

Visualising Magnetic Fields

By John Stuart Beeteson

Visualising Magnetic Fields

Numerical equation solvers in action

This Page Intentionally Left Blank

Visualising Magnetic Fields

Numerical equation solvers in action

John Stuart Beeteson



ACADEMIC PRESS

A Harcourt Science and Technology Company

San Diego San Francisco New York
Boston London Sydney Tokyo

This book is printed on acid-free paper.

Copyright © 2001 by ACADEMIC PRESS

All Rights Reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher.

The software accompanying this publication is furnished "as is" and without warranty or representation of any kind, either express or implied, all of which, including all warranties of merchantability, fitness for a particular purpose, use, or conformity to any description, are hereby disclaimed. The author, Academic Press, and its distributors will not be liable for any damages, including any damages for loss of data or profits, or for any incidental, consequential, or indirect damages, arising from the use of the software or its documentation. All users of the software will assume all risks associated with use of the software, and the sole remedy for any defect in the software media shall be the replacement thereof.

Academic Press

A Harcourt Science and Technology Company

Harcourt Place, 32 Jamestown Road, London NW1 7BY, UK

<http://www.academicpress.com>

Academic Press

A Harcourt Science and Technology Company

525 B Street, Suite 1900, San Diego, California 92101-4495, USA

<http://www.academicpress.com>

ISBN 0-12-084731-0

CD-ROM ISBN 0-12-084732-9

Library of Congress Catalog Number: 00-105294

A catalogue record for this book is available from the British Library

Typeset by M Rules, Southwark, London

Printed and bound in Great Britain by Cambrian Printers, Aberystwyth

01 02 03 04 05 CP 8 7 6 5 4 3 2 1

**To my wife, Dora, for all her
patience and encouragement.**

‘The study of these lines [of magnetic force] has at different times
been greatly influential in leading me to various results,
which I think prove their utility as well as fertility’.

Michael Faraday
in a paper to the Royal Society October 22nd 1851

This Page Intentionally Left Blank

Contents

Chapter 1:	Introduction	1
	Magnetic lines of force	1
	What is visualisation?	2
	The purpose of this book	3
	The software	4
	Communications to the author	5
	Reviewing the physics	5
Chapter 2:	Physics of the magnetic field	7
	Fields and field lines	7
	Gauss's Law	9
	The Biot–Savart law	9
	Ampere's Law	10
	Magnetic force on a conductor	12
	Magnetic materials and permanent magnets	13
	The magnetisation curve	14
	Hysteresis	14
	Maxwell's equations	15
	The magnetic circuit	16
	The first step in visualisation	18
Chapter 3:	The basic technique	19
	Overview	19
	Characteristics of the technique	24
	Generating the equations	25
	The matrix form	27
	Making use of the technique	27
Chapter 4:	Numerical algorithm theory	29
	Gaussian elimination	29
	The conjugate gradient method	31

	Double versus single precision arithmetic	33
	Theory of the conjugate gradient method	34
	The equations being solved	34
	The quadratic form	36
	The method of steepest descent	37
	The method of conjugate gradients	38
	Comparing the two algorithms	39
	Preconditioning	40
	Timing and storage results	40
	Indirect addressing for arrays	41
	Choice of analysis method and mesh size	41
	16 and 32 bit operating systems	42
	Progress indicator	43
	Error level setting	43
	Understanding the matrix solving algorithms	43
Chapter 5:	Visualising the algorithms	45
	Conjugate gradient visualisation	45
	Use of the options panel	47
	Example of use of the options	48
	Sample files included with SHOWALG	48
	Error level versus number of iterations	49
	Gaussian elimination visualisation	49
	Completing the theory	50
Chapter 6:	The boundary region, smoothing and external fields	53
	Why a boundary region is needed	53
	Mesh impedance characteristics	54
	Distortion performance	57
	Smoothing	58
	Using the boundary for external field simulation	60
	Magnetic field lines and magnetic flux density	61
Chapter 7:	The magnetic flux density function	63
	Magnetic flux density filter and log scale	65
	Putting the theory to work	67
Chapter 8:	Model creation	69
	Model functions provided	69
	Data structure	70
	Structure of object storage	70

	Model generation	70
	Equation generation	71
	Event driven code	71
	Editing	71
	Running the program	72
Chapter 9:	Program installation and use	73
	Installation	73
	Using VIZIMAG	73
	On-line help	76
	The source code	76
	Hints and tips on using VIZIMAG	76
Chapter 10:	Sample results	81
	Magnetic samples	81
References		83
Appendix:	The source code	85
	Running the source code	85
	Programming style	85
	Architecture	85
	Global variables	86
	Resizing arrays	86
	Code flow descriptions: overview	86
	Application start up	88
	Application close down	89
	Form.Update	89
	Form.Refresh	89
	Program procedures initiated by a menu item or speedbutton	90
	General program procedures in unit VIZGlobal	106
	General program procedures in unit VIZ1	106
	General program procedures in VIZ2	130
	General program procedures in VIZ3	135
	Other mouse and keyboard event initiated procedures	139
Index to source code procedures		145
Index		148

Colour plate section between pages 6 and 7

This Page Intentionally Left Blank

Introduction

Magnetic lines of force

Nature's mysterious force of magnetism has fascinated children and scientists alike over many centuries. We probably all remember playing with small bar magnets and wondering at their almost magical ability to attract, repel or even to levitate. One of the first experiments any child performs in the school laboratory is to sprinkle iron filings on a sheet of paper held over bar magnets, gently tap the paper and then to see strange lines, such as those shown in Figure 1-1, surround and link the magnets.

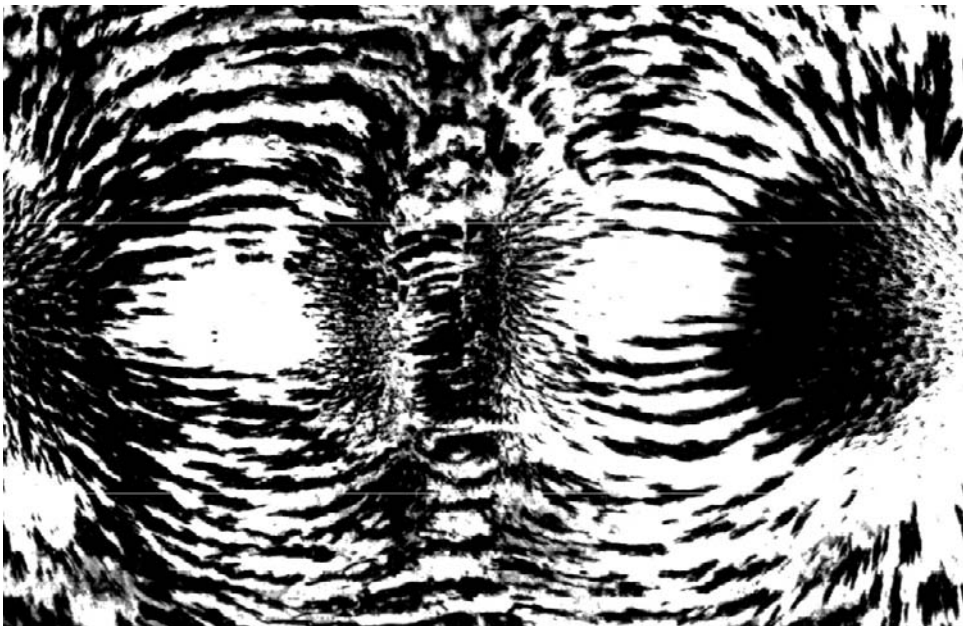


Figure 1-1: Iron filings over two bar magnets.

It is not only schoolchildren who have been fascinated by these patterns. John Tyndall, in his biography of the great 19th century physicist Michael Faraday, tells us that the idea of lines of magnetic force was suggested to Faraday by the linear arrangement of iron filings when scattered over a magnet. Faraday investigated the deflection of the lines when they pass through magnetic bodies and, in his later research, the idea of lines of force is extensively employed. In a paper to the Royal Society in 1851 Faraday devoted himself to the formal development and illustration of his favourite idea. The paper bears the title, *'On lines of magnetic force, their definite character, and their distribution within a magnet and through space'*. 'The study of these lines', Faraday says, 'has at different times been greatly influential in leading me to various results which I think prove their utility as well as fertility'.

Today, when first studying a new magnetic system, the first thing that scientists and engineers usually do is to draw out the system of magnets, coils and magnetic materials, and sketch how they think the lines of magnetic force will link everything together. They try to gain an intuitive feeling for the behaviour of the system, as a precursor to moving to a full quantitative analysis on a suitable computational platform. Thus magnetic lines of force are as useful a concept to us today, in the visualisation of magnetic fields, as they were to Michael Faraday.

What is visualisation?

The dictionary defines visualisation as 'the process of interpreting in visual terms or of putting into visible form'. The US National Science Foundation in Scientific Computing says that visualisation (and here is meant scientific visualisation) is characterised as a method that integrates the power of digital computers and human vision, and directs the result towards facilitating scientific insight. Albert Einstein and Richard Feynman both spoke of their use of visualisation. Einstein said 'The psychical entities . . . in thought . . . are more or less clear images which can be "voluntarily" reproduced and combined . . . the essential feature in productive thought before there is any connection with logical construction in words . . .'. Feynman described visualisation as 'trying to bring birth to clarity'. He added that visualisation was a way of seeing the character of the answer ' . . . before the mathematics could be really done'. The Leeds University Visualisation Research Group sums up visualisation succinctly 'as an approach in which a computer generated visual representation is used to improve our understanding'. We can also see that

mathematical exactness is not always a necessary requirement of a visualisation technique, especially if we are looking for insights rather than numerical results.

In the context of this book and software, visualisation is the picturing of the magnetic lines of force (or magnetic field lines as we call them today) presented in such a way that the patterns of linkage between different objects show with correct physical relationships and relative magnitudes. For this purpose the result is not intended to be analytically correct, but the visual relationships should give insights into the behaviour of the system. To be of value in the modelling of relationships prior to the use of a more exact modelling technique, it is also an aim that any method employed must allow rapid generation and modification of the models, and rapid production of the magnetic field line patterns, so that many different configurations can be tried very quickly.

For some purposes, for example to make illustrations for reports, or for some teaching purposes, this level of visualisation is sufficient in itself, but often it will be used as the first step before transferring a model to one of the several modern electromagnetic computational platforms now available – usually finite element or boundary element analysis. These computational techniques are extremely powerful and give high accuracy simulations, but a great deal of skill is needed in their use, and the modelling stage is time consuming. It is a boon to have a simple visualisation technique that provides a good basis for the first model creation, and also allows new ideas to be tried quickly during a long simulation.

Whatever modelling method is chosen, it is always necessary eventually to solve large matrices, produced from the solution simultaneous equations, and this in turn requires an appropriate numerical analysis technique.

The purpose of this book

Because speed is one of the aims of this particular visualisation technique, we need a simple method of model generation based on a graphical computer interface, and a fast way of solving and plotting the magnetic field lines. Bearing in mind that a full analytic solution of the field equations involves a surface integral for every point, this requirement is most important. The purpose of this book, and the accompanying software, is first to provide a method of visualising magnetic fields that fulfils these requirements, and second to demonstrate the two major numerical algorithms used for solving large numbers of simultaneous

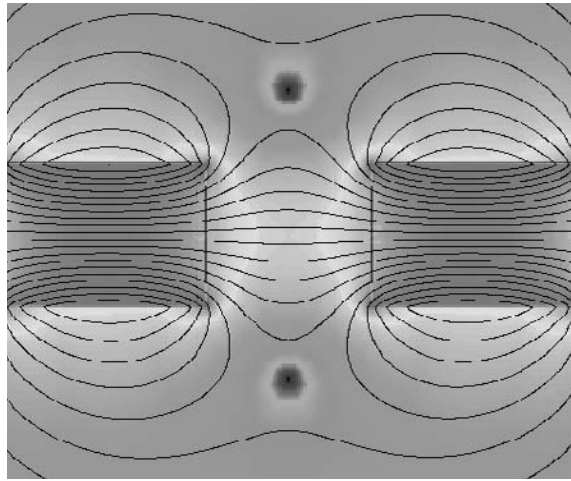


Figure 1-2: Computer generated magnetic field line plot of two bar magnets.

equations. Figure 1-2 was produced using the method to be described, and is the computed equivalent of the iron filings picture of Figure 1-1. From starting the model to the finished plot took less than 15 seconds, and a bonus is that the magnetic flux density can be shown along with the magnetic field lines. A colour plate of Figure 1-2 can be found between pages 6 and 7.

The modelling method to be described is a new one, and the approach adopted in this book has been to give a complete description of the theory and also to describe the various approaches to numerical equation solving algorithms. A complete software modelling application, named VIZIMAG, is provided on CD-ROM, but to make the method generally accessible, and to invite readers to contribute improvements and extensions, the complete source code with a plain English description of each procedure and event flow is provided. A second computer application named SHOWALG is also provided, and this gives a visualisation of the numerical algorithms in action.

The software

Included in the book is a CD-ROM containing application programs with sample data files providing worked examples. The software was written for use with the Windows 95, Windows 98 or Windows NT operating systems. Detailed descriptions of these programs will be found in Chapter 9. To use

the programs immediately, please turn to Chapter 9 and follow the installation instructions.

The CD-ROM also includes the full source code for the visualisation application program, written in Pascal using the Borland Delphi application package. In order to use the source code (and the associated graphical interface form design) it will be necessary to purchase the standard edition of Delphi, version 3 or 4. A detailed description of the source code will be found in Chapter 11.

Copyright in the application package VIZIMAG, the form design and the source code is retained by the author. However, the source code and the form may be reused free of charge by the purchaser of this book and companion CD-ROM. The source code may be used in any way the purchaser desires as part of his or her own software application. It may not, however, be distributed either free or for commercial gain in the original source form. If use is made of the code in other applications, an acknowledgement giving the title, author and publisher of this book is requested.

Communications to the author

Constructive comments and extensions to the visualisation technique described, suggestions for code changes, code and examples of use, are all welcomed by the author. However, material will be accepted only on the basis that it may be published or incorporated, free of charge, either in full or condensed form in future editions of this book and software, and hence may be reused by subsequent purchasers. The author may be e-mailed at vizimag@aol.com. Because of the problem of accidental exposure to viruses, it is required that no attachments be added to e-mails, and hence no executable code sent electronically. Please make all communications plain text, with source code only. Executable code and hard copy material may be sent via the publisher.

Reviewing the physics

Before describing the modelling method and the algorithms, the next chapter gives a brief review of the physics of magnetic fields, and illustrates the principles with magnetic field plots made using the VIZIMAG application.

This Page Intentionally Left Blank

Physics of the magnetic field

The properties of magnetite, the magnetic ore of iron, have been known since ancient times, and this has made the study of magnetism one of the oldest branches of experimental science. The Roman author Lucretius, writing in the 1st century BC, said that the word magnetism came from the Roman provincial city of Magnesia ad Sipylum (now Manisa in Turkey) where the ore was mined. However, Pliny, in the 1st century AD said that the name derived from the Greek shepherd Magnes, who found the nails in his boots sticking to magnetic rocks!

In this chapter a basic review of the physics of magnetic fields is provided, and to demonstrate how physical intuition may be built from the visualisation of magnetic field lines, examples generated using the VIZ-IMAG application are used. As a summary review, the equations given in this chapter are simply stated, rather than being derived from first principles. For a more in-depth approach the reader is directed to one of the many excellent physics textbooks available.

Fields and field lines

A bar magnet, or a wire carrying a current, produces a magnetic field permeating the space all around it. The direction of the field at any point is defined to be the direction in which a north magnetic pole would move due to the field at that point, and the path through which this pole would freely move is the magnetic line of force or magnetic field line. Like poles repel each other and unlike poles attract, and therefore magnetic lines of force are directed away from the north pole of a magnet and towards the south pole. In fact, although scientists have searched for, and are still searching for, isolated magnetic poles (i.e. magnetic monopoles), they have never been discovered.

The magnitude and direction of a magnetic field is \mathbf{B} , the magnetic flux

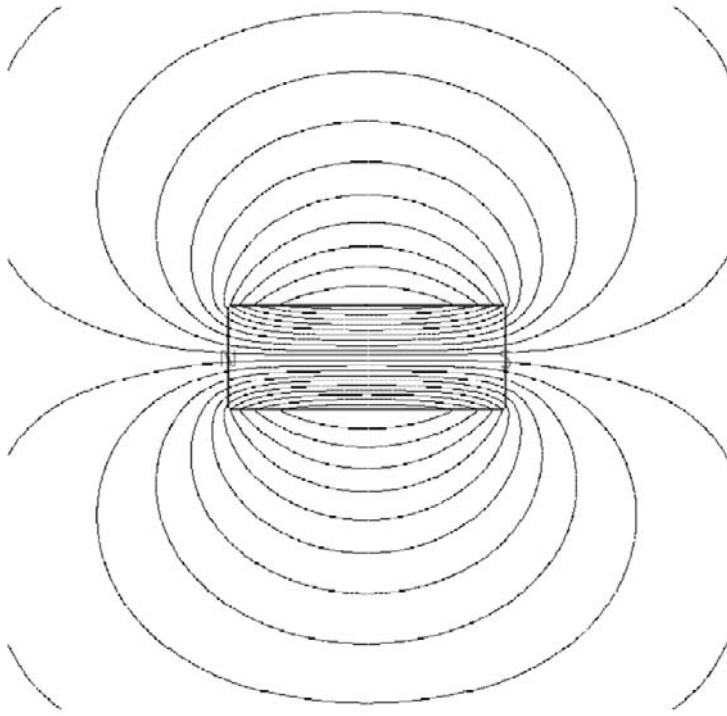


Figure 2-1: Air cored coil.

density. When we draw magnetic field lines to represent a magnetic field, the direction of the line at any point is tangential to the direction of the magnetic flux density, and the magnitude of the field is proportional to the number of field lines per unit area. There are two important rules to remember about magnetic field lines: first that field lines can never cross, and second that magnetic field lines must always close (i.e. a magnetic field line leaving a flux source must always eventually return to the same point).

Let us review these facts about magnetic field lines in relation to one of the examples included in the sample data files of the VIZIMAG application. Sample file number 5 (see Chapter 10) is reproduced in Figure 2-1.

This example shows the magnetic field from a simple wire coil or open solenoid. We can see that magnetic field lines near the centre of the coil have nearly uniform spacing, showing that the magnetic flux density (\mathbf{B}) is nearly constant in magnitude and direction. At the ends of the coil the magnetic field lines are more widely and less uniformly spaced, and for a long solenoid the magnetic flux density on axis at the ends is half that at the centre.

We also see that the field lines inside the solenoid are packed more closely together than the lines outside, showing that the magnitude of the field is much higher inside than outside. We can also see that field lines leaving the north pole always return to enter the solenoid again at the south pole and eventually close. A field line on the axis of the solenoid plainly must extend to infinity if there is to be closure of the line, implying that the effect of a magnetic field extends infinitely through space. The rapidly increasing spacing of the field lines away from the coil, however, shows that the magnetic flux density drops off rapidly so that, although a field line may enclose an infinitely large space, it will have a vanishingly small amplitude.

Gauss's Law

The magnetic flux through any region of space is represented by the total number of magnetic field lines passing through the region. If we look at one such region in Figure 2-1, for example the field lines leaving the solenoid at the north pole face, then we can see, because of the requirement for closure, that the number of lines leaving this face must equal the number of lines entering, so that the net magnetic flux out of any closed surface is always zero. This conclusion is represented by one of the basic laws of magnetism, Gauss's Law, which is also one of Maxwell's equations:

$$\oint \mathbf{B} \cdot d\mathbf{A} = 0 \text{ (for a closed surface)}$$

where \mathbf{B} is the magnetic flux density and dA is the surface area.

The Biot-Savart law

The magnetic flux density produced at any field point by a short length of current carrying conductor in vacuum is given by the Biot-Savart law, illustrated in Figure 2-2:

$$\delta\mathbf{B} = \frac{\mu_0}{4\pi} \frac{I\delta l \sin \theta}{r^2}$$

where μ_0 , the magnetic permeability of a vacuum, is unity, but the deviation in free air is so small that for most purposes this is also assumed to be unity. The Biot-Savart law can be tested only indirectly, since we cannot have just a δl length of current carrying conductor.

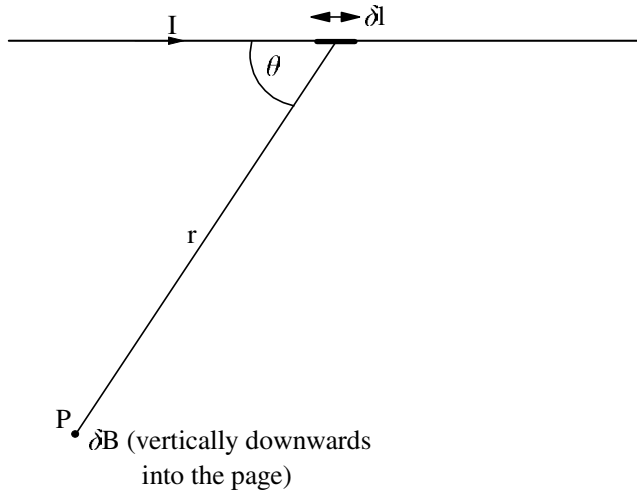


Figure 2-2: Magnetic flux density due to a current element δl .

Using the law to find the field due to a long straight conductor gives the result:

$$\mathbf{B} = \frac{\mu_0 I}{2\pi r}$$

where μ_0 is the permeability of air, I is the current and r is the distance of point from wire. This result, which can be verified experimentally, shows that the magnetic field due to a long straight conductor is circularly symmetric around the conductor, as is demonstrated in VIZIMAG sample file 4, shown in Figure 2-3.

Ampere's Law

In configurations of high geometric symmetry, Ampere's Law can be simpler to evaluate than the Biot-Savart Law. The symmetry requirement is that the line integral of the magnetic flux density taken around a closed path has a known solution:

$$\oint \mathbf{B} \cos \theta \, dl = \mu_0 I.$$

Figure 2-4 shows part of such a closed path, where θ is the angle between a small dl section of the path and the direction of \mathbf{B} .

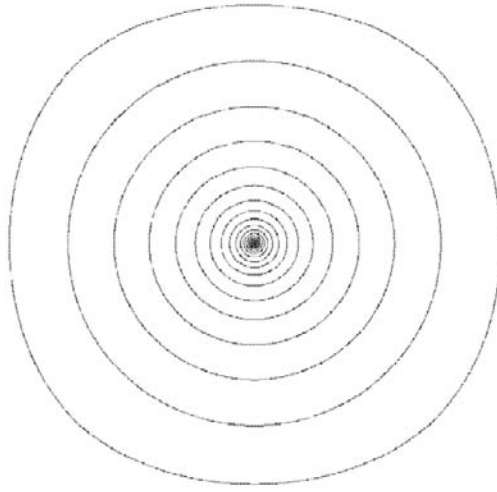


Figure 2-3: Field due to a current in a wire.

As an example take the field at the centre of a long open coil or solenoid. The line integral can be relatively easily obtained in this case (standard textbooks typically derive this example), to give the result

$$\mathbf{B} = \mu_0 n I$$

where n is the number of turns and I is the current. The result shows that the field inside a long solenoid is uniform, which can be seen in VIZIMAG sample file 5 shown in Figure 2-1.

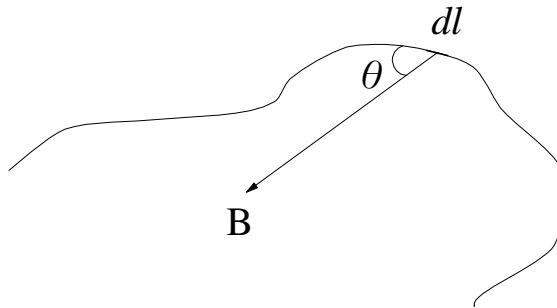


Figure 2-4: Part of closed section for Ampere's law.

Magnetic force on a conductor

If a current passes through a wire held in a magnetic field, such as shown in Figure 2-5, there is a force on the conductor $F = \mathbf{B}IL$.

The direction of the force can be found by Fleming's left-hand rule as shown in Figure 2-6.

In the case of Figure 2-5 the force acts leftwards across the paper. If the conductor is at an angle to the field, then only that component of current perpendicular to the field exerts a force and:

$$F = \mathbf{B}IL \sin \theta.$$

This, of course, is the principle of the electric motor, and is illustrated by VIZIMAG sample file 12, a wire between two poles, shown in Figure 2-7.

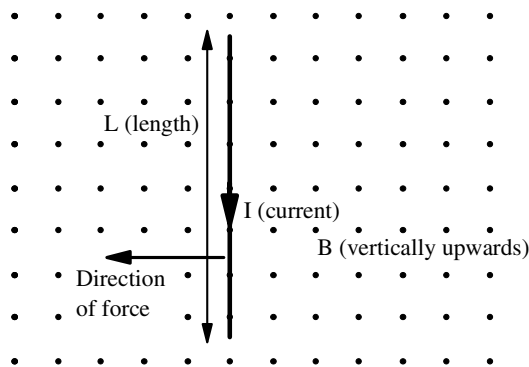


Figure 2-5: Magnetic force on a conductor.

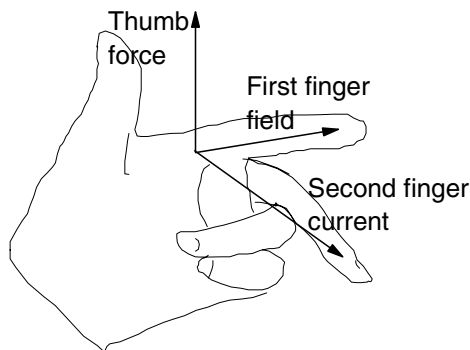


Figure 2-6: Fleming's left-hand rule.

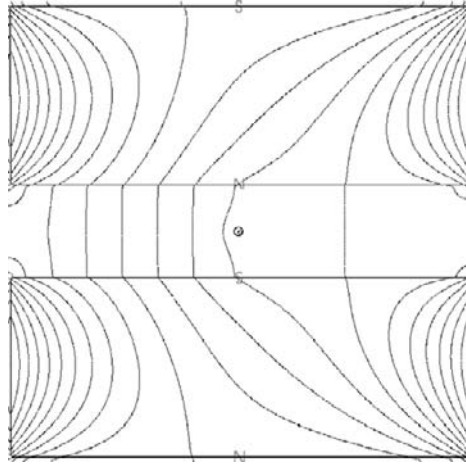


Figure 2-7: Wire between two poles.

In this example the magnetic field lines are bunched more closely together to the left of the wire, implying a higher magnetic flux density, and hence it would be expected that the wire would move to the right, as is found by the application of Fleming's left-hand rule.

Magnetic materials and permanent magnets

In the examples discussed so far, it has been assumed that the constant of proportionality in the equations is μ_0 and is unity for a vacuum and nearly unity for air. Some materials have strong magnetic properties and can have a permeability much greater than 1, for example iron, nickel and cobalt. Further, some materials can become magnetised, i.e. they retain a magnetic field even when the original source of magnetism is removed. For materials with a permeability greater than unity, an extra term of μ_r , or the relative permeability, is added to the equations. The equations are then modified by the term $\mu_0\mu_r$. Some examples of μ_r are:

Material	μ_r
Mild steel	5
Grain oriented magnetic iron	2000
Mumetal	10 000

The value of μ_r reduces with increasing temperature and, above a critical temperature known as the Curie point, the material will no longer be magnetic. In the case of a permanent magnet, increasing the temperature towards the Curie point will allow easier magnetising and demagnetising, but will reduce the magnetic field strength of the magnet. However, provided that the Curie temperature is not exceeded, the magnetic field will recover when the temperature drops. Taking a permanent magnet beyond the Curie point will destroy its magnetism, even when the temperature is reduced.

The magnetisation curve

When a ferromagnetic magnetic material is placed in a magnetic field, a magnetisation curve such as that shown in Figure 2-8 is obtained. At $B_{\text{external}} = 0$, the value of B for the material is also 0. As B_{external} increases, so does B , but not in a linear manner. At some value of B_{external} a point of saturation is reached, and B does not increase further.

Hysteresis

If a magnetic material is taken to saturation, as shown in Figure 2-8, and then the external magnetic field is first reduced and then increased again, it is found that the material exhibits a hysteresis effect as shown in Figure 2-9, which shows two different typical hysteresis curves. Figure 2-9 (i) shows a 'soft' magnetic material, where the hysteresis is quite small, such as might be used for transformer steel. Figure 2-9 (ii) shows a 'hard' magnetic material,

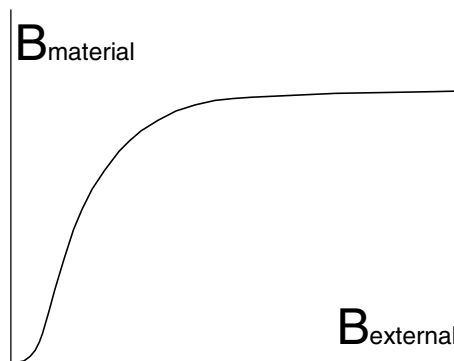


Figure 2-8: Magnetisation curve.

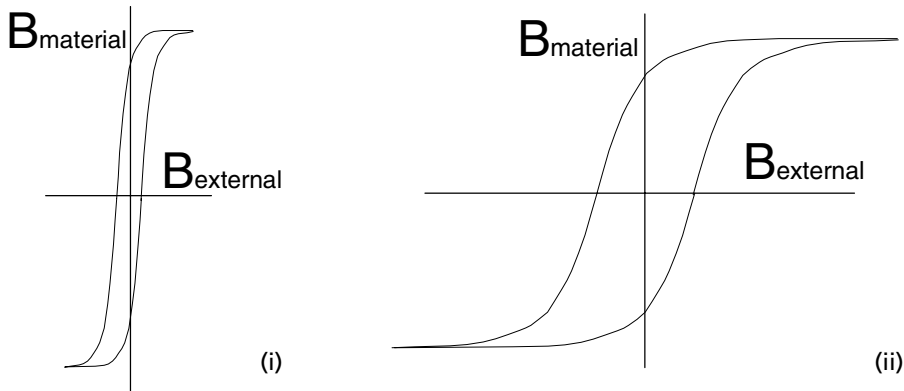


Figure 2-9 (i): Soft magnetic material. (ii): Hard magnetic material.

with a high remanence and a high coercivity, making the material suitable for a permanent magnet. Remanence is the magnetic field left after $\mathbf{B}_{\text{external}}$ is reduced to zero, and coercivity is the reverse value of $\mathbf{B}_{\text{external}}$ necessary to reduce $\mathbf{B}_{\text{material}}$ to zero.

Non-linear effects such as hysteresis are not modelled in the VIZIMAG application, which also assumes that the magnetic field of a material is linearly dependant on the external field. However, it is still possible to use a linear model to assess the performance of magnetic systems, because many systems are designed to operate in the linear region, and points of high magnetic flux density, where saturation would first occur, are easily seen. A linear model is nearly always the first step in visualising a magnetic system, prior to moving to more complex numerical modelling software (e.g. finite element, boundary element, etc.) which can use iterative processes to handle non-linearities.

Maxwell's equations

In 1855 and 1856 James Clerk Maxwell read papers on Faraday's lines of force to the Cambridge Philosophical Society. In these papers he showed that electric and magnetic fields were interrelated and could be summarised in just a few mathematical equations. In 1862 he used his equations to calculate that the speed of propagation of an electromagnetic field is equal to the speed of light, and proposed for the first time that light is an electromagnetic wave. His full theory, containing the four famous partial differential equations (in integral form), now known as Maxwell's equations, appeared in his

Treatise on Electricity and Magnetism in 1873. The first of Maxwell's equations is Gauss's law for electric fields:

$$\oint \mathbf{E} \cdot d\mathbf{A} = \frac{Q_{\text{enc}}}{\epsilon_0}.$$

The second of Maxwell's equations has already been given. It is Gauss's law for magnetic fields:

$$\oint \mathbf{B} \cdot d\mathbf{A} = 0.$$

The third equation is the Ampere–Maxwell law, which is a form of Ampere's law but extended by Maxwell to include the effect of displacement current:

$$\oint \mathbf{B} \cdot d\mathbf{l} = \mu_0 \left(I_c + \epsilon_0 \frac{d\Phi_E}{dt} \right).$$

The final equation is Faraday's induction law:

$$\oint \mathbf{E} \cdot d\mathbf{l} = - \frac{d\Phi_B}{dt}.$$

As stated above, the equations apply to electric and magnetic fields in a vacuum. If magnetic and electric materials are present, the terms ϵ_r and μ_r must be included, and generally will have to be moved inside the integrals on the left-hand side of the equations. The solutions to Maxwell's equations are generally obtained numerically using equation solvers such as are described in this book.

The magnetic circuit

Maxwell's paper and his four equations showed that electric and magnetic fields were inextricably entwined, and it is not surprising that (for linear materials) there are close analogies between electric and magnetic quantities. A system comprising one or more flux sources and magnetic materials is called a magnetic circuit and, just like an electric circuit, with voltages, resistance, currents, etc., magnetic circuits can be analysed in terms of magnetomotive force, reluctance, flux, and so on. A full treatment of this topic may be found in reference [1], but a few of these analogies are:

Electric circuit			Magnetic circuit		
Emf	E	(volts)	Magnetomotive force	F_m	(ampere-turns)
Current	I	(amps)	Flux	Φ	(webers)
Resistance	R	(ohms)	Reluctance	\mathfrak{R}	(ampere-turns per weber)

An example of such a magnetic circuit is VIZIMAG sample file 6, shown in Figure 2-10.

