INTERACTIVE MODELS OF ELECTRICAL MACHINES

DAVID DOWNES

A thesis submitted in partial fulfilment of the requirements of the Nottingham Trent University for the degree of Doctor of Philosophy

May 2003

Acknowledgments

My thanks to all those who have helped during the period of research and the duration the thesis has been laboured upon. A particular thank you to Dr. Patterson for his patience and understanding. Thank you also to Prof. Haydock for starting the research and for helpful reviewing of the later work.   Thank you also to the other kind souls with in the EEE section who have assisted. Lastly but not least thank you to my parents for their support over the years.

Abstract

Custom algorithms and stand-alone software have been originated that allow the automatic generation of complex Magnetic Equivalent Circuit (MEC) models from the description of a rotary machines physical geometry and materials. The models are automatically generated in a format that can be utilised in a standard circuit simulator package, such as SPICE, allowing the simulation of the full electrical, magnetic and mechanical interaction.

Fully interactive machine models have been developed by extending Magnetic Equivalent Circuits (MECs) to include mechanical torque by the application of the virtual work principle. Using a common finite element B-H formula, the models are further enhanced by the incorporation of non-linear permeance. These equivalent models use standard circuit components that are available in most circuit solver packages making them transportable and vendor independent.

A novel stand-alone software package has been created that simplifies the generation of the complex MEC models. By describing the machine geometry, flux paths and material characteristics in a simple file an electrical equivalent circuit can be automatically generated at different levels of abstraction. depending on the input information. The software allows the machine's geometry to be viewed and, after simulation, the flux information can be back annotated into the geometry viewer for further design refinement.

The modelling technique has been verified by automatically generating and simulating a MEC model of a 3-phase induction motor; this was checked against practical results obtained from a test-bed consisting of an induction motor and associated load. A further validation of the MEC modelling technique checks for consistency by ascertaining that the energy flow between the electrical, magnetic and mechanical sections balance for a synchronous generator and synchronous motor.

The automatic production of complex Magnetic Equivalent Circuits of rotary machines produces a realistic model giving a high degree of confidence in the simulation results when combined with the power and control electronics in a SPICE type simulator.

Table of contents

Appendices

Table of figures

Glossary

$\mathbf{F}$	vector force
q_n	discrete charge n
$\hat{\mathbf{r}}_{12}$	unit distance vector
ε_0	permittivity of free space
$\mathbf{E}$	electric field strength
ρ	charge density
τ	volume element
$\mathbf{D}$	electric displacement
ε_r	relative permittivity
ε	general permittivity
ϕ	electric potential
V	electric potential difference
A	area
d, l	distance
C	capacitance
U	energy
$\mathbf{j}$	current density
$\mathbf{v}$	velocity
B	flux density
μ_0	permeability of free space
i	current
$\mathbf{dl}$	line element
$\mathbf{ds}$	surface element
$d\tau$	volume element
$\int_S$	surface integral
$\oint_C$	surface contour integral

$\oint$	closed line integral
μ_r	relative permeability
μ	general permeability
Φ	magnetic flux
H	magnetic field strength
emf	electromotive force
n	number of turns
λ	flux linkage
L	inductance
M	mutual inductance
δ	skin depth
ω	angular velocity
mmf	magnetomotive force
Λ	permeance
S	reluctance
$\mathbf{A}$	magnetic vector potential
P	magnetic scalar potential
θ	displacement angle.
C_m	magnetic permeance

1 Introduction

1.1 Machine Models

Traditionally models of electric machines have used the simple electric equivalent circuit, which consists of inductors and resistors, as illustrated in Figure 1 below. This model has been adopted from simple circuit theory and used for power system studies. When used in simulation of complex power electronic systems poor results are obtained, because of the severe limits of the model used for the electrical machine or generator. These models take no account of the interaction between electrical, magnetic circuits and mechanical coupling found in a real machine.

