figures give the percentage increase in speed on 16K processors.

Λ	5	6	7	8	9
SP	36	27	20	20	20
DP	38	31	27	25	24

8.8 Further Architectural Details

In this section we delve more deeply into the architecture of the CM-2, in order to draw conclusions about how to implement a more efficient multigrid scheme. Earlier we stated that the CM-2 provides a simple parallel environment for novice SIMD programming. While this is true, simple implementations of algorithms (such as we have presented) will not achieve optimal performance from the CM — this requires a detailed knowledge of its architecture and mode of operation. We will only consider standard multigrid in this section, postponing a discussion of parallel multigrid methods to Chapter 9.

The Connection Machine CM-2 is constructed from replicated units called boards. Each fully-configured board contains two proprietary CM chips (sometimes called nodes), 256K of bit-addressable RAM on commercial chips, a Weitek floating-point interface chip (sometimes called a sprint chip) and a floating-point execution chip (see Figure 8.9).

Each CM chip consists of 16 bit-serial data processors, a communications controller, and an error-correcting code (ECC) unit. Each processor has four connections:

1. each processor is connected to an instruction bus, which distributes instructions broadcast from the sequencer
2. each processor is connected to a global bus, allowing for global results (such as `sum` or `max`) to be combined from all processors
3. each processor is connected to off-chip memory and the floating-point accelerator via the ECC unit (16 data signals plus 6 ECC bits)
4. each processor is connected to the communications controller, interfacing the 16 processors to 12 hypercube wires.

The CM-2 at the ANU has 2^{14} processors, hence only 10 of the 12 hypercube wires are used. The data processors are rated at 7 or 9 megahertz and are capable of performing a 32-bit add in about 21 microseconds [95]. For a VP ratio of n, any instruction is repeated n times, once for each data element in the n virtual processors. A program's performance (expressed in terms of megaflops,

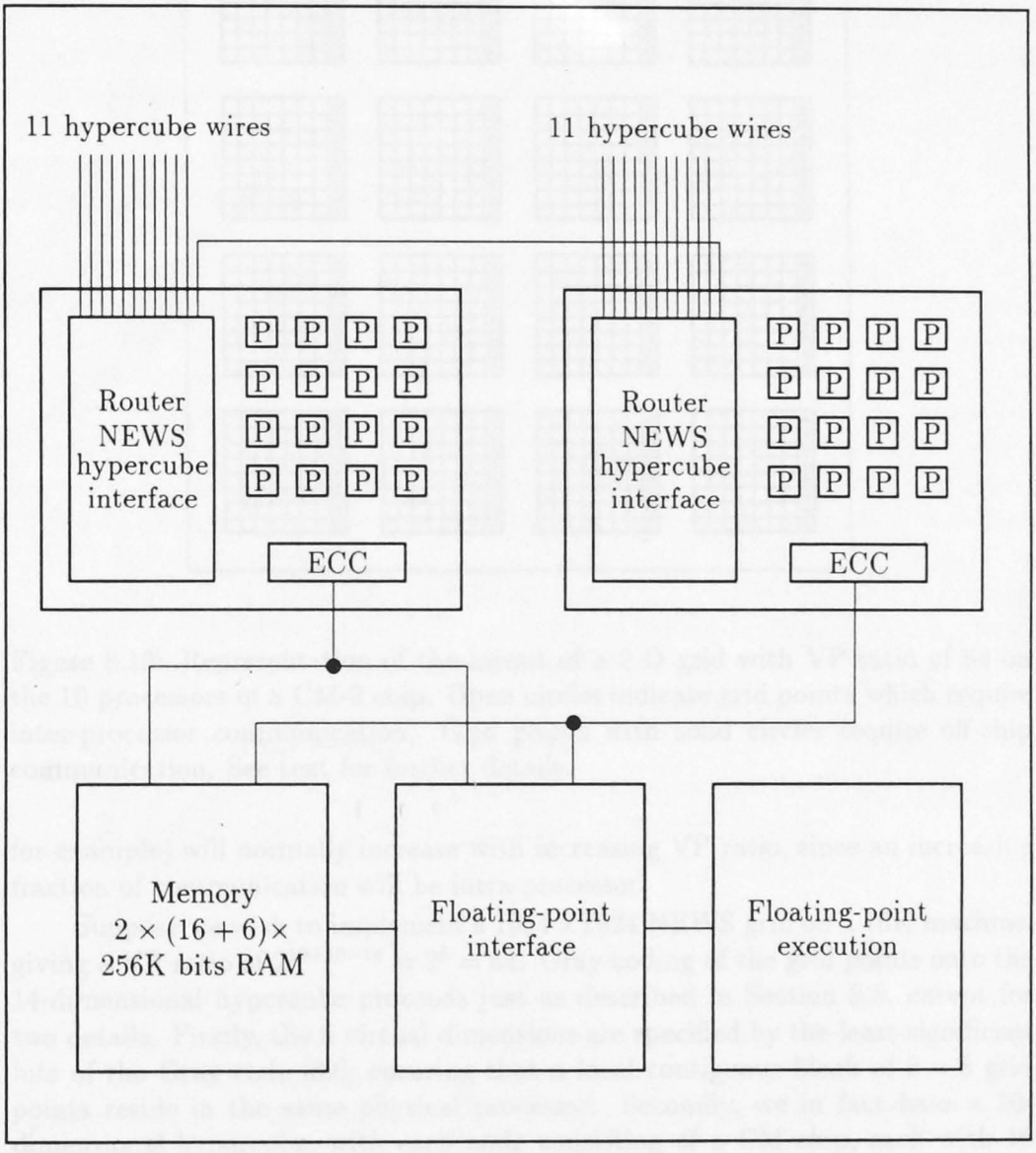


Figure 8.9: Diagram of the components of a CM-2 board. Each board contains a pair of CM-2 chips which share a group of memory chips, a floating-point interface ("sprint") chip, and a floating-point execution chip. The memory chips provide a 44-bit data path: 16 data and 6 ECC bits to each CM-2 chip. A CM-2 chip contains 16 single-bit processors (P), an error-correcting code (ECC) chip, and a hypercube communications interface chip. The internal wiring of the CM-2 chips is not shown. Also not shown are the instruction and global-result buses which connect to each CM-2 chip.

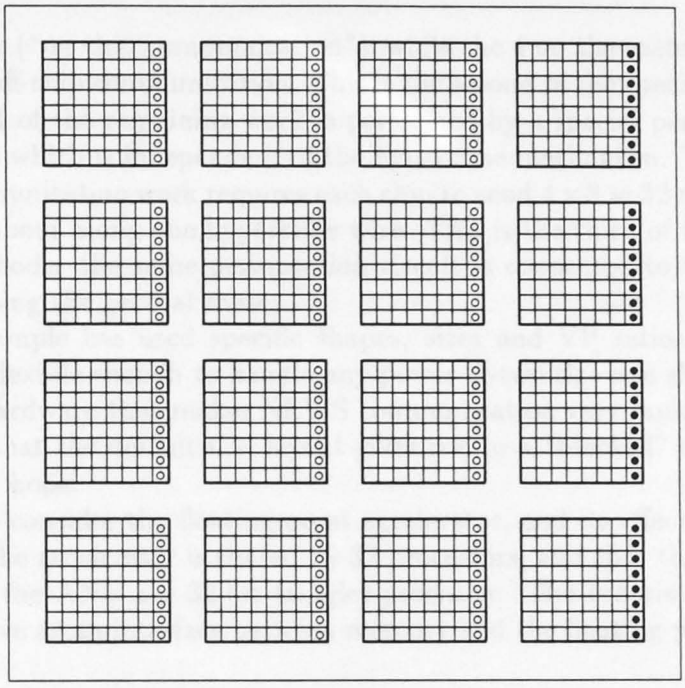


Figure 8.10: Representation of the layout of a 2-D grid with VP ratio of 64 on the 16 processors of a CM-2 chip. Open circles indicate grid points which require inter-processor communication. Grid points with solid circles require off-chip communication. See text for further details.

for example) will normally increase with increasing VP ratio, since an increasing fraction of communication will be intra-processor.

Suppose we wish to implement a 1024×1024 NEWS grid on a 16K machine, giving a VP ratio of $2^{10+10-14} = 2^6 = 64$. Gray-coding of the grid points onto the 14-dimensional hypercube proceeds just as described in Section 8.5, except for two details. Firstly, the 6 virtual dimensions are specified by the least-significant bits of the Gray code [89]; ensuring that a local contiguous block of 8×8 grid points reside in the same physical processor. Secondly, we in fact have a 10-dimensional hypercube, with each node consisting of a CM chip, each with 16 processors. Again, Gray-coding over our two-dimensional grid means that we can think of these 16 processors as forming a 4×4 group over the grid, each processor storing an 8×8 block of grid points (see Figure 8.10).

Consider a single NEWS communication on this VP set; say each processor sends a datum to its "eastern" neighbour. Within each group of 64 virtual processors, 56 of them send data within the same physical processor, while the 8 on the "eastern edge" of the block require inter-processor communication. Thus 7/8 of the work is done by a physical processor simply rearranging data in its memory; this is handled by sequencer software. The remaining 1/8 of the work requires each physical processor to send 8 messages to its physical processor neighbour to the east. Within each group of 16 processors, 12 of them send data within

the same chip ("on-chip communication"), while the 4 on the eastern edge of the chip require off-chip communication. This is the second of the specialised transfer methods: 3/4 of the remaining work is performed by a special per-node permutation circuit which is independent of the hypercube mechanism. The remaining 1/32 of communication work requires each chip to send $4 \times 8 = 32$ messages to its eastern neighbour along one hypercube wire. This is the third of the specialised transfer methods: the same permutation circuit is connected to the hypercube wires, bypassing the general router.

Our example has used specific shapes, sizes and VP ratios, but the CM hardware is flexible enough to handle any power-of-two size and shape. It is this specialised hardware that makes NEWS communication very rapid and efficient. We also see that the default CM layout gives rise to a "blocked" data structure, as one would hope.

Now we consider the floating-point accelerator, and its effect on execution. Recall that the accelerator is shared by 32 processors, and that the Weitek units currently at the ANU are 32-bit (single) precision. The function of the sprint chip is to serve as an interface between memory and the floating-point execution (FPE) chip.

Suppose we wish to add two 32-bit operands using the Weitek unit. Execution proceeds in five stages [95], each stage consisting of 32 "nano-instruction" cycles, one cycle for each of the 32 on-board processors.

1. The first operand (for each of the 32 processors) is moved from memory to the sprint chip.
2. Simultaneously, the first operands are moved to the FPE, and the second set of operands is moved from memory to the sprint chip.
3. The second operands are moved to the FPE, where the addition is performed.
4. The set of 32 results is moved from the FPE to the sprint chip.
5. The results are moved from the sprint chip to memory.

For a VP ratio of n, this process is pipelined to require $3n + 2$ stages, rather than $5n$. Since memory bandwidth is a limiting factor, simple 64-bit floating-point operations (on a CM with 64-bit FPU's) take precisely twice as long as 32-bit operations.

Careful consideration of the relationship between the 32 processors, memory and the floating-point unit reveals that there are two possible modes of operation. There is the standard approach whereby processors store data in contiguous words of memory; this is called the *fieldwise* execution model. There is room for improvement because the data is moved serially to the sprint chip, then "transposed" to feed the FPE all 32 bits at once. The alternative approach is to store the 32 bits of data *across* the memory, one bit corresponding to each processor on the board; this is called the *slicewise* execution model. Thus the data is in a form where it can be sent directly to the FPE. Slicewise operation generally gives significantly better performance; unfortunately, slicewise execution is only available for Connection Machines equipped with 64-bit floating-point

units. Moreover, Thinking Machines has recently written a good deal of slice-wise software which would very significantly improve the performance of MGLAB, including compilers designed to produce highly optimised micro-instructions for typical finite-difference stencil operations.

Given our fieldwise hardware, Thinking Machine software engineers [104] have advised that a substantial speed-up would only be obtained by writing a hand-coded blocked algorithm. In such an implementation, the coarsest grid is chosen such that all (or almost all) CM processors are utilised (near the point of processor saturation). For example, the coarsest grid on a 16K Connection Machine would be 128×128 ($L = 7$ in our standard grid hierarchy). Coarser grids are not employed at all, while finer grids are implemented by the many hand-coded grid interactions, in a somewhat similar way to the optimisation process of unrolling loops. Thus each CM processor acts like a serial multigrid implementation for the very deep (fine) levels, where v, f, r and so on are represented as single long arrays (see Section 3.7). Moreover, inter-processor communication is greatly reduced — most communication occurs within a physical processor, and consequently very high performance is achieved. This is a difficult programming exercise, however.

This approach works better for parabolic PDE's than it does for elliptic PDE's; parabolic equations seem to satisfactorily converge without moving to very coarse grids, unlike elliptic equations where attempting multigrid only on levels $L = 7, 8, 9, 10$ would generally result in poor convergence. Also, the hand-coded approach is more suited for three dimensional problems, since then saturation occurs on a much coarser grid. For example, a 3-D coarsest grid of $16 \times 16 \times 32$ processors results in a 100% processor utilisation on an 8K Connection Machine.

Chapter 9

Extensions

Research is what I'm doing when I don't know what I'm doing.

— Wernher von Braun

In this chapter, we firstly discuss how a multigrid algorithm can be modified for highly efficient execution on parallel processing systems, in particular for the Connection Machine CM-2. Research into parallel multigrid is becoming a significant topic in computational mathematics, especially as it has become clear that supercomputing will incorporate some degree of parallelism in the foreseeable future. Secondly, we consider how the `MGLAB` multigrid package could be extended to solve more difficult problems; in other words, how we might broaden the class of boundary value problems specified in Chapter 1 which are solvable using `MGLAB`.

9.1 Multigrid on Parallel Processing Systems

Chapter 8 described the simple CM-2 implementation of the current version of `MGLAB`, which is considered a standard parallel multigrid algorithm. Perhaps the most severe inefficiency in this implementation is the idle-processor problem, where large numbers of processors operating on coarse grids perform no useful computational work. This is a fundamental difficulty with the implementation of standard multigrid on massively-parallel systems. With N processors working on N grid points, standard parallel multigrid is an $O(\log N)$ rather than an $O(1)$ solution method (as it must be). In this section, we shall discuss multigrid schemes which are highly parallelisable; schemes designed to avoid the difficulties of standard multigrid, such as increasing numbers of idle processors on coarser grids. Time constraints in this project did not permit us to implement any of these schemes.

For a very recent discussion of multigrid methods on parallel computers, see McBryan *et al* [75]. See also Chan and Tuminaro [26].