In this example the current carrying coil, with a magnetomotive force given in Ampere-turns by the product of the number of turns and the current, nI , produces the flux in the circuit, with the path of the flux shown by the magnetic field lines. The permeability of the coil, magnetic domain material in the core, and the air gap form the reluctance elements of the magnetic circuit and (together with leakage flux into the surrounding air) determine the total amount of flux produced.

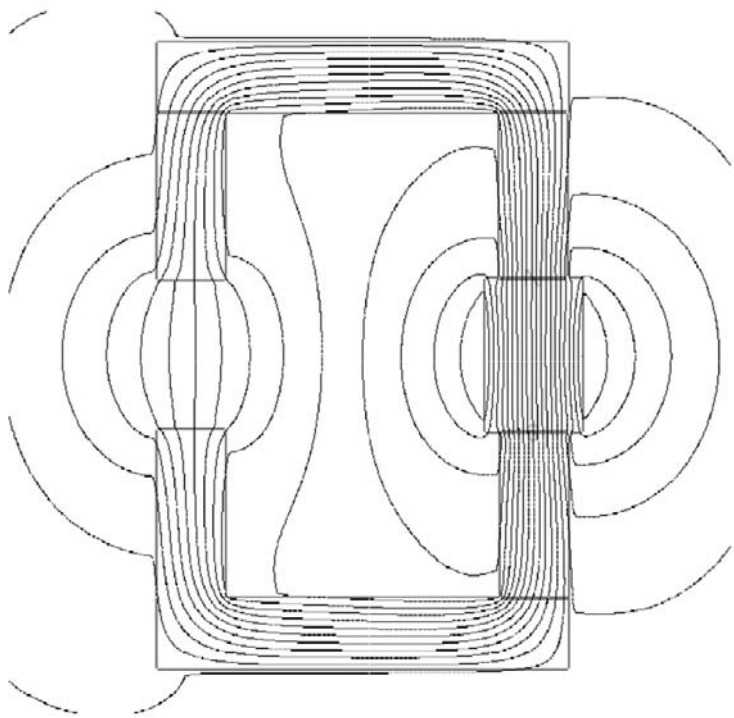


Figure 2-10: Magnetic circuit with air gap.

The first step in visualisation

The analogies between electric and magnetic circuits are so close that it is a reasonable assumption that an electric circuit analysis method might be extended to solve magnetic circuits. In creating a complete methodology for visualising magnetic fields, the first step is to find a technique that can provide a representation of the magnetic field line patterns and a method to do this, based on well known electric circuit solution techniques, is described in the next chapter.

The basic technique

In the previous chapter we saw that there were analogies between electric and magnetic units in the modelling of magnetic circuits. In the technique to be described in this chapter these analogies are exploited via the analysis of circuit meshes to form the basis of a visualisation technique. A non-mathematical overview of the method is given first, followed by a more detailed analysis.

Overview

Consider a set of resistors interconnected, so that, with four resistors forming a single mesh, we obtain a set of 4×4 meshes as shown in Figure 3-1. Now place a voltage source in one branch as shown in Figure 3-2.

In order to determine the resultant currents flowing throughout the mesh the method of loop mesh analysis, based on Kirchoff's Voltage Law [1] is typically used, which states that the algebraic sum of all the voltages taken around a closed loop in an electric circuit is zero. The first step in this

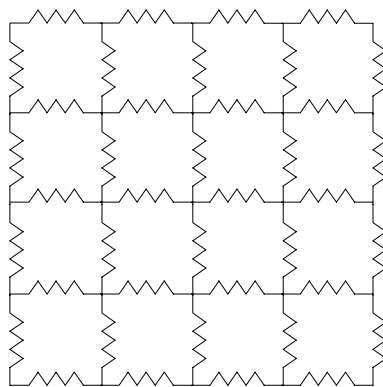


Figure 3-1: A basic resistor mesh.

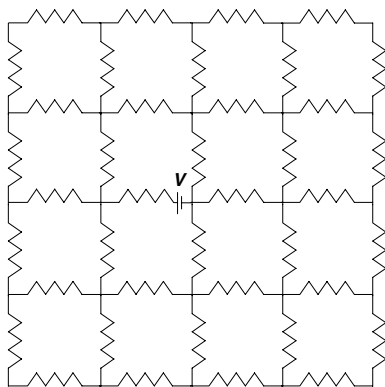


Figure 3-2: Resistor mesh with a single branch voltage source.

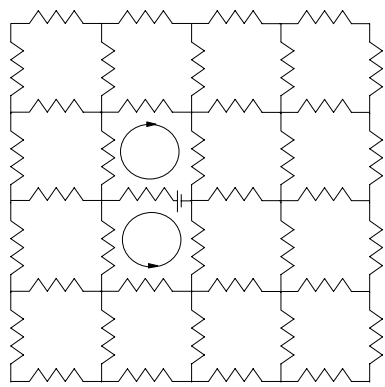


Figure 3-3: Mesh loop voltages.

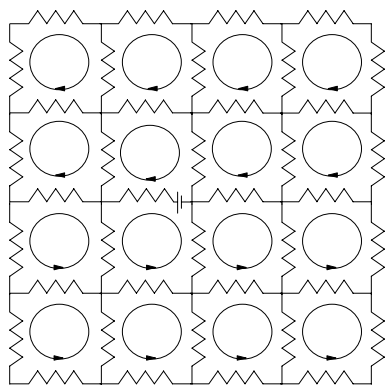


Figure 3-4: Mesh loop currents.

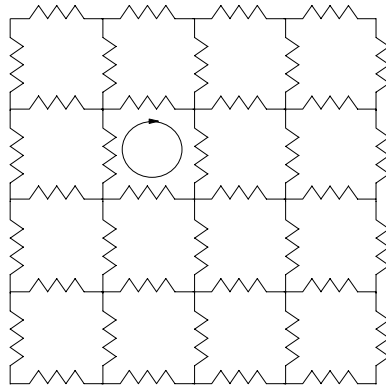


Figure 3-5: Resistor mesh with a single mesh loop voltage.

analysis method is to consider currents flowing round each mesh loop as a result of the voltage source in each mesh loop. For a 4×4 mesh the solution creates a set of 16 (i.e. 4^2) simultaneous equations relating the loop currents, loop voltages and the resistor values as coefficients. Solving the simultaneous equations gives the loop currents, which can then be summed and differenced to give the actual currents and hence voltages in each resistor. For the mesh with one branch voltage source of Figure 3-2, the mesh loop voltages (a mesh loop voltage is the sum of the four branch voltage sources in a particular mesh) and the resultant mesh loop currents are shown in Figures 3-3 and 3-4.

This method of analysis is mathematically convenient, but by itself, of course, a loop current has no physical existence and it is not possible to measure a loop current by placing a meter in any network branch. However, to examine if the loop currents derived from electrical mesh analysis had any analogy to magnetic field patterns, some experiments with mesh voltage sources were performed. Now the mesh equations (which will be described later) have the mesh loop voltages as the independent variables, and for our purposes we will only consider these and not the actual mesh branch voltage sources that create them. To begin with, consider a mesh of uniform resistor values with just one mesh loop voltage, as in Figure 3-5. A similar circuit but with 60×60 meshes was analysed to give the resultant loop currents shown in Figures 3-6 and 3-7.

With the exception of a slight cramping of the contour lines towards the edge of the picture, this does indeed provide an analogy to the way that magnetic fields propagate due to a flux source (in this case the flux from a current carrying wire perpendicular to the plane of the paper).

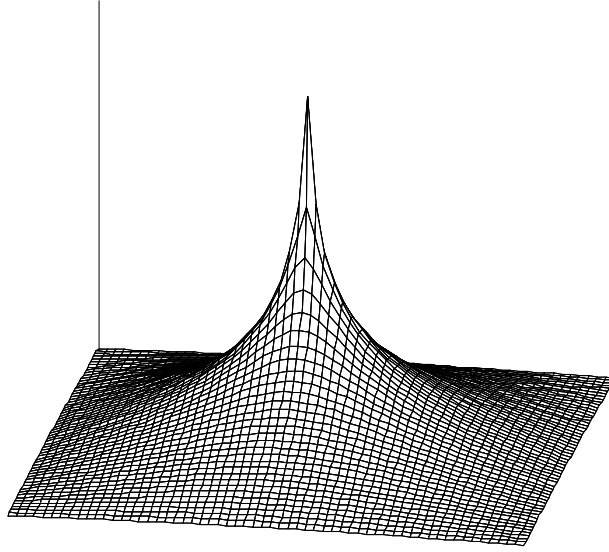


Figure 3-6: The solution plot with mesh loop current as the vertical axis.

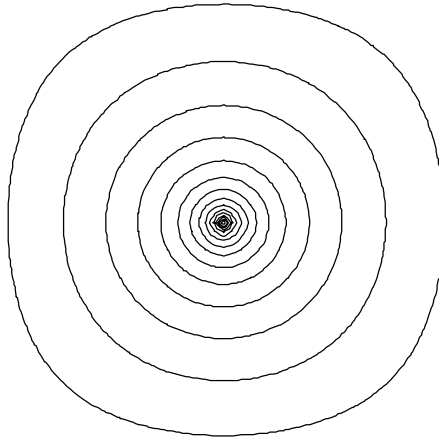


Figure 3-7: Contour plot of figure 3-6.

A circuit with two sets of mesh loop voltages, similar to Figure 3-8 but with a 60×60 mesh, was then generated and analysed. Note that the directions of the sets of loop voltages are of opposite sign. Figures 3-9 and 3-10 show the results of analysing this mesh, and this time we get a contour plot that shows an analogy to the magnetic field pattern of a coil or permanent magnet.

Further work showed that the resistor values could be taken as analogies to magnetic permeability, with a value of unity for free air or vacuum and values less than unity for more permeable material, e.g. resistor values of 0.01 to represent a region with a permeability of 100. This then, is the basis of the present visualisation technique. The mesh of resistors

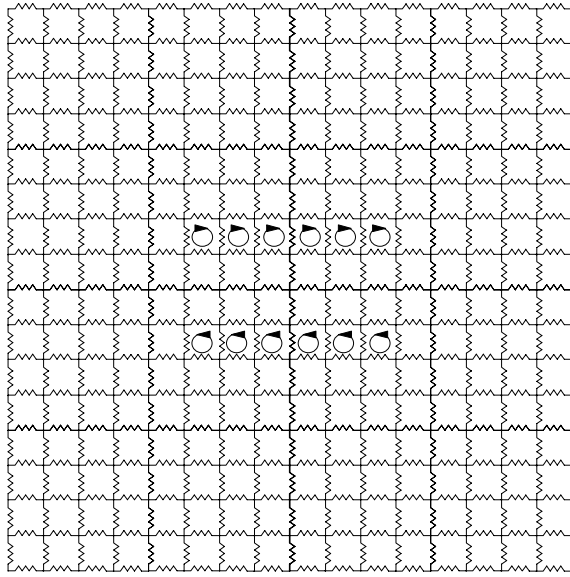


Figure 3-8: A mesh with two sets of opposing mesh voltages.

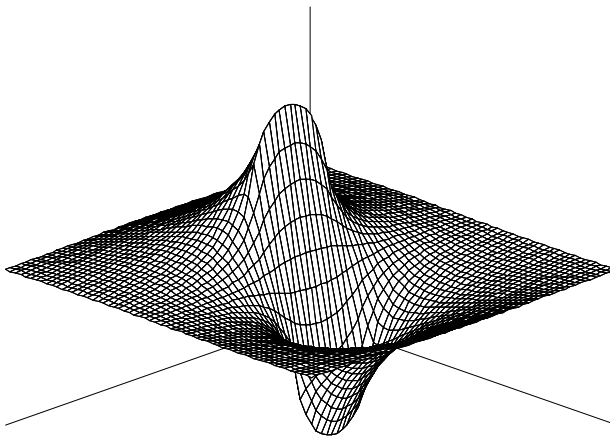


Figure 3-9: Mesh loop currents, surface plot.

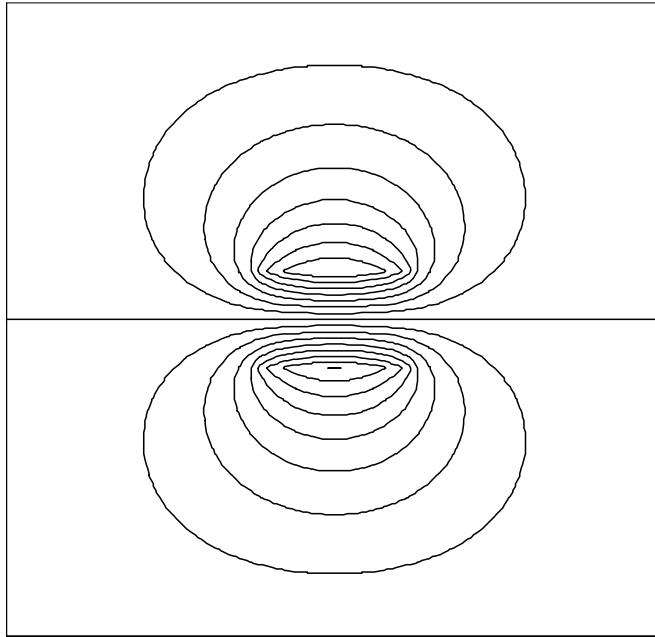


Figure 3-10: Mesh loop currents, contour plot.

represents the space to be analysed, with the resistor values an analogy of magnetic permeability and voltage sources representing magnetic flux sources. The mesh equations are solved to give the loop currents and a contour plot then gives a visualisation of the magnetic field pattern. An example for a 300×300 mesh is given in Figure 2-10. Within the right-hand rectangle in Figure 2-10 is a set of voltage sources representing a coil in air. Within the other enclosed areas the resistors are set to 0.01 to represent regions with a permeability of 100, and the primary and leakage magnetic field lines are clearly visualised.

Characteristics of the technique

A characteristic of this technique is that the mesh regions are of fixed size, so defining the spatial resolution of the analysis. Finite element techniques, on the other hand, allow variable size meshes, so that complex regions and regions of rapidly changing magnetic flux density can be modelled more finely.

The method does not pretend to produce an analytically correct result, but rather places an emphasis on speed of model creation, with a primary

purpose to produce a representation or visualisation of magnetic field lines (and magnetic flux density plots) to allow insights to be gained into different model configurations. The method also does not model non-linear effects such as magnetic saturation or permanent magnet demagnetisation.

Generating the equations

Reference [1] gives a rigorous derivation of the mesh loop analysis technique, and it is not necessary to repeat this here, since we are primarily concerned with the application of the results. It is sufficient to give an example of the equations resulting from a typical mesh. Consider the 3×3 resistor mesh with a single voltage source in one branch, shown in Figure 3-11. The meshes are numbered from 0 to 8. The resistor subscripts are R_h for horizontally drawn resistors and R_v for vertically drawn resistors, which is convenient here as this is how they will eventually be numbered and handled in the computer programs. The single branch voltage source has value V volts, which results in a loop voltage of $+V$ for mesh 4 and $-V$ for mesh 7, with all other meshes having a loop voltage of zero. This 3×3 mesh results in the following 3^2 , i.e. 9 simultaneous equations:

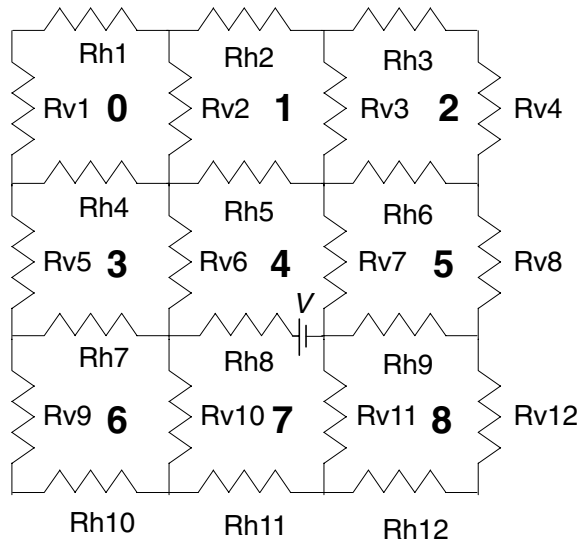


Figure 3-11: 3×3 resistor mesh with single branch voltage.

$$\begin{aligned}
R_{00}I_0 - R_{01}I_1 - R_{02}I_2 - R_{03}I_3 - R_{04}I_4 - R_{05}I_5 - R_{06}I_6 - R_{07}I_7 - R_{08}I_8 &= V_0 \\
-R_{10}I_0 + R_{11}I_1 - R_{12}I_2 - R_{13}I_3 - R_{14}I_4 - R_{15}I_5 - R_{16}I_6 - R_{17}I_7 - R_{18}I_8 &= V_1 \\
-R_{20}I_0 - R_{21}I_1 + R_{22}I_2 - R_{23}I_3 - R_{24}I_4 - R_{25}I_5 - R_{26}I_6 - R_{27}I_7 - R_{28}I_8 &= V_2 \\
-R_{30}I_0 - R_{31}I_1 - R_{32}I_2 + R_{33}I_3 - R_{34}I_4 - R_{35}I_5 - R_{36}I_6 - R_{37}I_7 - R_{38}I_8 &= V_3 \\
-R_{40}I_0 - R_{41}I_1 - R_{42}I_2 - R_{43}I_3 + R_{44}I_4 - R_{45}I_5 - R_{46}I_6 - R_{47}I_7 - R_{48}I_8 &= V_4 \\
-R_{50}I_0 - R_{51}I_1 - R_{52}I_2 - R_{53}I_3 - R_{54}I_4 + R_{55}I_5 - R_{56}I_6 - R_{57}I_7 - R_{58}I_8 &= V_5 \\
-R_{60}I_0 - R_{61}I_1 - R_{62}I_2 - R_{63}I_3 - R_{64}I_4 - R_{65}I_5 + R_{66}I_6 - R_{67}I_7 - R_{68}I_8 &= V_6 \\
-R_{70}I_0 - R_{71}I_1 - R_{72}I_2 - R_{73}I_3 - R_{74}I_4 - R_{75}I_5 - R_{76}I_6 + R_{77}I_7 - R_{78}I_8 &= V_7 \\
-R_{80}I_0 - R_{81}I_1 - R_{82}I_2 - R_{83}I_3 - R_{84}I_4 - R_{85}I_5 - R_{86}I_6 - R_{87}I_7 + R_{88}I_8 &= V_8
\end{aligned}$$

The subscripts have the following meanings:

R_{00} means the self-resistance of mesh 0, i.e. $R_{h1} + R_{v2} + R_{h4} + R_{v1}$.

R_{11} means the self-resistance of mesh 1, i.e. $R_{h2} + R_{v2} + R_{h5} + R_{v3}$.

R_{01} means the mutual resistance between mesh 0 and mesh 1, i.e. R_{v2} .

R_{10} means the mutual resistance between mesh 1 and mesh 0, i.e. again R_{v2} .
and so on.

V_1 etc. refer to the loop voltage of mesh 1 and so on.

Now some of these coefficient terms, for example R_{38} , are zero, since there is no mutual resistor shared between mesh 3 and mesh 8, and many of the loop voltages are zero also. The equations then reduce to a sparser form:

$$\begin{aligned}
R_{00}I_0 - R_{01}I_1 & 0 - R_{03}I_3 & 0 & 0 & 0 & 0 & 0 & 0 & = 0 \\
-R_{10}I_0 + R_{11}I_1 - R_{12}I_2 & 0 - R_{14}I_4 & 0 & 0 & 0 & 0 & 0 & 0 & = 0 \\
0 - R_{21}I_1 + R_{22}I_2 & 0 & 0 - R_{25}I_5 & 0 & 0 & 0 & 0 & 0 & = 0 \\
-R_{30}I_0 & 0 & 0 + R_{33}I_3 - R_{34}I_4 & 0 - R_{36}I_6 & 0 & 0 & 0 & 0 & = 0 \\
0 - R_{41}I_1 & 0 - R_{43}I_3 + R_{44}I_4 - R_{45}I_5 & 0 - R_{47}I_7 & 0 & = V \\
0 & 0 - R_{52}I_2 & 0 - R_{54}I_4 + R_{55}I_5 & 0 & 0 - R_{58}I_8 & = 0 \\
0 & 0 & 0 - R_{63}I_3 & 0 & 0 + R_{66}I_6 - R_{67}I_7 & 0 & = 0 \\
0 & 0 & 0 & 0 - R_{74}I_4 & 0 - R_{76}I_6 + R_{77}I_7 - R_{78}I_8 & = -V \\
0 & 0 & 0 & 0 & 0 - R_{85}I_5 & 0 - R_{87}I_7 + R_{88}I_8 & = 0
\end{aligned}$$

With the zero terms in the equations a diagonal matrix form emerges, with the mesh self-resistance as the centre diagonal, the mesh vertical resistor mutual resistance as the next diagonal (upper or lower) and the mesh horizontal resistor mutual resistance as the outer diagonal (upper or lower).

The matrix form

If we look at a large matrix we find the form shown in Figure 3-12.

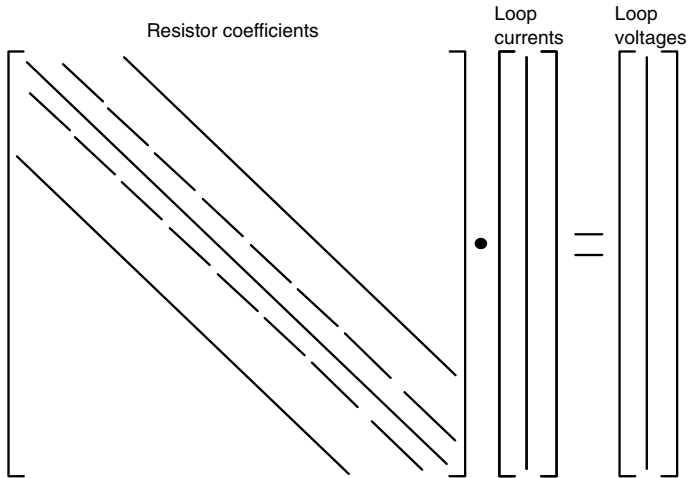


Figure 3-12: Matrix form.

The coefficients of the matrix, derived from self and horizontal and vertical mutual resistor values, lie on the diagonal lines and the form is known as a tridiagonal banded sparse matrix. The sparsity comes from all the coefficients not on the diagonal lines shown being zero.

Making use of the technique

To produce an acceptably fine resolution, there have to be a reasonably large number of meshes in the modelling space. We might expect to use, say, 50×50 meshes for small models, and many times this for more complex ones. Even 50×50 meshes results in 2500 simultaneous equations to solve, and larger models can require 100 000 or more. Before proceeding any further, therefore, we must demonstrate that this is possible in a sensible computation time, and the next chapter examines numerical algorithms for matrix solution.

This Page Intentionally Left Blank

Numerical algorithm theory

To obtain the loop current distribution in an $N \times N$ resistor mesh, N^2 simultaneous equations must be solved. The solution may be obtained numerically by a direct method based on Gaussian elimination, or by an iterative method usually based on the conjugate gradient technique.

From the user's point of view the difference between the two methods shows as a trade-off in the areas of memory size and computation time. The direct method requires substantial memory (moving into virtual memory for large N) but works with a predictable number of computational steps as a function of N . The conjugate gradient method requires a minimum amount of memory, but has a variable number of iteration steps and hence computation time. Both methods will be described here, together with typical timings and storage requirements, and both are implemented in the software.

Gaussian elimination

The Gaussian elimination algorithm has two phases, the first of which is *forward elimination* and is the most computationally intensive. In this phase all terms in the coefficient matrix below the centre diagonal become zero. In the second phase, *backward substitution*, the loop current values are extracted. As an example take the 3×3 mesh shown in Figure 3-11, and the associated nine simultaneous equations.

An operation that can be performed directly on an equation is to multiply both sides by a constant. Another operation that can be performed on sets of such simultaneous equations without altering the solution is to add two and replace one of these by the sum (the reader may try a simple example for proof of this). For example, the first two equations of our example are:

$$R_{00}I_0 - R_{01}I_1 - R_{03}I_3 = 0 \quad (1)$$

$$-R_{10}I_0 + R_{11}I_1 - R_{12}I_2 - R_{14}I_4 = 0. \quad (2)$$

Multiply (1) by R_{10}/R_{00} to give:

$$R_{10}I_0 - R_{01} \cdot (R_{10}/R_{00}) \cdot I_1 - (R_{10}/R_{00})R_{03}I_3 = 0. \quad (3)$$

Now add (2) and (3) to eliminate I_0 :

$$-((R_{01} \cdot (R_{10}/R_{00})) - R_{11}) \cdot I_1 - R_{12}I_2 - (R_{10}/R_{00})R_{03}I_3 - R_{14}I_4 = 0. \quad (4)$$

Equation (4) can then replace (2) in the original set. In doing this we have eliminated I_0 from the second equation, and we can similarly manipulate the first and fourth equation of the original set to eliminate I_0 from this, so leaving I_0 only in the first equation. We can repeat all these manipulations to gradually eliminate all the equation terms below the central diagonal line. When this is completed, the last two equations of the set of nine become:

$$\begin{aligned} K_1I_7 + K_2I_8 &= K_3V \\ K_4I_8 &= K_5V \end{aligned}$$

where the K constants are the results of all the manipulations carried out on the coefficients. This completes the forward elimination phase, and the backwards substitution phase is clear. Starting with the last equation we can solve for I_8 directly and then substitute the newly found I_8 value into the previous equation to solve for I_7 and so on.

One point to note in the forward elimination phase is the multiplication by terms similar to R_{10}/R_{00} . It will be found that a division by R_{00} , R_{11} , R_{22} , etc. is a common factor. If we remember that this is the self-resistance of a mesh (i.e. all four mesh resistors added together) then provided that the resistors never become zero there will be no chance of a floating point 'division by zero' overflow in the computation.

The amount of computation and storage necessary for the solution must be considered. When a larger network is used, the matrix form shown in Figure 3-12 is obtained, and two points should be noted. First, all coefficients not marked as lines are zero. Second, the mutual coefficient lines above the centre diagonal are identical to those below the centre diagonal and therefore do not need separate storage.

In the forward elimination stage of the Gaussian method all terms below the centre diagonal become zero, and no storage need be allocated for them.

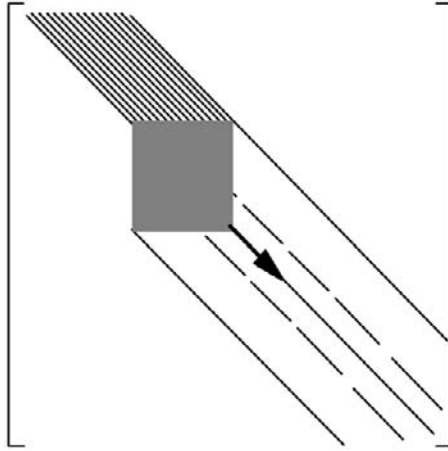


Figure 4-1: Computation block required during forward elimination phase, fill in of terms above the centre diagonal and elimination of terms below the centre diagonal.

However, all terms between the centre diagonal and the upper (horizontal resistor) mutual coefficient line become non-zero, and therefore storage must be allocated to these during the analysis. Also some non-zero terms are created below the centre diagonal, before being set to zero, and a storage block must be allocated as shown in Figure 4-1.

To graphically show how the row and column scanning is performed in practice, the computer program SHOWALG may be run (see chapter 5).

It is mainly the filling in of the upper diagonal terms that causes the larger memory requirement in the direct Gaussian method. In the application program VIZIMAG, the storage is allocated only during the ANALYSE procedure, so that outside of the analysis itself storage is primarily limited to the coefficient diagonals, and loop voltages and currents. In the source code procedure ANALYSE, the lines of code in the forward elimination phase that occupy the most computation time are commented. Conversion of these lines to machine code can be expected to show an appreciable speed improvement.

The conjugate gradient method

The method of conjugate gradients is an iterative procedure, i.e. an initial starting solution for the loop currents is assumed (usually simply all zeros), and then a computational iteration is performed to produce (hopefully) an improved solution. These iterations are repeated until the

procedure determines so that no further improvements are possible. There is no filling-in of matrix terms with the conjugate gradient method, and therefore storage is primarily limited to three diagonals (remembering that the two lower diagonals are identical to the upper two) and the loop voltages and currents. The actual algorithm will be presented first, together with a discussion of the use of double and single precision arithmetic. After this a brief overview of the theory of the method will be given.

Notation: A square matrix (such as the coefficient matrix) is written in bold capitals. A vector (i.e. a single column matrix) is written in bold lower case. A scalar (i.e. a single number) is written in italics. The notation $\mathbf{x}^T \mathbf{y}$ means the inner product of the vectors, i.e. the sum of $x_{[i]} \cdot y_{[i]}$, which is a scalar. For an $N \times N$ mesh we wish to solve the equation to within a given error value:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

where \mathbf{A} is the known coefficient matrix, \mathbf{b} is the known mesh voltage vector, and \mathbf{x} is the unknown mesh current loop vector.

Loop initialisation

$\mathbf{x} = 0$

$\mathbf{f} = \mathbf{g} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}$ \mathbf{f} and \mathbf{g} are internal vectors required in the algorithm

$h = j = \mathbf{f}^T \mathbf{f}$

$c = 0$ c is a counter

Main loop

$\mathbf{k} = \mathbf{A} \cdot \mathbf{g}$ \mathbf{k} is an internal vector required in the algorithm

$l = h / (\mathbf{g}^T \mathbf{k})$

$\mathbf{x} = \mathbf{x} + l \cdot \mathbf{g}$

If $c = 25$ then $\mathbf{f} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}$ and $c = 0$

else $\mathbf{f} = \mathbf{f} - l \cdot \mathbf{k}$ See explanation below for this line.

$m = h$

$h = \mathbf{f}^T \mathbf{f}$

$i = h/m$

$\mathbf{g} = \mathbf{f} + i \cdot \mathbf{g}$

If $h < \text{error}^2 \cdot j$ then finish.

See explanation below for this line.

If $c = 0$ and $l/i < 0.1$ then finish.

See explanation below for this line.

If number of loops = N^2 then finish. This line provides a fail safe termination.

Figure 4-2: Conjugate gradient algorithm.

From a computational viewpoint each iteration requires slightly over one matrix multiplication, which is coded in the procedure MATMULT: conversion of this routine to machine code can be expected to improve the speed of the program. Note that MATMULT is highly optimised for the tridiagonal banded sparse matrix of this application. To use the programs on less sparse matrices it is this routine which would have to be modified. To visualise what happens during successive iterations of the conjugate gradient method the computer program SHOWALG may be run (see Chapter 5).

Double versus single precision arithmetic

In published conjugate gradient algorithms there is generally an implicit assumption that double precision arithmetic will be used. This is because the algorithm is sensitive to floating point rounding errors if the number of iterations is large. Also a very high result accuracy is usually assumed as a requirement. In our case we are looking for a convergence point at which no further discernible *visual* difference can be seen, which is generally a lower accuracy than would be accepted for computational purposes. Some experiments showed that single precision arithmetic was adequate for this purpose, but unfortunately using the usual criteria of $h < (\text{error}^2 \cdot j)$ to terminate the loop would not always produce convergence when complicated models were run (typically problems show up if several objects of very high permeability are used). Double precision arithmetic always produced convergence, but at the expense of a large increase in computation time.

The reason that the loop would not always terminate with single precision arithmetic was that although an acceptable *visual* result had actually been computed, floating point rounding errors would not drive the accuracy criteria low enough. Simply reducing the accuracy criteria did not solve the problem since this produced visible errors. It was observed, however, that the factors l and i , which are ratios used in determining the next iteration value of \mathbf{x} , had a predictable behaviour as rounding errors began to predominate. A factor l/i was therefore introduced as an extra control over loop termination. While the algorithm is making significant improvement at each iteration l/i was observed to stay close to a value of 0.5, but as rounding errors increase this value reduced. After some experiments a value of 0.1 was selected as an additional loop termination criteria to indicate that rounding errors were preventing further improvement. There is one proviso to this, however. The correction $\mathbf{f} = \mathbf{f} - l \cdot \mathbf{k}$ in the algorithm is only an approximate equation. The accurate term is actually $\mathbf{f} = \mathbf{b} - \mathbf{A} \cdot \mathbf{x}$, but as can be seen this would introduce an extra matrix multiplication and

hence slow the algorithm considerably. However, for the l/i factor to work properly the accurate term is needed. The solution was to compute the accurate term only occasionally (every 25 iterations) and to check the l/i factor when this had been done.

Although this final algorithm has always produced a successful result on all the models run by the author, there is no mathematical proof that this is guaranteed, and therefore VIZIMAG includes analysis options to switch to double precision arithmetic or the Gaussian elimination method.

Note that the behaviour of the l/i term in the algorithm may well be specific to the type of tridiagonal banded sparse matrix found in this type of circuit analysis, so that it should be checked carefully before applying to more general cases.