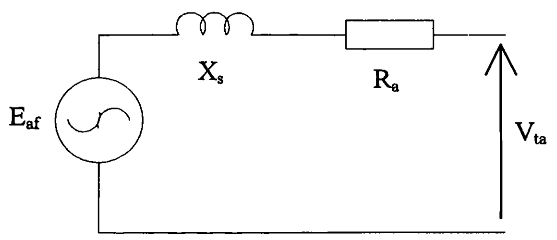


Figure 1: Simple synchronous generator equivalent circuit

The advances in circuit simulators that allow complex systems to be simulated with a high degree of confidence are now common, but these simulators require a more realistic model of the electrical machine. A true representation of the model should take into account the interaction between magnetic coupling, mechanical linkage, saturation and temperature effects. The weak point in all system simulations, that have a machine or generator included, is the limitations of the simple model that is normally used.

1.2 FE Models

At the present moment the only method that is accepted by designers for modelling mechanical, magnetic, saturation and electrical properties is the Finite Element (FE) technique. This has the disadvantage of long simulation times, but its greatest restriction is that it will not integrate with standard commercial circuit simulators. Therefore machine designers can use finite element solvers to find the magnetic characteristics of the machine. The finite element solution can be used to find the parameters for the two-axis model but does not always give realistic solutions when integrated with the electrical control system. An example of a FE mesh and magnetostatic field solution is shown in Figure 2 (Chaudhry et al (2)).

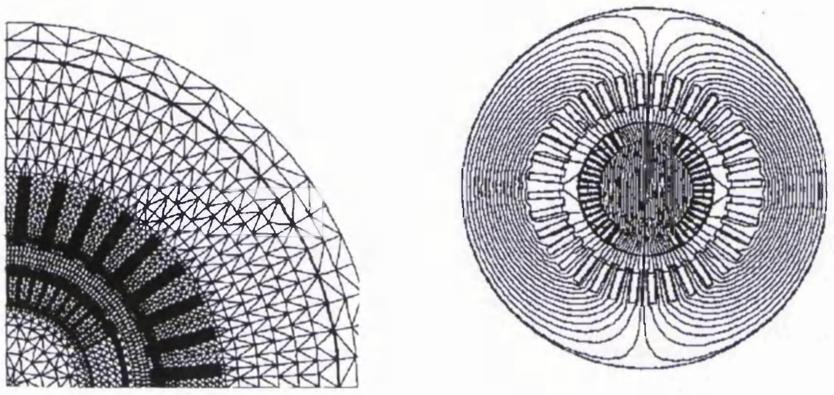


Figure 2: Example finite element mesh and magnetic field solution

1.3 Machine Model Requirements

What is required is a simulation model that can give the detail and characteristics of a finite element solution but will be compatible with standard circuit solvers, such as Simulation Program with Integrated Circuit Emphasis (SPICE) derived analogue circuit solvers. This research project builds on an idea proposed by Carpenter (3) and later developed by Haydock (4), where magnetic circuits can be represented by an electrical equivalent. This allows the coupling of magnetic and electrical circuits to be combined into the one simulation model, which is compatible with standard circuit simulators.

Generating more accurate models, using existing methods, generally incurs significant extra analysis. Simulation of machine control systems requires different levels of model complexity depending on the type of evaluation being carried out. If a new control methodology is to be evaluated then a small simple model, that allows fast simulation, will suffice. If the interaction of magnetic coupling in the machine, power electronic devices and supply is to be investigated, then a complex model of the machine is required.

This project has been concerned with the investigation of machine models, incorporating all pertinent physical properties, which can be used in standard circuit solvers. A modelling methodology has been developed allowing automatic generation of machine models from a textual description of the machine's geometry and materials.

A major objective has been the conception and design of software that allows the automatic production of complex machine models. This software has been designed to:

- produce a visual representation of the machine from a textual description;
- automatically produce a SPICE sub-circuit model of the machine with different levels of complexity dependent on the textual description;
- produce models in a form that can be directly used in a SPICE type simulator.