9.1.1 Parallel Superconvergent Multigrid

Perhaps the best-known parallel multigrid scheme is an algorithm called Parallel Superconvergent Multigrid (PSMG), introduced by Frederickson and McBryan [41] in 1988 (see also [42, 73, 74]). PSMG is based on the simple idea of solving many coarse-grid problems simultaneously, then combining these results to provide a better fine-grid approximation. Apart from this last step, no extra computation time is involved, since the coarse-grid problems are solved on processors which would otherwise have been idle. While PSMG is still an $O(\log N)$ algorithm, the method results in a smaller constant than for standard parallel multigrid, due to a more rapid convergence rate.

Let us assume periodic boundary conditions for the moment. For a d-dimensional regular grid, we have the choice of 2^d different coarse grids. Multigrid traditionally chooses the even points in each dimension to arrive at a single coarse grid Ω_{2h}. The idea of PSMG is to project the fine-grid approximation v_h to *all* of these possible coarse grids, since they all should provide equally good solutions. In general, these different coarse grids receive slightly different data from the fine grid, and so combining these complementary views of the fine-grid problem should produce a superior approximation to the solution. We let this combination operator be denoted by Q. The most common way to combine the 2^d coarse-grid approximations v_{2h}^k is simply to use a linear interpolation of all these N^d coarse grid points; in two dimensions for example, a simple choice for Q is the average of the four bilinear interpolations: $v_h^k = I_h^{2h} v_{2h}^k$.

The simplest approach to implementing Dirichlet or Neumann boundary conditions is to use reflection principles on an extended grid — see Section 9.2.2.

Algorithm 9.1 presents the PSMG version of the linear multigrid V-cycle, analogous to Algorithm 3.2 from Chapter 3. The extensions to a nonlinear scheme and to a full multigrid M-cycle scheme are as before (see Chapter 3).

PSMG can be viewed as a process which operates on a single grid of points Ω_Λ of size 2^Λ in each dimension, with operators of scales $L < \Lambda$; hence it may also be called a multiscale method.

The PSMG strategy is to choose Q_L and $\mathcal{R}_L$ as functions of $\mathcal{A}_L$ in order to optimise the convergence rate. In Section 3.4, we mentioned that optimisation of the multigrid convergence rate involves minimising the spectral radius of T_L. However it is well-known that for square matrices $\rho(\mathcal{A}) < 1$ if and only if $\|\mathcal{A}\| < 1$, hence the spectral radius is "similar" to a matrix norm [8]. In [41, 42], Frederickson and McBryan have chosen to define the convergence rate τ in terms of $\|T\|$. Moreover, they indicate how to calculate convergence rates for translation-invariant operators using the discrete Fourier transform. In the case of $\mathcal{A} = -\Delta$ with periodic boundary conditions, discretised by the usual five-point star

$$A_L^5 = {N_L}^2 \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix},$$

it is found that the optimal two-grid symmetric nine-point Q_L and $\mathcal{R}_L$ operators

Algorithm 9.1 (Linear PSMG V-cycle)
A recursive iteration for solving $A_\Lambda u_\Lambda = f_\Lambda$, given some initial guess v_Λ.

procedure PSMG-V-cycle (v, f, L)

begin
if $L = 1$ **then**
  $v_L \leftarrow \mathcal{R}_L^{\nu_0}(v_L, f_L)$
else
  $v_L \leftarrow \mathcal{R}_L^{\nu_1}(v_L, f_L)$
  forall $k = 1$ **to** 2^d **do**
    $r_{L-1}^k \leftarrow I_{L-1}^L (f_L - A_L v_L)$
    $v_{L-1}^k \leftarrow 0$
    PSMG-V-cycle $(v^k, r^k, L-1)$
    $v_L \leftarrow v_L + Q_L v_{L-1}^k$
  endforall
  $v_L \leftarrow \mathcal{R}_L^{\nu_2}(v_L, f_L)$
endif
end

are

$$Q_L^5 = \begin{bmatrix} 0.066460 & 0.132920 & 0.066460 \\ 0.132920 & 0.265840 & 0.132920 \\ 0.066460 & 0.132920 & 0.066460 \end{bmatrix}$$

and

$$\mathcal{R}_L^5 = {N_L}^2 \begin{bmatrix} 0.006308 & 0.041304 & 0.006308 \\ 0.041304 & 0.257070 & 0.041304 \\ 0.006308 & 0.041304 & 0.006308 \end{bmatrix},$$

giving a two-grid convergence rate of

$$\tau \equiv \sup_L \|T_L\| = 0.063$$

where

$$T_L = I - [\mathcal{R}_L + (I - \mathcal{R}_L A_L) Q_L A_{L-1}^{-1}] A_L$$

is the two-grid PSMG iteration operator (*cf.* equation (3.2)). (If A_L is singular, as above, then this equation has an appropriate interpretation in terms of the Moore-Penrose pseudo-inverse A_L^+.)

Choosing the nine-point *Mehrstellen* discretisation

$$A_L^9 = \tfrac{1}{6} {N_L}^2 \begin{bmatrix} -1 & -4 & -1 \\ -4 & 20 & -4 \\ -1 & -4 & -1 \end{bmatrix}$$

and the standard interpolation

$$Q_L^9 = \begin{bmatrix} 0.0625 & 0.1250 & 0.0625 \\ 0.1250 & 0.2500 & 0.1250 \\ 0.0625 & 0.1250 & 0.0625 \end{bmatrix}$$

leads to an optimal

$$\mathcal{R}_L^9 = {N_L}^2 \begin{bmatrix} 0.015666 & 0.046489 & 0.015666 \\ 0.046489 & 0.305900 & 0.046489 \\ 0.015666 & 0.046489 & 0.015666 \end{bmatrix}$$

with a convergence rate $\tau = 0.026$. These convergence figures compare favourably with a standard red-black Gauss-Seidel rate of 0.074 (see Decker [31, 32]).

The authors also compute optimal operators and convergence rates for multigrid V-cycles; see [41] for further details.

While these results prove the benefits of the PSMG algorithm for Poisson's equation, to our knowledge a *super*convergent rate has not yet been demonstrated for more complex problems. Moreover, while PSMG can produce extremely good convergence rates, it does so at the cost of communication-intensive operators. For example, the multigrid convergence rate for two smoothing steps per level, the Mehrstellen operator A^9, a nine-point relaxation operator and a 25-point interpolation operator Q is a remarkable 0.001. In fact, in 1990 Decker [31] carefully examined the PSMG algorithm and concluded that, while it achieves perfect processor utilisation, the actual efficiency (for the Poisson equation) is the same as for a parallelised version of standard red-black Gauss-Seidel multigrid. This is because, as we have said, the superior convergence rate of PSMG is counter-balanced by substantially more expensive operators $\mathcal{A}$, Q and $\mathcal{R}$ (in terms of both communication and computation).

9.1.2 Other Parallel Multigrid Schemes

In addition to the PSMG method of Frederickson and McBryan, there have been a number of other parallel multigrid methods proposed; for example, "filtering multigrid" by Chan and Tuminaro [25, 100], "aggregation/disaggregation multigrid" of Douglas and Miranker [35], "robust multigrid" of Hackbusch [52], "symmetric/antisymmetric multigrid" of Douglas and Smith [36], "concurrent multigrid" of Gannon and Van Rosendale [43], and the PVM algorithm of Lin, Proskurowski and Gaudiot [69]. A good discussion of many of these schemes appears in Tuminaro's doctoral thesis on parallel multigrid algorithms [100].

These novel methods are essentially based on the same principle used in PSMG: that new sub-problems are found for coarser grids, designed to be processed in parallel by otherwise idle processors. The crucial step is to determine suitable coarse-grid sub-problems which will combine effectively. Tuminaro [100, 27] has identified two general principles which are used in many parallel multigrid algorithms: "aliasing-error cancellation" and "non-interfering subspaces". The former refers to the annihilation of unwanted error components on

coarse grids; in standard multigrid this is the result of projecting high frequencies onto coarse grids (see Section 3.1). In PSMG, this aliasing-error cancellation is the mechanism which accelerates convergence. Non-interfering subspace methods create multiple approximate solutions corresponding to different subspaces; many of the parallel methods listed above fall into this category. These methods demand a proper choice of interpolation, restriction and coarse-grid operators.

The PSMG algorithm is quite simple to implement, as the same intergrid operators are used in the four sub-problems. In contrast, the non-interfering subspace methods of Hackbusch, Douglas and Miranker, and Douglas and Smith utilise different intergrid operators for each sub-problem, which are constructed so as to approximate the solution of the fine-grid problem in different subspaces. These must be chosen so that each approximation, when combined, does not adversely affect the approximation from the other subspaces. Tuminaro analyses these methods in terms of "A-orthogonal decompositions"; see [100] for more details.

To give a flavour of such subspace methods, we now give a brief overview of Hackbusch's "robust" multigrid. The method is motivated by the role of coarse-grid correction in standard multigrid, namely to reduce errors having low frequencies in both the x and y directions (see Figure 3.4). Hackbusch introduced the following three complementary corrections to reduce errors in the other frequency subspaces:

1. high frequencies in both the x and y directions,
2. low x-frequency and high y-frequency,
3. high x-frequency and low y-frequency.

The specific restriction stencils proposed are

$$R_1 = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \qquad R_2 = \frac{1}{8}\begin{bmatrix} -1 & 2 & -1 \\ -2 & 4 & -2 \\ -1 & 2 & -1 \end{bmatrix},$$

$$R_3 = \frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 2 & 4 & 2 \\ -1 & -2 & -1 \end{bmatrix}, \qquad R_4 = \frac{1}{8}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix},$$

which are applied at the even x - even y points, the even-odd, odd-even and odd-odd grid points, respectively. The corresponding prolongation and coarse-grid operators are

$$P_i = R_i^T \qquad \text{and} \qquad A_i = R_i A P_i.$$

This is as much as we shall say about subspace methods.

Instead of accelerating the convergence of the iteration, the other approach in creating a fast parallel multigrid method is to reduce the time per iteration. Concurrent multigrid aims to iterate on all grid levels simultaneously. It achieves this by first distributing the original problem over all grids, relaxing on all levels in parallel, and then recombining the approximations. Since the function of successively coarser grids is to reduce lower and lower frequency components of

the error, it is natural to distribute these various errors across the grids; this is achieved by restriction and interpolation of the residue. One then performs relaxation on all grids, and computes the recombination of the residue with some operator Q. A difficulty with concurrent multigrid is that there is no natural way to map a pyramidal hierarchy of grid points to a hypercube, hence many processors may remain idle: in two dimensions this fraction is approximately 1/3, while in three dimensions it rises to about 3/7 (see Chan and Saad [22]).

The field of parallel multigrid is still in its infancy, although a good deal of promising research is currently underway in the area, and an increasing number of papers are appearing.

9.2 More Difficult Problems

There are several ways in which MGLAB could be extended, so as to increase the range of boundary value problems that may be solved. We will also discuss more difficult problems that can be recast into a form suitable for numerical solution by MGLAB.

9.2.1 PDE's on Manifolds

Given an arbitrary basis $(\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d)$ of Euclidean space $\mathcal{E}^d$, the covariant components of a vector $\mathbf{x}$ are defined as

$$x_i = \mathbf{x} \cdot \mathbf{e}_i$$

while the contravariant components are numbers x^i such that

$$\mathbf{x} = x^i \mathbf{e}_i$$

(see for example [68]), using Einstein's summation convention over raised and lowered indices. By convention, covariant components are written with subscripts and contravariant with superscripts. With the definition

$$\mathbf{e}_i \cdot \mathbf{e}_j = g_{ij} = g_{ji}$$

we have the basic relations

$$x_i = g_{ij}\, x^j \qquad \text{and} \qquad x^j = g^{ij} x_i .$$

Now consider an arbitrary two-dimensional manifold $\mathcal{S}$. If we assume that $\mathcal{S}$ is embedded in $\boldsymbol{R}^3$, then we may introduce a parametric representation of $\mathcal{S}$ in 3-space as $\mathbf{x} = \mathbf{x}(u, v)$, where the components x, y and z of the position vector $\mathbf{x}$ are functions of two parameters u and v. We can then consider the *first* and *second fundamental forms* of the manifold (see for example [33])

$$ds^2 = E(u,v)\, du^2 + 2F(u,v)\, du\, dv + G(u,v)\, dv^2 = g_{ij}\, du^i\, du^j$$

and

$$L(u,v)\,du^2 + 2M(u,v)\,du\,dv + N(u,v)\,dv^2 = h_{ij}\,du^i\,du^j.$$

These relate to the material of Chapter 7 in the following way. The principal curvatures k_1 and k_2 are the solutions to the quadratic equation

$$k^2 - \frac{EN+GL-2FM}{EG-F^2}\,k + \frac{LN-M^2}{EG-F^2} = 0$$

and the mean curvature and Gauss curvature are given by

$$H = k_1 + k_2 = \frac{EN+GL-2FM}{EG-F^2}$$

and

$$K = k_1\,k_2 = \frac{LN-M^2}{EG-F^2}.$$

In fact, we need not restrict ourselves to such embedded manifolds. The following procedure allows us to consider arbitrary 2-manifolds, and by generalisation, arbitrary d-dimensional manifolds.

To investigate some differential equation $\mathcal{F}u = 0$ on $\mathcal{S}$ rather than simply on $\mathcal{E}^2$, our Cartesian coordinates (x,y) are replaced by more a general frame of curvilinear coordinates (x^1,x^2). We need to express all second derivatives in $\mathcal{F}$ in covariant form using a *connection* Γ

$$u_{;ij} = u_{,ij} - \Gamma^k_{ij}\,u_{,k}$$

with $i,j,k \in \{1,\ldots,d=2\}$, and where we use a standard notation of a subscripted semi-colon to indicate a covariant derivative and a subscripted comma to indicate a partial derivative: $u_{,i} \equiv \partial u/\partial x^i$. The Christoffel symbol Γ^k_{ij} is defined by

$$\Gamma^k_{ij} = \tfrac{1}{2}\,g^{kl}(g_{li,j} + g_{jl,i} - g_{ij,l})$$

where g^{ij} are the contravariant elements of the metric

$$[g^{ij}] = [g_{ij}]^{-1}.$$

Note that first derivatives are already frame-invariant. In principle, the same modifications apply to higher-order derivatives, but we shall restrict ourselves here to second-order equations.