Theory of the conjugate gradient method

The method of conjugate gradients is an iterative procedure for solving simultaneous equations. The development of the technique has a long history, but can be said to have come into its own when the use of digital computers became widespread. It is one of the most widely used algorithms, but is not easy to understand, especially for non-mathematicians. The best description I have found is by Shewchuk [2]. What will be given here is a simple overview of the method so as to allow a basic understanding of the various steps in the algorithm of Figure 4-2. For the detailed mathematics the reader is referred to the large body of literature on the subject.

The equations being solved

Referring back to Figure 3-11, a set of linear simultaneous equations was generated, where the resistor coefficients R and the voltages V are known, and the currents I are unknown. From Figure 3-11, nine simultaneous equations were created, but in a practical mesh there will be many thousands, and even though R is a sparse matrix the problem is formidable. Expressing the equations in matrix form gives:

$$\mathbf{R} \cdot \mathbf{i} = \mathbf{v}$$

but for the purposes of this theoretical overview the equations will be expressed as:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

$$3x_1 + 2x_2 = 2$$

$$2x_1 + 6x_2 = -8$$

or in matrix form

$$\begin{bmatrix} 4 & 3 \\ 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$$

Figure 4-3

because this is the notation most commonly employed in the mathematical literature and will therefore be the most convenient when making reference. For the purpose of illustrating the problem first consider a very simple case with just two values of x . The equations (Figure 4-3) give the solution $x_1 = 2, x_2 = -2$. The two lines are plotted in Figure 4-4, where the solution lies at their intersection.

Now in the general case we would have n values of x , where n is the number of equations (equal to the number of meshes in Chapter 3), and Figure 4-4 would then need to be drawn with n dimensions instead of just two. The general principle, however, is still true, that the solution values of x lie at the intersection of all the multi-dimensional lines. It is much too difficult to try to illustrate graphically a multi-dimensional solution, and therefore the examples in this chapter remain as the two-dimensional

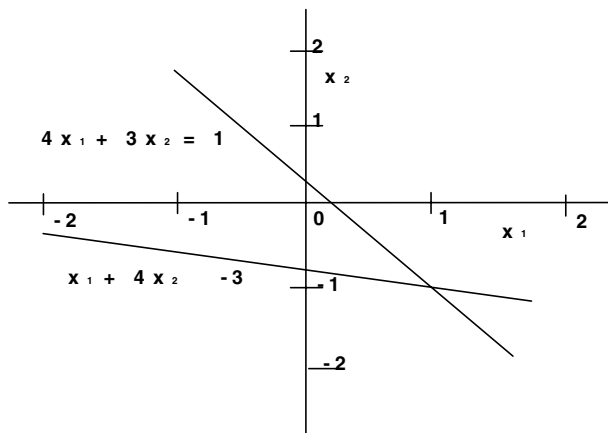


Figure 4-4: Intersection of lines.

equations of Figure 4-3. However, the general applicability to many dimensions should always be borne in mind.

The quadratic form

It will first be shown that the solution to the equations is also the minimum point of a surface that mathematicians call the quadratic form, where:

$$q(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{b}^T \mathbf{x} + c.$$

The notation $\mathbf{x}^T \mathbf{y}$ means the inner product of the vectors (i.e. the sum of $x_{[i]} \cdot y_{[i]}$, which is a scalar.) Since the product of $\mathbf{A} \mathbf{x}$ is a vector, then $\mathbf{x}^T \mathbf{A} \mathbf{x}$ is a scalar, as is $\mathbf{b}^T \mathbf{x}$, and (with c a constant) the quadratic form, $q(\mathbf{x})$, is therefore a scalar (i.e. a single number). If x_1 and x_2 in the equations of Figure 4-3 are allowed to range over a set of values, then the quadratic $q(\mathbf{x})$ forms the surface shown in figure 4-5.

We shall skip some maths here and just say that the gradient of the quadratic form can also be defined and shown to be:

$$q'(\mathbf{x}) = \frac{1}{2} \mathbf{A}^T \mathbf{x} + \frac{1}{2} \mathbf{A} \mathbf{x} - \mathbf{b}$$

and if \mathbf{A} is symmetric this reduces to:

$$q'(\mathbf{x}) = \mathbf{A} \mathbf{x} - \mathbf{b}.$$

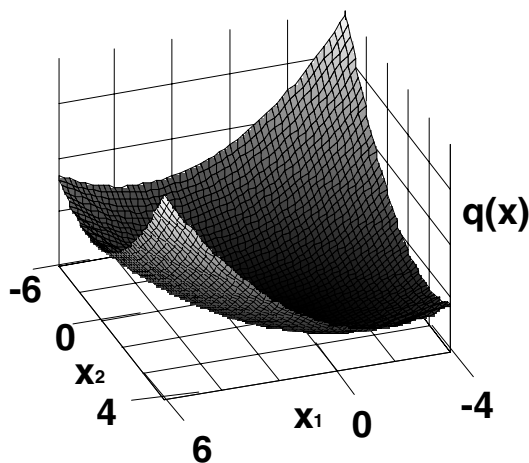


Figure 4-5: The surface produced by the quadratic, $q(\mathbf{x})$.

Notice that if the gradient $q'(x)$ is set to zero then the original equation $Ax = b$ is obtained. If A is a positive definite matrix, as well as symmetric, then this solution is also a minimum of $q(x)$. Positive definite is mathematically defined as $x^T Ax > 0$ for every non-zero value of x , and means that the quadratic surface will be as in Figure 4-5, with its minimum at the bottom of the surface and without turning points. The implication of this is that if the matrix A is symmetric and positive definite, then finding the value of x that minimises the quadratic form $q(x)$ will solve the equation $Ax = b$. It so happens that many, if not most, physical systems give rise to symmetric positive definite matrices, and hence a solution method will have wide applicability.

The method of steepest descent

Before considering the conjugate gradient method we will look at another iterative method called steepest descent. In this method a set of starting values for x are assumed and the algorithm then takes a series of steps, each time moving further down the surface of the quadratic form (Figure 4-5) until an x value is reached which is within a defined and acceptable error level from the minimum (i.e. the solution point). If the starting point that has been assumed for x is x_0 and the first step is to x_1 then:

$$x_1 = x_0 + s$$

and the question is: how should s be derived, such that successive steps will always head towards the minimum of $q(x)$? In the method of steepest descent s for the first step becomes:

$$s = k r_0$$

where k is a scalar and r_0 is the residual value of the vector b , showing how far that solution is from the correct value of b . Hence:

$$r_0 = b - Ax_0.$$

The residual value of b is a valuable solution indicator because the residual is a transformation (by A) of the error from x space to b space and because $r_0 = -q'(x_0)$, giving the direction of steepest descent. Thus when successive iterations of x are taken, the algorithm moves down the direction of steepest descent. No proofs have been given here for the above assertions and for these the reader is directed to the literature.

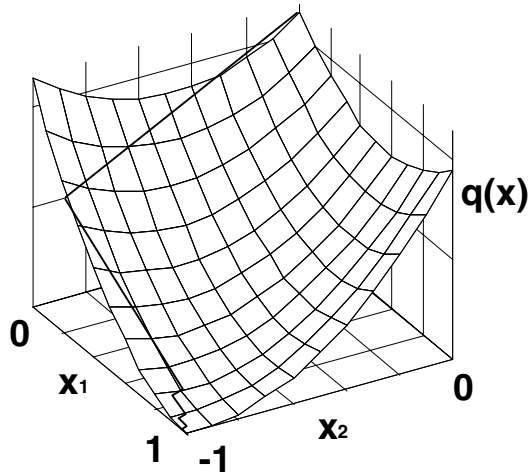


Figure 4-6: Solution steps for the method of steepest descent.

The next question is: what value should k take on successive iterations? Again, the (complex) mathematical derivation will not be given here, except to say that it is obtained by consideration of the gradients:

$$k = -\frac{\mathbf{r}_0^T \mathbf{r}_0}{\mathbf{r}_0^T \mathbf{A} \mathbf{r}_0}.$$

For the equations of Figure 4-3, the successive steps to solution are shown in Figure 4-6, which gives a larger scale view of a section of the quadratic function. A characteristic of the method of steepest descent is that each gradient is orthogonal to the next, so that a zigzag path is traced.

The method of conjugate gradients

Since the method of steepest descent produces a simple algorithm, and (for a large class of physical systems including the resistor mesh of Chapter 3) guarantees convergence, why look for an alternative? The reason is that the rate of convergence can be very slow, and it is an improved rate of convergence that the method of conjugate gradients seeks to provide. The algorithm starts off identically to steepest descent, but then requires that the search directions are produced by conjugation of the residuals \mathbf{r} .

Conjugation is mathematically defined as two vectors, \mathbf{c}_i and \mathbf{c}_j , that are \mathbf{A} orthogonal to each other if:

$$\mathbf{c}_i^T \mathbf{A} \mathbf{c}_j = 0.$$

To understand what this means unfortunately requires extensive consideration of extremely complex mathematics involving eigenvectors, eigenvalues, spectral condition numbers and many other difficult concepts. Here we will just say that in the method of conjugate gradients we are still headed down the gradient of the quadratic surface of Figure 4-5, but with successive steps constrained by conjugation to reduce the number of iterations to a minimum. Those readers wishing to understand the details of this refined and extremely elegant algorithm are referred to the literature (e.g. reference [2]).

Comparing the two algorithms

In the algorithm of Figure 4-2, all the initialisation and the steps up the calculation of the first residual by $\mathbf{f} = \mathbf{b} - \mathbf{A}\mathbf{x}$ are identical in steepest descent and conjugate gradients. However, in steepest descent this residual is used directly in the calculation of the next iteration of \mathbf{x} , but in conjugate gradients the residual is modified after the $\mathbf{b} - \mathbf{A}\mathbf{x}$ step to create the conjugated form, before being used to calculate the next \mathbf{x} . The successive steps to solution for conjugate gradients are shown in Figure 4-7, and compared to Figure 4-6 it can be seen that there are far fewer iterations than for steepest descent. In fact the conjugate gradient method will always (within the limits of floating point arithmetic referred to in an earlier section) converge in n iterations ($n = N \times N$

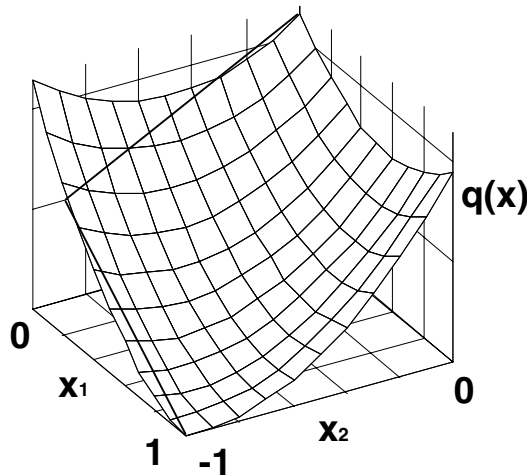


Figure 4-7: Solution steps for the method of conjugate gradients.

for the resistor mesh of Chapter 3). In most cases convergence is considerably faster than this, depending on the eigenvalue properties of the matrix \mathbf{A} .

Preconditioning

There are many variations on both the Gaussian elimination and the conjugate gradient methods described in the literature. Generally, a technique called preconditioning is employed in order to increase the rate of convergence. This changes the matrix equation to the form:

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}.$$

A secondary matrix then has to be computed also, and a net speed increase is realised if the overall rate of convergence is greater than the increased computation time due to the preconditioning matrix. The art, of course, is in the choice of \mathbf{M} . Preconditioning is not used in the VIZIMAG algorithms, but methods that mathematically inclined readers may care to experiment with are the partial Gauss–Seidel algorithm [3] and the preconditioned conjugate gradient scheme [4]. Also care must be taken not to cause filling in of diagonals in the case of the conjugate gradient method.

Timing and storage results

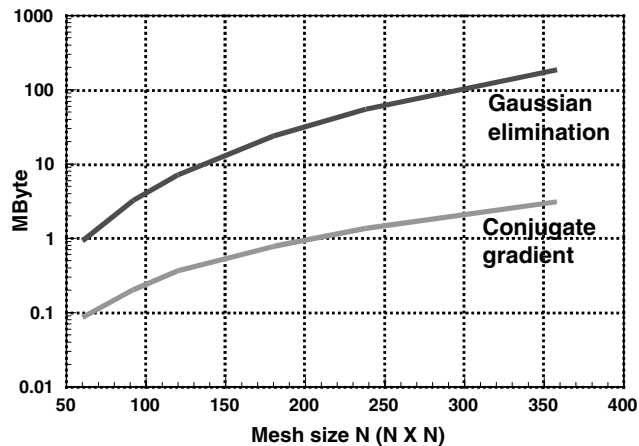


Figure 4-8: Storage (matrix coefficients, currents and voltages) during the analysis routine. Note that virtual storage came into use at 180×180 mesh size in the gaussian elimination algorithm. 200 MHz Pentium processor with 32 MByte RAM and no other applications running.

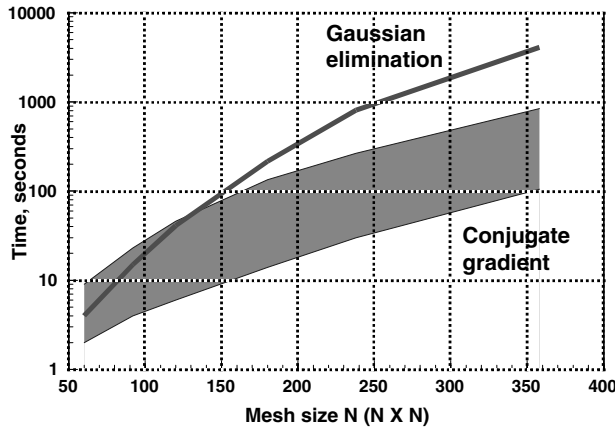


Figure 4-9: Analysis times. The variation in times for the conjugate gradient method is a function of model complexity. 200 MHz Pentium processor with 32 MByte RAM and no other applications running.

The timing results will be system dependent. Larger RAM size will reduce the onset of virtual storage. Higher processor clock frequencies will reduce time. Multiple applications running at the same time will reduce available memory and increase times.

Indirect addressing for arrays

The arrays used for matrix storage are resizable to allow for different mesh resolutions, which require the use of indirect addressing (commonly referred to as 'pointers'). However, modern compilers and computer architectures handle such structures very efficiently, and no increase in analysis times was measurable as a result of this.

Choice of analysis method and mesh size

A mesh size of 100×100 is a good choice for the first trials of a model, and higher resolution meshes are probably best left for final runs (because of execution time) unless a very fast system is used. The conjugate gradient method is usually the preferred solution method, both for speed of solution and memory usage. However, below a 130×130 mesh resolution the Gaussian elimination technique may be faster if complex models are run because the conjugate gradient method is quite variable in execution time.

For example the following analysis times (in seconds) were obtained depending on the complexity of the model.

Mesh size	Conjugate gradient		Gaussian elimination
	Simple model	Complex model	All models
60×60	2	9	4
92×92	3	23	15

It can be reasonably said that the objective of a fast solution time has been achieved with a typical computation time of under 30 seconds for a 150×150 mesh. With regard to larger mesh sizes and complex models, computation times are longer, for example 266 seconds for a 238×238 mesh. However, fast results can be obtained with coarser meshes, and the model then simply switched to a large mesh for a single run for final storage. Once stored, models can simply be taken from disk. Presenting the result from pre-computed mesh loop currents is always a matter of seconds compared to the longer analysis times.

PC hardware will continue to improve with time. For example, clock speeds will increase to over 1 GHz, system memory will be typically 128 megabytes or greater, synchronous memory will be used and system buses may be 64 bits or more. Combined together we might expect an improvement of 20× or more in the processing time of the numerically intensive algorithms described in this book and then the 266 second analysis time of a 238×238 mesh will reduce to under 15 seconds.

It should be emphasised that the numerically intensive computations needed to solve the mesh equations require that no other applications be running if the fastest solution time is required.

16 and 32 bit operating systems

A version of VIZIMAG was compiled with Delphi to run on Windows 3.1, which is a 16 bit operating system, and the algorithm timing compared to operation under Windows 95, a 32 bit operating system. The 32 bit version (as supplied with this book) ran nearly five times faster. Because of the computationally intense nature of matrix solving, it is not recommended to run VIZIMAG under 16 bit operating systems.

Progress indicator

When VIZIMAG runs, a progress indicator is shown. In the Gaussian elimination option, it is possible to compute the number of iterations needed, and therefore the indicator shows the exact state of the solution progress. However, the number of iterations varies considerably in the conjugate gradient option, and therefore the indicator is quite approximate. Generally, a solution will complete between about one third and two thirds of the maximum reading. If the indicator approaches maximum it is an indication that the region of floating point round-off error is being approached.

Error level setting

In the conjugate gradient option, the choice of error level with which to compute the solution is quite important (see Chapter 5 for a more detailed description). For the purposes of visualisation we are looking for a reasonable approximation rather than an exact solution, and some experiments showed that an error level of 0.2% gave magnetic field line patterns visually correct within a pixel or two, and therefore this has been selected as the default error setting. The ANALYSE-ERROR LEVEL option allows any other error level to be optionally set. The SHOWALG program (as described in the next chapter) allows the effect of different levels to be seen.

Understanding the matrix solving algorithms

This chapter has described two algorithms for the solution of large sets of sparse matrix form simultaneous equations. The detailed operations of the algorithms are quite complex, and for those interested in investigating them further the next chapter describes an application, included on the CD-ROM, for the visualisation of the algorithms in operation. If this is not of interest to you, then you may skip Chapter 5.

This Page Intentionally Left Blank

Visualising the algorithms

I must admit that when faced with a large set of many thousands of simultaneous equations, even in such a fascinating pattern as the tri-diagonal banded matrix, they do not exactly jump off the page shouting 'solve me'. One can only hold in awe the generations of mathematicians that have created such a masterpiece as the conjugate gradient algorithm. For those of us to whom such elegant vistas are somewhat shrouded in mist, some help is needed in visualising the operation of these algorithms. The application SHOWALG provides a visualisation of both the Gaussian elimination method, and the conjugate gradient option.

Conjugate gradient visualisation

When SHOWALG is opened the screen is very similar to the standard VIZIMAG application, except that an option panel appears as shown in Figure 5-1. Initially ignore this panel and create a simple permanent magnet on the screen. Clicking on the RUN speedbutton starts the normal VIZIMAG analysis routine, except that this time three graphs appear on the left of the screen, and contours are plotted every 20 iterations as the conjugate gradient algorithm gradually builds up the solution. The



Figure 5-1: Options panel for SHOWALG conjugate gradient mode.

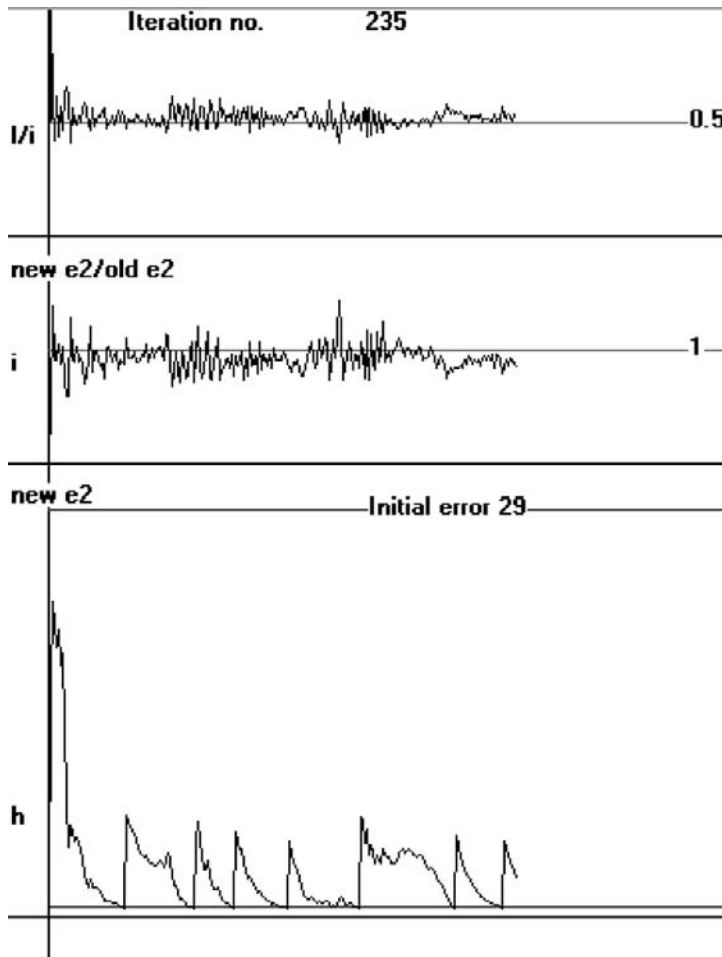


Figure 5-2: Graphs shown in the conjugate gradient mode.

graphs are shown in Figure 5-2 for a magnet after completion of the algorithm.

The graphs show some of the factors in the algorithm of Figure 4-2. The first graph is the l/i factor that is used to determine whether the algorithm has entered floating point round-off. The second graph is the factor i , which is the $(\text{new error})^2/(\text{old error})^2$, and the final graph is h , which is the $(\text{new error})^2$. Also shown is the iteration number.

On the h graph is printed the value of the $(\text{initial error})^2$ when the algorithm is initialised. The lower red line on the h graph is the $(\text{initial error})^2$ divided by 20. When the error drops below this value the graph value is

multiplied by 10. This happens continuously and is why the graph shows step changes; this technique allows small errors to be shown without resorting to a log scale. When the number of iterations takes the graphs to the extreme right-hand side, the plotting starts again from the left but in a different colour.

For the simple case of the magnet the algorithm can be seen building up the magnetic field line pattern gradually, starting from the source of flux, with the error dropping in a relatively simple exponential manner.

Now change the value in the 'plot every X iterations' box to 200 and load and run the file 'show_complex_pt001D' file. This time we see that the error initially grows quite considerably until it begins to fall, and the number of iterations is much larger so that several coloured graphs are drawn. Finally we get into floating point round-off and the l/i factor begins to fall until the iteration is truncated when l/i reaches 0.1.

Use of the options panel

Save Ref: If this box is checked a reference file is saved along with the data and results files following a FILE SAVE operation. The reference file is used if the **Load Ref** box is checked.

Load Ref: If this box is checked the reference file (if any) is loaded during a FILE-LOAD operation. When the model is run the reference magnetic field lines are displayed in red. The purpose is to allow a final target magnetic field line pattern to be permanently visible while the algorithm runs. Since the reference file can be created at any accuracy and error level, it becomes possible to create a highly accurate reference and to compare the results of running models at other error levels.

Show whole mesh: If this box is checked the whole mesh, including the normally undisplayed boundary region is shown. Note that models are not rescaled by this mode and hence models must be created specifically for this option. To initiate the whole mesh mode, first clear any models from the drawing area (by clicking on the NEW FILE speedbutton). Then check the box and click on the RUN speedbutton. Models can now be created or loaded.

Plot every X iteration: It is possible (and interesting) to set this number to 1 and see the effect of the pattern build up for every iteration. However, the plotting operation slows down the algorithm considerably, and at times it is

more helpful to set this to a larger number and hence to sample the pattern build up at larger intervals.

Set percentage error: This option allows any error level to be set (which can be in standard or high accuracy analysis mode) for completion of the algorithm.

Example of use of the options

Select the *Show whole mesh* mode as described above and check the *Load Ref* box. Set the *error level* to 0.1, the *Plot every X iterations* to 200 and then load and run the file 'Show_average_pt001D-all'. This file was created with the whole mesh visible and to 0.001% error level in double precision (i.e. the high accuracy mode). The run will show the effect of a higher error level in single precision arithmetic (the standard analysis mode). Now change the *error level* to 0.02 and run again to show the error at the 0.02% default level used in VIZIMAG.

Sample files included with SHOWALG

The following sample data files are included with SHOWALG. All the files were created with a reference and to 0.001% error level in the high accuracy analysis mode. Files suffixed with 'all' should be used in the *Show whole mesh* mode:

File name	Description
Show_simple_pt001D	A permanent magnet. This model runs with a small number of iterations.
Show_average_pt001D	Coils and regions of high permeability chosen to give an average number of iterations.
Show_complex_pt001D	A coil, core and solenoid which runs into round-off error in standard accuracy analysis mode.
Show_core_coil_pt001D	A core and coil similar to Figure 2-10. This also runs into round-off error.

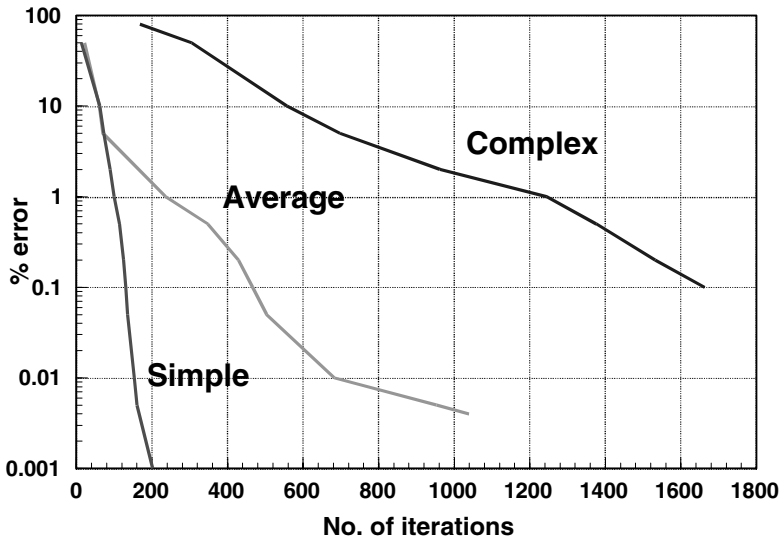


Figure 5-3: Error level versus number of iterations.

The simple, average and complex models were the ones used to assess the algorithm timings of figure 4-9.

Error level versus number of iterations

Using the simple, average and complex models described above, Figure 5-3 was generated to show how the number of iterations is influenced by the error level. These runs were all performed in the standard accuracy analysis mode. Note that the average and complex models are shown up to the point of round-off error termination, when no further improvement was possible. In the high accuracy analysis mode a lower error level would, of course, be possible.

Gaussian elimination visualisation

Selecting ANALYSE-ANALYSE OPTIONS-GAUSSIAN ELIMINATION brings up the options panel of Figure 5-4. The Gaussian elimination algorithm runs in a predetermined way, and this visualisation simply shows the row and column scanning of the computation block in operation, and the way that the upper diagonals fill in during the forward elimination phase. Initially ignore the options panel and set up and run a simple permanent

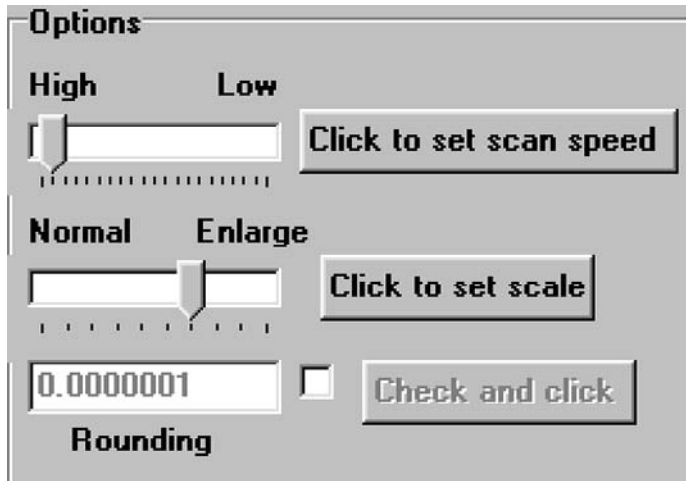


Figure 5-4: Gaussian elimination options panel.

magnet model. The diagonals of the coefficient matrix are drawn, the computation block is shown being loaded with data (in turquoise), scanned (in purple) and diagonal data left behind (olive). To give a reasonable scale only the top part of the matrix is drawn in. The computation block scanning may be slowed by clicking the *Set scan speed* button after setting the slider (this may be done during the scan). A larger or smaller scale may be set using the *Set scale* option, but this must be done before starting the run.

A point to note in the Gaussian elimination algorithm is that many of the lower diagonal points computed during scanning, and which should finish as zeros are, in fact, *computed zeros* and are thus subject to floating point round-off errors. As visualised so far, the algorithm has been deliberately set with a rounding error limit, to show the lower diagonals being set to zero. If the *Rounding* box is checked and the value set to zero (set the number and then click the button) then only true zeros will be plotted as such. In this case it will be seen that the lower diagonals are not left as zero. If the *Rounding* value is set to 0.0000001 then this rounding error limit is applied to all computed numbers. In this case it is seen that the upper diagonals are also left with many zeros as the scan starts, but full upper diagonal fill-in does eventually take place.

Completing the theory

So far we have seen how an electrical resistor mesh can be used to simulate a magnetic field pattern, how the equations of the mesh are created and how they can be efficiently solved.

Before we can move on to using the technique, however, the theory must first be completed by considering the necessary requirements of a boundary region, an external magnetic field function and the magnetic flux density function. Chapter 6 looks at the boundary region and external field, plus a way of smoothing the results. Chapter 7 shows how the magnetic flux density function is created.

This Page Intentionally Left Blank

The boundary region, smoothing and external fields

Why a boundary region is needed

Magnetic fields extend out from a source without limit, and any magnetic field line leaving a source must eventually return and close on itself. When these properties are coupled with the fact that magnetic field lines can never cross, then it becomes clear that any attempt to confine a magnetic field into a finite modelling space, such as our fixed number of meshes, must introduce distortions. We have already encountered this in Figure 3-7, where the magnetic field lines from a single wire became cramped near to the edge of the mesh space.

Finite element modelling techniques have exactly the same problem, and solve it by allowing a large boundary region around the model. Although there are no fixed rules, a boundary space of about five times the modelling space is typically allowed. The finite element technique, of course, has the advantage that mesh sizes can be varied, and so quite coarse meshes can be used in the far boundary field, with a finer mesh in the region of the model.

In our case, with fixed mesh sizes, a boundary region of this size would be uneconomic in both computation time and storage, and therefore there is a requirement to reduce the ratio of boundary space to model space, whilst keeping the deviations from the mathematically correct field patterns small enough to produce a sensible approximation. By itself a small boundary region will not produce an acceptable result. Consider Figure 6-1 where the use of a 5:1 boundary region is compared to a 10% region. Plainly there is a considerable distortion within the modelling space in the latter case. We need, therefore, a way of making the boundary region appear larger than it actually is, with the freedom that the distortion within the boundary region itself is of no importance.

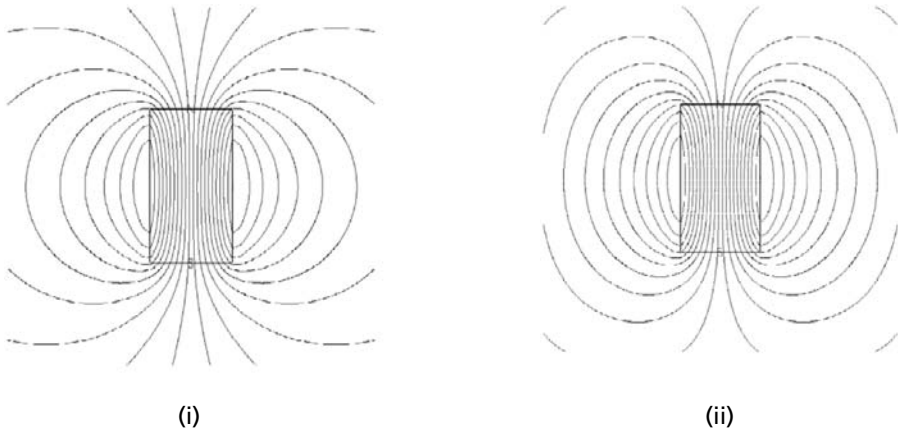


Figure 6-1 (i): $5 \times$ boundary region. (ii): 10% boundary region.

Mesh impedance characteristics

If we look at the basic mesh again, for example the mesh of Figure 6-2, the currents flowing in resistor R1 near to the centre of the mesh, will be determined by the parallel connection of all the other mesh resistors seen from resistor R1.

Intuitively we can see that changing the value of resistor R2 near the periphery of the mesh will have a very little effect on the current flowing in R1, compared to changing a resistor value close to R1. However, for a current flowing in resistor R3, changing the value of resistor R2 will have a large effect on the current in R3, simply because it is physically closer. One way of looking at this is to plot the total mesh resistance as seen from one particular resistor. If we plot the total mesh resistance as we move across the line A–A shown in Figure 6-2, from the centre to the periphery, we get the graph of Figure 6-3.