This flow is shown pictorially in Figure 3, with an example voltage plot typifying the simulation result. The arrows show the sequence of events.

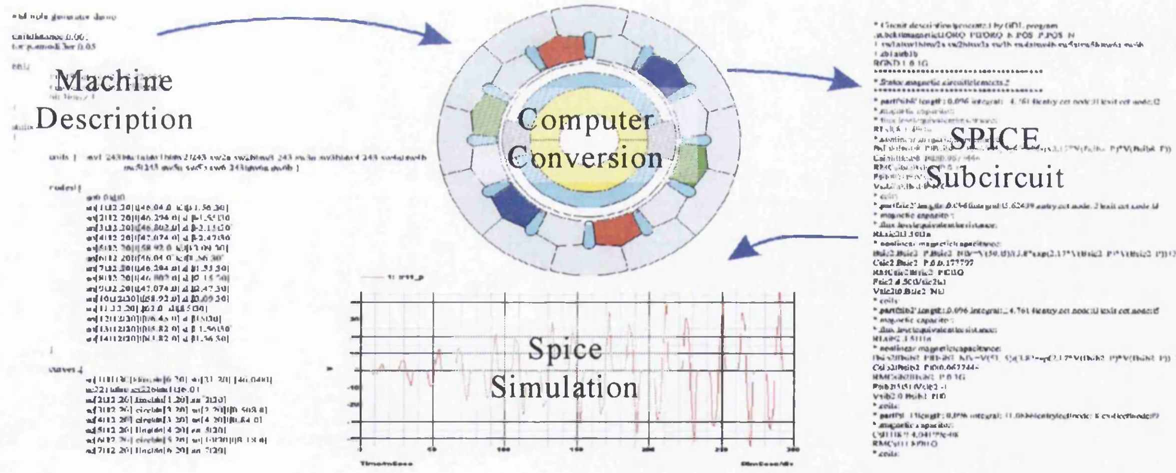


Figure 3: Example of modelling process

As stated the final model produced from the software can have different levels of complexity dependent on the requirements of simulation. A complex model will include the physical, electrical, magnetic and mechanical characteristics of the machine. Increased complexity of the machine model is achieved by increasing the information in the textual description of the machine. One of the main advantages of the software, which produces the models, is that a detailed knowledge of the machines magnetic characteristics is not required. This has an analogy with the development in electronic circuit design, where engineers can design complex integrated circuits with little or no knowledge of the silicon process parameters.

The textual description facilitates easy alteration of the machine design, which combined with the automatic machine model generation allows design changes to be quickly evaluated. The software has been designed to integrate into any standard Electronic Design Automation (EDA) system, facilitating the concurrent engineering philosophy.

1.4 Contribution of the Thesis

The following parts are believed to be original contributions to the advancement of knowledge in this field of study.

- Application of the virtual work principle to the calculation of torque, within standard circuit solver packages.
- Lessening the effect of rounding error within the torque calculation.
- Incorporation of a standard non-linear B-H permeance model, including hysteresis, in a MEC model using standard circuit components.
- Automatic generation of the machines equivalent circuit, from the geometry, flux paths and material characteristics.
- The reinterpretation of the electrical circuit voltages and currents to provide the equivalent magnetic equivalent circuit values. From these values the change in magnetic values (flux, permeance etc.) can be found over time.

1.5 Overview of the Thesis

Chapter 2 gives an overview of some of the basic electromagnetic concepts that are used as the basis for the rest of the thesis. Included is an introduction to electromagnetic quantities and the governing relationships between them. Magnetic equivalent circuits are also introduced along with Finite elements.

Chapter 3 discusses the various methods that in particular are used to model electrical machines. The majority of these are based on lumped element equivalent circuits, the exact details of which are dependent on the type of analysis and machine. Included in this chapter is an example of using a Magnetic Equivalent Circuit (MEC) in the optimisation of a transformer.