The generalisation of the Laplacian operator on a manifold is the *Laplace-Beltrami* operator

$$\Delta_g u \equiv g^{ij}\,u_{;ij}.$$

Thus we have a procedure by which we may apply an operator $\mathcal{F}$ on some arbitrary manifold $\mathcal{S}$. As an example of this procedure, consider Poisson's equation on a sphere of radius one:

$$g^{ij}\,u_{;ij} = f(x^1,x^2).$$

The transformation from Cartesian to spherical polar coordinates is given by

$$\begin{aligned} x &= r\sin\theta\cos\phi \\ y &= r\sin\theta\sin\phi \\ z &= r\cos\theta \end{aligned}$$

hence

$$\begin{aligned} dx &= \sin\theta\cos\phi\,dr + r\cos\theta\cos\phi\,d\theta - r\sin\theta\sin\phi\,d\phi \\ dy &= \sin\theta\sin\phi\,dr + r\cos\theta\sin\phi\,d\theta + r\sin\theta\cos\phi\,d\phi \\ dz &= \cos\theta\,dr - r\sin\theta\,d\theta. \end{aligned}$$

We have a surface $\mathcal{S}$ defined by $r = 1$ with coordinates $(x^1, x^2) \equiv (\theta, \phi)$, hence

$$\begin{aligned} ds^2 &= dx^2 + dy^2 + dz^2 \\ &= d\theta^2 + \sin^2\theta\,d\phi^2 \end{aligned}$$

therefore our metric is

$$[g_{ij}] = \begin{pmatrix} 1 & 0 \\ 0 & \sin^2\theta \end{pmatrix}$$

and

$$[g^{ij}] = \begin{pmatrix} 1 & 0 \\ 0 & \csc^2\theta \end{pmatrix}$$

so that $g_{11,1} = \partial g_{11}/\partial\theta = 0$, and so on. The first of the eight components of the connection is

$$\begin{aligned} \Gamma^1_{11} &= \tfrac{1}{2}\,g^{11}(g_{11,1} + g_{11,1} - g_{11,1}) + \tfrac{1}{2}\,g^{12}(g_{21,1} + g_{12,1} - g_{11,2}) \\ &= 0. \end{aligned}$$

In a similar fashion, we have

$$\begin{aligned} \Gamma^2_{11} &= 0 \\ \Gamma^1_{22} &= -\sin\theta\cos\theta \\ \Gamma^2_{22} &= 0 \end{aligned}$$

hence the Laplace-Beltrami operator $\Delta_g u \equiv g^{ij}\,u_{;ij}$ is

$$g^{11}\,u_{;11} + g^{12}\,u_{;12} + g^{21}\,u_{;21} + g^{22}\,u_{;22} = u_{;11} + \csc^2\theta\,u_{;22}$$

but these covariant derivatives are given by

$$\begin{aligned} u_{;11} &= u_{,11} - \Gamma^k_{11}\,u_{,k} \\ &= u_{,11} \\ &\equiv u_{\theta\theta} \\ u_{;22} &= u_{,22} - \Gamma^k_{22}\,u_{,k} \\ &= u_{,22} + \sin\theta\cos\theta\,u_{,1} \\ &\equiv u_{\phi\phi} + \sin\theta\cos\theta\,u_{\theta} \end{aligned}$$

and so Poisson's equation on a sphere is

$$u_{\theta\theta} + \csc^2\theta\, u_{\phi\phi} + \cot\theta\, u_\theta = f(\theta,\phi)$$

which is a well-known result.

To numerically solve this equation, we would renormalise from the domain $\{(\theta,\phi) \in [0,\pi] \times [0,2\pi)\}$ to $\{(\Theta,\Phi) \in [0,1] \times [0,1]\}$:

$$4u_{\Theta\Theta} + \csc^2(\pi\Theta)\, u_{\Phi\Phi} + 4\pi\cot(\pi\Theta)\, u_\Theta = 4\pi^2 f(\Theta,\Phi)$$

and then implement the solution in the usual way using MGLAB.

We want to solve this equation on the whole manifold $\mathcal{S}$, so care must be taken at the boundaries. Since $\Phi = 0$ corresponds to $\Phi = 1$, the "left" and "right" Dirichlet boundary conditions need to be replaced by appropriate regularity conditions (for example, if we desire a C^2-smooth solution, then we require matching first and second derivatives across the longitudinal cut in the sphere). Moreover, $\Theta = 0$ and $\Theta = 1$ correspond to the north and south pole respectively, so the "top" and "bottom" boundaries correspond to single points on $\mathcal{S}$ and so must remain constant. Additionally, regularity demands that all derivatives up to order n match across the north pole:

$$\begin{aligned} \lim_{\Theta\to 0^+} D^i u(\Theta,\Phi) &= \lim_{\Theta\to 0^-} D^i u(\Theta,\Phi) \\ &= \lim_{\Theta\to 0^+} D^i u(\Theta,\Phi+\tfrac{1}{2}) \qquad \text{for} \quad \Phi \in [0,\tfrac{1}{2}], \quad i = 1,2,\ldots,n \end{aligned}$$

and similarly for the south pole. This unpleasant constraint is difficult to achieve numerically. In fact, it is well-known that the sphere cannot be mapped isomorphically to the square.

These are the sort of additional considerations required to solve global problems on $\mathcal{S}$; local problems are more straight-forward. For example, the above procedure could be applied to solve $\mathcal{F}(u) = 0$ on a region of a minimal surface, with a certain amount of calculation in order to recast the problem and with no modification to MGLAB.

In general, solving PDE's on manifolds is a difficult problem, and there are many open questions in the field. A very recent paper by Lanza [66], for example, addresses the multigrid solution of certain boundary value problems on non-Euclidean manifolds arising in general relativity.

9.2.2 Non-Dirichlet Boundary Conditions

The functionality of MGLAB could be expanded by allowing the solution of PDE's with *Neumann* boundary conditions:

$$\tfrac{\partial u}{\partial n}(x,y) = g(x,y) \qquad \text{on} \quad \partial\Omega$$

where $\partial u/\partial n$ indicates the appropriate normal derivative, and also with *mixed* (or *Robbins*) boundary conditions:

$$\tfrac{\partial u}{\partial n}(x,y) + Cu(x,y) = g(x,y) \quad \text{on} \quad \partial\Omega$$

and, in principle, higher-order boundary derivative conditions.

In the finite-difference method, the Neumann condition $u_x(0,y) = g(y)$ for example, corresponds to

$$M(u_{1,j} - u_{0,j}) = g_j \qquad \text{for} \quad j = 0, 1, \ldots, N$$

but which is accurate only to $O(h)$. The standard technique is to introduce a fictitious column of grid points $u_{-1,j}$, so that we may use the $O(h^2)$ approximation

$$\tfrac{M}{2}(u_{1,j} - u_{-1,j}) = g_j \qquad \text{for} \quad j = 0, 1, \ldots, N.$$

The equation $\mathcal{A}(u_{ij}) = f_{ij}$ is then applied to the grid column $i = 0$, as well as to the interior points of Ω, $i = 1, 2, \ldots, M$. If the operator $\mathcal{A}$ is of order $2n$, then we would require n fictitious columns in this case.

Another common problem to arise is for a PDE to have periodic boundary conditions:

$$u(0,y) = u(1,y) \qquad \text{(continuity)}$$
$$u_x(0,y) = u_x(1,y) \qquad \text{(regularity)}$$

for example. We can then reduce our set of discrete grid points by one column, and maintain only the columns $i = 0, 1, \ldots, M-1$. One then applies relaxation sweeps uniformly to all grid points — the distinction between interior and exterior grid points vanishes. If we treat these columns as though they connect to form a torus, the periodic boundary conditions are automatically satisfied. Note that this situation is ideally suited to the architecture of a hypercube parallel processor, such as the Connection Machine CM-2, since tori embed perfectly into hypercubes.

9.2.3 Systems of PDE's

Suppose that we desire the solution of a system of PDE's, $\mathcal{A}(\mathbf{u}) = \mathbf{f}$, such as the steady-state Navier-Stokes equations describing the flow of an incompressible viscous fluid in three dimensions [11]

$$\sum_{j=1}^{3} \rho\, u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \mu \Delta u_i + \rho F_i \qquad \text{for} \quad i = 1, 2, 3$$

$$\sum_{i=1}^{3} \frac{\partial u_i}{\partial x_i} = 0$$

where the viscosity μ and density ρ are assumed to be constant, p is the pressure and $\mathbf{u} \equiv (u_1, u_2, u_3)$ is the velocity. This is a system of four equations in four unknowns:

$$\rho\mathbf{F} - \nabla p + \mu \Delta \mathbf{u} - \rho \mathbf{u} \nabla \mathbf{u} = 0$$
$$\text{div}\ \mathbf{u} = 0.$$

To extend MGLAB to enable the solution of vector equations would require a certain amount of additional house-keeping code, but no new ideas. In the simplest method, one would apply the elements of the multigrid process (relaxation, restriction, *etc.*) in turn to each unknown u_i, whose values are stored in an additional serial dimension of the multi-grid hierarchy. One might have to apply this idea more carefully if the PDE concealed a great deal of interdependence amongst the unknown functions.

These comments also apply to PDE's in n complex variables, which can be regarded as systems of $2n$ real variables.

9.2.4 PDE's in Higher Dimensions

The above example of the Navier-Stokes equations has a three-dimensional domain Ω. This is a very common situation for problems modelling the physical world. The current version of MGLAB is able to solve problems only in two dimensions, although it is obvious how it could be modified to handle d dimensions, in principle. The programming effort required for this would not be great, since the CM Fortran language incorporates the intrinsic array structures of Fortran 90, making the task much easier than it would be using FORTRAN 77, say. Of course, even on a supercomputer such as the Connection Machine, one would exhaust the memory capacity before reaching a very fine grid resolution. For example, on the CM-2 currently situated at the Australian National University, the total amount of memory is 2^{26} double-precision variables. Extending the results of Section 8.3 to $d = 3$, we see that a simple implementation of multigrid could probe no deeper than $\Lambda = 7$ levels.

As a final note regarding PDE's in d dimensions, we mention that the notion of line-relaxation extends to $d-1$ hyperplane relaxation. In three dimensions, we therefore have access to several variations of *plane-relaxation*; these involve the solution of a penta-diagonal system.

9.2.5 Evolution Equations

Suppose that we desire the solution $u(x,t)$ of some time-dependent PDE

$$\frac{\partial u}{\partial t} = \mathcal{F}(u, Du, D^2u, \ldots)$$

such as the Navier-Stokes equation

$$\rho \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = \rho\mathbf{F} - \nabla p + \mu\Delta\mathbf{u}.$$

Time-dependent partial differential equations in three dimensions represent the cutting edge of research in many areas of science, as such problems are difficult and demanding on computer resources.

We will consider only the most basic treatment of evolving systems (for further details, see for example Richtmyer and Morton [86]); to this end, suppose

we wish to solve for $u(x,t)$ the equation

$$\frac{du}{dt} = \lambda u.$$

Time derivatives are discretised almost exclusively using finite differences [39]. Defining $\overline{u} = u(x, t-dt)$, a quantity known from the previous time-step, we can choose to solve one of the following options:

$$\frac{u-\overline{u}}{dt} = \begin{cases} \lambda u & \text{(fully implicit)} \\ \lambda\left(\frac{u+\overline{u}}{2}\right) & \text{(Crank-Nicholson)} \\ \lambda\overline{u} & \text{(fully explicit)} \end{cases}$$

A more complex evolution problem is the two-dimensional *unnormalised Ricci flow* [55]

$$\frac{\partial g_{ij}}{\partial t} = -2K g_{ij}$$

On a 2-torus with flat metric, this may be written as

$$\frac{\partial u}{\partial t} = e^{-u}\Delta u,$$

where $g_{ij}(t) = e^{u(t)} g_{ij}(0)$. Applying the Crank-Nicholson method to this equation gives

$$\frac{u-\overline{u}}{dt} = e^{-(u+\overline{u})/2}\,\tfrac{1}{2}(\Delta u - \Delta\overline{u})$$

in other words

$$u - C_1\,\Delta u\, e^{-u/2}\,dt + C_2\, e^{u/2}\,dt = \overline{u}$$

where

$$C_1 = \tfrac{1}{2}\, e^{-\overline{u}/2} \qquad \text{and} \qquad C_2 = C_1\,\Delta\overline{u}.$$

Hence our time-dependent equation is transformed into a series of static nonlinear problems of the form $\mathcal{A}(u) = f$, where $\mathcal{A}$ and f change at each time-step; this problem is now in a form which MGLAB could solve.

This procedure allows us to solve parabolic PDE's which are elliptic at each time-step: $u_t = \mathcal{F}$, where $\mathcal{F}$ is an elliptic operator.

9.2.6 Finite-Volume Methods

The finite-volume method (see for example Fletcher [39]) is commonly used in the field of computational fluid dynamics (CFD), where the underlying concepts of flow, flux, continuity and conservation are more natural than for elliptic problems. However, we mention the technique since it results in a numerical scheme which is well-suited to the architecture of a hypercube parallel processor, such as the Connection Machine. Recall from Chapter 8 that the use of the standard finite-difference method results in grids of dimension 2^L+1, which we implement on the CM-2 as an implicit boundary scheme with grids of dimension 2^L-1. Using

finite-volume techniques, we work on grids of dimension exactly 2^L, and so this scheme is a natural one for any hypercube computer.

In contrast to the finite-difference method which is applied to a PDE in differential form, the finite-volume method is a discretisation of the governing equation in integral form. For example, one begins with Laplace's equation and integrates over a small subdomain Υ:

$$\int_\Upsilon 1\,(u_{xx} + u_{yy})\,dx\,dy \;=\; \int_\Upsilon (u_x dy - u_y dx)$$

and uses Green's theorem to obtain

$$\int_\Upsilon \mathbf{H} \cdot \mathbf{n}\,ds \;=\; 0$$

where $\mathbf{H} = (u_x, u_y)$ and $\mathbf{n}$ is the outward-pointing normal. One then chooses some discrete representation of this integral. For more details, see Fletcher [39].

The finite-volume method has advantages in situations where the equation obeys conservation properties; in complicated domains using curvilinear coordinates or nonuniform grids; in its ability to produce simple stencils, and also because Neumann boundary conditions can be handled as readily as those of Dirichlet type. Perhaps most interestingly, McCormick [76] and Liu [70] have recently described a hybrid finite-volume element method which is ideal for multigrid, and multilevel adaptive techniques in general. This is a most interesting development, but is beyond the scope of this report.