Here three conditions are plotted: firstly the mesh resistance, with all individual resistors having a value of unity, for a mesh size of 60×60 but plotted only up to 50×50 , so leaving a 10% boundary region all round. Next, the mesh resistance is plotted out to the 50×50 size, but with the actual mesh extended to 550×550 to provide a boundary region five times the modelling space all round. Finally the mesh resistance is plotted when the meshes extended to 60×60 , so giving him a 10% boundary region all round, but with the resistor values on the final two rows and columns reduced to a value of 0.2. It can be seen that the addition of the

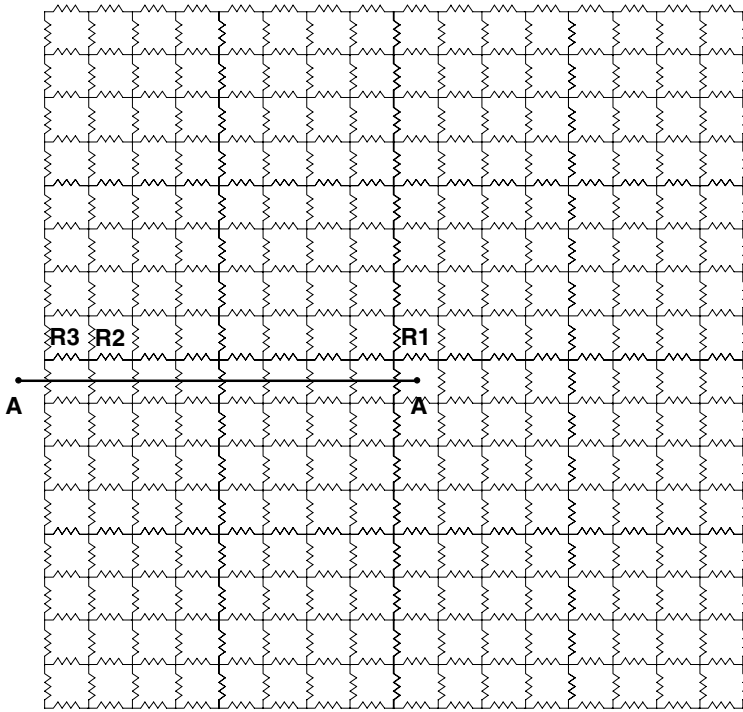


Figure 6-2: A basic mesh.

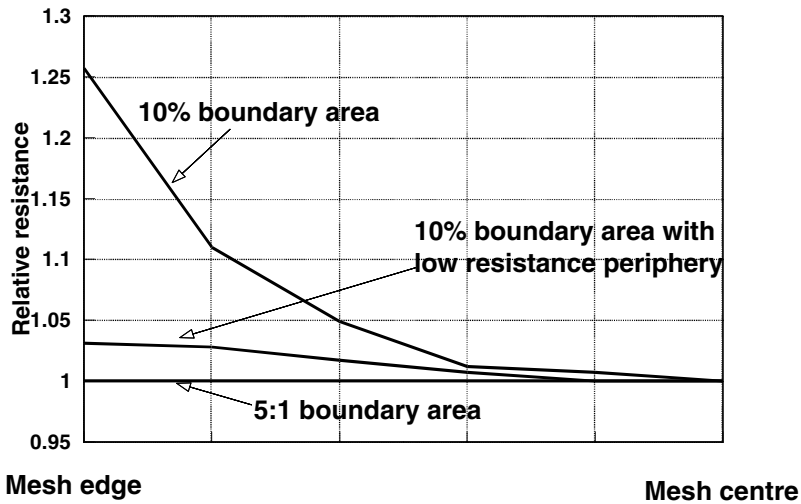


Figure 6-3: Mesh resistance from centre to edge.

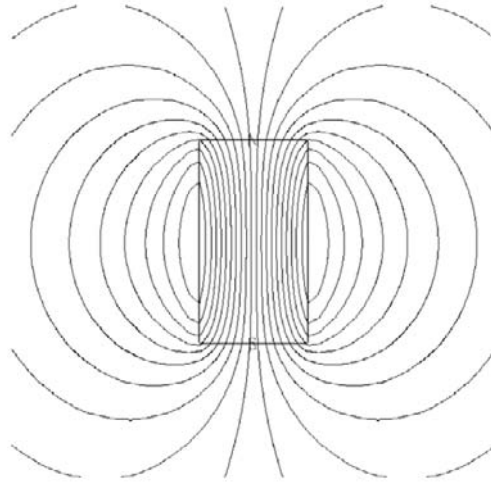


Figure 6-4: 10% boundary region, with low resistance periphery.

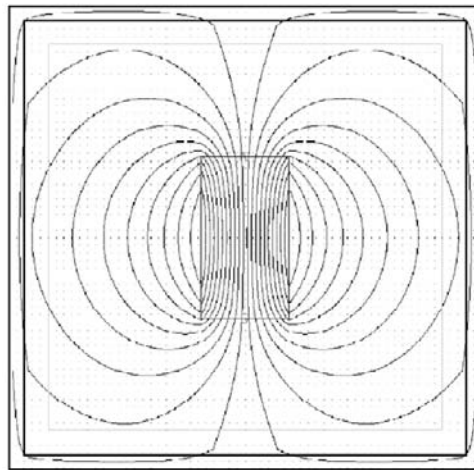


Figure 6-5: The whole mesh region, including boundary layer.

low resistance outer mesh resistors gives a closer approximation to a physically larger mesh.

Figure 6-4 repeats the model of Figure 6-1(ii), but with the peripheral mesh resistors reduced to a value of 0.2. We now get a closer approximation to the large boundary region case of Figure 6-1(i). Figure 6-5 shows the

same model as Figure 6-4, but with the boundary region made visible and with the low resistance peripheral mesh resistors outlined.

Distortion performance

The technique reduces, but does not eliminate, distortion, and objects drawn close to the edge of the visible mesh region will show magnetic field line patterns that are less well balanced than for objects close to the centre of the drawing area. For example, Figure 6-6 shows a bar magnet near the edge of the modelling space with a low resistance 10 per cent boundary region and a 5 \times boundary region for comparison.

Some improvements to the technique may be possible: e.g., some gradation of the peripheral resistor values to the edge and to the corners. It is left to readers to experiment further in this area. Nevertheless, the overall compromise produced by the method as implemented is a reasonable approximation for visualisation purposes. The program SHOWALG can be run to show the whole magnetic field pattern of a model, including the boundary region. Also, in the VIZIMAG application, setting the variable 'Plotall' to zero in the source code *FormCreate* event handler will cause the whole mesh area to be drawn, including the boundary region.

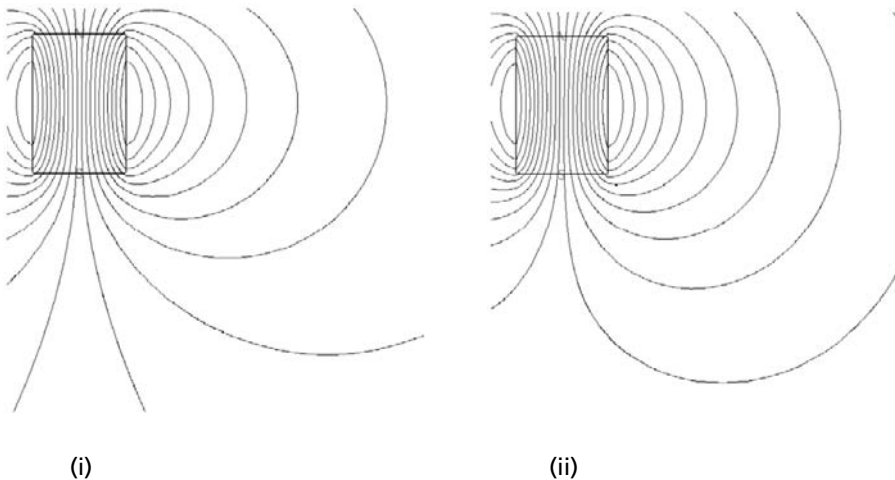


Figure 6-6 (i): 5 \times boundary region. (ii): 10% boundary region.

Smoothing

When low resolution meshes are used (generally to achieve a high computation speed), there can be problems with flux sources that are created at shallow angles to the mesh axes. This is because the mesh resolution is not great enough to accurately model the line slope. The effect is of a distortion of the magnetic field line pattern close to the object. An example is shown in Figure 6-7. Note that the distortion rapidly disappears away from the actual flux source. The effect can be reduced by choice of smoothing in the ANALYSE-SMOOTHING OPTION menu. Figure 6-8 shows the same model with smoothing set to high.

Smoothing is performed by taking an average of the loop current and a proportion of the loop currents of the surrounding mesh squares. The proportions used are 0.1 for the normal (default) setting or 0.25 for the high setting. The effect is not a problem if a 'quick look' result is all that is required, and can be eliminated by increasing the mesh resolution. Figure 6-9 shows the same model, with no smoothing, on a 150×150 mesh.

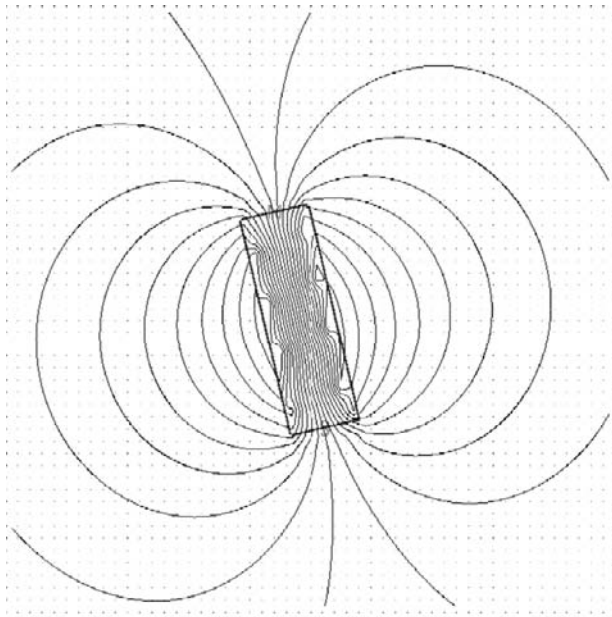


Figure 6-7: Coil of permeability 500 on a 50×50 mesh (no smoothing).

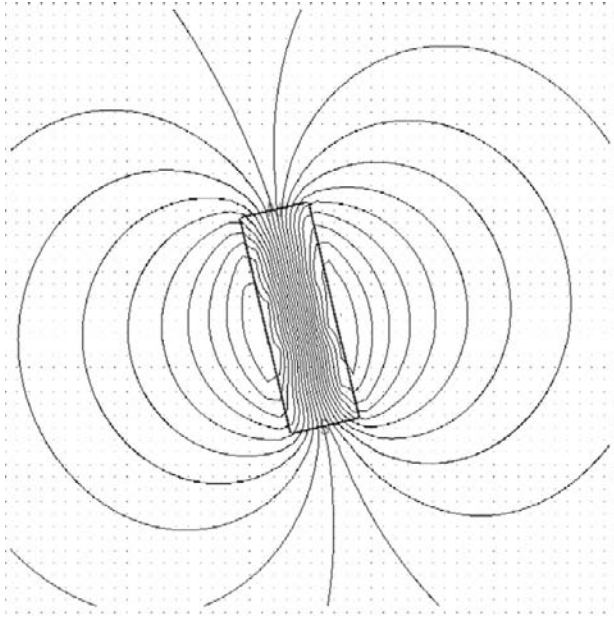


Figure 6-8: Smoothing set to high.

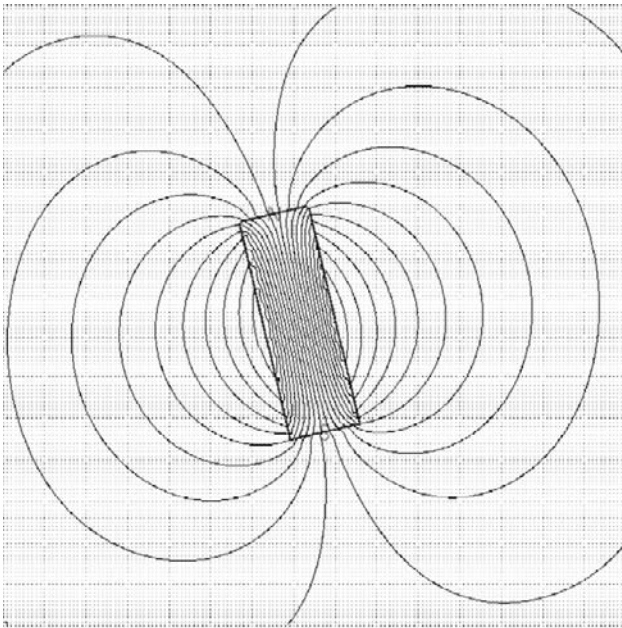


Figure 6-9: 150×150 mesh, no smoothing.

Using the boundary for external field simulation

An extremely useful function in any magnetic field visualisation method is that of generating an external magnetic field (e.g. the earth's field). Now that an unplotted boundary region is present, it becomes possible to use this as an area where such an external field can be generated. By placing voltage sources on the penultimate meshes around the periphery of the whole mesh, a flux source is created, where the internal field lines become an approximation to an external field source when viewed in the modelling area of the mesh (i.e. excluding the boundary region) as shown in Figure 6-10. Placing voltage sources at the periphery of the mesh (and in the low resistance extreme periphery region) does, of course, produce a highly distorted pattern, since there is very little area left for field line closure.

To linearise the pattern, two sets of orthogonally placed sources are used. Figure 6-11 shows how the individual sources are placed (for a left to right field) to build up the composite pattern of Figure 6-10. This figure includes the whole boundary region, and it should be remembered that the purpose is to produce as linear a pattern as possible within the visible mesh region only. It is important, of course, that the source placements can be scaled so that the field pattern is satisfactory over a wide range of mesh sizes: in the case of VIZIMAG from 60×60 to 300×300 .

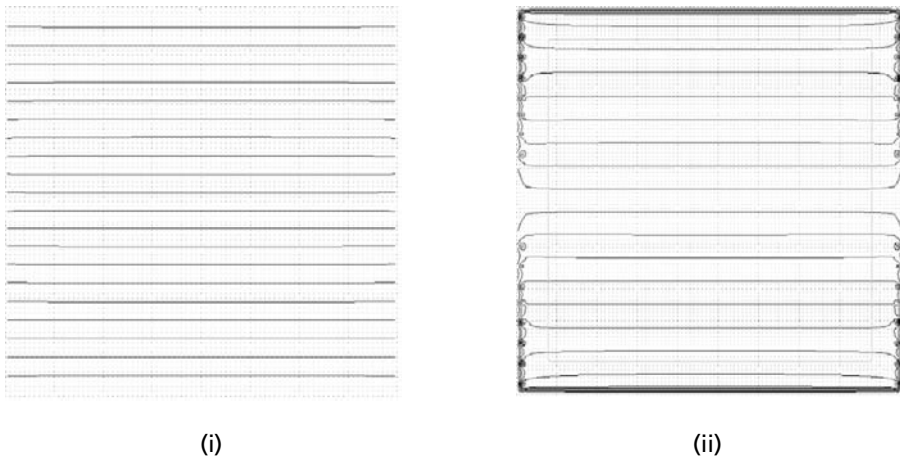


Figure 6-10 (i): External magnetic field. (ii): External magnetic field including view of the usually undisplayed boundary region.

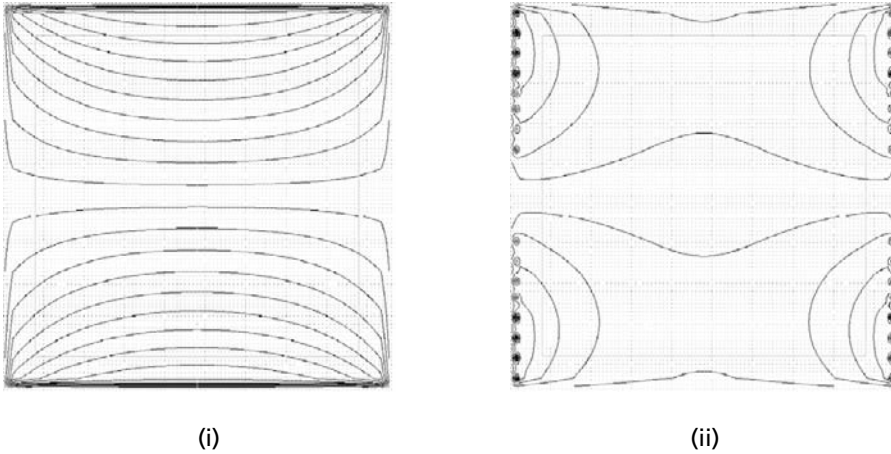


Figure 6-11 (i): Primary source. (ii): Orthogonal sources.

The algorithm to create the external field pattern will be found in the source code procedures *Eset*, *AddTBWires* and *AddLRWires*. It should be noted that the source strengths and placements have been obtained entirely by experiment, so no logical pattern should be sought. Readers may care to attempt a more theoretical treatment or to develop the method further.

Magnetic field lines and magnetic flux density

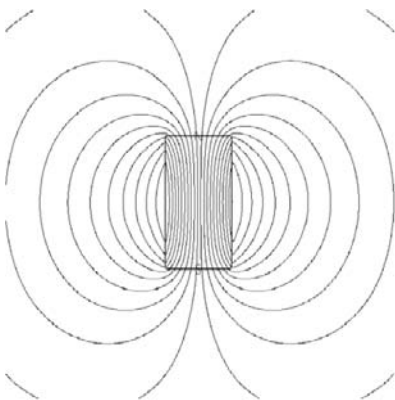
So far we have concentrated only on the production of magnetic field lines. A final step before considering the actual model creation is a method of visualising the magnetic flux density, which is described in the next chapter.

This Page Intentionally Left Blank

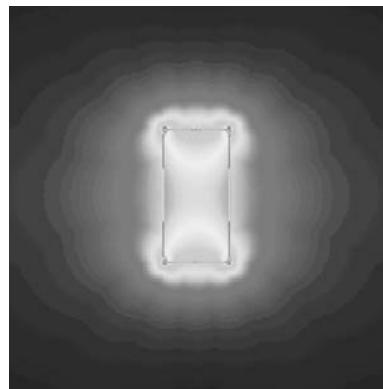
The magnetic flux density function

Magnetic field lines are instructional in themselves, but often the density function is also required as an aid to visualisation, for example to see where the points of peak magnetic flux density occur.

If we look at the field pattern of the simple coil shown in Figure 7-1 we know that the magnetic flux density is highest where the magnetic field lines are closest together. Since these field lines are produced by contouring a surface (refer back to Figures 3-9 and 3-10), then it is apparent that an analogy to magnetic flux density may be obtained by finding the gradient of the surface at any point. The gradient is simply found by examining each mesh loop current value and taking the modulus of the difference to the adjacent eight connected mesh points as shown in Figure 7-2. A colour plate of Figure 7-1 (ii) can be found between pages 6 and 7.



(i)



(ii)

Figure 7-1 (i): Field pattern of coil. (ii): Magnetic flux density function.

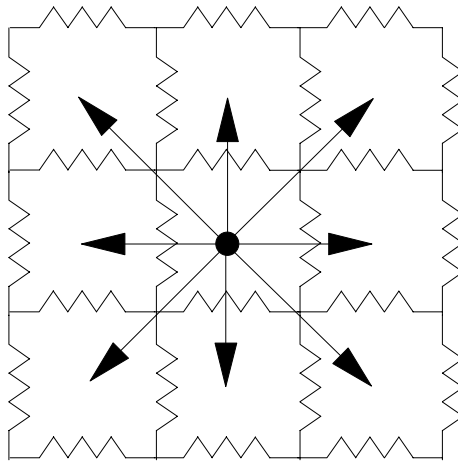


Figure 7-2: Eight connected mesh points for magnetic flux density determination.

A correction factor of 0.7071 is applied to the corner values because of the longer axis. Note that the limited number of points examined (i.e. 8) gives rise to some artefacts in the plots, which can be seen as a form of 'ray' pattern (see Figure 7-7 as an example).

The largest of the eight values found becomes a representation of the density function. This is plotted as a grey scale intensity from white as maximum to black as minimum, or using a colour scale with red as maximum and black as minimum. The colour scale is created as shown on the CIE colour diagram in Figure 7-3.

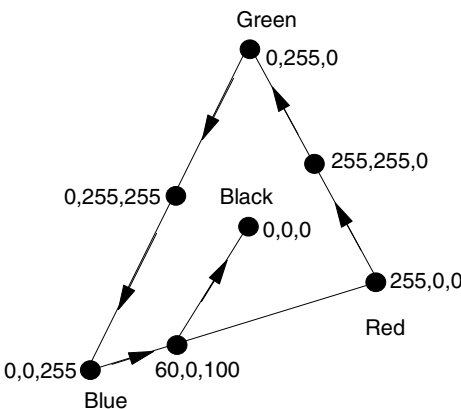


Figure 7-3: Program colour coding for R,G,B applied to magnetic flux density function.

Magnetic flux density filter and log scale

Magnetic flux density is high on the flux sources themselves (e.g. magnets, coils, etc.) and falls away quite rapidly, so that detail may be hard to make out in regions remote from the sources. If we plot the magnetic flux density from the centre of the magnet shown in Figure 7-1 horizontally outwards, this rapid fall-off can be seen in Figure 7-4.

To assist in visualising magnetic flux density detail in the tails of the function two different plotting options are provided. The first is a form of filter function, in which a mesh loop current level can be set and above

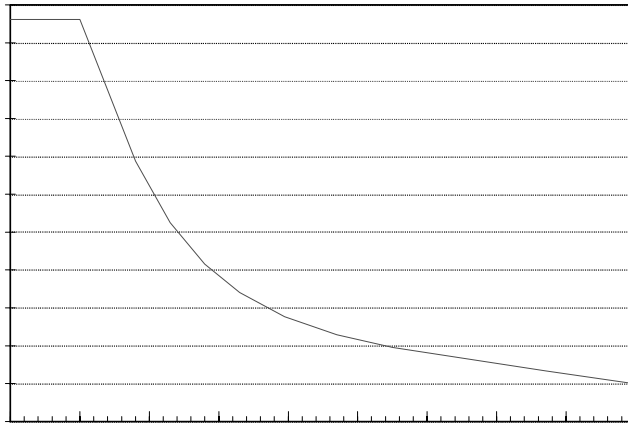


Figure 7-4: Magnetic flux density fall off for bar magnet.

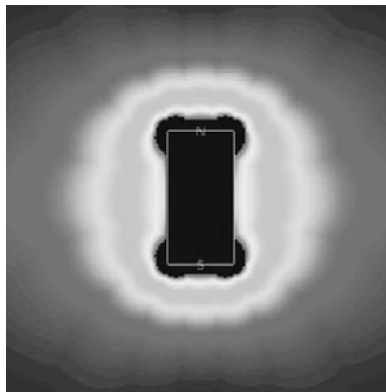


Figure 7-5: Linear plot, with filter.

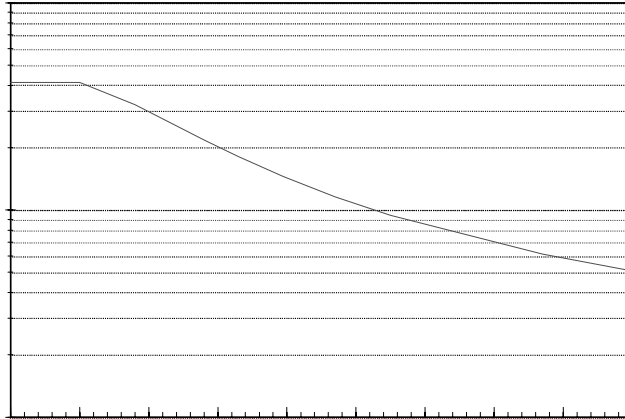


Figure 7-6: Log plot of magnetic flux density fall-off.

which the mesh current is set to zero for plotting purposes. This has the effect of progressively blanking out the higher density levels and allowing detail in the far field to be seen more clearly, as can be seen in Figure 7-5.

Far field detail can also be enhanced by the use of a logarithmic scale. In Figure 7-6 the magnetic flux density graph of Figure 7-4 is plotted using a log scale, showing how the lower level detail in the far field is accentuated. In creating the log function it is necessary to manipulate the data to avoid taking any logs of numbers less than 1. The details of how this is done will be found in the source code for the function *Fluxfill* in unit VIZ3. The result of using the log function option is shown in figure 7-7. Colour plates of Figures 7-5 and 7-7 can be found between pages 6 and 7.

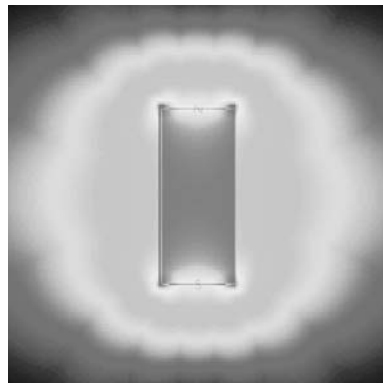


Figure 7-7: Log function plot.

Magnetic field lines are a very useful way of visualising the linkage and interaction of magnetic components, but can give the impression that the field is very clearly delineated. Viewing the magnetic flux density function, particularly with grey scale and with a filter in place possibly shows more clearly the true nature of the magnetic field that surrounds a flux source, and diffusely permeates into neighbouring regions of space.

Putting the theory to work

We have now completed all the theoretical steps needed to analyse models and hence to plot the magnetic lines of force and the magnetic flux density function. In the next chapter we move on to the application of the theory, with the creation of models in a practical computer program.

This Page Intentionally Left Blank

Model creation

In Chapter 1 some of the aims of this particular visualisation technique were stated to be the rapid creation of models, fast and automatic translation of the models to the underlying equations and solving those equations very quickly.

We have seen how the equations can be solved in a few seconds for mesh sizes of 50×50 and in less than a minute for a 200×200 mesh. However, the methodology described in this book would be incomplete without a working computer program to perform the modelling function.

Model functions provided

The first step is to decide what basic models should be included. The VIZ-IMAG application provides the following, selected by menu or application speedbutton icon:

Guidelines	Lines used as a drawing aid, and not stored on disk or printed
Rectangular magnetic domain or air gap	A rectangular area of specified permeability, or an air gap
Circular magnetic domain	A circular area (torus) of specified permeability
Permanent magnet	Defined by strength and with a fixed permeability of 1.1
Coil	A rectangular area defined by strength and permeability
Solenoid	A circular coil (torus) defined by strength, permeability and magnetic field line direction
Wire	Assumed to be a current carrying wire running perpendicular to the screen, defined by strength and current direction
External field	Specified by strength and direction

Data structure

Data is stored in three files: one with a .MAG extension uses single precision floating point numbers to store object data and loop current results; a second with a .FLG extension stores integer number flag data; a third with a .TLE extension stores the image title. A further preference file (.PRF extension) is also automatically created and a file giving the working directory (.DIR extension).

Structure of object storage

The various objects (e.g. magnets, coils, etc.) that VIZIMAG can create are stored in a *primitive* form, where the data for each object consists of the significant drawing co-ordinates, the permeability and the strength. The primitive arrays are of fixed size, and allowance has been made for the storage of 300 wires and 100 of each of the other objects.

As an example the coil data is stored in *primitive_coil* and each object is stored in single precision arithmetic format as:

First point x co-ordinate
First point y co-ordinate
Second point x co-ordinate
Second point y co-ordinate
Third point x co-ordinate
Third point y co-ordinate
Fourth point x co-ordinate
Fourth point y co-ordinate
Permeability
Strength

primitive_coil[4] will therefore refer to the second point y co-ordinate.

Model generation

The logical steps in creating a model are:

- placing the main co-ordinates on the screen (and storing them);
- interpolating lines and curves between the points;
- determining which mesh numbers lie within the objects;
- setting the resistor values of these meshes to a value determined by the permeability;
- setting voltage sources in the meshes down the edge of the object.

Equation generation

Once the resistor values for all objects are determined, the coefficient matrix can be obtained by calculating the self- and mutual resistor values.

Event driven code

To speed model creation, the entering of co-ordinates is entirely visual using the mouse. Data entry (e.g. permeability and strength) is via entry panels automatically created on the screen. This is all done with event driven code to perform the geometric manipulations and calculations and the other logical steps. This is fully described for each program procedure, and in the code flow for each model, in chapter 11.

Editing

Once models have been entered, a practical program must also contain ways of editing the models. The functions provided by VIZIMAG (also fully described in chapter 11) are:

- COPY
- MOVE
- ROTATE (not circular magnetic domain, solenoid or wire)
- SWITCH POLES (not circular magnetic domain, solenoid or wire)
- CHANGE DATA
- FLIP LEFT/RIGHT (not circular magnetic domain, solenoid or wire)
- FLIP TOP/BOTTOM (not circular magnetic domain, solenoid or wire)
- SIZE HORIZONTAL (not circular magnetic domain, solenoid or wire)
- SIZE VERTICAL (not circular magnetic domain, solenoid or wire)
- SIZE BOTH (not wire)
- DELETE

All objects may be deleted with `TOOLS-CLEAR-ALL`, or by `NEWFILE`. Guidelines may be deleted by `EDIT-CLEAR ALL GUIDELINES` (all guidelines are deleted at once). `COPY`, `MOVE`, `ROTATE`, `SIZE` require the mouse button to be held down (i.e. dragged) while performing the action. Of course, many other edit functions could be provided, but since the full source code is provided, readers are free to expand the list as they wish with their own code.

Running the program

We now have all the theory and functions in place to allow the program to be run. This is discussed in chapter 9, whilst chapter 10 describes the models that are included in the program as examples of typical magnetic circuits.

Program installation and use

Installation

The VIZIMAG and SHOWALG programs and their source code are contained on the CD-ROM that accompanies this book. Place in the CD-ROM reader and the installation program should start automatically. If it does not then select START–RUN from Windows, type D:\Install (where D is the drive letter of your CD-ROM) and click OK. The installer program will run and will prompt for directories to install the two applications. Normally the default directories can be accepted. The installer will also create sub-directories (SOURCE and DATA) for the application source code and data files. The source code may be used directly as projects in standard Borland Delphi. To place a shortcut to the executable files on the desktop: open Windows Explorer and click on the Vizimag1.exe or the Show_Alg.exe file to highlight it. Then, in Explorer, select FILE–CREATE SHORTCUT and drag the shortcut to the desktop with the left mouse button. Alternatively right click on the Vizimag1.exe or Show_Alg.exe files and select CREATE SHORTCUT or SEND TO DESKTOP AS SHORTCUT. The programs may be uninstalled simply by deleting the two directories and any desktop icons created.

In use the VIZIMAG application shows the typical screen appearance shown in Figure 9-1.

Using VIZIMAG

The first three speedbuttons have the usual functions of NEW FILE, OPEN FILE and SAVE FILE. When the application is first opened, the drawing area comes up in NEW FILE mode, with a (visible) mesh size of 100×100 and with a grid pattern of dots and centre lines. The TOOLS menu can be used to change mesh sizes, apply a line grid, or remove the grid.

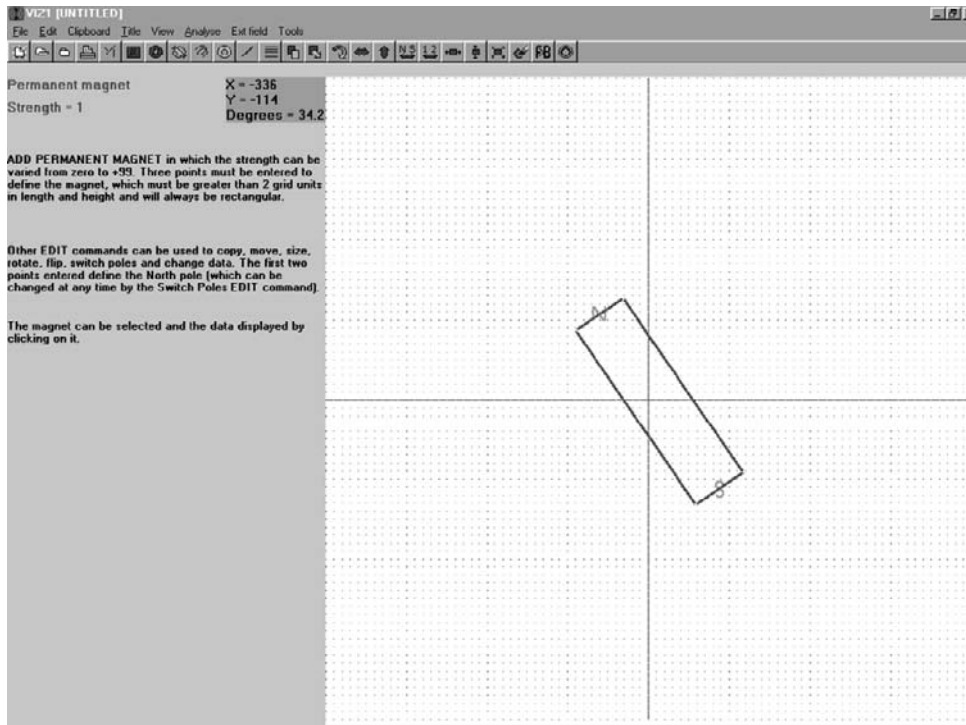


Figure 9-1: Typical VIZIMAG screen.

TOOLS-PREFERENCES can also be used to define the default state when the application is next run. Help panels display when functions are selected, but these can be disabled in the TOOLS menu. The left mouse button and the keyboard are used for data entry.

To enter objects on the drawing area, first select (add) an object type from the EDIT menu or speedbuttons. After selecting an object type, click and release the mouse button to define the first object point on the screen. Then, for all objects except a wire, move the mouse to the next point (a black line or circle will be drawn, following the mouse pointer). Click and release the mouse button to define the next point. Repeat until all points are entered and a display panel appears requiring keyboard entry. Continue until the object is complete. Note that in the case of a rectangular magnetic domain, entering a value of 'a' for permeability will create an air gap.