Chapter 10

Summary and Conclusions

> *I think that computers will bring about great changes in the aspect of both stating and proving theorems .... Certainly I believe in the heuristic or experimental value of computers, where by working examples, one will get intuitions about the more general fact. Ultimately the computers will be able to make formal proofs and operate symbolically the way we do now in thinking about mathematics.*
>
> — Stanislaw Ulam [10]

> *So, is the computer important to mathematics? My answer is "no". It is important, but not to mathematics.*
>
> — Paul Halmos [2]

> *On my list [of the twenty most important events in the history of mathematics] appears the modern computer as one of the very most important mathematical events of all time. ... It would be closer to the truth to say that the development of the electronic computer is one of the major events not just in scientific history, but in world history.*
>
> — Kenneth O. May [72]

This thesis has reported the results of our research into multigrid methods for solving elliptic boundary value problems, and how these methods may be implemented on parallel processing systems. We have generally restricted ourselves to consideration of the case of static scalar two-dimensional Dirichlet problems with finite-difference discretisation, and we have discussed aspects of the implementation of the multigrid algorithm specifically for the massively-parallel Connection Machine CM-2.

We began by giving an overview of the fundamental concepts of discretisation and relaxation for elliptic equations (Chapter 2), then proceeded to discuss the elements which comprise multigrid methods for linear problems (Chapter 3),

and the generalisation to nonlinear problems (Chapter 5). We also considered the theoretical smoothing and convergence properties of relaxation and multigrid iteration. In Section 3.6, we formulated an initial guess based on a discrete-Laplacian function; this proved to give substantial computational benefits for smooth problems with non-constant boundary data. To learn more about the practicalities of multigrid solution, we wrote a software package designed to function as a laboratory for multigrid experiments on a broad range of boundary value problems. `MGLAB` allows the user to freely and conveniently experiment with the many possible multigrid parameters. The standard FORTRAN 77 implementation of this package is discussed in Chapter 3, while the CM Fortran parallel implementation is discussed in detail in Chapter 8. This chapter included a discussion of the architecture of the Connection Machine, which was necessary for understanding the issues of parallel performance. Some CM-2 code is presented in the Appendices, which demonstrate the suitability of the CM Fortran language and the ease of use of `MGLAB`. We selected several sets of model problems to illustrate and examine the multigrid solution process; these ranged from simple linear problems (Chapter 4) to relatively simple nonlinear equations (Chapter 6), to difficult nonlinear problems involving surfaces of prescribed curvature (Chapter 7). We were able to validate the numerical results of `MGLAB` by comparing these with certain theoretical results and published data. Some of the model problems involved singularities; we examined the behaviour of multigrid iteration in these cases. Finally, in Chapter 9 we considered highly-efficient parallel multigrid methods which are currently the focus of much research; also we looked at techniques for extending the range of problems which may be numerically solved using `MGLAB`.

We have found that, while multigrid is not yet a mature field of computational mathematics, it has the potential to become the pre-eminent numerical method — for the class of elliptic PDE's at least. A great deal of effort is currently underway into placing multigrid methods on a firm theoretical basis; substantial progress has been made in this regard. We have demonstrated that serial multigrid is an $O(N)$ method, while parallel multigrid is an $O(\log N)$ method; in other words, multigrid is an asymptotically optimal iteration for elliptic PDE's. It is clear that multigrid is also the most efficient method for well-behaved problems in practice; however, while this may be true for more difficult nonlinear PDE's, the application of multigrid methods to such problems is much less understood.

The efficient implementation of multigrid on massively-parallel computing systems remains a difficult issue; in contrast, it is a simple exercise to implement a fast and efficient multigrid algorithm on a vector supercomputer. Researchers are investigating various hybrid schemes in an attempt to overcome inefficiencies such as the idle-processor problem. Perhaps the advent of "universal" machines such as the CM-5 will foster the development of some highly efficient multigrid-type schemes.

Appendix A
CM Fortran Listing of `mg10.fcm`

This appendix contains a listing of `mg10.fcm`, the CM Fortran implementation of Problem MG10, a type of Poisson equation (see Chapter 4). It gives an example of a driver program, which combines with the back-end multigrid library package `MGLAB` (a partial listing of which appears in Appendix C). We are demonstrating that programs written in CM Fortran are often simpler and more readable than those in FORTRAN 77, and that the subroutines to be written by the user of `MGLAB` can require minimal programming effort. For further details of the organisation and structure of `MGLAB`, and the format of programs which link with it, see Section 3.7.
The program listing is preceded by the two include files `nmax_declarations.inc` and `mg_common.inc`, which are used throughout `MGLAB` and its driver programs. Sample output from the complete program MG10 appears in Appendix B. These listings are for the "unrolled" version of `MGLAB` (see Section 8.4).

```
C========================== Start of 'nmax_declarations.inc' ==========================
C
      LOGICAL red(2:Lmax,nmax,nmax), black(2:Lmax,nmax,nmax)
CMF$  LAYOUT red(:SERIAL,:NEWS,:NEWS), black(:SERIAL,:NEWS,:NEWS)
      DOUBLE PRECISION x(nmax,nmax), y(nmax,nmax), x2(nmax,nmax), y2(nmax,nmax)
CMF$  LAYOUT x(:NEWS,:NEWS), y(:NEWS,:NEWS), x2(:NEWS,:NEWS), y2(:NEWS,:NEWS)
C
      DOUBLE PRECISION right__bc(nmax,nmax), left___bc(nmax,nmax), top____bc(nmax,nmax)
CMF$  LAYOUT right__bc(:NEWS,:NEWS), left___bc(:NEWS,:NEWS), top____bc(:NEWS,:NEWS)
      DOUBLE PRECISION bottom_bc(nmax,nmax)
CMF$  LAYOUT bottom_bc(:NEWS,:NEWS)
C
      DOUBLE PRECISION v(2:Lmax,nmax,nmax), f(2:Lmax,nmax,nmax), old_v(2:Lmax,nmax,nmax)
CMF$  LAYOUT v(:SERIAL,:NEWS,:NEWS), f(:SERIAL,:NEWS,:NEWS), old_v(:SERIAL,:NEWS,:NEWS)
      DOUBLE PRECISION u(nmax,nmax), e(nmax,nmax), r(nmax,nmax)
CMF$  LAYOUT u(:NEWS,:NEWS), e(:NEWS,:NEWS), r(:NEWS,:NEWS)
C
C========================== End of 'nmax_declarations.inc' ==========================
```

```
C============================== Start of 'mg_common.inc' ==============================
C
      DOUBLE PRECISION zero, sixteenth, eighth, quarter, half, one, two, three, four
      DOUBLE PRECISION five, six, pi, pi2
C
      INTEGER max_Lmax, max_nmax
      PARAMETER (max_Lmax = 10, max_nmax = 1023)
C
      INTEGER ni(max_Lmax), step(max_Lmax)
CMF$  LAYOUT ni(:SERIAL), step(:SERIAL)
      DOUBLE PRECISION n(max_Lmax), n2(max_Lmax), h(max_Lmax), h2(max_Lmax)
CMF$  LAYOUT n(:SERIAL), n2(:SERIAL), h(:SERIAL), h2(:SERIAL)
      DOUBLE PRECISION cc_centre(3,max_Lmax), cc_right(3,max_Lmax)
CMF$  LAYOUT cc_centre(:SERIAL,:SERIAL), cc_right(:SERIAL,:SERIAL)
      DOUBLE PRECISION cc__left(3,max_Lmax), cc____up(3,max_Lmax), cc__down(3,max_Lmax)
CMF$  LAYOUT cc__left(:SERIAL,:SERIAL), cc____up(:SERIAL,:SERIAL), cc__down(:SERIAL,:SERIAL)
      INTEGER Lmax, nmax, mid, nu_pre, nu_coarsest, nu_post, nu_guess, num_cycles
      INTEGER user_relax, work_scale, debug, rotate_pause_flag, initial_guess_type
      INTEGER colour_graphics_flag, plot_scale_type
      DOUBLE PRECISION relax_weight, restrict_weight, work, error_2_norm
      DOUBLE PRECISION old_error_2_norm, error_inf_norm, old_error_inf_norm
      DOUBLE PRECISION residue_2_norm, old_residue_2_norm, residue_inf_norm
      DOUBLE PRECISION old_residue_inf_norm, line_relax_tolerance, v_cycle_convergence
      DOUBLE PRECISION u_scale, f_scale, r_scale, plane_a, plane_b, plane_c
      DOUBLE PRECISION v1, f1, e1, r1, u1, top__left, top_right, bot__left, bot_right
      LOGICAL non_linear_PDE, imm_rep, solution_given, graphics
C
      INTEGER num_theta
      PARAMETER (num_theta = 32)
      REAL theta, phi, this_theta     ! Used by CMFB graphics: must be single precision.
C
      CHARACTER*7 relax_type
      CHARACTER*6 restrict_type
      CHARACTER*5 relax_method
      CHARACTER*3 norm_type
      CHARACTER*2 timing_type
      CHARACTER*1 method
C
      COMMON / constants / zero, sixteenth, eighth, quarter, half, one, two, three,
     &                     four, five, six, pi, pi2
      COMMON / front_end / ni, step, n, n2, h, h2, cc_centre, cc_right, cc__left,
     &                     cc____up, cc__down, Lmax, nmax, mid, nu_pre, nu_coarsest,
     &                     nu_post, nu_guess, num_cycles, user_relax, work_scale, debug,
     &                     rotate_pause_flag, initial_guess_type, colour_graphics_flag,
     &                     plot_scale_type, relax_weight, restrict_weight, work,
     &                     error_2_norm, old_error_2_norm, error_inf_norm,
     &                     old_error_inf_norm, residue_2_norm, old_residue_2_norm,
     &                     residue_inf_norm, old_residue_inf_norm,
     &                     line_relax_tolerance, v_cycle_convergence, u_scale, f_scale,
     &                     r_scale, plane_a, plane_b, plane_c, non_linear_PDE, imm_rep,
     &                     solution_given, graphics, theta, phi, this_theta
      COMMON / level_one / v1, f1, e1, r1, u1
      COMMON / corners   / top__left, top_right, bot__left, bot_right
      COMMON / fe_strings/ relax_type, restrict_type, relax_method, norm_type, method,
     &                     timing_type
C
C============================== End of 'mg_common.inc' ================================
```

```
C===============================================================================C
C                                                                               C
C  Solving  - /\ u  =  -2 [ (1-6xx)yy(1-yy) + (1-6yy)xx(1-xx) ]    on (0,1)x(0,1)   C
C             u  =   0                                            on the boundary  C
C                                                                               C
C  with initial guess     v  =  0                                               C
C  The exact solution is  u  =  xxyy(1-xx)(1-yy)                                 C
C                                                                               C
C===============================================================================C
C
      PROGRAM mg10
      IMPLICIT NONE
C
      INCLUDE 'mg_common.inc'
C
      CALL calculate_solution
C
      STOP
      END
C
C===============================================================================
C
      SUBROUTINE print_pde_heading
      IMPLICIT NONE
C
      PRINT*, 'Solving  - /\ u  = -2 [ (1-6xx)yy(1-yy) + (1-6yy)xx(1-xx) ]'
      PRINT*, 'with zero boundary conditions on [0,1]x[0,1]'
      PRINT*, 'with initial guess      v = 0'
      PRINT*, 'The exact solution is   u = xxyy(1-xx)(1-yy)'
C
      RETURN
      END
C
C===============================================================================
C
      SUBROUTINE initialise_pde_params
      IMPLICIT NONE
C
C  Define the parameters and relaxation coefficients of the PDE.
C
      INCLUDE 'mg_common.inc'
C
      INTEGER L
C
      non_linear_PDE = .FALSE.
      solution_given = .TRUE.
C
      DO L = 1, max_Lmax
         cc_centre(2,L) = -four
         cc_right(2,L) = -one
         cc__left(2,L) = -one
         cc____up(2,L) = -one
         cc__down(2,L) = -one
      ENDDO
C
      RETURN
      END
```

```
C
C=======================================================================================
C
      SUBROUTINE initialise_bdy_conds (right__bc, left___bc, top____bc, bottom_bc,
     &                                 x, y, x2, y2)
      IMPLICIT NONE
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
      right__bc = zero
      left___bc = zero
      top____bc = zero
      bottom_bc = zero
C
      top__left = zero
      top_right = zero
      bot__left = zero
      bot_right = zero
C
      RETURN
      END
C
C=======================================================================================
C
      SUBROUTINE user_calc_initial_guess_v (v, x, y, x2, y2, L)
      IMPLICIT NONE
      INTEGER L
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
      IF (L == 1) THEN
         v1 = zero
      ELSE
         v(L,:,:) = zero
      ENDIF
C
      RETURN
      END
C
C=======================================================================================
C
      SUBROUTINE calc_rhs_function_f (f, x, y, x2, y2, L)
      IMPLICIT NONE
      INTEGER L
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
      IF (L == 1) THEN
         f1 = three * eighth
      ELSE
         f(L,:,:)  =  -two *
     &        ( (one - six*x2) * y2 * (one - y2)  +  (one - six*y2) * x2 * (one - x2) )
      ENDIF
C
      RETURN
      END
C
```

```
C=======================================================================================
C
      SUBROUTINE calc_exact_soln_u (u, x, y, x2, y2)
      IMPLICIT NONE
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
      u  =  x2 * y2 * (one-x2) * (one-y2)
C
      RETURN
      END
C
C=======================================================================================
C
      SUBROUTINE user_calc_residue_r
      STOP '### USER_CALC_RESIDUE_R was called!'
      END
C
C=======================================================================================
C
      SUBROUTINE user_relaxation
      STOP '### USER_RELAXATION was called!'
      END
C
C=======================================================================================
C
      SUBROUTINE user_jacobi_relaxation
      STOP '### USER_JACOBI_RELAXATION was called!'
      END
C
C=======================================================================================
C
      SUBROUTINE user_red_black_relaxation
      STOP '### USER_RED_BLACK_RELAXATION was called!'
      END
C
C=======================================================================================
```

Appendix B
Sample Output from `mg10.fcm`

This appendix contains sample output from the program MG10 (see Appendix A). In this case, we are using $1 \times M_8^{2,2,2}$-cycle of $\omega = 0.8$ weighted Jacobi point relaxation and full-weighting restriction to solve this boundary value problem.

```
Solving  - /\ u  = -2 [ (1-6xx)yy(1-yy) + (1-6yy)xx(1-xx) ]
with zero boundary conditions on [0,1]x[0,1]
with initial guess       v = 0
The exact solution is    u = xxyy(1-xx)(1-yy)

 1. Method = m  (P = pure relax, V = V-cycles, W = W-cycles, M = full multigrid)
 2. No of levels = 9                                                    (LMAX)
 4. No of m-cycles =  1                                            (NUM_CYCLES)
 5. No of pre-relaxations (moving to coarser grids) = 2               (NU_PRE)
 6. No of relaxations on the coarsest grid = 2                     (NU_COARSEST)
 7. No of post-relaxations (moving to finer grids) = 2                (NU_POST)
 8. User-relax =  2  (-1=-Lapl, 0=user-defined, 1=+Lapl, 2=auto relax & residue)
 9. Relaxation method = point        (POINT = pointwise, LINE = Jacobi linewise)
10. Relaxation type = jacobi   (JACOBI = Jacobi, GAUSS = red-black Gauss-Seidel)
11. Weight (SOR) factor for relaxation =  0.80000000000000004     (RELAX_WEIGHT)
12. Restriction = fullwt           (INJECT = injection, FULLWT = full weighting)
13. Weighting factor for restriction =  1.00000000000000000    (RESTRICT_WEIGHT)
14. Norm type = two        (INF = infinity-norm, TWO = 2-norm, ALL = both norms)
15. Show internal steps = 0     (0=no, 1=v once, 2=vrf once, 3=v all, 4=vrf all)
16. Use immediate replacement = F                          (T = true, F = false)
17. Scale WORK to level L =  9                                      (WORK_SCALE)
18. Use graphic display = F                                (T = true, F = false)
22. Timing type = cm                    (WU = work-unit cost, CM = CM busy time)
24. Initial guess = 0 (0=user,1=plane,2=diag,3=lin,4=quad,5=quint,6=Lapl,7=soln)

Which option do you wish to change ?  (0 = OK)
0

        Warning: Paris safety has been turned off.
        CM speed = 7.00 MHz
```

Level	Error norm (two) Converg	Residue norm (two) Converg	CM (secs)
1	0.18750000 = -0.73 (0.000)	0.61237244 = -0.21 (0.000)	0.15756
2	.00326471 = -2.49 (.017)	.04084632 = -1.39 (.066)	2.83482
3	.00140017 = -2.85 (.429)	.04950647 = -1.31 (1.212)	9.27129
4	.00049932 = -3.30 (.357)	.03857822 = -1.41 (.779)	18.54574
5	.00015791 = -3.80 (.316)	.02756616 = -1.56 (.715)	30.36103
6	.00004629 = -4.33 (.293)	.01951418 = -1.71 (.708)	44.59921
7	.00001292 = -4.89 (.279)	.01383591 = -1.86 (.709)	61.23992
8	.00000349 = -5.46 (.270)	.00983982 = -2.01 (.711)	80.24603
9	.00000092 = -6.03 (.265)	.00703635 = -2.15 (.715)	101.60307

Breakdown of CM-Busy Times (seconds)

Relaxation (point jacobi)	61.788	60 %
Residue calculation	13.971	14 %
Restriction (fullwt)	9.082	9 %
Prolongation (bilinear interp)	7.198	7 %
Result calculation	5.058	5 %
Other V-cycle calculation	3.202	3 %
Other multigrid calculation	1.844	2 %
Total of the above times	102.143	100 %