Repeat with new objects until the desired configuration is complete. Clicking on an object will select it, change the colour to red and display

the data for that object. Clicking off the object will deselect it and change the colour to green. For permanent magnets and coils the first line entered defines the North Pole.

These instructions for object entry also appear on an instruction panel on the screen.

An object may be edited by first selecting it (by clicking on it and changing the colour to red) and then using the EDIT menu or one of the speedbuttons.

The only object data stored (or saved on disk by FILE SAVE) will be the points and data defining the object, plus flags defining mesh size, etc. Guidelines are not stored.

To set up the resistor values of the mesh defined by the object permeability choose ANALYSE–SET UP MESH. ANALYSE–RUN will then compute and plot the magnetic field lines. Alternatively the RUN speedbutton will perform both operations.

Once computed, FILE SAVE will also store the mesh current loop values, so that a FILE OPEN operation will then automatically display the magnetic field lines. Magnetic field lines may be removed by ANALYSE–SETUP MESH.

Other options from the TOOLS menu are the number of magnetic field lines displayed, which may be selected as high, low or normal, and SHOW DATA, where the strength and permeability of all objects will be shown on the objects themselves.

The VIEW–FLUX DENSITY/FLUX LINES menu or F>B speedbutton changes the display from magnetic field lines to magnetic flux density, as described in Chapter 7. Note that the display must be changed back to magnetic field lines to permit further editing. VIEW–OPTIONS allows the magnetic field lines to be superimposed on magnetic flux density plots.

The FILE–PRINT or the PRINT speedbutton will print out whatever is displayed on the drawing area of the screen (except guidelines). Thus the GRID and SHOW DATA options should be set before printing.

The CLIPBOARD menu will copy the screen display to the clipboard in bitmap format (.BMP file extension), to allow pasting into other applications such as word processors. Sometimes copied bitmaps look 'grainy' when displayed on monitor screens, but they normally print with

acceptable quality. Many applications are available to enable the capture of images in other formats (jpg, gif, tif, cgm, wmf, etc.)¹.

Options are also provided under the ANALYSE menu for double precision conjugate gradient and Gaussian elimination matrix solvers, and for smoothing and error level.

On-line help

Context sensitive help is not provided in VIZIMAG, but on-line help text panels are automatically shown when a function is selected from the menu or from the tool bar, which give a brief overview of what each function does. In addition a prompt line is shown to act as a guide through each step in an edit function.

The source code

The source code for the main application programs is in the directory C:\VIZ\SOURCE and is described more fully in Chapter 11. To use the source code the standard version of Borland Delphi (version 3 or 4) is required.

Hints and tips on using VIZIMAG

By following a few simple common sense guidelines, VIZIMAG will make model building and analysis simple and uncomplicated. The key things to bear in mind when creating a model are **resolution**, **balance** and **symmetry**.

Resolution refers to the need for a sensible number of mesh elements in a model. For good results aim at ten meshes (i.e. grid squares) across any dimension of a flux source (i.e. magnet, coil or solenoid) and five across any dimension of a permeability region. Too few elements will cause distortion in the magnetic field line pattern. If a model is created with a high resolution mesh, remember that changing to a lower resolution will reduce the number of elements in the model. It is better to first create a model at the lowest resolution that is intended to be used.

The need for enough mesh resolution to cope with angled models has already been discussed in Chapter 6. Remember that the

¹ Search on the internet for 'screen capture software'.

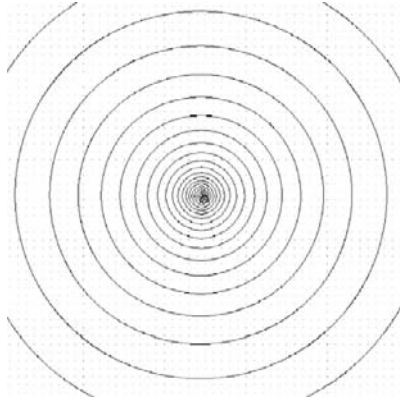


Figure 9-2: Model offset to mesh.

ANALYSE-SMOOTHING option can be used to increase or reduce the smoothing level.

A further effect of resolution is the quantisation of the computed magnetic field lines relative to the model, which can be particularly noticeable on low resolution meshes. The computed result has to assume that a flux source, e.g. a wire, is at the centre of a mesh, whereas the model entry may have placed a point anywhere on the mesh. If, for example, a wire is placed at the corner of a mesh with 50×50 resolution, the result is as shown in Figure 9-2. If, however, the wire is placed at the

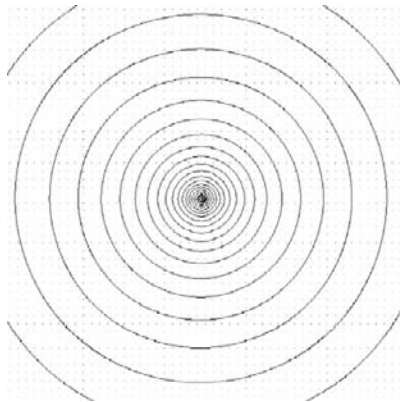


Figure 9-3: Model centred on mesh.

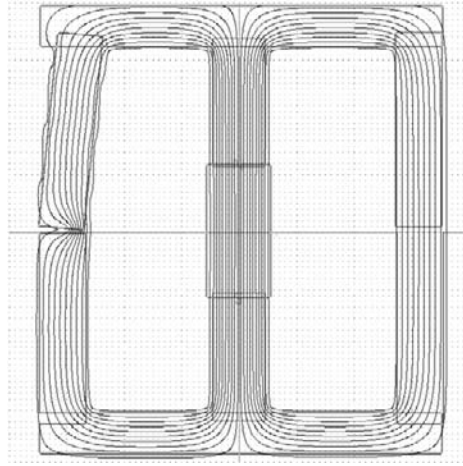


Figure 9-4: Overlap.mag example.

centre of the mesh the plot looks like Figure 9-3. Thus when entering models the best result is obtained if the model is placed at the centres of meshes. Alternatively – but with longer computation time – the effect is less noticeable at higher resolutions.

Balance refers to what happens when regions of high permeability are overlapped. Take as an example a coil with a permeability of 999 overlapping a region also with a permeability of 999, shown in Figure 9-4. The two coils are identical except that the left-hand coil dimension is the same as that of the high permeability region, whilst the right-hand coil is wider by one mesh square. Remember that the magnetic flux is simulated by voltage sources placed round the coil and it can be seen that in the case of the left-hand coil the induced current (analogous to flux) has been 'short-circuited'. Thus always leave a one-mesh boundary on the appropriate dimension of flux sources.

Another example of balance can be seen in Figure 9-5, where a high permeability region is slightly misaligned, so that a one-mesh area of free space occurs on one side. Not surprisingly the flux takes a preferred path through the high permeability overlap. Both of the above examples are contained in the magnetic samples file (see Chapter 10).

Symmetry refers to the distorting effects caused by a finite boundary region. This is not usually a problem, but where there are opposing fields

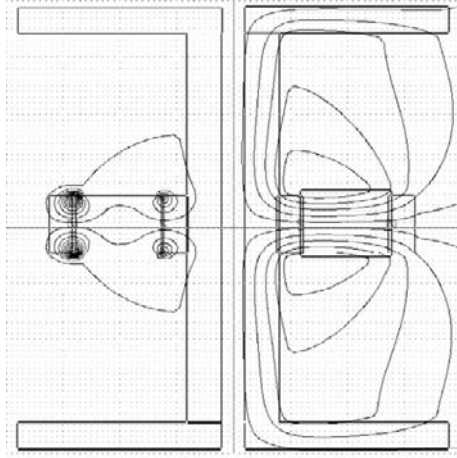


Figure 9-5: Misalign example.mag.

balancing each other, it is wise to create the model so that they are symmetrically placed around the centre lines.

The sample file 4 pole dc motor armature.mag (in the C:\VIZ\DATA file) is an example where both balance and symmetry are important, since proper positioning of the model elements is needed to show the balanced nature of the flux.

This Page Intentionally Left Blank

Sample results

Magnetic samples

The sample files have all been prepared on a 300×300 mesh unless stated otherwise. Use **TOOLS–MESH SIZE** to change to smaller mesh sizes if desired (for example to run modified models with faster analysis speed). The samples are contained in the **C:\VIZ\DATA** file and are automatically listed when **FILE–OPEN** is selected.

File name	Description
1. Permanent magnet	
2. Two permanent magnets N–S faces	Two permanent magnets with adjacent N–S faces.
3. Two permanent magnets N–N faces	Two permanent magnets with adjacent N–N faces.
4. Field due to a wire	
5. Air cored coil.	
6. μ 100 circuit with air gap	Magnetic circuit, $\mu = 100$, with air gap.
7. μ 10 shield in external field	Closed magnetic shield, $\mu = 10$ in external field.
8. μ 500 shield in external field	Closed magnetic shield, $\mu = 500$ in external field.
9. Solenoid	Closed solenoid.
10. Solenoid with air gap	
11. μ 100 core with coil	$\mu = 100$ ring with small coil.
12. Wire between two poles	
13. 4 pole dc motor armature	
14. Misalign example (80×80 mesh)	See figure 9-5.
15. Overlap (80×80 mesh)	See figure 9-4.

This Page Intentionally Left Blank

References

- [1] Boctor S A. Electric circuit analysis. Prentice-Hall International; 1992.
- [2] Shewchuk J R. An introduction to the conjugate gradient method without the agonizing pain, Aug 4, 1994, <http://atol.ucsd.edu/~pflatau/scatlib/conjugate.html>
- [3] Burch R et al. A new matrix solution technique for general circuit simulation. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems; 1993, 12(2): 225–241.
- [4] Burch R et al. PGS and PLUCGS – two new matrix solution techniques for general circuit simulation. IEEE Trans; 1989, 408–411.

This Page Intentionally Left Blank

Appendix

The source code

Running the source code

The code contains the graphical interface form design and the procedures and event handlers. Borland Delphi Standard Edition was used for the creation of this code, and hence is required in order to examine and modify it (version 3 or 4).

The source code can be loaded as a complete project and run immediately from the Delphi RUN menu. Alternatively the code in the various units may be viewed using the Delphi VIEW menu.

Programming style

The programming language used in Delphi is object Pascal which is a highly typed, object oriented environment. Object oriented programming and rigid typing, whilst allegedly producing reliable reusable code, does not always result in program code that is easy for non-programmers to follow. Therefore, this style has been avoided wherever possible in the VIZIMAG code. General purpose reusable code was also not a requirement as the code been intended as an illustration of this visualisation technique, and therefore extensive use has been made of global variables. Since the code is made freely available, however, the reader is free to re-write it in any style that he or she favours.

Architecture

The architecture, or structure, of the program code flow in large event driven graphical interface application is never easy to follow simply by reading the code, even if commented. Development of the program is certainly eased by the use of tools such as Delphi, particularly for the

graphical interface and by the use of the ready built components for file handling, printing, etc. Nevertheless it is still necessary to write all the code associated with handling the mouse and keyboard events, and to break the code down into easy to use sub-routines or procedures. To try to ease the understanding of the code, the approach taken here has been to provide simplified flow descriptions of each event driven function, briefly describing the purpose of each part, showing all the procedures called, and describing the next event required in multi-event procedures. These flow descriptions describe the basic architecture of the application. Similarly flow descriptions are given for each procedure in the code. A plain English style has been adopted for the code description rather than any stylised programming form. It is hoped that readers interested in using or adapting the code will find this approach understandable. However, it has to be admitted that writing easily comprehensible descriptions of software code is not easy, and any suggestions as to ways to improve this aspect of the book are welcome. It would be appreciated if such suggestions could be accompanied by some specific example of what is being asked for.

Global variables

Global variables used in the program are listed in the VIZGlobal unit, with the major ones described.

Resizing arrays

The matrix and vector arrays used in VIZIMAG have to be resizable in order to accommodate variable mesh sizes. Before version 4, Delphi (or more precisely the Object Pascal language) did not explicitly support resizable arrays and therefore such structures had to be designed using pointer constructs. Version 4 of Delphi now contains the support but the arrays in VIZIMAG have been left in the pointer form to permit the use of either version 3 or version 4.

Code flow descriptions: overview

The source code is described here as a set of flow descriptions for each procedure in the application, which give a brief outline of the most important code actions plus a list of all other procedures called. Where

appropriate there is a discussion of the logical methods used within the procedure. The flow diagram type faces have the following meanings:

- **Bold**: a procedure name.
- *Italics*: a description of the code action.
- Underlined normal face: the result of a code action.
- SMALL CAPITALS: a program variable.

The flow descriptions are listed in the following order:

- Procedures called during application start up.
- Procedures called during application close down.
- Procedures called during **Form.Update** calls.
- Procedures called during **Form.Refresh** calls.
- Program procedures initiated by a menu item or speedbutton.
- General program procedures in unit *VIZGLOBAL*.
- General program procedures in unit *VIZ1*.
- General program procedures in unit *VIZ2*.
- General program procedures in unit *VIZ3*.
- Other mouse and keyboard event initiated procedures.

In Delphi a form (.dfm extension) is associated with a unit. In this case it is unit *VIZ1*.

Application start-up

In VIZIMAG, the following two events occur in this order:

Form1.OnShow event

TForm1.FormShow

The ONLINEHELP_FLAG is set.

Form1.OnCreate event

TForm1.FormCreate

Set PLOTALL to 1 (to show only the inner meshes) Set the CENTRELINES_FLAG to 1. Read the Preferences file from disk. From the data read set:

GRIDDOTS_FLAG, SHOWGRID_FLAG, number of contour lines, MESH SIZE. Reset the GRIDLINES_FLAG, initialise the magnetic flux density mode to color, linear.

*Now read and set the working directory file from disk. **Resize_Arrays** depending on the mesh size.*

Form1.OnPaint

TForm1.FormPaint

*Set FORM_PAINT_FLAG. If FIRST_CREATED_FLAG = 0 then **BitmapCreate** and calculate the CANVASWIDTH, Else **BitmapToCanvas**.*

*If SOMETHING_SELECTED_FLAG = 0 then **Reset_Data_Labels**.*

Reset FORM_PAINT_FLAG.

Make the labels fill the screen.

*If FIRST_CREATED_FLAG = 0 then **Nfile** (Bring up a canvas and grid when the application first runs). Set FIRST_CREATED_FLAG.*

Application close down

The following event occurs on application close down.

Form1.OnClose event

TForm1.FormClose

*If the current file has changed the user is prompted to save it with **Ask_About_Save**.*

*Reset FILECHANGED_FLAG. **Free_Memory** allocated to arrays. Send **Application.Terminate**.*

Form.Update

Form1.Update calls the Windows API UpdateWindow function to process any pending paint messages, and therefore will cause all controls on the screen to be updated.

Form.Refresh

Form1.Refresh erases the whole screen and repaints it after calling the Windows API UpdateWindow function to process any pending paint messages, and therefore causing all controls on the screen to be updated. Calls the formpaint event handler.

Program procedures initiated by a menu item or speedbutton

File-New File or Speedbutton 19

Nfile

If the current file has changed the user is prompted to save it
Ask_About_Save.

*The data labels are blanked out and the magnetic flux density function blanked and reset. The PRIMITIVE stores are set to zero. The screen is cleared and (if Show Grid is checked) the grid is drawn with **Draw_GridLines** or **Draw_GridDots**. **Enable_menus**. Set the title bar and reset the external field data entry panel. **Reset_All_Flags**, **CanvasToBitmap**.*

File-Open or Speedbutton 2

Fopen

*If the current file has changed the user is prompted to save it **Ask_About_Save**. The data labels are blanked out and then the **OpenDialog** box is displayed (a standard Delphi function) to allow the user to enter file data.*

*If CANCEL is pressed: **Reset_All_Flags** and exit.*

*If OK is pressed: **Clear_Screen**, then initialise the external field function and get the name of the file from the Dialog box.*

Call File_open

*The File is checked for a valid format and then read from disk. If not a valid format an error message is sent, then **Reset_All_Flags** and exit. If the file is a valid format then the data is scaled (if the file data was written at a different screen resolution) with **Scale_Data**.*

*Create the canvas bitmap and other data with **BitMapCreate**. Set the centreline check box then **Clear_Screen** and draw all the objects from the stored PRIMITIVE data with **Draw_Primitives**. If the PRINT_DATA_FLAG is set then set the **PrintData** check box, display the object data on screen with **Show_Data**, and save the **CanvasToBitmap**. Send **Form1.Refresh**. If the CONTOUR_FLAG is set then **Contour**. **Enable_menus_in_Edit**, **Reset_Most_Flags**, set the file name in the caption bar.*

File-File Save or Speedbutton 1

Fsave

*Calls **Save_As** and then sets the caption title to the new name.*

File_Print or Speedbutton 3

Fprint

Fprint draws a copy of the screen display to the printer canvas.

*Disable_menus_in_Edit to grey out and disable the menus and speedbuttons. **CanvastoBitmap** then show the print dialog box (a standard Delphi function).*

*If CANCEL is pressed **BitmapToCanvas**, **Reset_Most_Flags**, **Enable_Menus_in_Edit** and then exit.*

*If OK is pressed then set the number of copies from the dialog box. Obtain a scaling factor for the printer pen using the **pixelsperinch** property of the printer canvas font (since printer resolutions vary, it is necessary to scale the printer pen width setting depending on the particular printer). A margin is set all round the canvas on the page and the canvas size in pixels is set depending on whether the user has chosen portrait or landscape format. The grid scaling factor for the printer is calculated, and a scale factor obtained to scale the object data in the **PRIMITIVE** stores (which are set to suit the screen resolution) to the printer resolution. A frame is drawn round the canvas area and then the printer canvas is cleared with **Clear_Screen** and the objects drawn with **Draw_Primitives**. If the **CONTOUR_FLAG** is set then **Contour**. If magnetic flux density is checked then **Fluxfill**. If **PrintData** is checked then put the object data onto the printer canvas with **ShowData**. The pen is spaced above the canvas by a distance dependant on the printer text height and the title is printed (note that the printer canvas brush style must be set to **CLEAR** to avoid picking up a background colour from the canvas). Finally reset to the screen resolution, **BitmapToCanvas** and **Reset_Most_Flags**. If the screen is in magnetic flux density mode then only **F>B** and **Print** menus are set, otherwise **Enable_Menus_in_Edit**.*

File_Exit or Application icon double click

*If the current file has changed then the user is prompted to save it **Ask_About_Save**. The memory allocated to the program is released with **Free_Memory**, and the application is terminated.*

Edit-Add Guideline or Speedbutton 25

Eguide

Disable_Menus_in_Edit then set the `GUIDELINE` and `EDIT` flags to 1. Exit to wait for a mouse up event.

Edit-Add Magnetic Domain or Speedbutton 4

Emag

Disable_Menus_in_Edit then set the `MAGDOM` and `EDIT` flags to 1. Exit to wait for a mouse up event.

Edit-Add Circular Magnetic Domain or Speedbutton 23

eCmag

Disable_Menus_in_Edit then set the `CMAGDOM` and `EDIT` flags to 1. Exit to wait for a mouse up event.

Edit-Add Permanent Magnet or Speedbutton 5

EPermag

Disable_Menus_in_Edit then set the `PERMAG` and `EDIT` flags to 1. Exit to wait for a mouse up event.

Edit-Add Coil or Speedbutton 6

ECoil

Disable_Menus_in_Edit then set the `COIL` and `EDIT` flags to 1. Exit to wait for a mouse up event.

Edit-Add Solenoid or Speedbutton 22

ESolenoid

Disable_Menus_in_Edit then set the SOLENOID and EDIT flags to 1. Exit to wait for a mouse up event.

Edit-Add Vertical Wire or Speedbutton 7

EWire

Disable_Menus_in_Edit then set the WIRE and EDIT flags to 1. Exit to wait for a mouse up event.

Edit-Copy or Speedbutton 8

ECopy

Exit if no object is selected. Else **Disable_Menus_in_Edit** then set the COPY and EDIT flags to 1 and DRAWBLACK flag to 0. Exit to wait for a mouse move event.

Edit-Move or Speedbutton 9

EMove

Exit if no object is selected. Else **Disable_Menus_in_Edit** then set the MOVE and EDIT flags to 1 and DrawBlack flag to 0. **Centroid** to calculate the centroid of the object. Exit to wait for a mouse move event.

Edit-Rotate or Speedbutton 10

ERotate

Exit if no object is selected, or if the object is a circular magnetic domain, solenoid or wire. Else **Disable_Menus_in_Edit** then set the ROTATE and EDIT flags to 1 and the DRAWBLACK flag to 0. **Centroid** to calculate the centroid of the object, **Draw_Hline** to draw a short horizontal line from the centroid co-ordinates as a guide to the angle to be rotated. Exit to wait for a mouse move event.

Edit-Flip-Left/right or Speedbutton 11

Efliplr

Exit if no object is selected, or if the object is a circular magnetic domain, solenoid or wire.

Else set LR and TB values and call.

Flip

Disable_Menus_in_Edit then set the FLIP and EDIT flags to 1. **Centroid** to calculate the centroid of the object. Remove the object lines with **Draw_Red_using_BS** and (if a magnet or coil) **DrawNS**. Reverse the X co-ordinates of the object and redraw with **Draw_Red_using_BS** and **DrawNS**. Store the new object co-ordinates in the PRIMITIVE store. **Reset_Most_Flags**, **Enable_Menus_in_Edit**, **CanvasToBitmap**.

Edit-Flip-Top/bottom or Speedbutton 16

Efliptb

Exit if no object is selected, or if the object is a circular magnetic domain, solenoid or wire.

Else set LR and TB values and call.

Flip

Disable_Menus_in_Edit then set the FLIP and EDIT flags to 1. **Centroid** to calculate the centroid of the object. Remove the object lines with **Draw_Red_using_BS** and (if a magnet or coil) **DrawNS**. Reverse the Y co-ordinates of the object and redraw with **Draw_Red_using_BS** and **DrawNS**. Store the new object co-ordinates in the PRIMITIVE store. **Reset_Most_Flags**, **Enable_Menus_in_Edit**, **CanvasToBitmap**.

Edit-Switch Poles or Speedbutton 12

ESwitchPoles

Exit if no object is selected, or if the object is a magnetic domain, solenoid or wire.

Disable_Menus_in_Edit then set the SWITCHPOLES and EDIT flags to 1. Copy the contents of the BUFFER_SELECTED store into the BUFFER_ANGLE store and then exit to wait for a mouse up event.

Edit-Change Data or Speedbutton 13

EChangeData

Exit if no object is selected. **Disable_Menus_in_Edit** then set the CHANGEDATA and EDIT flags to 1. The next action depends on the type of object, determined by the value of TTYPE_SELECTED:

TYPE_SELECTED = 0 (Magnetic domain) **CanvasToBitmap**, then display the permeability data entry panel with **Show_Perm_Frame**. Exit to wait for a mouse up event on the OK button of the frame.

TYPE_SELECTED = 1 (Permanent magnet) **CanvasToBitmap**, then display the strength data entry panel with **Show_Strength_Frame**. Exit to wait for a mouse up event on the OK button of the frame.

TYPE_SELECTED = 2 (Coil) **CanvasToBitmap**, then display the permeability data entry panel with **Show_PermStrength_Frame**. Exit to wait for a mouse up event on the OK button of the frame.

TYPE_SELECTED = 3 (Wire) **CanvasToBitmap**, then display the strength data entry panel with **Show_Strength_Frame**. Exit to wait for a mouse up event on the OK button of the frame.

TYPE_SELECTED = 4 (Solenoid) **CanvasToBitmap**, then display the solenoid data entry panel with **Show_SolData_Frame**. Exit to wait for a mouse up event on the OK button of the frame.

TYPE_SELECTED = 5 (Circular magnetic domain) **CanvasToBitmap**, then display the permeability data entry panel with **Show_Perm_Frame**. Exit to wait for a mouse up event on the OK button of the frame.

Edit-Size-H or Speedbutton 17

ESizeH

Size H is defined as the size of the object in the direction of the NS poles or the first line of the object entered.

Exit if no object is selected, or if the object is a circular magnetic domain, solenoid or wire. **Disable_Menus_in_Edit** then set the SIZE, H and EDIT flags to 1, and the DRAWBLACK flag to 0. Calculate the centroid of the object with **Centroid**, find the angle of the North line to the horizontal with **Find_N_Angle** and then exit to wait for a mouse move event.

Edit-Size-V or Speedbutton 18

ESizeV

Size V is defined as the size of the object in the direction of the none pole lines or the second line of the object entered.

Exit if no object is selected, or if the object is a circular magnetic domain, solenoid or wire.

Disable_Menus_in_Edit then set the SIZE, V and EDIT flags to 1, and the DRAWBLACK flag to 0. Calculate the centroid of the object with **Centroid**, find the angle of the North line to the horizontal with **Find_N_Angle** and then exit to wait for a mouse move event.

Edit-Size-Both or Speedbutton 14

ESizeBoth

Exit if no object is selected, or if the object is a wire.

Disable_Menus_in_Edit then set the SIZE and EDIT flags to 1, and the DRAWBLACK flag to 0. Calculate the centroid of the object with **Centroid** and then exit to wait for a mouse move event.

Edit-Clear or Speedbutton 15

EClear

*If no object is selected **Enable_Menus_in_Edit** and exit.*

*Else **Disable_Menus_in_Edit** then set the **CLEAR** and **EDIT** flags to 1. Remove the object data from the **PRIMITIVE** store with **Remove_Data** and then **Delete_Selected**. **Reset_Most_Flags**, set the **SOMETHING_SELECTED_FLAG** to 0 and **Enable_Menus_in_Edit**.*

Edit-Clear All

TForm1.MnuEditClearClick

***Disable_Menus_in_Edit** then ask for confirmation that all objects are to be removed. If **NO** then **Enable_Menus_in_Edit** and exit.*

*If **YES** then reset the **PRIMITIVE** stores with **Set_Primsizes-values_to_Zero** and **Clear_Screen**. If **ShowGrid** is checked then **Draw_GridDots** or **DrawGridLines**. Then **Reset_All_Flags**, **Enable_Menus_in_Edit**, **CanvasToBitmap**.*

Edit-Clear all guidelines

TForm1.ClearAllGuidelinesClick

Clear_Screen, **Draw_Primitives**.

Clipboard – Copy to clipboard

TForm1.ClipClick

CanvasToBitmap then assign the bitmap to the clipboard.

Title

TForm1.MnuTitleClick

*If in edit mode then exit. Else **Disable_Menus_in_Edit** and make the title entry panel visible. Exit to wait for a TForm1.TitleOKClick event from the panel.*

View – Fluxlines

TForm1.MnuFluxlinesClick

If Fluxlines is checked then exit, else toggle to magnetic flux density display by FtoB.

View – Flux density

TForm1.MnuFluxdensityClick

If Fluxdensity is checked then exit, else toggle to magnetic field lines display by FtoB.

View – Options – Flux density alone

TForm1.BaloneClick

Toggle the check box.

View – Options – Flux density plus field lines

TForm1.BpluslinesClick

Toggle the check box.

Analyse – Set up mesh

TForm1.MnuAnalyseSetClick

Eset.

Analyse – Run

TForm1.MnuAnalyseRunClick

Depending on the state of the VIEW-ANALYSE OPTIONS menu, call Analyse_CG for the standard conjugate gradient matrix solver, or Analyse_CG_DoublePrec for the double precision solver, or Analyse_Gauss for the gaussian elimination solver.

Analyse – Analyse options – Conjugate gradient normal

TForm1.CGNormalClick

Toggle the analyse options check boxes.

Analyse – Analyse options – Conjugate gradient high

TForm1.CGHighClick

Toggle the analyse options check boxes.

Analyse – Analyse options – Gaussian elimination

TForm1.GaussClick

Toggle the analyse options check boxes.

Speedbutton 21

Eset followed by **Analyse_GC** or **Analyse_CG_DoublePrec** or **Analyse_Gauss** depending on the state of the **VIEW-ANALYSE OPTIONS** menu.

Analyse – Smoothing options – None

TForm1.SmoothnoneClick

Toggle the smoothing options check boxes.

Analyse – Smoothing options – Normal

TForm1.SmoothnormalClick

Toggle the smoothing options check boxes.

Analyse – Smoothing options – High

TForm1.SmoothhighClick

Toggle the smoothing options check boxes.

External field

TForm1.MnuExtFieldClick

Disable_Menus_in_Edit, CanvasToBitmap, show the external field data entry panel.

*Exit, waiting for a **TForm1.ExtFldButtonClick** or **TForm1.ExtFldCancelClick** event.*

Speedbutton 20

TForm1.Speedbutton20Click

Disable_Menus_in_Edit, CanvasToBitmap, show the external field data entry panel.

Exit, waiting for a *TForm1.ExtFldButtonClick* or *TForm1.ExtFldCancelClick* event.

Tools – Grid – Dots

TForm1.MnuToolsDotsClick

If GRIDDOTS_FLAG *is set* (grid dots are already displayed) then exit.

If ShowGrid *is not checked* then exit.

If GRIDLINES_FLAG *is set* then **Erase_GridLines, Draw_GridDots, CanvasToBitmap**.

Set GRIDDOTS_FLAG, *reset* GRIDLINES_FLAG *set the check boxes*.

Tools – Grid – Lines

TForm1.MnuToolsLinesClick

If GRIDLINES_FLAG *is set* (grid lines are already displayed) then exit.

If ShowGrid *is not checked* then exit.

If GRIDDOTS_FLAG *is set* then **Erase_GridDots, Draw_GridLines, CanvasToBitmap**.

Set GRIDLINES_FLAG, *reset* GRIDDOTS_FLAG *set the check boxes*.

Tools – Show Grid

TForm1.MnuToolsShowGridClick

Toggle the ShowGrid check box and set or reset SHOWGRID_FLAG. If a grid is displayed then

Erase_GridDots or Erase_GridLines and CanvasToBitmap.

Tools – Show centre lines

TForm1.MnuToolsShowCentreLinesClick

Toggle the CentreLines check box and set or reset CENTRELINES_FLAG.

Draw_CentreLines if required.

Tools – Field Lines – High

TForm1.MnuFluxHighClick

Set NUMCONT, the number of contour lines to 41. Set the field lines check boxes.

*If CONTOUR_FLAG is set then remove contour lines by **Clear_Screen, Draw_Primitives.***

*Redraw with the new number of lines by **Contour. Enable_menus_in_edit.***

Tools – Field Lines – Normal

TForm1.MnuFluxNormalClick

Set NUMCONT, the number of contour lines to 21. Set the field lines check boxes.

*If CONTOUR_FLAG is set then remove contour lines by **Clear_Screen, Draw_Primitives.***

*Redraw with the new number of lines by **Contour. Enable_menus_in_edit.***

Tools – Field Lines – Low

TForm1.MnuFluxLowClick

Set NUMCONT, the number of contour lines to 9. Set the field lines check boxes.

*If CONTOUR_FLAG is set then remove contour lines by **Clear_Screen**, **Draw_Primitives**.*

*Redraw with the new number of lines by **Contour**. **Enable_menus_in_edit**.*

Tools – Show Data

TForm1.MnuPrintDataClick

*Toggle the PRINT_DATA flag and check box. If PRINT_FLAG = 0 (i.e. data should be sent to the screen and not the printer) and the box is checked then put object data on the screen with **ShowData**. If the box is not checked then remove data from the screen with **Clear_Screen** and **Draw_Primitives**. If CONTOUR_FLAG is set then **Contour**. **Enable_Menus_in_Edit**.*

Tools – On Line Help

TForm1.MnuHelpClick

On line help gives an on-screen outline description of the major functions when they are called. This procedure toggles the ONLINEHELP_FLAG.