Total CM-busy time = 101.603 sec
Total CM-elapsed time = 103.320 sec

Appendix C
CM Fortran Listing of `mglab.fcm`

This appendix contains a partial listing of `mglab.fcm`, the CM Fortran implementation of the back-end multigrid library package MGLAB. For further details of the organisation and structure of MGLAB, see Section 3.7. A complete listing was not feasible due to its length, nor desirable since much of the code is of little interest. We list here the kernel routines essential to the multigrid process: relaxation, restriction, prolongation, coarse-grid correction, and so on.

```
C=============================================================================C
C                                                                             C
C     2-D  MULTIGRID  LIBRARY  ROUTINE  PACKAGE                               C
C                                                                             C
C              (Double Precision Connection Machine CM-2 Version)             C
C                                                                             C
C=============================================================================C
C                                                                             C
C             Author  :  Nicholas Keith Spencer                               C
C             Date    :  May 1991                                             C
C             Place   :  Australian National University                       C
C                                                                             C
C=============================================================================C
C                                                                             C
C     Purpose and Structure of this Package:                                  C
C     ....                                                                    C
C                                                                             C
C=============================================================================C
C                                                                             C
C  This package contains the following routines:                              C
C                                                                             C
C     SUBROUTINE calculate_solution                                           C
C     SUBROUTINE general_initialisation                                       C
C     SUBROUTINE initialise_constants                                         C
C     SUBROUTINE read_user_options                                            C
C     SUBROUTINE solve_pde                                                    C
C     SUBROUTINE specific_initialisation                                      C
C     SUBROUTINE print_heading                                                C
C     SUBROUTINE calc_initial_guess_v (v, L)                                  C
C     SUBROUTINE v_cycle (v, f, r, u, e, old_v, start_L, stop_L)              C
C     SUBROUTINE relax_v (v, f, L, num_sweeps, depth)                         C
C     SUBROUTINE relax_on_level_1 (num_sweeps, depth)                         C
```

```
C      SUBROUTINE jacobi_point_relax (v, f, L, num_sweeps, depth)                C
C      SUBROUTINE rb_gauss_seidel_point_relax (v, f, L, num_sweeps, depth)       C
C      SUBROUTINE horiz_jacobi_line_relax (v, f, L, num_sweeps, depth)           C
C      SUBROUTINE vert_jacobi_line_relax (v, f, L, num_sweeps, depth)            C
C      SUBROUTINE calc_residue_r (v, f, r, L, depth)                             C
C      SUBROUTINE restrict_r_to_f (f, r, L)                                      C
C      SUBROUTINE injection_restriction_of_r_to_f (f, r, L)                      C
C      SUBROUTINE weighted_restriction_of_r_to_f (f, r, L)                       C
C      SUBROUTINE immediate_correction (v, f, r, u, e, old_v, Ltop, L)           C
C      SUBROUTINE bilinear_interpolate_v (v, L)                                  C
C      SUBROUTINE coarse_grid_correct_v (v, L)                                   C
C      SUBROUTINE calc_result (v, f, r, u, e, L, immediate_correction, from_L)   C
C      DOUBLE PRECISION FUNCTION log_10 (x)                                      C
C      SUBROUTINE plot_vf (input_array, L, depth, vf)                            C
C      SUBROUTINE plot_ur (input_array, ur)                                      C
C      SUBROUTINE pause                                                          C
C                                                                                C
C================================================================================C
C
      SUBROUTINE calculate_solution
      IMPLICIT NONE
C
      INCLUDE 'mg_common.inc'
C
      CALL general_initialisation
      CALL initialise_pde_params
      CALL print_pde_heading
      CALL read_user_options
      CALL solve_pde
C
      RETURN
      END
C
C================================================================================
C
      SUBROUTINE general_initialisation
C
C  Initialise problem-independent data structures and parameters.
C
      ....
C
C================================================================================
C
      SUBROUTINE initialise_constants
C
C  Initialise the common block 'constants'.
C
      ....
C
C================================================================================
C
      SUBROUTINE read_user_options
C
C  Query the user for the desired way to solve the PDE.  Validate these options.
C
      ....
C
C================================================================================
```

```
C
      SUBROUTINE solve_pde
C
C  Solve the PDE using the multigrid parameters defined by the user.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE specific_initialisation (red, black, x, y, x2, y2, right__bc,
     &                                    left___bc, top____bc, bottom_bc)
C
C  Initialise problem-specific data structures and parameters.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE print_heading
C
C  Print the appropriate heading line(s) for subsequent results.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE calc_initial_guess_v (v, right__bc, left___bc, top____bc, bottom_bc,
     &                                 x, y, x2, y2, L)
C
C  Choose one of several possible methods for calculating an initial guess v(L,:,:).
C
      ....
      ELSEIF (initial_guess_type == 6) THEN
         x1 = one - x
         y1 = one - y
         x12 = x2 * x1 * x1
         y12 = y2 * y1 * y1
         FORALL (i = 1 : nmax, j = 1 : nmax)
     &      v(L,i,j)  =  ( x12(i,j) * (top____bc(i,1)*y(i,j) + bottom_bc(i,1)*y1(i,j))
     &                   + y12(i,j) * (right__bc(1,j)*x(i,j) + left___bc(1,j)*x1(i,j)) )
/  ( x12(i,j) + y12(i,j) )
      ....
C
C=======================================================================================
C
      SUBROUTINE v_cycle (v, f, r, u, e, old_v, x, y, x2, y2, red, black, right__bc,
     &                    left___bc, top____bc, bottom_bc, start_L, stop_L)
      IMPLICIT NONE
      INTEGER start_L, stop_L
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  Perform a V-cycle from level 'start_L' down to level 1, and back up to level 'stop_L'.
C  Normally start_L = stop_L; otherwise this indicates we are doing a W-cycle composed
C  of several pieces of V-cycles.  We do 'nu_pre' relaxations moving to the coarsest
C  grid, 'nu_coarsest' relaxations on the coarsest grid, and 'nu_post' relaxations
C  moving back to the finest grid.  'imm_rep' indicates immediate replacement: an error
```

```
C  correction is made directly to the fine grid after every coarse grid relaxation
C  (simply for display purposes - this has no effect on the iteration).  We must have
C  V         =  v  (the discretized fine-grid current guess) upon entry to this routine.
C   start_L     j
C
      INTEGER L
      DOUBLE PRECISION old_f1
      CHARACTER*13 word(4)
      DOUBLE PRECISION av(nmax,nmax), old_f(nmax,nmax)      ! Used only for FAS cycle.
CMF$  LAYOUT av(:NEWS,:NEWS), old_f(:NEWS,:NEWS)
C
      DATA word(1) / 'neg Laplacian' /
      DATA word(2) / 'user-defined ' /
      DATA word(3) / 'pos Laplacian' /
      DATA word(4) / 'auto-defined ' /
C
      IF (timing_type == 'cm') THEN
         CALL cm_timer_stop (7)
         CALL cm_timer_start (6)
      ENDIF
C
      IF ( (start_L < 2) .OR. (start_L > Lmax) ) STOP '### V_CYCLE: start_L not in [2,Lmax]'
      IF ( (stop_L < 2) .OR. (stop_L > Lmax) ) STOP '### V_CYCLE: stop_L not in [2,Lmax]'
C
C  V-cycle initialisation.
C
      IF (.NOT. non_linear_PDE) THEN
         v1 = zero
         v(2:start_L-1,:,:) = zero
      ENDIF
      IF (start_L == stop_L) CALL calc_rhs_function_f (f, x, y, x2, y2, start_L)
C
      IF (debug >= 3) THEN
         WRITE(*,10) start_L
10       FORMAT(/1X, 'At the start of this V(', I1, ')-cycle:')
         IF (graphics) CALL plot_vf (v, right__bc, left___bc, top____bc, bottom_bc,
     &                               start_L, 0, 'v')
      ENDIF
      IF ( (debug == 4) .AND. graphics ) THEN
         CALL plot_vf (f, right__bc, left___bc, top____bc, bottom_bc, start_L, 0, 'f')
         CALL calc_residue_r (v, f, r, right__bc, left___bc, top____bc, bottom_bc,
     &                        x, y, x2, y2, start_L, 0)
         CALL plot_ur (r, 'r')
      ENDIF
C
C  Step A: Moving from the fine grid (L=start_L) to the coarsest grid (L=1), do
C  ------  (1) 'nu_pre' x pre-relaxations  v(L) = Rel(v(L),f(L))
C          (2) calculate the residue  r = f(L) - A(L).v(L)
C          (3) restrict the residue to the coarser grid  f(L-1) = I(L,L-1) r
C
      DO L = start_L, 2, -1
C
C  (1) 'nu_pre' x pre-relaxations  v(L) = Rel(v(L),f(L))
C
         IF ( .NOT. ( (start_L<>stop_L) .AND. (start_L==L) ) ) THEN
            IF (timing_type == 'cm') THEN
               CALL cm_timer_stop (6)
               CALL cm_timer_start (1)
```

```
            ENDIF
            CALL relax_v (v, f, right__bc, left___bc, top____bc, bottom_bc, red, black,
     &                    x, y, x2, y2, L, nu_pre, start_L-L)
            IF (timing_type == 'cm') THEN
               CALL cm_timer_stop (1)
               CALL cm_timer_start (6)
            ENDIF
            IF (debug >= 3) THEN
               WRITE(*,20) nu_pre, relax_type, relax_method, word(user_relax+2)
20             FORMAT(/1X, '1. After', I2, 1X, A7, 1X, A5, 1X, A13, ' relaxations:')
               IF (graphics) CALL plot_vf (v, right__bc, left___bc, top____bc,
     &                                     bottom_bc, L, start_L-L, 'v')
            ENDIF
            IF (imm_rep) CALL immediate_correction (v, f, r, u, e, old_v, right__bc,
     &                                              left___bc, top____bc, bottom_bc,
     &                                              x, y, x2, y2, start_L, L)
         ENDIF
C
C  (2) calculate the residue  r = f(L) - A(L).v(L)
C
         IF (timing_type == 'cm') THEN
            CALL cm_timer_stop (6)
            CALL cm_timer_start (2)
         ENDIF
         CALL calc_residue_r (v, f, r, right__bc, left___bc, top____bc, bottom_bc,
     &                        x, y, x2, y2, L, start_L-L)
         IF (timing_type == 'cm') THEN
            CALL cm_timer_stop (2)
            CALL cm_timer_start (6)
         ENDIF
         IF ( (debug == 4) .AND. graphics )  CALL plot_ur (r, 'r')
C
C  (3) restrict the residue to the coarser grid  f(L-1) = I(L,L-1) r
C
         IF (timing_type == 'cm') THEN
            CALL cm_timer_stop (6)
            CALL cm_timer_start (3)
         ENDIF
         CALL restrict_r_to_f (f, r, L)
         IF (non_linear_PDE) THEN        ! Use Full Approximation Scheme (FAS):
            IF (L == 2) THEN
               old_f1 = f1
               f1 = zero
               v1 = v(2,mid,mid)
               CALL calc_residue_r (v, f, av, right__bc, left___bc, top____bc, bottom_bc,
     &                              x, y, x2, y2, L-1, 0)
               f1 = old_f1 - av(mid,mid)
            ELSE
               old_f = f(L-1,:,:)              ! Save a copy of just-calculated f(L-1)
               f(L-1,:,:) = zero               ! Prepare to call special calc_residue_r
               v(L-1,:,:) = v(L,:,:)           ! Injected initial guess on coarse grid
               CALL calc_residue_r (v, f, av, right__bc, left___bc, top____bc, bottom_bc,
     &                              x, y, x2, y2, L-1, 0)
                                               ! Calc temp av = -A*v(L-1) [actual BC's]
               f(L-1,:,:) = old_f - av         ! f(L-1) = f(L-1) + A*v(L-1)
            ENDIF
            IF ( (debug == 4) .AND. graphics ) THEN
               CALL plot_vf (f, right__bc, left___bc, top____bc, bottom_bc, L-1, 0, 'f')
```

```
            CALL plot_vf (v, right__bc, left___bc, top____bc, bottom_bc, L-1, 0, 'v')
         ENDIF
      ENDIF
      IF (timing_type == 'cm') THEN
         CALL cm_timer_stop (3)
         CALL cm_timer_start (6)
      ENDIF
      IF (debug == 4) THEN
         WRITE(*,30) restrict_weight, restrict_type
30       FORMAT(/1X, '2. After', F5.2, ' * ', A6, ' restriction:')
         IF (graphics) CALL plot_vf (f, right__bc, left___bc, top____bc, bottom_bc,
     &                               L-1, start_L-L+1, 'f')
      ENDIF
C
      ENDDO
C
C  Step B: On the coarsest grid (L=1), do
C  ------  (1) 'nu_coarsest' x relaxations  v(L) = Rel(v(L),f(L))
C
      IF (timing_type == 'cm') THEN
         CALL cm_timer_stop (6)
         CALL cm_timer_start (1)
      ENDIF
      CALL relax_v (v, f, right__bc, left___bc, top____bc, bottom_bc, red, black, x, y,
     &              x2, y2, 1, nu_coarsest, start_L-1)
      IF (timing_type == 'cm') THEN
         CALL cm_timer_stop (1)
         CALL cm_timer_start (6)
      ENDIF
      IF (debug >= 3) THEN
         WRITE(*,40) nu_coarsest, relax_type, relax_method, word(user_relax+2)
40       FORMAT(/1X, '3. After', I2, 1X, A7, 1X, A5, 1X, A13,
     &          ' relaxations on the coarsest grid:')
         IF (graphics) CALL plot_vf (v, right__bc, left___bc, top____bc, bottom_bc, 1,
     &                               start_L-1, 'v')
      ENDIF
      IF ( (debug == 4) .AND. graphics ) THEN
         CALL calc_residue_r (v, f, r, right__bc, left___bc, top____bc, bottom_bc, x,
     &                        y, x2, y2, 1, start_L-1)
         CALL plot_ur (r, 'r')
      ENDIF
      IF (imm_rep) CALL immediate_correction (v, f, r, u, e, old_v, right__bc,
     &                                        left___bc, top____bc, bottom_bc,
     &                                        x, y, x2, y2, start_L, L)
C
C  Step C: Moving from the coarsest grid (L=1) to the fine grid (L=stop_L), do
C  ------  (1) coarse-grid correction  v(L) = v(L) + I(L-1,L) v(L-1)
C          (2) 'nu_post' x post-relaxations  v(L) = Rel(v(L),f(L))
C
      DO L = 2, stop_L
C
C  (1) coarse-grid correction  v(L) = v(L) + I(L-1,L) v(L-1)
C
         IF (timing_type == 'cm') THEN
            CALL cm_timer_stop (6)
            CALL cm_timer_start (4)
         ENDIF
         IF (non_linear_PDE) THEN
```

```
            IF (L == 2) THEN
               v1 = v1 - v(2,mid,mid)
            ELSE
               v(L-1,:,:) = v(L-1,:,:) - v(L,:,:)
            ENDIF
         ENDIF
         CALL coarse_grid_correct_v (v, L)
         IF (timing_type == 'cm') THEN
            CALL cm_timer_stop (4)
            CALL cm_timer_start (6)
         ENDIF
         IF (debug >= 3) THEN
            WRITE(*,50)
50          FORMAT(/1X, '4. After coarse-grid correction by bilinear interpolation:')
            IF (graphics) CALL plot_vf (v, right__bc, left___bc, top____bc, bottom_bc,
     &                                  L, stop_L-L, 'v')
         ENDIF
         IF ( (debug == 4) .AND. graphics ) THEN
            CALL calc_residue_r (v, f, r, right__bc, left___bc, top____bc, bottom_bc,
     &                           x, y, x2, y2, L, stop_L-L)
            CALL plot_ur (r, 'r')
         ENDIF
C
C  (2) 'nu_post' x post-relaxations  v(L) = Rel(v(L),f(L))
C
         IF (timing_type == 'cm') THEN
            CALL cm_timer_stop (6)
            CALL cm_timer_start (1)
         ENDIF
         CALL relax_v (v, f, right__bc, left___bc, top____bc, bottom_bc, red, black,
     &                 x, y, x2, y2, L, nu_post, stop_L-L)
         IF (timing_type == 'cm') THEN
            CALL cm_timer_stop (1)
            CALL cm_timer_start (6)
         ENDIF
         IF (debug >= 3) THEN
            WRITE(*,60) nu_post, relax_type, relax_method, word(user_relax+2)
60          FORMAT(/1X, '5. After', I2, 1X, A7, 1X, A5, 1X, A13, ' relaxations:')
            IF (graphics) CALL plot_vf (v, right__bc, left___bc, top____bc, bottom_bc,
     &                                  L, stop_L-L, 'v')
         ENDIF
         IF ( (debug == 4) .AND. graphics ) THEN
            CALL calc_residue_r (v, f, r, right__bc, left___bc, top____bc, bottom_bc,
     &                           x, y, x2, y2, L, stop_L-L)
            CALL plot_ur (r, 'r')
         ENDIF
         IF ( imm_rep .AND. (L <> stop_L) )
     &      CALL immediate_correction (v, f, r, u, e, old_v, right__bc, left___bc,
     &                                 top____bc, bottom_bc, x, y, x2, y2, stop_L, L)
C
      ENDDO
C
C
C
      IF (timing_type == 'cm') THEN
         CALL cm_timer_stop (6)
         CALL cm_timer_start (5)
      ENDIF
```

```
      CALL calc_result (v, f, r, u, e, right__bc, left___bc, top____bc, bottom_bc,
     &                  x, y, x2, y2, stop_L, .FALSE., 0)
      IF (timing_type == 'cm') THEN
         CALL cm_timer_stop (5)
         CALL cm_timer_start (6)
      ENDIF
C
      IF (debug > 0) THEN
         WRITE(*,80) stop_L
80       FORMAT(/1X, 'At the end of this V(', I1, ')-cycle:')
         IF (graphics) CALL plot_vf (v, right__bc, left___bc, top____bc, bottom_bc,
     &                               stop_L, 0, 'v')
      ENDIF
      IF ( (debug == 2) .OR. (debug == 4) ) THEN
         CALL calc_residue_r (v, f, r, right__bc, left___bc, top____bc, bottom_bc,
     &                        x, y, x2, y2, stop_L, 0)
         IF (graphics) CALL plot_ur (r, 'r')
      ENDIF
C
      IF (timing_type == 'cm') THEN
         CALL cm_timer_stop (6)
         CALL cm_timer_start (7)
      ENDIF
C
      RETURN
      END
C
C=======================================================================================
C
      SUBROUTINE relax_v (v, f, right__bc, left___bc, top____bc, bottom_bc, red, black,
     &                    x, y, x2, y2, L, num_sweeps, depth)
C
C  Select the appropriate relaxation routine.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE relax_on_level_1 (right__bc, left___bc, top____bc, bottom_bc,
     &                             num_sweeps, depth)
      IMPLICIT NONE
      INTEGER num_sweeps, depth
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  V(1) is updated by 'num_sweeps' of weighted relaxation.
C
      INTEGER ur
      DOUBLE PRECISION wbar, wtemp
C
      wbar = one - relax_weight
      IF (user_relax == 2) THEN
         ur = 2
      ELSE
         ur = user_relax + 2
      ENDIF
      wtemp = relax_weight / cc_centre(ur,1)
C
```

```
      IF ( (depth == 0) .OR. non_linear_PDE ) THEN     ! Use actual boundary conditions.
         DO (num_sweeps) TIMES
            v1  =  wtemp * (  cc_right(ur,1) * right__bc(nmax,mid)
     &                      + cc__left(ur,1) * left___bc(1,mid)
     &                      + cc____up(ur,1) * top____bc(mid,nmax)
     &                      + cc__down(ur,1) * bottom_bc(mid,1)
     &                      - f1 * h2(1)  )
     &           + wbar * v1
         ENDDO
      ELSE     ! Use zero boundary conditions (since this is a coarse grid).
         DO (num_sweeps) TIMES
            v1  =  -wtemp * f1 * h2(1)  +  wbar * v1
         ENDDO
      ENDIF
C
      RETURN
      END
C
C=======================================================================================
C
      SUBROUTINE jacobi_point_relax (v, f, right__bc, left___bc, top____bc, bottom_bc,
     &                               L, num_sweeps, depth)
      IMPLICIT NONE
      INTEGER L, num_sweeps, depth
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  V(L) is updated by 'num_sweeps' of pointwise weighted (SOR) Jacobi relaxation.
C
      INTEGER ur, s
      DOUBLE PRECISION wbar, wtemp
C
      s = step(L)
      IF (user_relax == 2) THEN
         ur = 2
      ELSE
         ur = user_relax + 2
      ENDIF
      wbar = one - relax_weight
      wtemp = relax_weight / cc_centre(ur,L)
C
      IF ( (depth == 0) .OR. non_linear_PDE ) THEN     ! Use actual boundary conditions.
         DO (num_sweeps) TIMES
            v(L,:,:)  =  wtemp * (  cc_right(ur,L) *
     &           EOSHIFT(v(L,:,:), DIM = 1, SHIFT = +s, BOUNDARY = right__bc(nmax,:))
     &                      + cc__left(ur,L) *
     &           EOSHIFT(v(L,:,:), DIM = 1, SHIFT = -s, BOUNDARY = left___bc(1,:))
     &                      + cc____up(ur,L) *
     &           EOSHIFT(v(L,:,:), DIM = 2, SHIFT = +s, BOUNDARY = top____bc(:,nmax))
     &                      + cc__down(ur,L) *
     &           EOSHIFT(v(L,:,:), DIM = 2, SHIFT = -s, BOUNDARY = bottom_bc(:,1))
     &                      - f(L,:,:) * h2(L)  )
     &               + wbar * v(L,:,:)
         ENDDO
      ELSE     ! Use zero boundary conditions (since this is a coarse grid).
         DO (num_sweeps) TIMES
            v(L,:,:)  =  wtemp * (  cc_right(ur,L) *
     &                    EOSHIFT(v(L,:,:), DIM = 1, SHIFT = +s, BOUNDARY = zero)
```

```
     &                          + cc__left(ur,L) *
     &                        EOSHIFT(v(L,:,:), DIM = 1, SHIFT = -s, BOUNDARY = zero)
     &                          + cc____up(ur,L) *
     &                        EOSHIFT(v(L,:,:), DIM = 2, SHIFT = +s, BOUNDARY = zero)
     &                          + cc__down(ur,L) *
     &                        EOSHIFT(v(L,:,:), DIM = 2, SHIFT = -s, BOUNDARY = zero)
     &                          - f(L,:,:) * h2(L)  )
     &              + wbar * v(L,:,:)
        ENDDO
      ENDIF
C
      RETURN
      END
C
C=======================================================================================
C
      SUBROUTINE rb_gauss_seidel_point_relax (v, f, right__bc, left___bc, top____bc,
     &                                        bottom_bc, red, black, L, num_sweeps, depth)
C
C  V(L) is updated by pointwise weighted (SOR) red-black Gauss-Seidel relaxation.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE horiz_jacobi_line_relax (v, f, right__bc, left___bc, top____bc,
     &                                    bottom_bc, L, num_sweeps, depth)
      IMPLICIT NONE
      INTEGER L, num_sweeps, depth
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  V(L) is updated by 'num_sweeps' of horizontal linewise weighted Jacobi relaxation.
C
      INCLUDE '/usr/include/cm/cmssl-cmf.h'
C
      INTEGER s, us, status
      DOUBLE PRECISION wbar
      DOUBLE PRECISION temp_row(nmax,nmax)
CMF$  LAYOUT temp_row(:NEWS,:NEWS)
      DOUBLE PRECISION upper_diag(nmax,nmax), diag(nmax,nmax), lower_diag(nmax,nmax)
CMF$  LAYOUT upper_diag(:NEWS,:NEWS), diag(:NEWS,:NEWS), lower_diag(:NEWS,:NEWS)
C
      s = step(L)
      IF (user_relax == 2) THEN
         us = 2
      ELSE
         us = user_relax + 2
      ENDIF
      wbar = one - relax_weight
C
C  Assemble the tridiagonal matrix.
C
      upper_diag = -cc_right(us,L)
      diag = cc_centre(us,L)
      lower_diag = -cc__left(us,L)
C
      DO (num_sweeps) TIMES
```

```
C
C  Assemble temp_row.
C
        IF ( (depth == 0) .OR. non_linear_PDE ) THEN      ! Use actual boundary conds.
           temp_row   =   cc____up(us,L) *
     &              EOSHIFT(v(L,:,:), DIM = 2, SHIFT = +s, BOUNDARY = top____bc(:,nmax))
     &                  +  cc__down(us,L) *
     &              EOSHIFT(v(L,:,:), DIM = 2, SHIFT = -s, BOUNDARY = bottom_bc(:,1))
     &                  -  f(L,:,:) * h2(L)
        ELSE    ! Use zero boundary conditions (since this is a coarse grid).
           temp_row   =   cc____up(us,L) *
     &                     EOSHIFT(v(L,:,:), DIM = 2, SHIFT = +s, BOUNDARY = zero)
     &                  +  cc__down(us,L) *
     &                     EOSHIFT(v(L,:,:), DIM = 2, SHIFT = -s, BOUNDARY = zero)
     &                  -  f(L,:,:) * h2(L)
        ENDIF
C
C  Incorporate the left and right boundary conditions for each temp_row.
C
        IF ( (depth == 0) .OR. non_linear_PDE ) THEN      ! Use actual boundary conds.
           temp_row(s,:)  =  temp_row(s,:)  +  cc__left(us,L) * left___bc(1,:)
           temp_row(ni(L)*s,:)
     &                    =  temp_row(ni(L)*s,:)  +  cc_right(us,L) * right__bc(nmax,:)
        ENDIF
C
C  Update each temp_row by solving the ni(L) tridiagonal systems.
C
        status = 0
        CALL gen_tridiag_solve (temp_row(s:nmax:s,s:nmax:s),
     &                          upper_diag(s:nmax:s,s:nmax:s),
     &                          diag(s:nmax:s,s:nmax:s),
     &                          lower_diag(s:nmax:s,s:nmax:s),
     &                          1, line_relax_tolerance, status)
        IF (status <> 0)
     &     STOP '### HORIZ_JACOBI_LINE_RELAX: error occured in GEN_TRIDIAG_SOLVE'
C
C  Update v.
C
        v(L,s:nmax:s,s:nmax:s)  =  wbar * v(L,s:nmax:s,s:nmax:s)
     &                          + relax_weight * temp_row(s:nmax:s,s:nmax:s)
      ENDDO
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE vert_jacobi_line_relax (v, f, right__bc, left___bc, top____bc,
     &                                   bottom_bc, L, num_sweeps, depth)
C
C  V(L) is updated by 'num_sweeps' of vertical linewise weighted Jacobi relaxation.
C
      ....
C
C=============================================================================
C
      SUBROUTINE calc_residue_r (v, f, r, right__bc, left___bc, top____bc, bottom_bc,
     &                           x, y, x2, y2, L, depth)
```

```
      IMPLICIT NONE
      INTEGER L, depth
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  r  =  f  -  A *v
C         L     L  L
C
      INTEGER s
C
      IF (user_relax <> 2) THEN
         CALL user_calc_residue_r (v, f, r, right__bc, left___bc, top____bc, bottom_bc,
     &                             x, y, x2, y2, L, depth)
         RETURN
      ENDIF
C
      IF (L == 1) THEN
         IF ( (depth == 0) .OR. non_linear_PDE ) THEN     ! Use actual boundary conditions.
            r1  =  f1  -  n2(L) * (  cc_right(2,L) * right__bc(nmax,mid)
     &                             + cc__left(2,L) * left___bc(1,mid)
     &                             + cc____up(2,L) * top____bc(mid,nmax)
     &                             + cc__down(2,L) * bottom_bc(mid,1)
     &                             - cc_centre(2,L) * v1  )
         ELSEIF (depth > 0) THEN     ! Use zero boundary conditions (this is a coarse grid).
            r1  =  f1  +  n2(L) * cc_centre(2,L) * v1
         ELSE
            STOP '### CALC_RESIDUE_R: depth < 0'
         ENDIF
         RETURN
      ENDIF
C
      s = step(L)
      IF ( (depth == 0) .OR. non_linear_PDE ) THEN     ! Use actual boundary conditions.
         r  =  f(L,:,:)  -  n2(L) * (  cc_right(2,L) *
     &              EOSHIFT(v(L,:,:), DIM = 1, SHIFT = +s, BOUNDARY = right__bc(nmax,:))
     &                             + cc__left(2,L) *
     &              EOSHIFT(v(L,:,:), DIM = 1, SHIFT = -s, BOUNDARY = left___bc(1,:))
     &                             + cc____up(2,L) *
     &              EOSHIFT(v(L,:,:), DIM = 2, SHIFT = +s, BOUNDARY = top____bc(:,nmax))
     &                             + cc__down(2,L) *
     &              EOSHIFT(v(L,:,:), DIM = 2, SHIFT = -s, BOUNDARY = bottom_bc(:,1))
     &                             - cc_centre(2,L) * v(L,:,:)  )
      ELSEIF (depth > 0) THEN     ! Use zero boundary conditions (this is a coarse grid).
         r  =  f(L,:,:)  -  n2(L) * (  cc_right(2,L) *
     &                        EOSHIFT(v(L,:,:), DIM = 1, SHIFT = +s, BOUNDARY = zero)
     &                             + cc__left(2,L) *
     &                        EOSHIFT(v(L,:,:), DIM = 1, SHIFT = -s, BOUNDARY = zero)
     &                             + cc____up(2,L) *
     &                        EOSHIFT(v(L,:,:), DIM = 2, SHIFT = +s, BOUNDARY = zero)
     &                             + cc__down(2,L) *
     &                        EOSHIFT(v(L,:,:), DIM = 2, SHIFT = -s, BOUNDARY = zero)
     &                             - cc_centre(2,L) * v(L,:,:)  )
      ELSE
         STOP '### CALC_RESIDUE_R: depth < 0'
      ENDIF
C
      RETURN
      END
```

```
C
C=======================================================================================
C
      SUBROUTINE restrict_r_to_f (f, r, L)
C
C  Select the appropriate restriction routine.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE injection_restriction_of_r_to_f (f, r, L)
      IMPLICIT NONE
      INTEGER L
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  F(L-1) = I(L,L-1) R,   where I = restriction by 'restrict_weight'-injection.
C
      IF (L == 2) THEN
         f1 = restrict_weight * r(mid,mid)
      ELSE
         f(L-1,:,:) = restrict_weight * r
      ENDIF
C
      RETURN
      END
C
C=======================================================================================
C
      SUBROUTINE weighted_restriction_of_r_to_f (f, r, L)
      IMPLICIT NONE
      INTEGER L
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  F(L-1) = I(L,L-1) R,  where I = restriction by 'restrict_weight'-full-weighting.
C
      INTEGER i1, i2, i3, s
      DOUBLE PRECISION stencil(nmax,nmax)
CMF$  LAYOUT stencil(:NEWS,:NEWS)
C
      IF (L == 2) THEN
         i2 = mid
         i1 = i2 / 2
         i3 = 3 * i1
         f1  =  sixteenth * restrict_weight *
     &                    (  four * r(i2,i2)
     &                     + two * (r(i1,i2) + r(i2,i1) + r(i3,i2) + r(i2,i3))
     &                     + r(i1,i1) + r(i1,i3) + r(i3,i1) + r(i3,i3)  )
         RETURN
      ENDIF
C
      s = step(L)
      stencil = quarter *
     &          (  EOSHIFT(r, DIM = 1, SHIFT = +s, BOUNDARY = zero)
     &           + two * r
     &           + EOSHIFT(r, DIM = 1, SHIFT = -s, BOUNDARY = zero) )
```

```
      stencil = quarter *
     &          (  EOSHIFT(stencil,  DIM = 2, SHIFT = +s, BOUNDARY = zero)
     &           + two * stencil
     &           + EOSHIFT(stencil,  DIM = 2, SHIFT = -s, BOUNDARY = zero) )
      f(L-1,:,:) = restrict_weight * stencil
C
      RETURN
      END
C
C=======================================================================================
C
      SUBROUTINE immediate_correction (v, f, r, u, e, old_v, right__bc, left___bc,
     &                                 top____bc, bottom_bc, x, y, x2, y2, Ltop, L)
C
C  newv(Ltop) = v(Ltop) + I(Ltop,L) v(L) ,   where I = bilinear interpolation.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE bilinear_interpolate_v (v, right__bc, left___bc, top____bc, bottom_bc, L)
      IMPLICIT NONE
      INTEGER L
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  V(L) = I(L-1,L) V(L-1) ,   where I = bilinear interpolation.
C
      INTEGER i1, i2, i3, s, t
      DOUBLE PRECISION v1l, v1r, v1b, v1t
C
      IF (L == 2) THEN
         i2 = mid
         i1 = i2 / 2
         i3 = 3 * i1
         v1l = left___bc(1,i2)
         v1r = right__bc(nmax,i2)
         v1b = bottom_bc(i2,1)
         v1t = top____bc(i2,nmax)
         v(2,i2,i2) = v1
         v(2,i1,i2) = half * (v1l + v1)
         v(2,i3,i2) = half * (v1r + v1)
         v(2,i2,i1) = half * (v1b + v1)
         v(2,i2,i3) = half * (v1t + v1)
         v(2,i1,i1) = quarter * (bot__left + v1l + v1b + v1)
         v(2,i1,i3) = quarter * (bot_right + v1r + v1b + v1)
         v(2,i3,i1) = quarter * (top__left + v1l + v1t + v1)
         v(2,i3,i3) = quarter * (top_right + v1r + v1t + v1)
         RETURN
      ENDIF
C
      s = step(L-1)
      t = step(L)
      v(L,:,:) = zero
      v(L,s:nmax:s,s:nmax:s) = v(L-1,s:nmax:s,s:nmax:s)
      v(L,:,:)  =  v(L,:,:)  +  half *
     &          (  EOSHIFT(v(L,:,:), DIM = 1, SHIFT = +t, BOUNDARY = right__bc(nmax,:))
     &           + EOSHIFT(v(L,:,:), DIM = 1, SHIFT = -t, BOUNDARY = left___bc(1,:))   )
```

```
      v(L,:,:)  =  v(L,:,:)  +  half *
     &          (  EOSHIFT(v(L,:,:), DIM = 2, SHIFT = +t, BOUNDARY = top____bc(:,nmax))
     &           + EOSHIFT(v(L,:,:), DIM = 2, SHIFT = -t, BOUNDARY = bottom_bc(:,1))   )
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE coarse_grid_correct_v (v, L)
      IMPLICIT NONE
      INTEGER L
      INCLUDE 'mg_common.inc'
      INCLUDE 'nmax_declarations.inc'
C
C  V(L) = V(L) + I(L-1,L) V(L-1) ,   where I = bilinear interpolation.
C
      INTEGER i1, i2, i3, s, t
      DOUBLE PRECISION temp_v(nmax,nmax)
CMF$  LAYOUT temp_v(:NEWS,:NEWS)
C
      IF (L == 2) THEN
         i2 = mid
         i1 = i2 / 2
         i3 = 3 * i1
         v(2,i2,i2) = v(2,i2,i2) + v1
         v(2,i1,i2) = v(2,i1,i2) + half * v1
         v(2,i3,i2) = v(2,i3,i2) + half * v1
         v(2,i2,i1) = v(2,i2,i1) + half * v1
         v(2,i2,i3) = v(2,i2,i3) + half * v1
         v(2,i1,i1) = v(2,i1,i1) + quarter * v1
         v(2,i1,i3) = v(2,i1,i3) + quarter * v1
         v(2,i3,i1) = v(2,i3,i1) + quarter * v1
         v(2,i3,i3) = v(2,i3,i3) + quarter * v1
         RETURN
      ENDIF
C
      s = step(L-1)
      t = step(L)
      temp_v = v(L,:,:)
      v(L,:,:) = zero
      v(L,s:nmax:s,s:nmax:s) = v(L-1,s:nmax:s,s:nmax:s)
      v(L,:,:)  =  v(L,:,:)  +  half *
     &                    (  EOSHIFT(v(L,:,:), DIM = 1, SHIFT = +t, BOUNDARY = zero)
     &                     + EOSHIFT(v(L,:,:), DIM = 1, SHIFT = -t, BOUNDARY = zero)  )
      v(L,:,:)  =  v(L,:,:)  +  half *
     &                    (  EOSHIFT(v(L,:,:), DIM = 2, SHIFT = +t, BOUNDARY = zero)
     &                     + EOSHIFT(v(L,:,:), DIM = 2, SHIFT = -t, BOUNDARY = zero)  )
      v(L,:,:) = v(L,:,:) + temp_v
C
      RETURN
      END
C
C=============================================================================
C
      SUBROUTINE calc_result (v, f, r, u, e, right__bc, left___bc, top____bc, bottom_bc,
     &                        x, y, x2, y2, L, immediate_correction, from_L)
C
```

```
C  Calculate and print the error and residual norms.
C
      ....
C
C=======================================================================================
C
      DOUBLE PRECISION FUNCTION log_10 (x)
C
C  Return LOG10(x) while checking for a zero argument.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE plot_vf (input_array, right__bc, left___bc, top____bc, bottom_bc, L,
     &                    depth, vf)
C
C  Plot input_array(L,:,:) on the Connection Machine frame buffer or an X-terminal.
C  For coarser grids, this means expanding the value of each coarse-grid point to the
C  step(L) x step(L) sized block surrounding it.
C  Note: rotate_pause_flag = -2 means rotate after each plot,
C        rotate_pause_flag = -1 means rotate and pause after each plot,
C        rotate_pause_flag =  0 means don't pause after each plot,
C        rotate_pause_flag = +1 means pause after each plot.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE plot_ur (input_array, ur)
C
C  Plot input_array(:,:) in a similar fashion to the above routine 'plot_vf'.
C
      ....
C
C=======================================================================================
C
      SUBROUTINE pause
      IMPLICIT NONE
C
      CHARACTER*1 dummy
C
      WRITE(*,10)
10    FORMAT(1X, 'Hit <CR> to continue .... ' $)
      READ(*,20) dummy
20    FORMAT(A1)
C
      RETURN
      END
C
C=======================================================================================
```