Tools – Mesh size – 50 × 50

TForm1.N50Click

*Free memory previously allocated to arrays with **Free_Memory**. Set the outer mesh size MESHO to 60 and the inner (visible) mesh size MESHI to 50. Calculate the screen scaling factors, xs and ys (pixels per grid unit). Set the mesh size check boxes. Allocate new memory to arrays with **Resize_Arrays**. **Clear_Screen**, **BitMapCreate**, **Draw_Primitives**.*

Tools – Mesh size – 80×80

TForm1.N76Click

*Free memory previously allocated to arrays with **Free_Memory**. Set the outer mesh size MESHO to 96 and the inner (visible) mesh size MESHI to 80. Calculate the screen scaling factors, xs and ys (pixels per grid unit). Set the mesh size check boxes. Allocate new memory to arrays with **Resize_Arrays**. **Clear_Screen**, **BitMapCreate**, **Draw_Primitives**.*

Tools – Mesh size – 100×100

TForm1.N100Click

*Free memory previously allocated to arrays with **Free_Memory**. Set the outer mesh size MESHO to 120 and the inner (visible) mesh size MESHI to 100. Calculate the screen scaling factors, xs and ys (pixels per grid unit). Set the mesh size check boxes. Allocate new memory to arrays with **Resize_Arrays**. **Clear_Screen**, **BitMapCreate**, **Draw_Primitives**.*

Tools – Mesh size – 150×150

TForm1.N150Click

*Free memory previously allocated to arrays with **Free_Memory**. Set the outer mesh size MESHO to 180 and the inner (visible) mesh size MESHI to 150. Calculate the screen scaling factors, xs and ys (pixels per grid unit). Set the mesh size check boxes. Allocate new memory to arrays with **Resize_Arrays**. **Clear_Screen**, **BitMapCreate**, **Draw_Primitives**.*

Tools – Mesh size – 200×200

TForm1.N200Click

*Free memory previously allocated to arrays with **Free_Memory**. Set the outer mesh size MESHO to 238 and the inner (visible) mesh size MESHI to 200. Calculate the screen scaling factors, xs and ys (pixels per grid unit). Set the mesh size check boxes. Allocate new memory to arrays with **Resize_Arrays**. **Clear_Screen**, **BitMapCreate**, **Draw_Primitives**.*

Tools – Mesh size – 300 × 300

TForm1.N300Click

*Free memory previously allocated to arrays with **Free_Memory**. Set the outer mesh size **MESHO** to 358 and the inner (visible) mesh size **MESHI** to 300. Calculate the screen scaling factors, **xs** and **ys** (pixels per grid unit). Set the mesh size check boxes. Allocate new memory to arrays with **Resize_Arrays**. **Clear_Screen**, **BitMapCreate**, **Draw_Primitives**.*

Tools – Preferences

TForm1.mnuPreferencesClick

*Disable_Menus_in_Edit, **CanvastToBitmap**. Show the Preferences data entry panel. Exit to wait for a **TForm1.Button2Click** event (OK button) or **TForm1.Button3Click** event (CANCEL button).*

General program procedures in unit VIZGlobal

DrawGrid

*If **ShowGrid** box is checked then **Draw_GridDots** or **Draw_GridLines** depending on the states of the **GRIDLINES_FLAG** and **GRIDDOTS_FLAG**.*

General program procedures in unit VIZ1

Convert_Buffer_Values_to_Integers

*Convert from floating point to canvas integer scale: **BUFFER_BLACK** into **XC/YC**, **BUFFER_SELECTED** into **XBS/YBS** and **TEMP** values into **TXC/TYC**. Remove **CLEFT**, **CTOP**, etc. from these conversions when sending to the printer and scale the screen to the printer.*

Rectangle_Points

When entering rectangular magnetic domains, permanent magnets and coils a rectangle must be drawn. This routine uses the first two object points entered (in `BUFFER_BLACK[0-3]`), plus the mouse position to calculate the last two co-ordinates of a rectangle such that the projected line would pass through the mouse pointer. The points are left in `BUFFER_BLACK`. The last two co-ordinate points are also held in `TEMP` to allow erasure during the exclusive or drawing routines. This is a geometric routine, and some of the code is concerned with resolving ambiguities of +/- square roots and division by zero.

Draw_Black

*When an object is entered or edited, black lines are drawn to represent changes to the object. These black lines are constantly drawn, erased and redrawn using the exclusive or function. **Rectangle_Points** is used to get a complete rectangle with data from `BUFFER_BLACK`, and*

***Convert_Buffer_Values_to_Integers** is used to convert the floating point `PRIMITIVE` data to integers for the canvas drawing routines. `FLAG` values passed to the routine define different actions at different times as an object is created or edited (some `FLAG` values are changed by mouse up events). `DRAWBLACK_FLAG` is used to define whether the black line is to be erased or not (since the first black line drawn does not have a previous line to erase). `FLAG = 1` is the first point of an object and is not drawn, so exit. `FLAG = 2` is the second point (i.e. the first line of an object). `FLAG = 3` is when a complete rectangle is defined for the first time with **Rectangle_Points**, but before this is called a check is made that the rectangle does not have a zero dimension (if it does then exit with no action). `FLAG = 4` is for second and subsequent rectangles. `FLAG = 5` defines that the draw part of the object is complete.*

Dot

For drawing a wire object a dot at the centre of the wire is made by drawing a single pixel line. If `PRINT_FLAG = 0` then the dot is drawn on the screen using data direct from `BUFFER_SELECTED` (truncated to convert to integers). If the `PRINT_FLAG = 1` then the `BUFFER_SELECTED` values are shifted and scaled to suit the printer canvas.

Cross

For drawing a wire object a cross at the centre of the wire is made by drawing lines. If `PRINT_FLAG = 0` then the lines are drawn on the screen using data from `BUFFER_SELECTED` (truncated to convert to integers). To keep the cross the same size for different meshes, the `BUFFER_SELECTED` values are scaled by a factor `s`. If the `PRINT_FLAG = 1` then the `BUFFER_SELECTED` values are shifted and further scaled to suit the printer canvas.

Circle

For drawing a wire object a circle round the wire is drawn. If `PRINT_FLAG = 0` then the circle is drawn on the screen using data from `BUFFER_SELECTED` (truncated to convert to integers). To keep the circle the same size for different meshes, the `BUFFER_SELECTED` values are scaled by a factor `s`. If the `PRINT_FLAG = 1` then the `BUFFER_SELECTED` values are shifted and further scaled to suit the printer canvas.

Change_Selected_to_Green

The red lines of a selected object are erased (by redrawing with the exclusive-or function) and redraw as green. `SOMETHING_SELECTED_FLAG` is reset.

*Data taken from `BUFFER_SELECTED`. `TYPE_SELECTED` are used to determine the drawing action to take for the particular object (i.e. lines for rectangular magnetic domain, permanent magnet or coil and circles for circular magnetic domain or solenoid). If `TYPE_SELECTED = 3` (wire) the procedures **Circle**, **Cross** and **Dot** are called for the drawing action.*

Check_Plot_limits

If any point in `buffer_black` exceeds plot area, the `off_area_flag` is set.

in_mouseup

The **formmouseup** event is used for several actions within the application, the particular action being dependent on the state of various flags and TYPE_SELECTED. During the creation of rectangular magnetic domains, permanent magnets and coils the **in_mouseup** procedure described here is called.

A FLAG is passed to the procedure on entry. The mouse co-ordinates are checked to ensure they are within plot limits (exit if not), any selected object is changed to green, BUFFER_BLACK is loaded and flags set as each point in the object is defined in turn. Line length is checked for greater than two grid units, and rectangle points is used to ensure only rectangles are drawn. The permeability and strength data entry panels are shown as required. The instruction label changes at each FLAG value change.

Check that the mouse is within the plot area using RLIMIT **ect**. Beep and exit if not. If SOMETHING_SELECTED_FLAG is set then **Change_Selected_to_Green**.

If FLAG = 1 (the first point of the object) load BUFFER_BLACK[0,1] with the mouse co-ordinates. Retain these co-ordinates in MOUSE_LASTUPX/Y. Set FLAG = 2 and exit.

If FLAG = 2 (the second point of the object) **Check_Size0** to check the line is greater than two grid units long. On return from this procedure if TOO_SMALL_FLAG = 1 then Beep, set TOO_SMALL_FLAG to 0 and exit. Check that MOUSEX/Y do not equal MOUSE_LASTUPX/Y (otherwise there would not be four lines). If equal then exit. **Convert_Buffer_Values_to_Integers** and erase the old line (using BUFFER_BLACK and MOUSE_OLDX/Y data) and draw the final line using BUFFER_BLACK and MOUSEX/Y). Set flag = 3 and exit.

FLAG = 3 is not valid in this procedure so exit.

If FLAG = 4 **Rectangle_Points** to draw a complete rectangle.

Check_Plot_Limits to ensure that no point in buffer_black is outside the plot area. On return, if OFF_AREA_FLAG = 1 then Beep and exit. **Check_Size** to ensure no line is less than two grid units long. On return if TOO_SMALL_FLAG = 1 then exit.

If BUFFER_BLACK[4] = BUFFER_BLACK[2] and BUFFER_BLACK[5] = BUFFER_BLACK[3] then load BUFFER_BLACK with TEMPL and exit, since this would not give a complete rectangle.

Convert_Buffer_Values_to_Integers, erase the old lines and draw the final lines.

Set FLAG = 5, **CanvasToBitmap**.

If MAGDOM_FLAG = 5 then **Show_Perm_Frame**.

If PERMAG_FLAG = 5 then **Show_Strength_Frame**.

If COIL_FLAG = 5 then **Show_PermStrength_Frame**.

Exit to wait for a button click event from the data entry panels.

Guideline_in_mouseup

The **formmouseup** event is used for several actions within the application, the particular action being dependent on the state of various flags and TYPE_SELECTED. During the creation of guidelines the **Guideline_in_mouseup** procedure described here is called.

The mouse co-ordinates are checked to ensure they are within plot limits (exit if not), any selected object is changed to green, BUFFER_BLACK is loaded and flags set as each point in the object is defined in turn. Line length is checked for greater than two grid units, and rectangle points is used to ensure only rectangles are drawn. The permeability and strength data entry panels are shown as required. The instruction label changes at each FLAG value change.

Check that the mouse is within the plot area using RLIMIT **ect**. Beep and exit if not. If SOMETHING_SELECTED_FLAG is set then **Change_Selected_to_Green**.

If GUIDELINE_FLAG = 1 (the first point) load BUFFER_BLACK[0,1] with the mouse co-ordinates. Retain these co-ordinates in MOUSE_LASTUPX/Y. Set GUIDELINE_FLAG = 2 and exit.

If GUIDELINE_FLAG = 2 (the second point of the object) **Check_Size0** to check the line is greater than two grid units long. On return from this procedure if TOO_SMALL_FLAG = 1 then Beep, set TOO_SMALL_FLAG to 0 and exit.

Convert_Buffer_Values_to_Integers and erase the old line (using BUFFER_BLACK and MOUSE_OLDX/Y data) and draw the final line using BUFFER_BLACK and MOUSEX/Y). Set flag = 3 and exit. **Store_MouseXY** which also has the action of updating the on screen read out of the line angle. **Reset_most_Flags and Enable_menus_in_edit**.

Solenoid_in_up

*The **formmouseup** event is used for several actions within the application, the particular action being dependent on the state of various flags and TYPE_SELECTED. During the creation of circular magnetic domains and solenoids the **Solenoid_in_up** procedure described here is called.*

A FLAG is passed to the procedure on entry. Any selected object is changed to green, BUFFER_BLACK is loaded and FLAG set as each point in the object is defined in turn. Diameter and solenoid thickness is checked for minimum limits. The permeability and strength data entry panels are shown as required. The instruction label changes at each FLAG value change.

*If SOMETHING_SELECTED_FLAG is set then **Change_Selected_to_Green**.*

If FLAG = 1 (the first point of the object) load BUFFER_BLACK[0,1] with the mouse co-ordinates (the centre of the circle). Set FLAG = 2 and exit.

If FLAG = 2 (the inner circle of the solenoid), test for the circle greater than two grid units in diameter and not off limits, using MOUSEX/Y and BUFFER_BLACK. If so then beep and exit. If not then set FLAG = 3 and DRAWBLACK_FLAG = 0. Exit.

*If FLAG = 3 (the outer circle of the solenoid), test for the circle more than four grid units larger than inner circle and not off limits. If so then beep and exit. If not then FLAG = 4, **CanvasToBitmap**. If SOLENOID_FLAG set then **Show_SolData_Frame**, or if CMAGDOM_FLAG set then **Show_Perm_Frame**. Exit to wait for a button click event from the data entry panels.*

Get_Data

Gets data from permeability and strength data entry panels and puts into buffer selected.

*If in permeability object entry mode for a magnetic domain or coil (defined by the value of MAGDOM_FLAG, COIL_FLAG or CMAGDOM_FLAG), or in change data mode for a magnetic domain (defined by TYPE_SELECTED and CHANGEDATA_FLAG), then read the permeability text and perform a **NonNumCheck** to ensure that no non-numeric text has been entered (the only exception is if 'a' has been entered, so defining an air gap, in which case AIRGAP_FLAG is set). Exit if ERROR_FLAG has been set. If correct then convert text to numeric, and check that it is greater than 1 and less than 999. Exit if in error. If correct set BUFFER_SELECTED[8] to the permeability value (or to 1 if the AIRGAP_FLAG is set).*

*Else if in strength object entry mode for a permanent magnet, coil or wire (defined by the value of PERMAG_FLAG, COIL_FLAG or WIRE_FLAG), or in change data mode for a permanent magnet or wire (defined by TYPE_SELECTED and CHANGEDATA_FLAG), then read the strength text and perform a **NonNumCheck** to ensure that no non-numeric text has been entered. Exit if ERROR_FLAG has been set. If correct then convert text to numeric, and check that it is greater than 0 and less than 99. Exit if in error. If correct set BUFFER_SELECTED[8] (or BUFFER_SELECTED[2] for a wire) to the strength value.*

Set_and_Store_Data

Stores data in primitive stores, sets PRIMSIZE and TYPE_SELECTED and leaves with POINTER_SELECTED showing start of last data stored.

If not in change data mode (defined by the CHANGEDATA_FLAG) and if in object entry mode (defined by MAGDOM_FLAG, PERMAG_FLAG, COIL_FLAG, SOLENOID_FLAG or CMAGDOM_FLAG) set TYPE_SELECTED to the appropriate value, set POINTER_SELECTED, increase the size of the store by increasing PRIMSIZE and copy BUFFER_SELECTED into the PRIMITIVE store.

Else if in change data mode (defined by the CHANGEDATA_FLAG) then set permeability and strength as appropriate to the object defined by TYPE_SELECTED in the location in the PRIMITIVE store defined by POINTER_SELECTED.

Wire_in_mouseup

The **formmouseup** event is used for several actions within the application, the particular action being dependent on the state of various flags and **TYPE_SELECTED**. During the creation of wires the **Wire_in_mouseup** procedure described here is called.

A **FLAG** is passed to the procedure on entry. Any selected object is changed to green. The mouse is checked to ensure it is not off the canvas. A black circle is drawn and the strength data entry panel is shown. The instruction label changes as the **FLAG** value is changed from 1 to 2.

Change_black_to_selected

For rectangular magnetic domains, permanent magnets or coils, **Convert_Buffer_Values_to_Integers** and erase the black lines in **BUFFER_BLACK**. Redraw in red and load **BUFFER_BLACK** values into **BUFFER_SELECTED**. Set **SOMETHING_SELECTED_FLAG**.

Direction_Button

When a direction button is selected for wire data entry this procedure is called.

If **CHANGEDATA_FLAG** is set then **Hide_Frame**. Draw red **Circle** and **Dot** or **Cross** depending on the sign of the wire strength. Make **BUFFER_SELECTED[2]** (the strength) value negative if **WIRE_FLAG** = 3 (denoting a negative value). Store the value in the location in the **PRIMITIVE** store defined by **POINTER_SELECTED**. **CanvasToBitmap**, **Enable_Menus_in_Edit**, **Reset_Most_Flags**, **BitMapToCanvas**, **exit**.

Else if in object creation mode then **Hide_Frame**. Erase black **Circle** and draw red **Circle** and **Dot** or **Cross** depending on the sign of the wire strength. Make **BUFFER_SELECTED[2]** (the strength) value negative if **WIRE_FLAG** = 3 (denoting a negative value). Set **SOMETHING_SELECTED_FLAG** to 1 and **TYPE_SELECTED** to 3. Increase the size of the **PRIMITIVE** store and store the **BUFFER_SELECTED** data into the location in the **PRIMITIVE** store defined by **POINTER_SELECTED**. **CanvasToBitmap**, **Enable_Menus_in_Edit**, **Reset_Most_Flags**, **BitMapToCanvas**.

Clear_Buffer_Selected

Sets the nine elements of BUFFER_SELECTED to zero.

Wire_Search

This procedure is called when looking to see if the mouse pointer is on a wire. A search is performed in a small area around the wire co-ordinates. If a wire is found, leave with SOMETHING_SELECTED_FLAG = 1 and POINTER_SELECTED pointing at the particular wire in the PRIMITIVE store.

Calculate a scale factor SCALE to keep the search area constant with different mesh sizes.

Begin to examine each wire in the PRIMITIVE store. Calculate a small area in grid units around the wire co-ordinates. If the mouse co-ordinates are within the search area then set SOMETHING_SELECTED_FLAG, set TYPE_SELECTED to 3 (wire), set POINTER_SELECTED to the particular wire in store, load BUFFER_SELECTED with the wire data and exit.

Repeat for all wires in store.

Change_Green_to_Selected

Using TYPE_SELECTED to determine the object type, change green solid lines to double width solid red. Also change wire circles and wire cross/points to red.

Convert_Buffer_Values_to_Integers

If the PRINT_FLAG = 0 then set screen canvas drawing mode, else set printer canvas drawing mode (in this case SCALEPW is used to scale the printer values to ensure optimum line width).

Green lines or circles (depending on the object defined by TYPE_SELECTED) are first erased by the exclusive or function and then redrawn in red.

Normalised_Angle

Calculates four normalised angles for an object and stores the largest in NORMALISED_ANGLE[0], and the number of the line (i.e. 1 to 4) in LINE_OFFSET. Also sums the four un-normalised angles and sets POINT_INSIDE_FLAG if they equal nearly 360 degrees (nearly 360 to account for floating point rounding error in the calculations). Enter the routine with the four lines of the object stored in BUFFER_ANGLE.

A normalised angle is the angle (in radians) from the mouse X,Y points (passed to the routine) to the line, divided by the length of the line. The largest normalised angle is the line nearest the mouse X,Y point, and if the four un-normalised angles equal (nearly) 360 degrees then point XY lies within the boundary of the four lines – in which case the POINT_INSIDE_FLAG is set.

This is a geometric routine and contains many code lines concerned with resolving ambiguities and divide by zero situations.

Search

This procedure is called when searching through the objects stored in the PRIMITIVE stores (except for wires) to find which (if any) the mouse pointer is on. Exits with SOMETHING_SELECTED_FLAG set (if object found), POINTER_SELECTED at start of the object data in the PRIMITIVE store, LINE_SELECTED at the closest line to the pointer, TYPE_SELECTED defining the object type.

*If the object (defined by TYPE_SELECTED) is not a solenoid or circular magnetic domain then begin with the first object in the primitive store passed to the procedure. Find the maximum and minimum grid points of a square surrounding the object. If the mouse co-ordinates are within this square then **normalised_angle**. Repeat for all objects in this primitive store and exit the search with (if the mouse is on an object) SOMETHING_SELECTED_FLAG, TYPE_SELECTED, POINTER_SELECTED, LINE_SELECTED all set and BUFFER_SELECTED loaded with the primitive data.*

If the object (defined by TYPE_SELECTED) is a solenoid or circular magnetic domain then begin with the first object in the primitive store passed to the procedure. Test whether the mouse lies between the inner and outer radius and if so exit the search with SOMETHING_SELECTED_FLAG, TYPE_SELECTED, POINTER_SELECTED, LINE_SELECTED (on the inner or outer radius) all set and BUFFER_SELECTED loaded with the primitive data.

Find_North_Angle

This procedure finds the angle between a line stored in `TEMPL` and the horizontal. The routine is typically used to find the angle of the North line of an object (i.e the first line of the stored object). The previous value of the angle is left stored in `OLD_N_ANGLE` and the new angle in `N_ANGLE`.

This is a geometric routine and contains many code lines concerned with resolving ambiguities and divide by zero situations.

Find_Rot_Angle

This procedure finds the rotated angle between the line joining the mouse `X/Y` co-ordinates and the centroid of an object, and the horizontal. The previous value of the angle is left in `OLD_ROT_ANGLE`, the new angle in `ROT_ANGLE`. On entry to the routine the object centroid co-ordinates are stored in `CENTXY`.

This is a geometric routine and contains many code lines concerned with resolving ambiguities and divide by zero situations.

Rotate_it

This procedure uses geometric transformations to rotate the `x,y` co-ordinates of an object stored in `TEMPL`, and stores to rotated co-ordinates in `TEMPLATE`.

Interpolate_line

A matrix, `BOUNDARY_MESH_NO.`, with row and columns representing the mesh grid numbers, is filled with a character at grid points interpolated between the end co-ordinates of a line, using Brezenham's algorithm. On entry to the routine the line co-ordinates are passed together with the character to be used.

Set_resistors_and_voltages

For every object in the `PRIMITIVE` store check if a grid point is inside or outside the object. If inside then set the resistance value of the four surrounding mesh resistors to a value determined from the permeability parameter by:

1. Subtracting the existing resistor and adding the new resistor for self resistance.
2. Subtracting the existing resistor and replacing by the new resistor for mutual resistance.

This is only done if the new resistance is less than the old, except in the case of permanent magnets where a permeability of 1.1 (resistance = 0.9) will always be used, or an air gap where a resistance of 0.0001 will be set to mark that this is an air gap (this will be changed to a value or 1 in `setup_diagonals`). If the object is a magnet, coil or solenoid then voltage sources will be set around the edges of the object. The resistors are set into vectors `RH` and `RV`, and the voltages into `VLOOP`. Code lines are included to visually show how the resistors and voltages are set. Remove the comment marks to include this code.

If `SOMETHING_SELECTED_FLAG` is set then **Change_Selected_to_Green**. Initialise the `BOUNDARY_MESH_NO` array with blank characters.

If the object is not a wire (defined by `TYPE_SELECTED`) then use the object co-ordinates to define a search square around the object in grid units (this saves scanning the whole mesh grid area). If the object is not a solenoid, magnetic domain or wire then find which line of the object should have positive and which negative voltages: **Find_North_Angle**, store the object centroid in `CENX/Y`, put the object co-ordinates minus centroid into `TEMPL`, set `ROT_ANGLE` to `N_ANGLE` and **Rotate_it**. By determining the start of the north line set volt + or – using the value from `PRIMITIVE[9]`.

If the object is not a solenoid then fill the `BOUNDARY_MESH_NO` matrix with character 'b' along the north and south lines by **Interpolate_line**. Fill the lines between north and south with 'b' or with '+' or '-'.

If the object is a solenoid or circular magnetic domain then fill the `BOUNDARY_MESH_NO` matrix with a '+' or '-' voltage sign or with a 'b' on the outer and inner circles by **Interpolate_circle**.

Now, for all objects except a wire, fill the area outside the object with a character 'o' and then set the RH and RV vectors, with the action determined by the character in the BOUNDARY_MESH_NO matrix. Similarly set the voltages in VLOOP.

If the object is a wire then find the grid point directly from the wire co-ordinates. Then set all surrounding mesh resistors to 1 (a wire is assumed to have a clear space around it) and set the VLOOP voltage by adding the wire voltage to it.

SetUp_Diagonals

Set the resistors in the self (DS), vertical mutual (DV) and horizontal mutual (DH) diagonals, from the resistor values in the horizontal (RH) and vertical (RV) resistor vectors. Put zeros at intervals equal to the mesh size on the vertical diagonal (DV).

PrintMu

Print a 'mu' symbol underneath the value for strength at the object centroid (found in CENT). If the PRINT_FLAG = 0 then the character is put on the screen canvas, else it is put on the printer canvas.

PrintStrength

Print Strength data first, followed by mu underneath if required, on either the screen or printer canvas, at the object centroid (found in CENT). On entry to the routine the strength value is found in TEMP.

ShowData

Prints permeability and strength data in green or red onto objects, on either the screen or printer canvases. On entry to the routine the permeability and strength are in TEMP.

*The routine goes through all object type PRIMITIVES in turn using PRIMSIZE, calculates the object centroid, puts permeability and strength data into TEMP, sets the centroid into CENT and CURRENTX/Y and then calls **PrintMu** and **PrintStrength**. Data intended for the printer (PRINT_FLAG set) is scaled using CLEFT, CTOP, PRTOSCRN and PMARGIN. The colour for the character is determined using the SOMETHING_SELECTED_FLAG and POINTER_SELECTED.*

Form_Select

Changes any selected object from red to green and then uses search routines to determine if the mouse pointer is on an object. If the mouse is on an object, it is changed to red and the data labels loaded to display the object data.

If no objects are stored in primitives then exit. If SOMETHING_SELECTED_FLAG is set then

Change_Selected_to_Green, CanvasToBitmap.

Clear_Buffer_Selected.

*If PRIMSIZE_WIRE > 0 then set SEARCH_TYPE to 3 and **Wire_Search**. If a wire is found (SOMETHING_SELECTED_FLAG = 1) then **Change_Green_to_Selected**.*

ShowData if PRINT_DATA_FLAG = 1, **CanvasToBitmap**, **Set_Data_Labels**, exit.

*If PRIMSIZE_MAGDOM > 0 then set SEARCH_TYPE to 0 and **Search**.*

*If PRIMSIZE_PERMAG > 0 then set SEARCH_TYPE to 1 and **Search**.*

*If PRIMSIZE_COIL > 0 then set SEARCH_TYPE to 2 and **Search**.*

*If PRIMSIZE_SOLENOID > 0 then set SEARCH_TYPE to 4 and **Search**.*

*If PRIMSIZE_CMAGDOM > 0 then set SEARCH_TYPE to 5 and **Search**.*

If an object has been found (SOMETHING_SELECTED_FLAG = 1) then

Change_Green_to_Selected, CanvasToBitmap, Set_Data_Labels. ShowData if PRINT_DATA_FLAG = 1.

Centroid

Calculate the centroid of the BUFFER_SELECTED points and store in CENTXY and CENTROIDXY. Subtract from BUFFER_SELECTED and store the result in TEMPLATE and TEMPL. Also find the max/min limits of template and store in X/YMIN and X/YMAX.

Draw_Mline

Draw a line from the centroid of an object towards the mouse, and an arc between this line and the horizontal. Used as a rotation indicator in edit-rotate function. On entry the centroid co-ordinates are found in CENTXY and the mouse co-ordinates and rotate angle (ROT_ANGLE) in MLINE_BUFFER.

Shift_black

Used in Size, Copy, Move, Rotate functions to draw black lines when the mouse button is down and the mouse moves. Points are in TEMPLATE, BUFFER_BLACK, with centroid in CENTXY or MOUSEXY. Checks that the plot area would not be exceeded. Draws an arc and line in rotate mode.

If MOUSE_IS_DOWN_FLAG = 0 then exit.

If the object is not a wire, solenoid or circular magnetic domain (determined by TYPE_SELECTED) then check that the object would not exceed plotting area by using X/YMIN and X/YMAX (which are the maximum and minimum sizes of the object around the centroid), MOUSEX/Y and LLIMIT, RLIMIT, TLIMIT, BLIMIT (which are the canvas limit areas for drawing).

*If in Size or Rotate mode the check is done using TEMPLATE and CENTXY. If in Rotate mode then draw an arc and line with **Draw_Mline**. DRAWBLACK_FLAG is used to check if the old arc and line should be erased first. If DRAWBLACK_FLAG is set then **Convert_Buffer_Values_to_Integers** and erase the old black object lines. Load BUFFER_BLACK with the new object co-ordinates, using TEMPLATE and MOUSEX/Y if not in Size or Rotate mode, or using TEMPLATE and CENTXY if in Size or Rotate mode.*

***Convert_Buffer_Values_to_Integers** and draw the new black object lines. Set DRAWBLACK_FLAG to indicate that old lines will be erased from now on.*

Set `BLACK_WAS_DRAWN_FLAG` to indicate that a black object has been drawn, and exit.

If the object is a wire (determined by `TYPE_SELECTED`) then check that the plotting area would not be exceeded by using `MOUSEX/Y` and `RLIMIT`, etc. If `DRAWBLACK_FLAG = 1` then erase the old black circle first. Then draw a new black circle, set `DRAWBLACK_FLAG` and `BLACK_WAS_DRAWN_FLAG` and exit. Note that the circles will be scaled to keep the same size regardless of mesh size.

If the object is a solenoid or circular magnetic domain (determined by `TYPE_SELECTED`): first set scaling factors (if not in Size mode) `SOLSCALE`, `S_OLD` and `ss` all to 1. Then check the plot area limits by calculating the size of the outer object circle that would be drawn, using `MOUSEX/Y` as the centre point, unless in Size mode when `BUFFER_SELECTED` is used. `SOLSCALE` is used to scale the circle (which will be 1 unless in Size mode). If the plot area would be exceeded then beep and exit, but first (if `BEEP_FLAG` is 0) store the old scaling factor in `ss` and the old mouse co-ordinates in `MOX/Y`, to ensure that the old black circles are erased properly when the mouse comes back within limits. Set `BEEP_FLAG` to 1. When the mouse is properly within limits then, if `BEEP_FLAG = 1`, restore `MOUSE_OLDX/Y` and `SOLSCALE` from `MOX/Y` and `ss`, and set `BEEP_FLAG = 0`.

Set `BLACK_WAS_DRAWN_FLAG` to 1. If `DRAWBLACK_FLAG = 1` then erase old circles using `S_OLD`. Draw new circles using `SOLSCALE` and set `DRAWBLACK_FLAG = 1`. Retain the old scaling factor by transferring `SOLSCALE` to `S_OLD`.

Change_Copy_to_Selected

Changes black lines to solid double width red and leaves the result in BUFFER_SELECTED.

*If not a wire (determine by TYPE_SELECTED) then, if BLACK_WAS_DRAWN_FLAG = 1, erase black line using **Convert_Buffer_Values_to_Integers**. If not in Size or Rotate mode then load BUFFER_SELECTED using TEMPLATE and MOUSEX/Y. If in Size or Rotate mode then load BUFFER_SELECTED using TEMPLATE and CENTXY. **Convert_Buffer_Values_to_Integers** and draw a new line in red. If a permanent magnet or coil (determined by TYPE_SELECTED) then **DrawNS** using BUFFER_SELECTED values, **CanvasToBitmap**.*

*If a wire then, if BLACK_WAS_DRAWN_FLAG = 1, erase old circles from BUFFER_BLACK. Draw a new circle in red using **Circle**. Draw **Cross** or **Dot** depending on the sign of BUFFER_SELECTED[2]. Set SOMETHING_SELECTED_FLAG to 1, WIRE_FLAG to 3, **CanvasToBitmap**.*

Delete_Selected

Deletes the object red lines from the display and sets SOMETHING_SELECTED_FLAG to 0.

*If a rectangular magnetic domain, magnet or coil (determined from TYPE_SELECTED) **Convert_Buffer_Values_to_Integers** and erase the red lines using BUFFER_SELECTED.*

*If a magnet or coil then erase NS using **DrawNS** from BUFFER_SELECTED values.*

If a circular magnetic domain or solenoid then erase the red circles using BUFFER_SELECTED.

*If a wire then remove with **Circle** and **Cross** or **Dot**. Set SOMETHING_SELECTED_FLAG = 0, **Reset_Data_Labels**, **CanvasToBitmap**.*

Draw_Red_using_BS

*Draws red lines using **Convert_Buffer_Values_to_Integers**.*

Draw_Green_using_BB

*Draws green lines using **Convert_Buffer_Values_to_Integers** and **BUFFER_BLACK** data. If **PRINT_FLAG** = 0 then the screen canvas is used. If **PRINT_FLAG** = 1 then the printer canvas is used and the data is scaled.*

Shuffle_BS

*For the **SwitchPoles** function, move round **BUFFER_SELECTED** data to put new North line in **BUFFER_SELECTED**[0-3]. On entry, **LINE_OFFSET** points to the line required to become the new North line (i.e. the first line in the object).*

*Copy **BUFFER_SELECTED** from **LINE_OFFSET** to the end of the object into **TEMPLATE**[0] onwards. Then copy **BUFFER_SELECTED** up to **LINE_OFFSET** into the remainder of **TEMPLATE**. Then transfer template into **BUFFER_SELECTED**.*

Rotate_Back

*This procedure uses geometric transformations to rotate the x,y co-ordinates in **TEMPLATE** in the opposite direction to the angle in **ROT_ANGLE**. An intermediate buffer, **BUFFER_SIZE**, is used.*

Scale_H

Calculate the length of the line from the mouse to the object centroid and scale the object along NS as the mouse is moved.

*If **DRAWBLACK_FLAG** = 0, i.e. the first entry to the procedure, calculate **LTEMPLATE** as the line length in **TEMPL**.*

*Calculate a scale factor **LMOUSE** as a factor mouse to centroid line divided by **LTEMPLATE**.*

*Set **ROT_ANGLE** = -**N_ANGLE** and **Rotate_it** from **TEMPL** to **TEMPLATE**. Multiply **TEMPLATE** by the scale factor, set **ROT_ANGLE** = **N_ANGLE** and **Rotate_Back**.*

Scale_V

Calculate the length of the line from the mouse to the object centroid and scale the object along the non-NS direction as the mouse is moved.

If DRAWBLACK_FLAG = 0, i.e. the first entry to the procedure, calculate LTEMPLATE as the line length in TEMPL.

Calculate a scale factor LMOUSE as a factor mouse to centroid line divided by LTEMPLATE.

*Set ROT_ANGLE = -N_ANGLE and **Rotate_it** from TEMPL to TEMPLATE. Multiply TEMPLATE by the scale factor, set ROT_ANGLE = N_ANGLE and **Rotate_Back**.*

Scale_it

Calculate the length of the line from mouse point to the object centroid and scale the object uniformly in all directions.

If not a solenoid or circular magnetic domain then if DRAWBLACK_FLAG = 0, i.e. the first entry to the procedure, calculate LTEMPLATE as the line length in TEMPL (TEMPLATE and TEMPL contain (BUFFER_SELECTED minus centroid) on entry). Calculate a scale factor LMOUSE as a factor mouse to centroid line divided by LTEMPLATE. Multiply TEMPLATE by the scale factor.

If a solenoid or circular magnetic domain then calculate a scale factor SOLSCALE. If DRAWBLACK_FLAG = 0 then retain SOLSCALE in S_OLD.

Remove_Data

Delete data from a PRIMITIVE store for a particular object, move data up the store, and reset PRIMSIZE.

Draw_Primitives

Draws grid, draws screen objects from PRIMITIVE stores, sets selected object to red, sets data labels. If PRINT_FLAG = 0 the screen canvas is used, otherwise the printer canvas is used and data is scaled.

DrawGrid

*If not a circular magnetic domain, solenoid or wire then load BUFFER_BLACK with data from the PRIMITIVE store, **Draw_Green_using_BB**. If a magnet or coil then **DrawNS** using BUFFER_BLACK data.*

If a circular magnetic domain or solenoid then draw directly from the primitive store.

*If a wire then load BUFFER_SELECTED from the PRIMITIVE store and draw **Circle** and **Dot** or **Cross**.*

*Now change a selected object to red and set data labels. If SOMETHING_SELECTED_FLAG = 1 then load BUFFER_SELECTED from the PRIMITIVE store at the POINTER_SELECTED point, **Change_Green_to_Selected**, **Set_Data_Labels**, **CanvasToBitmap**.*

File_Save

Check that a .MAG file extension has been specified and then write a file of flags, a file of data and a title file. A header is included to allow later file format updates.

Save_As

*Puts the caption (if a file name) as the name in the dialog box. Then **File_Save**, **Reset_Most_Flags**.*

ReSizeArray1

Dynamically resizes arrays of T1arr type (see global variables).

ReSizeArray2

Dynamically resizes arrays of T2arr type (see global variables).

ReSizeArrayUD

Dynamically resizes the array storage for the upper diagonals in gaussian elimination.

ReSizeArrayBMesh

Dynamically resizes the boundary mesh array.

ReSizeArrayW

Dynamically resizes the array storage for the matrix arithmetic block in gaussian elimination.

File_Open

Checks for an extension of .MAG. Reads the file of flags (checking for the correct header), and sets all the flags and other integer data.

Free_Memory and Resize_Arrays.

Reads the file of data and sets all the PRIMITIVE stores and ILOOP.

Reads the title file.

Ask_about_Save

Used when exiting Vizimag, if there have been unsaved changes. If required then Save_As.

Draw_Hline

Draw horizontal line for a rotation indicator, for a fixed distance from the object centroid co-ordinates found in CENTXY.

Scale_Data

If the current display is a different resolution to the written disk file then scale the PRIMITIVE values. The variables WRITTENCANVASWIDTH and CANVASWIDTH are used to determine a scale factor.

Fopen

If FILECHANGED_FLAG = 1 Then Ask_About_Save. Set FILECHANGED_FLAG = 0. Reset_Data_Labels, display the openfile dialog box. Clear_Screen, initialise the external field panel, check that the file has .MAG extension, File_Open, if the current CANVASWIDTH is different to WRITTENCANVASWIDTH then Scale_Data.

BitMapCreate, Clear_Screen, Draw_Primitives, ShowData if required, CanvasToBitmap. If CONTOUR_FLAG = 1 then Contour.

Enable_menus_in_Edit, Reset_Most_Flags, set the form caption.

PrintMatrix(M: string; A: P1arr; LENGTH: integer)

A debug aid to print a vector in row and column matrix form. The printer must be in landscape format from Windows control panel. M is a name to be printed. A is the vector to be printed. LENGTH is the length of the vector.

e.g. PrintMatrix('Iloop', Iloop, (mesho-1))

PrintW(I:integer)

A debug aid to print the computation block in gaussian elimination in row and column matrix form. The printer must be in landscape format from Windows control panel.

Writelloop

A debug aid which writes the lloop vector in BCD format to disk. The file is named lloop.dat which may be read into other applications.

MatMult

A sparse matrix multiplication routine using single precision arithmetic. A vector MATOUT is created by multiplying a vector MATIN (provided on entry) with the coefficient matrix defined by the diagonals DS, DV, DH, the remainder of the matrix being assumed to be zero.

MatMultD

A sparse matrix multiplication routine using double precision arithmetic. A vector MATOUT is created by multiplying a vector MATIN (provided on entry) with the coefficient matrix defined by the diagonals DS, DV, DH, the remainder of the matrix being assumed to be zero.

Analyse_CG

The default single precision conjugate gradient analysis routine. The algorithm for this is provided in chapter 4. After completing the analysis, Contour, Enable_Menus_in_Edit.

Analyse_CG_DoublePrec

Double precision conjugate gradient analysis routine. The result is left in single precision for Contour.

Analyse_Gauss

See chapter 4 for a description of the gaussian elimination routine. Calls **Contour**.

Eset

Disable_Menus_in_Edit. If `CONTOUR_FLAG = 1` then remove contour lines with **Clear_Screen**, **Draw_Primitives**, set `CONTOUR_FLAG = 0`.

Initialise resistor R_H and R_V vectors with unity. Set resistors for $\mu = 5$ two layer boundary around the periphery of the mesh. Initialise diagonals with $D_S = 4$, $D_V = -1$ and 0 , $D_H = -1$, and V_{LOOP} and I_{LOOP} vectors with zeros. Set voltages for external fields, calling **AddTBwires** and **AddLRwires**. Then set up resistor R_H and R_V vectors and V_{LOOP} vector with

Set_resistors_and_voltages. Go through all resistors in R_H and R_V vectors and change any value of 0.0001 (a marker for an air gap) to 1 , and any V_{LOOP} voltage in the corresponding mesh grid point to zero. Set up the D_S , D_H , D_V diagonals with the resistor values from the resistor vectors with **Setup_Diagonals**.

Enable_Menus_in_Edit.

Change_Solenoid_to_Selected

Erase black circles from `BUFFER_BLACK`, redraw as red and load `BUFFER_SELECTED` with `BUFFER_BLACK` values. Set `SOMETHING_SELECTED_FLAG = 1`.

Edit_Solenoid

Called after a form mouse up event in Move, Copy, Size, Change data.

Set scaling factors SOLSCALE, S_OLD, ss to 1 if not in size mode.

Check that plot area limits would not be exceeded by calculating the circle diameter that would be obtained, using MOUSEX/Y if not in Size mode, or BUFFER_SELECTED if in Size mode.

If CHANGEDATA_FLAG = 0 then erase black circles if BLACK_WAS_DRAWN_FLAG = 1. Note that scale factors are retained as required if the mouse is off the plot limits. Erase red circles using BUFFER_SELECTED. If COPY_FLAG = 1 then draw green circles using BUFFER_SELECTED data. Change BUFFER_SELECTED centre position (BUFFER_SELECTED[0,1]) to its new value and scale BUFFER_SELECTED with SOLSCALE if in Size mode. Draw new red circles. CanvasToBitmap, set SOMETHING_SELECTED_FLAG = 1, set SOLENOID_FLAG or CMAGDOM_FLAG to 4, Set_and_Store_Data,

Set_Data_Labels, Reset_Most_Flags, Enable_Menus_in_Edit.

General program procedures in unit VIZ2

DrawNS

Draws lines to give N and S characters using data either from BUFFER_SELECTED or BUFFER_BLACK. Depending on PRINT_FLAG either the screen or printer canvas is used, and data is scaled for the printer.

Check_Size0

Leaves TOO_SMALL_FLAG = 1 if length of line from BUFFER_BLACK[0,1] to MOUSEX/Y is < 2.

Store_MouseXY

Stores the x,y coordinates of the mouse in MOUSEX/Y, retaining the previous values in MOUSE_OLDX/Y, and also showing the coordinates on screen (relative to zero at screen centre) together with the angle of the line. For the angle, if SOMETHING_SELECTED_FLAG = 1 then the angle of the North line (i.e. the first line of the object) to the horizontal is shown. If in Rotate mode the angle of the rotating copy is shown.

Set_Data_Labels

Shows the object data on screen for a selected object.

Hide_Frame

*Hides data entry frames, putting the bitmap back to restore the screen display with **BitmapToCanvas**.*

Show_PermStrength_Frame

If COIL_FLAG = 5 then make the permeability data entry box visible. If SOMETHING_SELECTED_FLAG = 1 then get the permeability data from BUFFER_SELECTED[8], convert to text and show it in the permeability text box. If SOMETHING_SELECTED_FLAG = 0 then put '1' in the text box. Set focus on the text box.

If COIL_FLAG = 6 then make the strength data entry box visible. If SOMETHING_SELECTED_FLAG = 1 then get the strength data from BUFFER_SELECTED[9], convert to text and show it in the strength text box. If SOMETHING_SELECTED_FLAG = 0 then put '1' in the text box. Set focus on the text box.

Reset_Most_Flags

Reset all flags except SOMETHING_SELECTED_FLAG, GRIDLINES, GRIDDOTS, SHOWGRID, SHOWCENTRELINES *and* CONTOUR_FLAG.

Reset_All_Flags

Reset all flags except GRIDLINES, GRIDDOTS, SHOWGRID *and* SHOWCENTRELINES.

BitmapToCanvas

Stores the screen canvas onto the internal bitmap.

Clear_Screen

Fills the screen or printer canvas with white.

Set_Primsizes_values_to_Zero

Sets PRIMSIZE value of all PRIMITIVE stores to zero.

BitMapCreate

Creates the bitmap used for storing the canvas, calculates the grid scaling factors xs and ys (pixels per grid unit), sets the canvas size square by ensuring xs and ys are equal by modifying the data label widths, sets the limit values, LLIMIT, RLIMIT, TLIMIT, BLIMIT used for checking if objects are within the mesh area and sets XBIAS, YBIAS values used in the x,y and degree on screen readout.

NonNumCheck

Checks if a data entry is a valid number, but allowing the character 'a' as a marker for an airgap.

Reset_Data_Labels

Resets all the data label captions to blank.

Draw_GridDots

Draws a grid of dots to screen or printer canvas (depending on the setting of PRINT_FLAG), with darker dots every tenth row and column. For large meshes of 200×200 and 300×300 then use SCALE to plot one dot every two grid points to avoid overcrowding the display.

First plot a dot every grid point. Then erase every tenth row and column and redraw every tenth black. If PLOTALL = 0 (i.e. the whole area including the margin is to be plotted) draw a line at the margin. Check the tools-dots box, and if required DRAW_CENTRELINES.

Draw_GridLines

Draws a grid of lines to screen or printer canvas (depending on the setting of PRINT_FLAG), with darker lines every tenth row and column. For large meshes of 200×200 and 300×300 then use SCALE to plot one line every two grid points to avoid overcrowding the display.

First plot a line every grid point. Then erase every tenth row and column and redraw every tenth black. If PLOTALL = 0 (i.e. the whole area including the margin is to be plotted) draw a line at the margin. Check the tools-lines box, and if required DRAW_CENTRELINES.

Erase_GridDots

Call Draw_GridDots to erase the grid in exclusive or mode.

Erase_GridLines

Call Draw_GridLines to erase the grid in exclusive or mode.

Enable_menus

Set the enabled property true for all menus, speedbuttons and showhints.

Disable_Menus_in_Edit

Set the enabled property false for all menus, speedbuttons and showhints. Send an update command to ensure the glyphs are updated immediately on the screen.

Enable_Menus_in_Edit

Set the enabled property true for all menus, speedbuttons and showhints.

Show_Perm_Frame

Make the permeability data entry box visible. If `SOMETHING_SELECTED_FLAG = 1` then get the permeability data from `BUFFER_SELECTED[8]`, convert to text and show it in the permeability text box. If `SOMETHING_SELECTED_FLAG = 0` then put '1' in the text box. Set focus on the text box.

Show_Strength_Frame

Make the strength data entry box visible. If `SOMETHING_SELECTED_FLAG = 1` then get the strength data from `BUFFER_SELECTED[9]` (`BUFFER_SELECTED[2]` FOR A WIRE), convert to text and show it in the strength text box. If `SOMETHING_SELECTED_FLAG = 0` then put '1' in the text box. Set focus on the text box.

Show_Direction_Frame

Make the direction data entry panel for a wire visible.

HelponLine

Depending on the value of the FLAG passed to the procedure, select on-line help text for display.

General program procedures in unit VIZ3

Contour

Disable_menus_in_Edit, set CONTOUR_FLAG = 1, **Smooth** the ILOOP values and store the smoothed values in ILOOPS. Take the lower left four adjacent meshes and first use the ILOOPS values of the two lower points. If there is a contour line, then (for every contour) find the co-ordinates of its position using real numbers. Plot a point. Then interpolate up towards the upper two points and repeat. The interpolation is by one pixel since the object is to plot so that dots merge. Now repeat the whole thing except move from left to right instead of vertically. This ensures that no point will be lost because of quantisation. Then gradually increment along the meshes of a whole row horizontally, and then vertically. The number of contour lines (in NUM) should be odd for best symmetry. The interval for contour lines is stored in CONINT, and the values for each line in CONT. Depending on the value of PRINT_FLAG the screen or printer canvases will be used. For increased plotting speed each set of four meshes is first checked to see if there are any contour lines, if not they are skipped. **CanvasToBitmap**.

Resize_Arrays

Call **ResizeArray1** for DS, DH, DV, VLOOP, ILOOP, ILOOPS, RV, RH, BD.

Call **ReSizeArrayBMesh**.

Free_Memory

Unlock and free allocated memory for arrays.

AddTBwires

Add orthogonal wire voltages in vertical external field.

AddLRwires

Add orthogonal wire voltages in horizontal external field.

Show_SolData_Frame

Make the solenoid data entry panel visible. If `SOMETHING_SELECTED_FLAG = 1` then get permeability data from `BUFFER_SELECTED[8]`, convert to text and display in the permeability text box. Get strength data from `BUFFER_SELECTED[9]`, negate depending on sign, convert to text and display in the strength text box. Set field direction depending on strength sign. If `SOMETHING_SELECTED_FLAG = 0` then put '0' in strength and '1' in permeability text boxes.

Draw_black_Solenoid

If `DRAWBLACK_FLAG = 1` then erase the old black circle using `BUFFER_BLACK` data and `MOUSE_OLDX/Y`. If the `FLAG` value (passed to the routine on entry) is 2 then set the inner circle radius into `BUFFER_BLACK`, else if `FLAG = 3` set the outer radius into `BUFFER_BLACK`. Draw new circle using `BUFFER_BLACK` and `MOUSEX/Y`. Set `DRAWBLACK_FLAG = 1`.

Interpolate_circle

Interpolates a circle in grid unit values from the co-ordinate data defining the circle. Real arithmetic is used for best accuracy and symmetry, by solving the equation $x^2 + y^2 = r^2$. Variables `p` and `q` are used to set the increment directions for the four quadrants. A character 'b' is set into `BOUNDARY_MESH_No` array where a point of the circle lies.

Draw_CentreLines

Horizontal and vertical centre lines are drawn. Either the screen or printer canvases are used depending on the value of PRINT_FLAG.

Fluxfill

Go through LOOP square by square and determine the slope of the line by taking the largest modulus value of the slope to the eight connected squares. The result is a representation of the flux density. Plot the result as a colour or mono fill of each mesh square, with colours ranging from red (max) to black (min). If no contours have been calculated (CONTOUR_FLAG = 0) then the procedure exits.

Make the flux density option panels (colour or mono, log or linear, filter level) visible. Start at upper left, calculate the line slope for each grid square and store the result in BD. Now, for each point, take the absolute value, find the maximum and normalise.

If a log display is required then take a log function, find the maximum value, normalise and invert.

Go through each BD point and compare to the value set in the filter function. If the value is greater than the filter value then set the value to zero, find the maximum value and normalise.

Now plot the data. Each grid square in turn (starting at the upper left) is checked and a colour or grey level determined from the value of BD. For colour the display is split into six colour segments as shown in figure 6.4. The whole grid square is filled using the canvas.rectangle function. Depending on the setting of PRINT_FLAG either the screen or printer canvas is used.

*Then **Draw_Primitives** and **Contour** (if the option of adding field lines has been selected). In flux density mode the menu options of PRINT, F>B and CLIPBOARD only are then enabled (to continue editing return to field lines plot with F>B). **CanvasToBitmap**.*

FtoB

Toggle the display between field lines plot and flux density plot.

If CONTOUR_FLAG = 0 then exit.

*If flux density mode is required then **Disable_Menus_in_Edit**, **Clear_Screen**, **Fluxfill**. Enable **PRINT**, **CLIPBOARD** and **F>B** functions only.*

*If field lines are required then **Disable_Menus_in_Edit**, **Clear_Screen**, make the flux density option panels invisible, **Draw_Primitives**, **Contour**, **Enable_Menus_in_Edit**.*

Smooth

Smooth ILOOP currents by averaging with a portion of the surrounding eight squares. Smoothing is not performed if the whole mesh is being displayed. Store the result in ILOOPS. Options of no smoothing, addition of 0.1 of the surrounding squares or 0.25 of the surrounding squares is available.

Other mouse and keyboard event initiated procedures

TForm1.FormMouseMove

*Put mouse X co-ordinate in MOUSEX and mouse Y co-ordinate in MOUSEY.
Store_MouseXY.*

If GUIDELINE_FLAG = 1 then exit.

*If GUIDELINE_FLAG = 2 then **Draw_Black**, exit.*

*If MAGDOM_FLAG, PERMAG_FLAG OR COIL_FLAG > 0 then **Draw_Black.***

*If SOLENOID_FLAG OR CMAGDOM_FLAG > 0 then **Draw_Black_Solenoid.***

*If SIZE_FLAG = 1 (and not a solenoid or circular magnetic domain) then if
H_FLAG = 1 **Scale_H**, or if V_FLAG = 1 **Scale_V**. If H_FLAG and V_FLAG = 0 then
Scale_it (scale both H&V directions).*

*If ROTATE_FLAG = 1 (and not a solenoid or circular magnetic domain) then
Rotate_it.*

*If COPY_FLAG, MOVE_FLAG, SIZE_FLAG or ROTATE_FLAG = 1 then **Shift_Black.***

TForm1.FormMouseUp

If CHANGEDATA_FLAG = 1 *then* exit. Set MOUSE_IS_DOWN_FLAG = 0. Put mouse X co-ordinate in MOUSEX and mouse Y co-ordinate in MOUSEY. **Store_MouseXY**.

If GUIDELINE_FLAG > 0 *then* **Guideline_in_mouseup**, exit.

If EDIT_FLAG = 0 *then* select an object with **Form_Select**, exit.

If MAGDOM_FLAG, PERMAG_FLAG or COIL_FLAG > 0 *then* **in_mouseup**, exit.

If SOLENOID_FLAG or CMAGDOM_FLAG > 0 *then* **Solenoid_in_up**, exit.

If WIRE_FLAG > 0 *then* **Wire_in_mouseup**, exit.

If TYPE_SELECTED = 4 (solenoid) or TYPE_SELECTED = 5 (circular magnetic domain) *then* **Edit_Solenoid**, exit.

If COPY_FLAG = 1 *then* check that the plotting area would not be exceeded using XMIN, XMAX and RLIMIT, etc. **Change_Selected_to_Green**, **Change_Copy_to_Selected**, **CanvasToBitmap**. Set flags ready for data entry. Set SOMETHING_SELECTED_FLAG = 1. **Set_and_Store_Data**, **Set_Data_Labels**, **Reset_Most_Flags**, **Enable_Menus_in_Edit**, exit.

If MOVE_FLAG, SIZE_FLAG or ROTATE_FLAG = 1 *then* check that the plotting area would not be exceeded using XMIN, XMAX and RLIMIT, etc. in MOVE, or TEMPLATE and CENTXY in SIZE or ROTATE modes. *If* SIZE_FLAG = 1 *then* check that no line is less than 2 grid units in length. *If* ROTATE_FLAG = 1 *then* (if BLACK_WAS_DRAWN_FLAG = 1) erase the angle line and arc with **Draw_Mline**, and the horizontal guide line with **Draw_Hline**. **Delete_Selected**, **Change_Copy_to_Selected**, **CanvasToBitmap**. Set flags ready for data entry. Set SOMETHING_SELECTED_FLAG = 1. **Set_and_Store_Data**, **Set_Data_Labels**, **Reset_Most_Flags**, **Enable_Menus_in_Edit**, exit.

If (SWITCHPOLES_FLAG = 1) *then* find the nearest line to the mouse pointer with **Normalised_Angle**, erase the NS characters with **DrawNS** using BUFFER_SELECTED data, move the data around to make the new N line the first line in store with **Shuffle_BS**. Draw the new NS characters with **DrawNS**. **CanvasToBitmap**, set flags ready for next procedure, **Set_and_Store_Data**, set SOMETHING_SELECTED_FLAG = 1, **Reset_Most_Flags**, **Enable_Menus_in_Edit**, exit.

If a wire and COPY_FLAG = 1 *then* check that the plotting area would not be exceeded using MOUSEX/Y and RLIMIT, etc. **Change_Selected_to_Green**, load BUFFER_SELECTED with MOUSEX/Y, set POINTER_SELECTED to the start of new data in the PRIMITIVE store, increase the size of the store (PRIMSIZE) and store

BUFFER_SELECTED data in the store. **Change_Copy_to_Selected**, **CanvasToBitmap**, set **SOMETHING_SELECTED_FLAG = 1**, **Set_Data_Labels**, **Reset_Most_Flags**, **Enable_Menus_in_Edit**, **exit**.

*If a wire and MOVE_FLAG = 1 then check that the plotting area would not be exceeded using MOUSEX/Y and RLIMIT, etc. **Delete_Selected**, load BUFFER_SELECTED with MOUSEX/Y, store BUFFER_SELECTED in the PRIMITIVE store. **Change_Copy_to_Selected**, **CanvasToBitmap**, set SOMETHING_SELECTED_FLAG = 1, **Set_Data_Labels**, **Reset_Most_Flags**, **Enable_Menus_in_Edit**.*

TForm1.PermOKClick

When the OK button is clicked, get permeability data and either change black to selected or go on to show strength frame. Then set and store data.

Get_Data, check if a data error was detected by **ERROR_FLAG**.

*If COIL_FLAG = 5: Set it to 6, **Hide_Frame**. **Set_instructions_label**, **Show_PermStrength_Frame**, **exit**.*

*If COIL_FLAG not = 5 then **Hide_Frame**.*

*If CHANGEDATA_FLAG = 0 then if CMAGDOM_FLAG = 0 **Change_black_to_selected**, else if CMAGDOM_FLAG not = 0 **Change_Solenoid_to_selected**.*

CanvasToBitmap.

Set_and_Store_Data, **Set_Data_Labels**, **Enable_Menus_in_Edit**, **Reset_Most_Flags**.

TForm1.UpButtonClick

Up direction for wire.

Direction_Button.

TForm1.DownButtonClick

Down direction for wire.

Set WIRE_FLAG = 3 (indicates negative). **Direction_Button.**

TForm1.TitleOKClick

Gets title and shows this on caption bar.

TForm1.StrengthOKClick

Gets strength data, completes black to selected change and sets and stores data. Goes on to show direction frame if a wire.

Get_Data, check if a data error was detected by ERROR_FLAG.

If COIL_FLAG or PERMAG_FLAG > 0 then:

If CHANGEDATA_FLAG = 0 **Hide_Frame, Change_black_to_selected, DrawNS, CanvasToBitmap.**

If CHANGEDATA_FLAG = 1 **Hide_Frame.**

Set_and_Store_Data, Set_Data_Labels, Enable_Menus_in_Edit, Reset_Most_Flags, exit.

If WIRE_FLAG > 0, or TYPE_SELECTED = 3 (wire) and CHANGEDATA_FLAG = 1:

Hide_Frame, save BUFFER_SELECTED[2] in BS2_OLD, load BUFFER_SELECTED[2] with strength value, CanvasToBitmap, Show_Direction_Frame, exit.

TForm1.ExtfldButtonClick

Get external field data.

*Check that data has been set, check the value of strength for errors, make the data entry panel invisible. **BitmapToCanvas, Enable_Menus_in_Edit, Reset_Most_Flags.***

TForm1.Button2Click

OK button in Preferences panel, writes a new preferences file to disk.

*Calls **BitMaptoCanvas, Enable_Menus_in_Edit.***

TForm1.Button3Click

*Preferences cancel button. Calls **Enable_Menus_in_Edit, BitmaptoCanvas.***

TForm1.ExtfldcancelClick

*External field cancel button. Calls **BitmapToCanvas, Enable_Menus_in_Edit, Reset_Most_Flags.***

TForm1.SolButtonClick

Solenoid data entry OK button.

*Get the permeability text, check for errors and store in **BUFFER_SELECTED[8].***

*Get the strength text and check for errors. Negate if direction negative, and store in **BUFFER_SELECTED[9].***

*If **CHANGEDATA_FLAG** is not set then put **BUFFER_BLACK** co-ordinate data into **BUFFER_SELECTED**. **Set_and_Store_Data, Hide_Frame, Change_Solenoid_to_selected, CanvasToBitmap, Set_Data_Labels, Enable_Menus_in_Edit, Reset_Most_Flags.***

TForm1.SolStrenClick

Set focus on solenoid strength panel when clicked.

TForm1.SolStrenMouseDown

Set focus on solenoid strength.

TForm1.FormMouseDown

Set MOUSE_IS_DOWN_FLAG = 1 to show mouse is down when dragging in COPY, MOVE, ROTATE, SIZE.

TForm1.BcolorMouseUp

Colour/mono mode on flux density plot.

Fluxfill.

TForm1.BmodeMouseUp

Log/linear mode on flux density plot.

Fluxfill.

TForm1.BsetClick

Set flux density filter level as $(100 - Bfilter_level.position)/100$.

Fluxfill.

TForm1.FormKeyPress

If key number = 27 (Esc key on keyboard) then ESCAPEPRESSED = 1.

Index to Source Code Procedures

AddLRwires, 136
AddTBwires, 136
Analyse – Analyse options – Conjugate
 gradient high, 100
Analyse – Analyse options – Conjugate
 gradient normal, 100
Analyse – Analyse options – Gaussian
 elimination, 100
Analyse – Run, 100
Analyse – Set up net, 100
Analyse – Smoothing options – High, 101
Analyse – Smoothing options – None, 101
Analyse – Smoothing options – Normal,
 101
Analyse_CG, 128
Analyse_CG_DoublePrec, 128
Analyse_Gauss, 129
Application close down, 89
Application start up, 88
Ask_about_Save, 126

BaloneClick, 99
BcolorMouseUp, 144
BitMapCreate, 132
BitmapToCanvas, 132
BmodeMouseUp, 144
BpluslinesClick, 99
BsetClick, 144
Button2Click, 143
Button3Click, 143

Centroid, 120
CGHighClick, 100
CGNormalClick, 100
Change_black_to_selected, 113
Change_Copy_to_Selected, 122
Change_Green_to_Selected, 114
Change_Selected_to_Green, 108
Change_Solenoid_to_Selected, 129
Check_Plot_limits, 108
Check_Size0, 130

Circle, 108
Clear_Buffer_Selected, 114
Clear_Screen, 132
ClearAllGuidelinesClick, 98
Clipboard – Copy to clipboard, 98
ClipClick, 98
Contour, 135
Convert_Buffer_Values_to_Integers, 106,
 114
Cross, 108

Delete_Selected, 122
Direction_Button, 113
Disable_Menus_in_Edit, 134
Dot, 107
DownButtonClick, 142
Draw_Black, 107
Draw_black_Solenoid, 136
Draw_CentreLines, 137
Draw_Green_using_BB, 123
Draw_GridDots, 133
Draw_GridLines, 133
Draw_Hline, 126
Draw_Mline, 120
Draw_Primitives, 124
Draw_Red_using_BS, 123
DrawGrid, 106, 125
DrawNS, 130

EChangeData, 96
EClear, 98
eCmag, 93
ECoil, 93
ECopy, 94
Edit_Solenoid, 130
Edit-Add Circular Magnetic Domain, 93
Edit-Add Coil, 93
Edit-Add Guideline, 93
Edit-Add Magnetic Domain, 93
Edit-Add Permanent Magnet, 93
Edit-Add Solenoid, 94

- Edit-Add Vertical Wire, 94
- Edit-Change Data, 96
- Edit-Clear, 98
- Edit-Clear All, 98
- Edit-Clear all guidelines, 98
- Edit-Copy, 94
- Edit-Flip-Left/right, 95
- Edit-Flip-Top/bottom, 95
- Edit-Move, 94
- Edit-Rotate, 94
- Edit-Size-Both, 97
- Edit-Size-H, 97
- Edit-Size-V, 97
- Edit-Switch Poles, 96
- Efliplr, 95
- Eflipth, 95
- Eguide, 93
- Emag, 93
- EMove, 94
- Enable_menus, 134
- Enable_Menus_in_Edit, 134
- EPermag, 93
- Erase_GridDots, 133
- Erase_GridLines, 133
- ERotate, 94
- Eset, 129
- ESizeBoth, 97
- ESizeH, 97
- ESizeV, 97
- ESolenoid, 94
- ESwitchPoles, 96
- EWire, 94
- External field, 101
- ExtfldButtonClick, 143
- ExtfldcancelClick, 143

- File_Exit, 92
- File_Open, 126
- File_Print, 92
- File_Save, 125
- File-File Save, 91
- File-New File, 90
- File-Open, 91
- Find_North_Angle, 116
- Find_Rot_Angle, 116
- Flip, 95
- Fluxfill, 137
- Fopen, 91, 127
- Form.Refresh, 89
- Form.Update, 89
- Form_Select, 119
- FormKeyPress, 144
- FormMouseDown, 144
- FormMouseMove, 139
- FormMouseUp, 140
- Fprint, 92
- Free_Memory, 135

- Fsave, 91
- FtoB, 138

- GaussClick, 100
- Get_Data, 112
- Guideline_in_mouseup, 110

- HelponLine, 135
- Hide_Frame, 131

- in_mouseup, 109
- Interpolate_circle, 136
- Interpolate_line, 116

- MatMult, 128
- MatMultD, 128
- MnuAnalyseRunClick, 100
- MnuAnalyseSetClick, 100
- MnuEditClearClick, 98
- MnuExtFieldClick, 101
- MnuFluxdensityClick, 99
- MnuFluxHighClick, 103
- MnuFluxlinesClick, 99
- MnuFluxLowClick, 104
- MnuFluxNormalClick, 103
- MnuHelpClick, 104
- mnuPreferencesClick, 106
- MnuPrintDataClick, 104
- MnuTitleClick, 99
- MnuToolsDotsClick, 102
- MnuToolsLinesClick, 102
- MnuToolsShowCentrelinesClick, 103
- MnuToolsShowGridClick, 103

- N100Click, 105
- N150Click, 105
- N200Click, 105
- N300Click, 106
- N50Click, 104
- N76Click, 105
- Nfile, 90
- NonNumCheck, 132
- Normalised_Angle, 115

- PermOKClick, 141
- PrintMatrix, 127
- PrintMu, 118
- PrintStrength, 118
- PrintW, 127

- Rectangle_Points, 107
- Remove_Data, 124
- Reset_All_Flags, 132
- Reset_Data_Labels, 133
- Reset_Most_Flags, 132
- Resize_Arrays, 135
- ReSizeArray1, 125

-
- ReSizeArray2, 125
 - ReSizeArrayBMesh, 126
 - ReSizeArrayUD, 126
 - ReSizeArrayW, 126
 - Rotate_Back, 123
 - Rotate_it, 116
 - Save_As, 125
 - Scale_Data, 127
 - Scale_H, 123
 - Scale_it, 124
 - Scale_V, 124
 - Search, 115
 - Set_and_Store_Data, 112
 - Set_Data_Labels, 131
 - Set_Primsizes_values_to_Zero, 132
 - Set_resistors_and_voltages, 117
 - SetUp_Diagonals, 118
 - Shift_black, 120
 - Show_Direction_Frame, 134
 - Show_Perm_Frame, 134
 - Show_PermStrength_Frame, 131
 - Show_SolData_Frame, 136
 - Show_Strength_Frame, 134
 - ShowData, 119
 - Shuffle_BS, 123
 - Smooth, 138
 - SmoothhighClick, 101
 - SmoothnoneClick, 101
 - SmoothnormalClick, 101
 - SolButtonClick, 143
 - Solenoid_in_up, 111
 - SolStrenClick, 144
 - SolStrenMouseDown, 144
 - Speedbutton 1, 91
 - Speedbutton 10, 94
 - Speedbutton 11, 95
 - Speedbutton 12, 96
 - Speedbutton 13, 96
 - Speedbutton 14, 97
 - Speedbutton 15, 98
 - Speedbutton 16, 95
 - Speedbutton 17, 97
 - Speedbutton 18, 97
 - Speedbutton 19, 90
 - Speedbutton 2, 91
 - Speedbutton 21, 101
 - Speedbutton 22, 94
 - Speedbutton 23, 93
 - Speedbutton 25, 93
 - Speedbutton 3, 92
 - Speedbutton 4, 93
 - Speedbutton 5, 93
 - Speedbutton 6, 93
 - Speedbutton 7, 94
 - Speedbutton 8, 94
 - Speedbutton 9, 94
 - Speedbutton20Click, 102
 - Speedbutton 20, 102
 - Store_MouseXY, 131
 - StrengthOKClick, 142
 - Title, 99
 - TitleOKClick, 142
 - Tools – Flux Lines – High, 103
 - Tools – Flux Lines – Low, 104
 - Tools – Flux Lines – Normal, 103
 - Tools – Grid – Dots, 102
 - Tools – Grid – Lines, 102
 - Tools – Mesh size – 100 X 100, 105
 - Tools – Mesh size – 150 X 150, 105
 - Tools – Mesh size – 200 X 200, 105
 - Tools – Mesh size – 300 X 300, 106
 - Tools – Mesh size – 50 X 50, 104
 - Tools – Mesh size – 80 X 80, 105
 - Tools – On Line Help, 104
 - Tools – Preferences, 106
 - Tools – Show centre lines, 103
 - Tools – Show Data, 104
 - Tools – Show Grid, 103
 - UpButtonClick, 142
 - View – Flux density, 99
 - View – Fluxlines, 99
 - View – Options – Flux density alone, 99
 - View – Options – Flux density plus flux lines, 99
 - Wire_in_mouseup, 113
 - Wire_Search, 114
 - Writelloop, 128