Bibliography

[1] J. C. Adams. FMG results with the multigrid software package MUDPACK. In J. Mandel, S. F. McCormick, et al., editors, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, pages 1–12, Philadelphia, 1989. Society for Industrial and Applied Mathematics.

[2] D. J. Albers. *Paul Halmos.* In D. J. Albers and G. L. Alexanderson, editors, *Mathematical People: Profiles and Interviews*, pages 119–132. Birkhäuser, Boston, 1985.

[3] D. J. Albers and L. A. Steen. *Donald Knuth.* In D. J. Albers and G. L. Alexanderson, editors, *Mathematical People: Profiles and Interviews*, pages 181–204. Birkhäuser, Boston, 1985.

[4] W. F. Ames. *Nonlinear Partial Differential Equations in Engineering.* Academic Press, New York, 1965.

[5] W. F. Ames. *Numerical Methods for Partial Differential Equations.* Academic Press, New York, 1977. 2nd edition.

[6] K. Appel and W. Haken. Every planar map is four colorable. *Bull. Amer. Math. Soc.*, 82:711–714, 1976.

[7] D. G. Aronson. The porous medium equation. In A. Fasano and M. Primicerio, editors, *Nonlinear Diffusion Problems*, pages 1–46. Springer-Verlag, Berlin, 1986. Volume 1224 of Lecture Notes in Mathematics.

[8] K. E. Atkinson. *An Introduction to Numerical Analysis.* John Wiley and Sons, New York, 1978.

[9] N. S. Bakhvalov. Convergence of a relaxation method with natural constraints on an elliptic operator. *Ž. Vyčisl. Mat. i Mat. Fiz.*, 6:861–885, 1966. Math. Rev. 35 #6378.

[10] A. Barcellos. *Stanislaw M. Ulam.* In D. J. Albers and G. L. Alexanderson, editors, *Mathematical People: Profiles and Interviews*, pages 353–361. Birkhäuser, Boston, 1985.

[11] G. K. Batchelor. *An Introduction to Fluid Dynamics.* Cambridge University Press, 1967.

[12] J. H. Bramble and J. E. Pasciak. New convergence estimates for multigrid algorithms. *Math. Comp.*, 49:311–329, 1987.

[13] J. H. Bramble, J. E. Pasciak, and J. Xu. The analysis of multigrid algorithms for nonsymmetric and indefinite elliptic problems. *Math. Comp.*, 51:389–414, 1988.

[14] A. Brandt. Multi-level adaptive technique (MLAT) for fast numerical solution to boundary value problems. In *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*, pages 82–89, New York, 1973. Springer-Verlag.

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

[16] A. Brandt. Multi-level adaptive techniques (MLAT) for singular-perturbation problems. In P. W. Hemker and J. J. H. Miller, editors, *Numerical Analysis of Singular Perturbation Problems*, pages 53–142, New York, 1979. Academic Press.