Index

Air cored coil, 8
Air gap, 69
Ampere's Law, 10
Ampere-Maxwell law, 16
Analysis times, 41, 42
Artefacts, 64

B, 7
Backward substitution, 29
Balance, 78
Bar magnet, 7
Bar magnets, 1, 4
Biot-Savart, 9
Borland, 5, 73
Boundary element, 3, 15
Boundary region, 47, 53

CHANGE DATA, 71
CIE colour diagram, 64
Circuit meshes, 19
Circular magnetic domain, 69
Clock, 41
Coefficients, 21
Coercivity, 15
Coil, 69
Colour scale, 64
Communications, 5
Computation time, 29
Conjugate gradient, 29, 31
Conjugate gradient algorithm, 32
Conjugate Gradient visualisation, 45
Conjugate gradients, 38
Conjugation, 38
Contour lines, 21
Contour plot, 24
Convergence, 33, 38
COPY, 71
Copyright, 5
Curie point, 14
Current, 17

Data structure, 70
DELETE, 71
Delphi, 5, 73
Diagonal matrix, 26
Displacement current, 16
Distortion, 57
Distortions, 53
Division by zero, 30
Double precision, 33

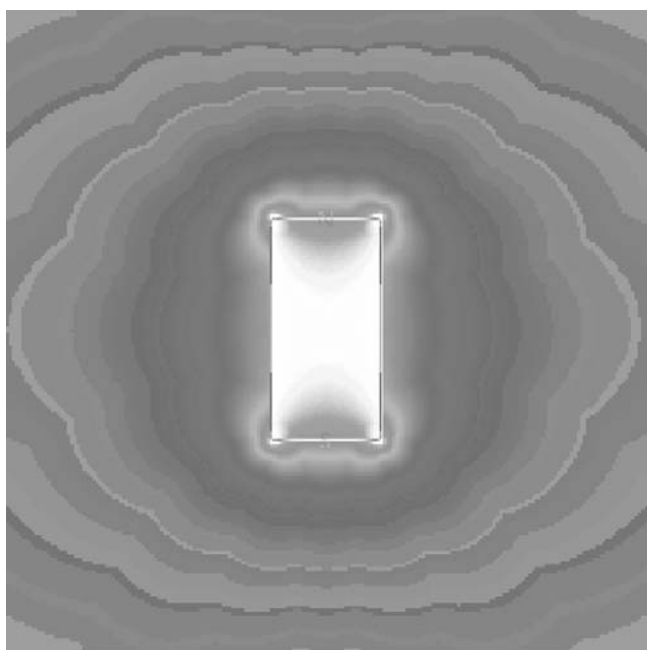
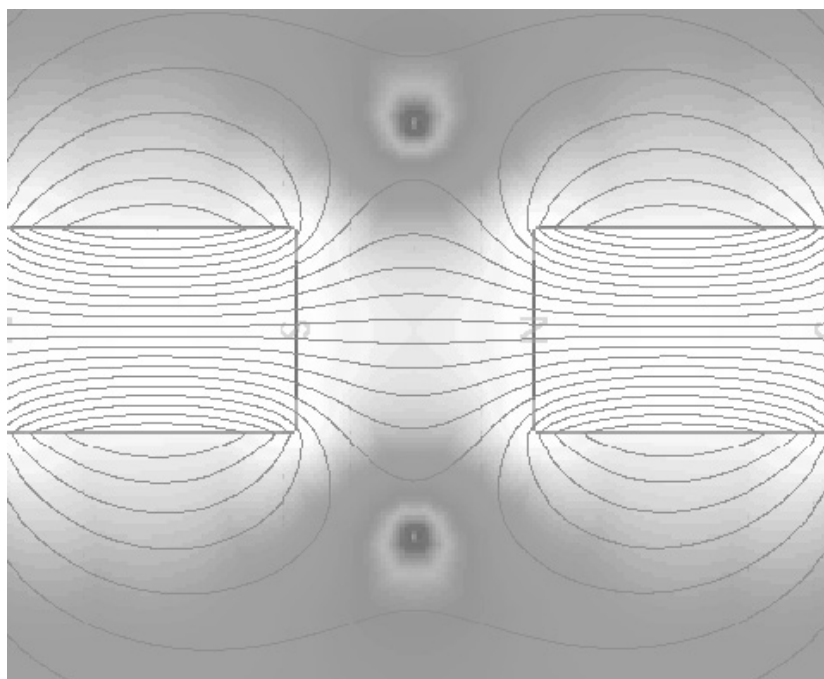
EDIT-CLEAR ALL GUIDELINES, 71
Eigenvalues, 39
Eigenvectors, 39
Electric motor, 12
E-mails, 5
Emf, 17
Error level, 43, 49
Error value, 32
Event initiated procedures, 139
External field, 60, 69
External field source, 60
External fields, 53

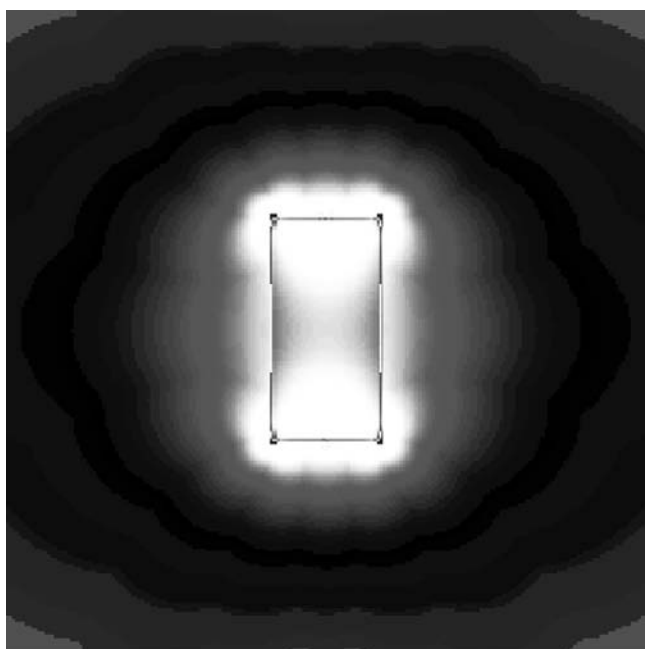
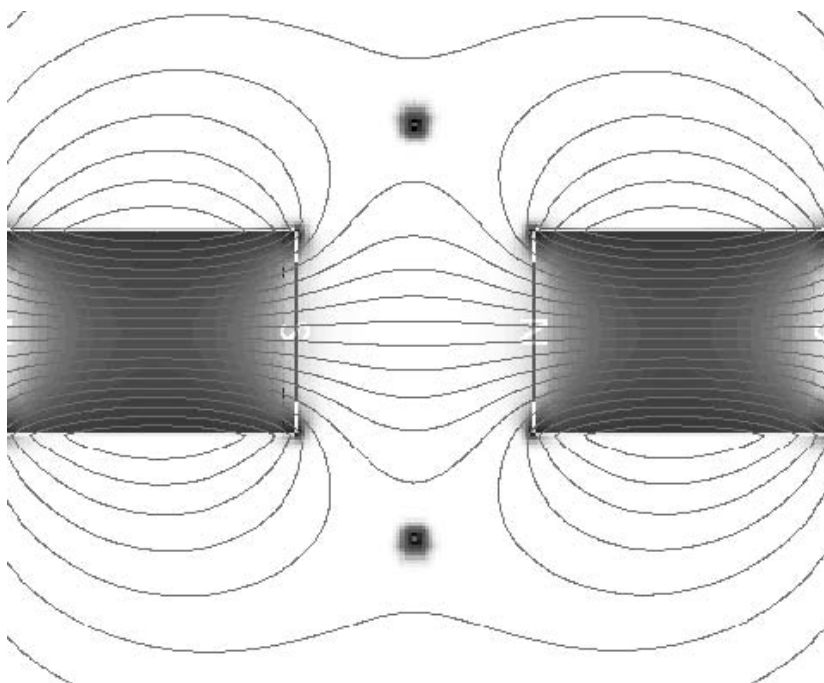
Faraday, 2
Faraday's induction law, 16
Field lines, 7
Fill in, 31
Finite element, 3, 15, 24
Fleming's left hand rule, 12
FLIP LEFT/RIGHT, 71
FLIP TOP/BOTTOM, 71
Floating point round off, 43
Floating point rounding, 33
Flux, 16, 17
Flux density filter, 65
Formats, 76
Forward elimination, 29

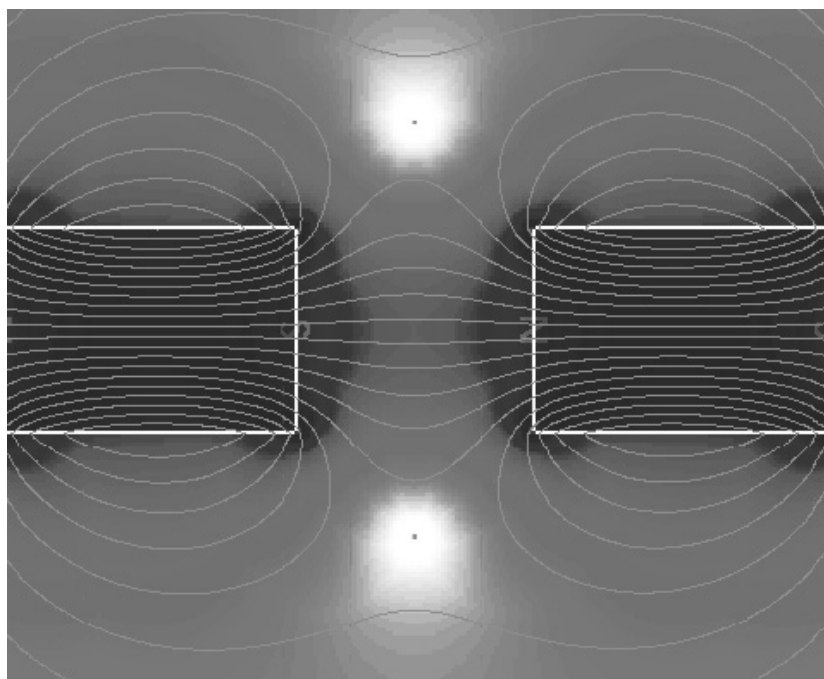
Gauss - Seidel algorithm, 40
Gauss's Law, 9, 16
Gaussian elimination, 29

-
- Gaussian elimination visualisation, 49
 - Generating the equations, 25
 - Gradient, 37, 63
 - Graphs, 46
 - Guidelines, 69
 - h*, 46
 - Hard magnetic material, 14
 - Hints and tips, 76
 - Horizontal resistor mutual, 26
 - Hysteresis, 14
 - i*, 46
 - Indirect addressing, 41
 - Inner product, 32
 - Installation, 73
 - Iron filings, 1
 - Iteration number, 46
 - Kirchoff's Voltage Law, 19
 - l/i*, 33, 46
 - Leakage flux, 17
 - Linear model, 15
 - Load Ref, 47
 - Log scale, 65, 66
 - Long solenoid, 8
 - Loop currents, 21, 24
 - Loop voltages, 21
 - Lucretius, 7
 - Machine code, 31, 33
 - Magnes, 7
 - Magnetic circuit, 16
 - Magnetic field, 7
 - Magnetic field lines, 3
 - Magnetic flux density, 7, 63
 - Magnetic flux sources, 24
 - Magnetic force on a conductor, 12
 - Magnetic iron, 13
 - Magnetic lines of force, 1
 - Magnetic materials, 13
 - Magnetic permeability, 23
 - Magnetic samples, 81
 - Magnetisation curve, 14
 - Magnetite, 7
 - Magnetomotive force, 16, 17
 - Magnitude, 8
 - Matrix multiplication, 33
 - Maxwell's equations, 15
 - Memory size, 29
 - Mesh, 19
 - Mesh loop, 21
 - Mesh self resistance, 26
 - Mesh vertical resistor, 26
 - Mild steel, 13
 - Model functions, 69
 - Monopoles, 7
 - MOVE, 71
 - Multi dimensional solution, 35
 - Mumetal, 13
 - NEWFILE, 71
 - Non-linear, 15
 - North magnetic pole, 7
 - Number of iterations, 49
 - Object storage, 70
 - On-line help, 76
 - Operating systems, 42
 - Options, 48
 - Options panel, 47
 - Pascal, 5
 - Permanent magnet, 69
 - Permanent magnets, 13
 - Permeability, 13
 - Pliny, 7
 - Plot every X iteration, 47
 - Plotall, 57
 - Pointers, 41
 - Positive definite, 37
 - Preconditioned conjugate-gradient, 40
 - Preconditioning, 40
 - Progress indicator, 43
 - Quadratic form, 36
 - RAM, 41
 - Rate of convergence, 38
 - Rectangular magnetic domain, 69
 - Reluctance, 16, 17
 - Remanence, 15
 - Resistance, 17
 - Resistor values, 21
 - Resolution, 76
 - ROTATE, 71
 - Sample files, 48
 - Saturation, 15
 - Save Ref, 47
 - Scalar, 32
 - Secondary matrix, 40
 - Set percentage error, 48
 - Set rounding, 50
 - Set rounding value, 50
 - Set scale, 50
 - Show whole mesh, 47
 - SHOWALG, 31, 33, 45
 - Simultaneous equations, 21, 25
 - Single precision, 33
 - SIZE BOTH, 71
 - SIZE HORIZONTAL, 71
 - SIZE VERTICAL, 71

- Slow scan speed, 50
- Smoothing, 53, 58
- Soft magnetic material, 14
- Solenoid, 69
- Source code, 5, 76
- Sparse matrix, 27
- Spatial resolution, 24
- Spectral condition, 39
- Square matrix, 32
- Steepest descent, 37
- Storage, 30, 31, 40
- SWITCH POLES, 71
- Symmetric, 36
- Symmetry, 78
- Timing, 40
- TOOLS-CLEAR-ALL, 71
- Tyndall, 2
- Unit VIZ2, 130
- Unit VIZ3, 135
- Unit VIZGlobal, 106
- Using VIZIMAG, 73
- Variable size meshes, 24
- Vector, 32
- VIEW-OPTIONS, 75
- Virtual memory, 29
- Virtual storage, 41
- Visualisation, 2, 25
- Visualising the algorithms, 45
- Voltage source, 19
- Wire, 7, 69
- Wire between two poles, 13







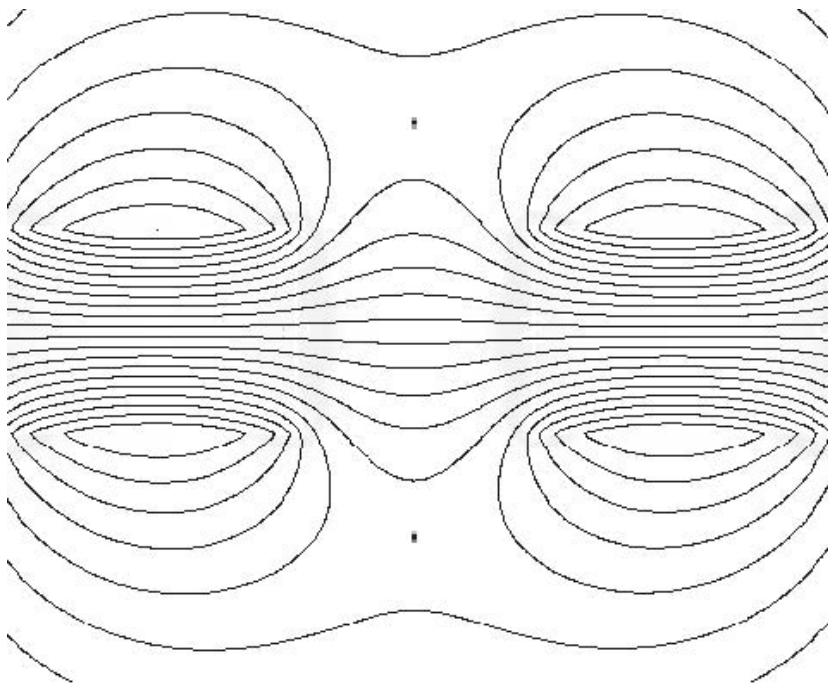


Figure 1-2: Computer generated magnetic field line plot of two bar magnets.

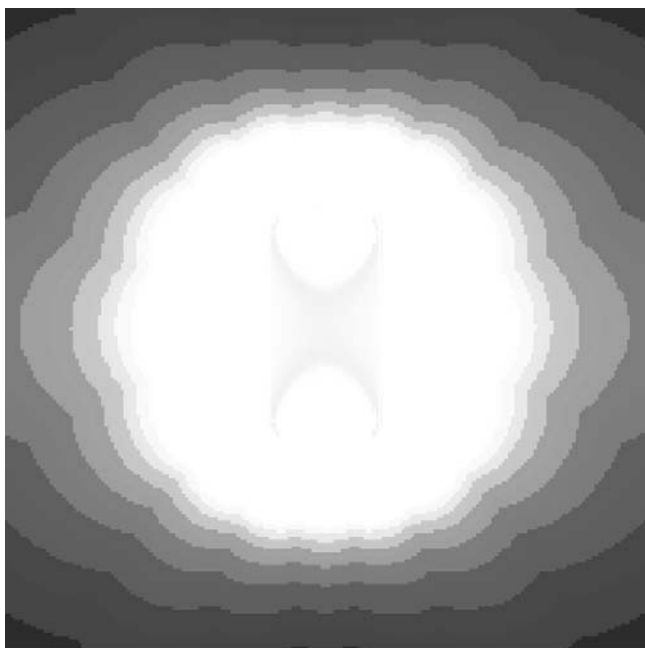
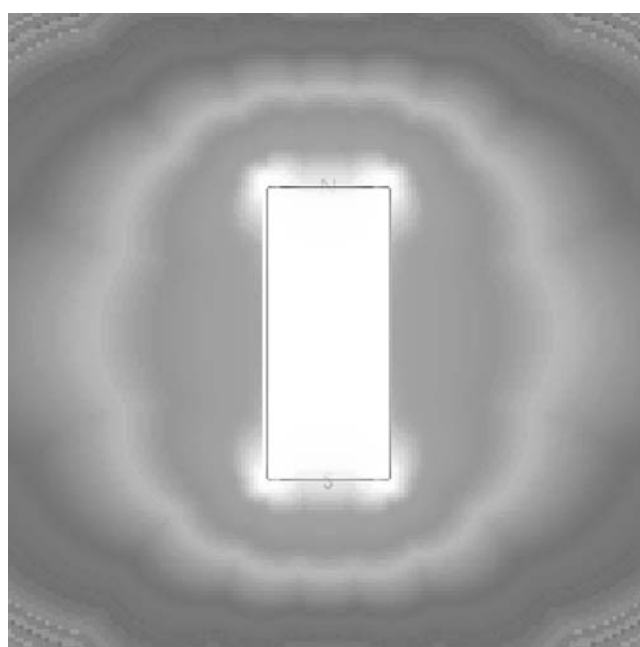
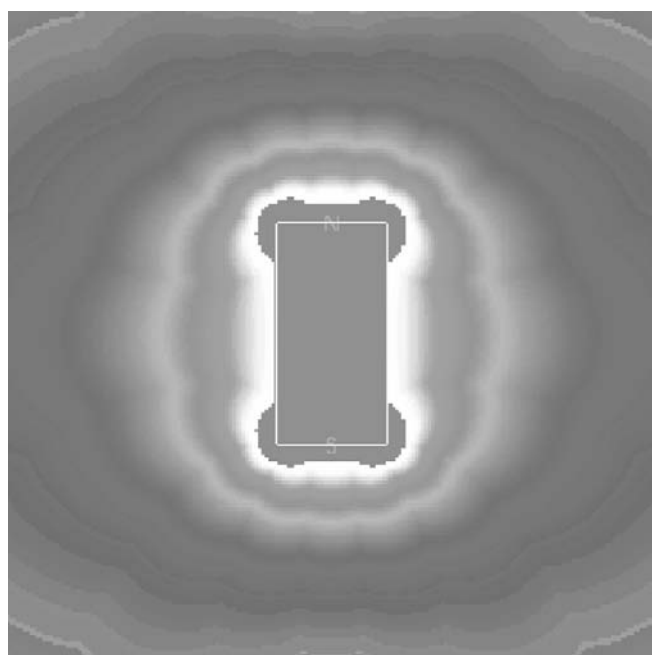
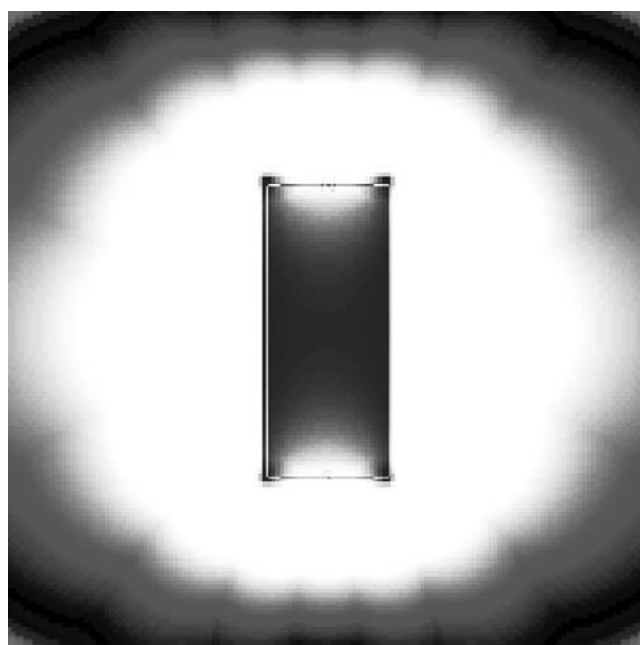
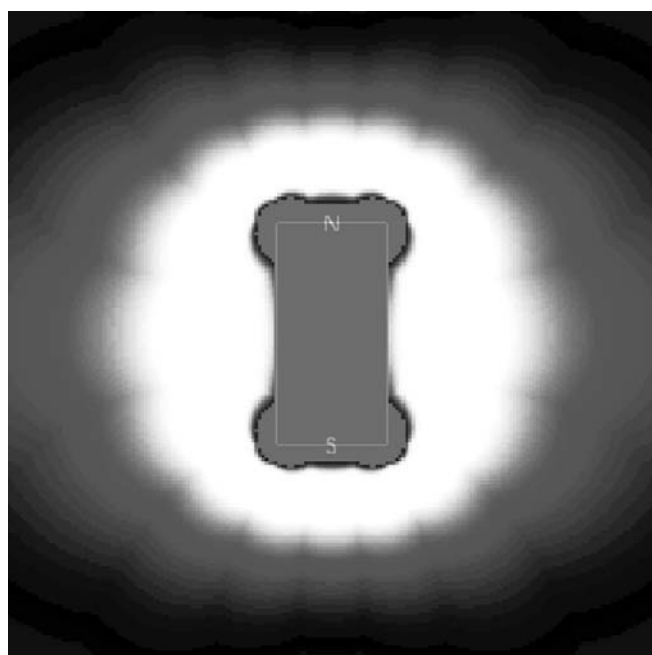
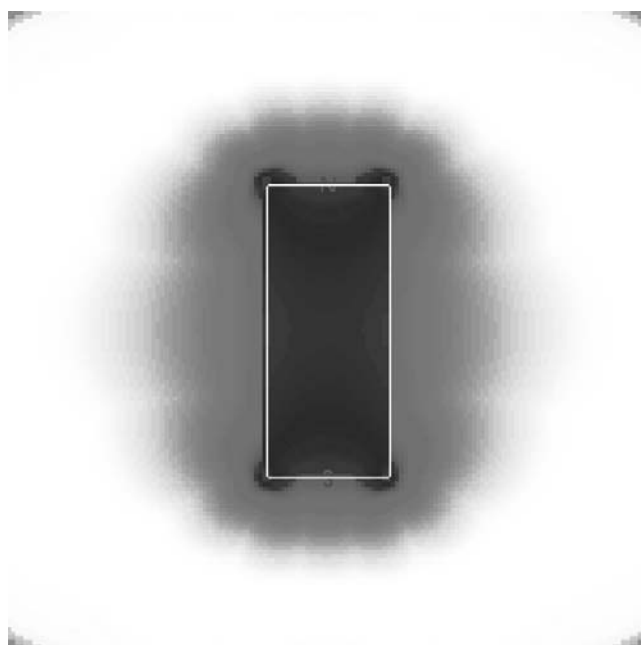


Figure 7-1 (ii): Magnetic flux density function.







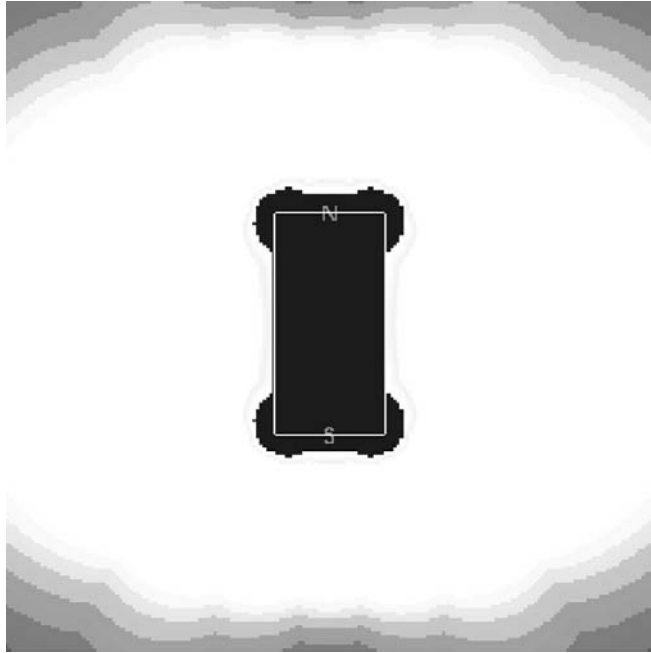


Figure 7-5: Linear plot, with filter.

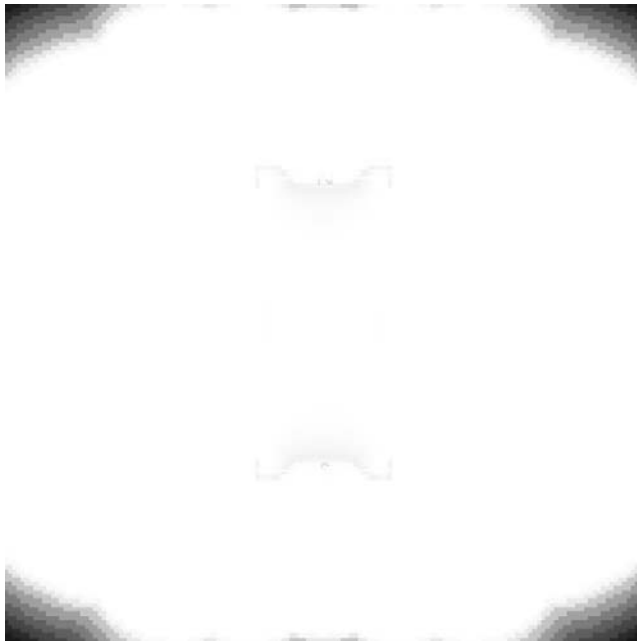


Figure 7-7: Log function plot.

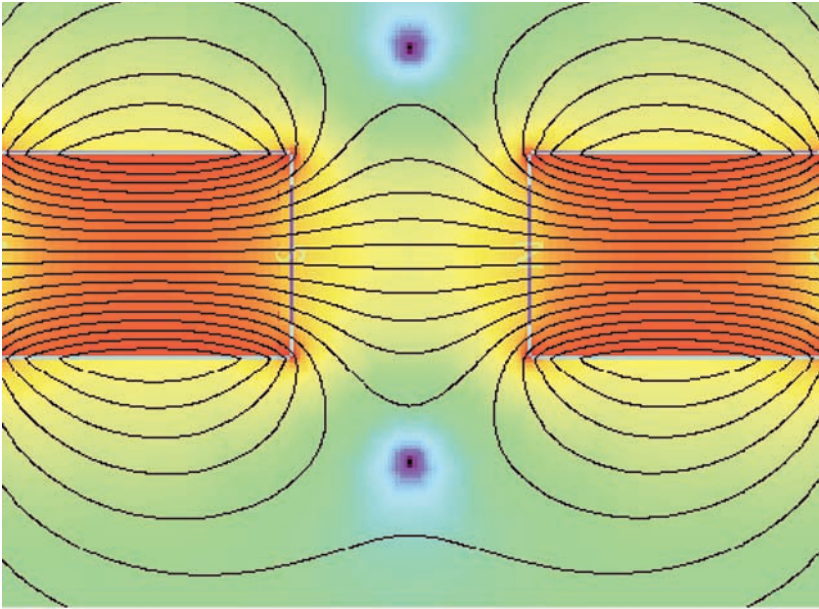


Figure 1-2: Computer generated magnetic field line plot of two bar magnets.

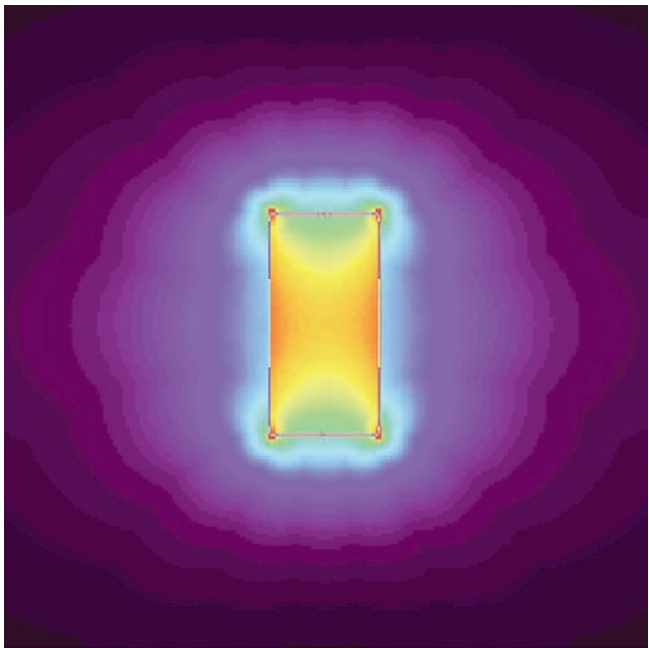


Figure 7-1 (ii): Magnetic flux density function.

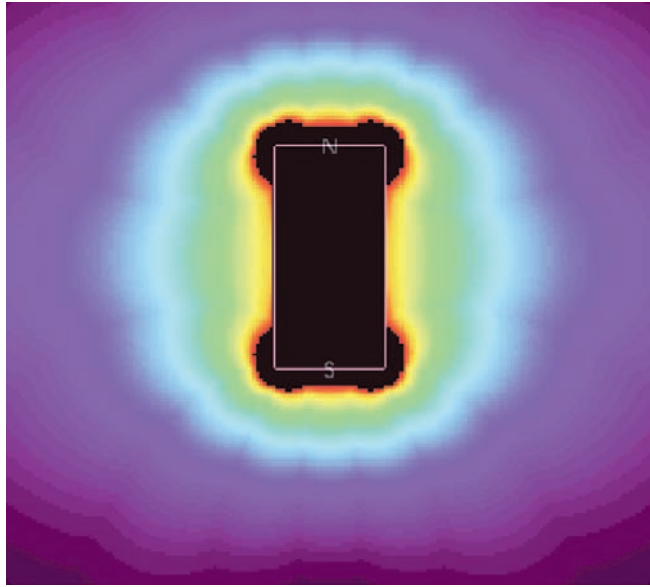


Figure 7-5: Linear plot, with filter.

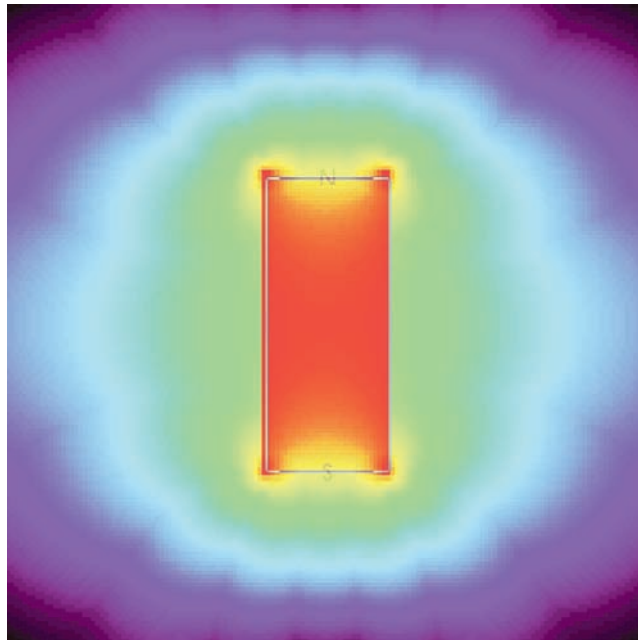


Figure 7-7: Log function plot.