[17] A. Brandt. Guide to multigrid development. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, pages 220–312. Springer-Verlag, Berlin, 1981.

[18] A. Brandt. The Weizmann Institute research in multilevel computation: 1988 report. In J. Mandel, S. F. McCormick, et al., editors, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, pages 13–53, Philadelphia, 1989. Society for Industrial and Applied Mathematics.

[19] W. Briggs, L. Hart, S. McCormick, and D. Quinlan. Multigrid methods on a hypercube. In S. F. McCormick, editor, *Multigrid Methods: Theory, Applications and Supercomputing*, pages 63–83, New York, 1988. Marcel Dekker. Proceedings of the Third Copper Mountain Conference on Multigrid Methods.

[20] W. L. Briggs. *A Multigrid Tutorial.* Society for Industrial and Applied Mathematics, Philadelphia, 1987.

[21] G. Caginalp and P. C. Fife. Elliptic problems involving phase boundaries satisfying a curvature condition. *IMA J. Appl. Math.*, 38:195–217, 1987.

[22] T. F. Chan and Y. Saad. Multigrid algorithms on the hypercube multiprocessor. *IEEE Trans. Comput.*, C-35:969–977, 1986.

[23] T. F. Chan, Y. Saad, and M. H. Schultz. Solving elliptic partial differential equations on hypercubes. In M. T. Heath, editor, *Proceedings of the First Conference on Hypercube Multiprocessors*, pages 198–210, Philadelphia, 1985. Society for Industrial and Applied Mathematics.

[24] T. F. Chan and R. S. Tuminaro. Implementation of multigrid algorithms on hypercubes. In M. T. Heath, editor, *Proceedings of the Second Conference on Hypercube Multiprocessors*, pages 730–737, Philadelphia, 1986. Society for Industrial and Applied Mathematics.

[25] T. F. Chan and R. S. Tuminaro. Design and implementation of parallel multigrid algorithms. In S. F. McCormick, editor, *Multigrid Methods: Theory, Applications and Supercomputing*, pages 101–115, New York, 1988.

Marcel Dekker. Proceedings of the Third Copper Mountain Conference on Multigrid Methods.

[26] T. F. Chan and R. S. Tuminaro. A survey of parallel multigrid algorithms. In A. K. Noor, editor, *Parallel Computations and their Impact on Mechanics*, pages 155–170. American Society of Mechanical Engineers, 1988.

[27] T. F. Chan and R. S. Tuminaro. Analysis of a parallel multigrid algorithm. In J. Mandel, S. F. McCormick, et al., editors, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, pages 66–86, Philadelphia, 1989. Society for Industrial and Applied Mathematics.

[28] L. Collatz. Über die konvergenzkriterien bei iterationsverfahren für lineare gleichungssysteme. *Math. Z.*, 53:149–161, 1950.

[29] D. N. de G. Allen. *Relaxation Methods.* McGraw-Hill, London, 1954.

[30] F. de la Vallée Poussin. An accelerated relaxation algorithm for iterative solution of elliptic equations. *SIAM J. Numer. Anal.*, 5:340–351, 1968.

[31] N. H. Decker. On the parallel efficiency of the Frederickson-McBryan multigrid. ICASE Report 90-17, NASA, 1990.

[32] N. H. Decker. Note on the parallel efficiency of the Frederickson-McBryan multigrid algorithm. *SIAM J. Sci. Stat. Comput.*, 12:208–220, 1991.

[33] M. P. do Carmo. *Differential Geometry of Curves and Surfaces.* Prentice-Hall, New Jersey, 1976.

[34] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. *LINPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[35] C. C. Douglas and W. L. Miranker. Constructive interference in parallel algorithms. *SIAM J. Numer. Anal.*, 25:376–398, 1988.

[36] C. C. Douglas and B. F. Smith. Using symmetries and antisymmetries to analyze a parallel multigrid algorithm: the elliptic boundary value problem case. *SIAM J. Numer. Anal.*, 26:1439–1461, 1989.

[37] R. P. Fedorenko. A relaxation method for solving elliptic difference equations. *Ž. Vyčisl. Mat. i Mat. Fiz.*, 1:922–927, 1961. Math. Rev. 25 #766.

[38] R. P. Fedorenko. On the speed of convergence of an iteration process. *Ž. Vyčisl. Mat. i Mat. Fiz.*, 4:559–564, 1964. Math. Rev. 31 #6386.

[39] C. A. J. Fletcher. *Computational Techniques for Fluid Dynamics, Volume 1.* Springer-Verlag, Berlin, 1991. 2nd edition.

[40] H. Foerster and K. Witsch. Multigrid software for the solution of elliptic problems on rectangular domains: MG00 (release 1). In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, pages 427–460. Springer-Verlag, Berlin, 1981.

[41] P. O. Frederickson and O. A. McBryan. Parallel superconvergent multigrid. In S. F. McCormick, editor, *Multigrid Methods: Theory, Applications and*

Supercomputing, pages 195–210, New York, 1988. Marcel Dekker. Proceedings of the Third Copper Mountain Conference on Multigrid Methods.

[42] P. O. Frederickson and O. A. McBryan. Normalized convergence rates for the PSMG method. *SIAM J. Sci. Stat. Comput.*, 12:221–229, 1991.

[43] D. Gannon and J. Van Rosendale. On the structure of parallelism in a highly concurrent PDE solver. *J. Parallel Dist. Comput.*, 3:106–135, 1986.

[44] D. Gilbarg and N. S. Trudinger. *Elliptic Partial Differential Equations of Second Order.* Springer-Verlag, Berlin, 1983. 2nd edition.

[45] E. Giusti. On the equation of surfaces of prescribed mean curvature. *Invent. Math.*, 46:111–137, 1978.

[46] H. H. Goldstine and J. von Neumann. On the principles of large scale computing machines. *Proc. Symp. Pure Math.*, 50:179–183, 1990.

[47] G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins University Press, Baltimore, 1983.

[48] W. D. Gropp and I. C. F. Ipsen. Recursive mesh refinement on hypercubes. *BIT*, 29:186–211, 1989.

[49] W. Hackbusch. Convergence of multi-grid iterations applied to difference equations. *Math. Comp.*, 34:425–440, 1980.

[50] W. Hackbusch. Multi-grid convergence theory. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, pages 177–219. Springer-Verlag, Berlin, 1981.

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

[52] W. Hackbusch. A new approach to robust multi-grid solvers. In *ICIAM'87: Proceedings of the First International Conference on Industrial and Applied Mathematics*, pages 114–126, Philadelphia, 1988. Society for Industrial and Applied Mathematics.

[53] W. Hackbusch. The frequency decomposition multi-grid method i: applications to anisotropic equations. *Numer. Math.*, 56:229–245, 1989.

[54] M. B. Hall. *Nature and Nature's Laws.* Macmillan, New York, 1970.

[55] R. S. Hamilton. The Ricci flow on surfaces. *Contemporary Mathematics*, 71:237–262, 1988.

[56] W. D. Hillis. *The Connection Machine.* MIT Press, Cambridge, Massachusetts, 1985.

[57] W. D. Hillis, 1991. Personal communication.

[58] D. C. Jespersen. Multigrid methods for partial differential equations. In Gene Golub, editor, *Studies in Numerical Analysis.* Mathematical Association of America, 1984. Volume 24 of Studies in Mathematics.

[59] F. John. *Partial Differential Equations.* Springer-Verlag, New York, 1971.

[60] S. L. Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. Stat. Comput.*, 8:354–390, 1987.

[61] D. Kamowitz. Multigrid applied to singular perturbation problems. ICASE Report 87-1, NASA, 1987.

[62] J. L. Kazdan. *Prescribing the Curvature of a Riemannian Manifold.* American Mathematical Society, Rhode Island, 1985. Number 57 of Regional Conference Series in Mathematics.

[63] M. Kline. *Mathematics and the Physical World.* Thomas Y. Crowell, New York, 1959.

[64] S. Kobayashi and K. Nomizu. *Foundations of Differential Geometry, Volume 1.* Wiley (Interscience), New York, 1963.

[65] H.-J. Kuo and N. S. Trudinger. Discrete methods for fully nonlinear elliptic equations. CMA Research Report R14-90, Australian National University, 1990.

[66] A. Lanza. Multigrid in general relativity ii: Kerr spacetime. *Class. Quantum Grav.*, 9:677–696, 1992.

[67] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufman, San Mateo, California, 1992.

[68] A. Lichnerowicz. *Elements of Tensor Calculus.* Methuen, London, 1962.

[69] C.-M. Lin, W. Proskurowski, and J.-L. Gaudiot. A parallel multigrid method for data-driven multiprocessor systems. In J. Mandel, S. F. McCormick, et al., editors, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, pages 299–318, Philadelphia, 1989. Society for Industrial and Applied Mathematics.

[70] C. Liu. The finite volume element (FVE) and fast adaptive composite grid methods (FAC) for the incompressible Navier-Stokes equations. In J. Mandel, S. F. McCormick, et al., editors, *Proceedings of the Fourth Copper Mountain Conference on Multigrid Methods*, pages 319–336, Philadelphia, 1989. Society for Industrial and Applied Mathematics.

[71] S. B. Maurer. *Albert Tucker.* In D. J. Albers and G. L. Alexanderson, editors, *Mathematical People: Profiles and Interviews*, pages 337–348. Birkhäuser, Boston, 1985.

[72] K. O. May. Historiography: A perspective for computer scientists. In N. Metropolis, J. Howlett, and G. Rota, editors, *A History of Computing in the Twentieth Century.* Academic Press, New York, 1980.

[73] O. A. McBryan. The Connection Machine: PDE solution on 65536 processors. TMC Report CS86-1, Thinking Machines Corporation, 1986.

[74] O. A. McBryan. New architectures: performance highlights and new algorithms. *Parallel Comput.*, 7:477–499, 1988.

[75] O. A. McBryan, P. O. Frederickson, et al. Multigrid methods on parallel computers — a survey of recent developments. *IMPACT Comp. Sci. Eng.*, 3:1–75, 1991.

[76] S. F. McCormick. *Multilevel Adaptive Methods for Partial Differential Equations.* Society for Industrial and Applied Mathematics, Philadelphia, 1989.

[77] S. F. McCormick and W. L. Briggs. FMV program listing. In S. F. McCormick, editor, *Multigrid Methods*, pages 179–186. Society for Industrial and Applied Mathematics, Philadelphia, 1987. Appendix 1.

[78] S. F. McCormick and J. Ruge. Multigrid methods for variational problems. *SIAM J. Numer. Anal.*, 19:924–929, 1982.

[79] S. F. McCormick and J. Thomas. The fast adaptive composite grid (FAC) method for elliptic equations. *Math. Comp.*, 46:439–456, 1986.

[80] A. R. Mitchell and D. F. Griffiths. *The Finite Difference Method in Partial Differential Equations.* Wiley (Interscience), New York, 1973.

[81] B. Montgomery and L. Evans. *You and Stress.* Nelson, 1984.

[82] J. C. C. Nitsche. *Lectures on Minimal Surfaces, Volume 1.* Cambridge University Press, 1989. Translated from German.

[83] J. Noye. Finite difference methods for partial differential equations. In J. Noye, editor, *Numerical Solutions of Partial Differential Equations.* North-Holland, Amsterdam, 1982.

[84] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New York, 1970.

[85] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms.* Prentice-Hall, New Jersey, 1977.

[86] R. D. Richtmyer and K. W. Morton. *Difference Methods for Initial-Value Problems.* Wiley (Interscience), New York, 1967. 2nd edition.

[87] J. W. Ruge and K. Stüben. Algebraic multigrid (AMG). In S. F. McCormick, editor, *Multigrid Methods*, pages 73–130. Society for Industrial and Applied Mathematics, Philadelphia, 1987.

[88] J. Serrin. The problem of Dirichlet for quasilinear elliptic differential equations with many independent variables. *Philos. Trans. Roy. Soc. London Ser. A*, 264:413–496, 1969.

[89] D. B. Singleton. Multigrid on the cm-2, 1992. In preparation.

[90] R. V. Southwell. Stress-calculation in frameworks by the method of systematic relaxation of constraints. *Proc. Roy. Soc. London Ser. A*, 151:56–95, 1935.

[91] R. V. Southwell. *Relaxation Methods in Theoretical Physics.* Oxford University Press, 1946.

[92] E. L. Stiefel. Über einige methoden der relaxationsrechnung. *Z. Angew. Math. Phys.*, 3:1–33, 1952. Math. Rev. 13 p. 874,1140.

[93] K. Stüben and U. Trottenberg. Multigrid methods: Fundamental algorithms, model problem analyis and applications. In W. Hackbusch and

U. Trottenberg, editors, *Multigrid Methods*, pages 1–176. Springer-Verlag, Berlin, 1981.

[94] R. Temam. *Numerical Analysis.* D. Reidel, Dordrecht, 1973.

[95] Thinking Machines Corporation, Cambridge, Massachusetts. *Connection Machine Model CM-2 Technical Summary*, 1989. Revised edition.

[96] Thinking Machines Corporation, Cambridge, Massachusetts. *CM Fortran Reference Manual*, 1991. Version 1.1.

[97] Thinking Machines Corporation, Cambridge, Massachusetts. *CMSSL for CM Fortran*, 1991. Version 2.2.

[98] Thinking Machines Corporation, Cambridge, Massachusetts. *The Connection Machine CM-5 Technical Summary*, 1991.

[99] N. S. Trudinger and J. I. E. Urbas. The Dirichlet problem for the equation of prescribed Gauss curvature. *Bull. Austral. Math. Soc.*, 28:217–231, 1983.

[100] R. S. Tuminaro. *Multigrid Algorithms on Parallel Processing Systems.* PhD thesis, Stanford University, California, December 1989. Dept of Computer Science, report STAN-CS-90-1299.

[101] R. Varga. *Matrix Iterative Analysis.* Prentice-Hall, New Jersey, 1962.

[102] E. L. Wachspress. *Iterative Solution of Elliptic Systems and Applications to the Neutron Diffusion Equations of Reactor Physics.* Prentice-Hall, New Jersey, 1966.

[103] W. R. Wasow. Discrete approximations to elliptic difference equations. *Z. Angew. Math. Phys.*, 6:81–97, 1955.

[104] R. Whaley, J. Myczkowski, and L. M. Hood, 1991. Personal communication.

[105] D. Young. *Iterative Solution of Large Linear Systems.* Academic Press, New York, 1971.

I have nothing to write about.

— N. K. Spencer, November 1991

I'm never going to run out of things to write about.

— N. K. Spencer, April 1992