Chapter 4 is concerned with the development of the model elements that are used in the simulation of the electrical machine. The models of the electrical machines contained in the thesis are based upon the MECs of Haydock and Carpenter. The model elements are restricted by the requirement that they be compatible with SPICE version 3 based circuit simulation packages.

Chapter 5 gives details of how the models are automatically constructed from a description of the flux paths and machine geometry. A language and syntax for describing the machine is described which provides a means for concisely describing repeating shapes found within the machine geometry. The generation of the equivalent circuit requires the determination of how the equivalent circuit elements are interconnected. The algorithm for determining the interconnection details is described. The software design is also described along with its implementation in C++.

Chapter 6 provides the details of simulation models that were constructed of a synchronous generator, synchronous motor and induction machine. The synchronous machine models demonstrate the transfer of energy from the mechanical to the electrical parts of equivalent circuit models and in the other direction from the electrical to the mechanical parts of the model. An induction motor model is also included which is based on a manufactured motors design and data sheets.

Chapter 7 discusses the models developed and the software implemented to build the machine models with reference to relevant literature. The usefulness of the techniques developed in the thesis, along with limitations, are described. An extension of the technique of building Magnetic Equivalent Circuits based on a generic pattern of elements is also described. It is shown that this has limitations in terms of its general applicability.

Chapter 8 summarises the work presented in the thesis. Suggestions for further future work are also presented in this chapter.

2 General Electromagnetic Models

2.1 Introduction

The aim of this chapter is to present the basic electromagnetic concepts and quantities used in the following sections that are pertinent to the analysis of electrical machines.

Electromagnetics can be described as the study of the interaction of electrical charges (Carpenter (5)). Three different situations can occur depending whether the charges are static, at constant relative velocity or varying relative velocity. The three situations lead to respectively electrostatic, magnetostatic and finally electromagnetic analysis.

It is possible to derive the magnetostatic equations by applying Einstein's special theory of relativity to the electrostatic equations. Thus it could be said that the whole of electromagnetism depends on only one equation, that being coulombs law (Dobbs E.R. (6) ch. 8, this book is also used as reference for the rest of the chapter, Carpenter (1) also gives a very good introduction).

At the end of this chapter, the Magnetic Equivalent Circuit (MEC) and Bond Graphs are introduced. The MEC is the basis for the representation of the magnetic circuit and its interaction with both the electrical and mechanical systems.

2.2 Elementary electrostatics

The electrostatics situation can be described by considering two charges at rest, and the force developed between them in vacuum.

Coulombs law states that the force between two charges is given by:

$$\mathbf{F} = \frac{q_1 q_2 \hat{\mathbf{r}}_{12}}{4\pi\varepsilon_0 r_{12}^2} \text{ (N)} \qquad (1)$$

Where q_1 and q_2 are the charges, r_{12} is the distance between them and ε_0 is the permittivity of free space. When the flux of the electric field is plotted for positive and negative charge, the flux starts on one charge and ends on the other i.e. a bi-polar field. Converting Eq. 1 to the force on a hypothetical test charge of one coulomb and infinitesimal size:

Electric field $$\mathbf{E} = \frac{q\hat{\mathbf{r}}_{12}}{4\pi\varepsilon_0 r_{12}^2} \text{ (V/m)} \qquad (2)$$

Generalising and applying Gauss's law.

$$\int_S \mathbf{E}\cdot d\mathbf{s} = \frac{1}{\varepsilon_0}\int_V \rho\, d\tau \qquad (3)$$

If the charges have no relative velocity (magnitude is constant due to the law of conservation of charge) then another definition of an electrostatic situation is that the change in electrostatic field strength with time is zero i.e. $\partial\mathbf{E}/\partial t = 0$.

Often in calculations a simplification can be made by introducing a new variable.

Electric displacement $$\mathbf{D} = \varepsilon_r\varepsilon_0\mathbf{E} \text{ (FVm}^{-2}\text{)} \qquad (4)$$

The relative permeability ε_r relates the value of permittivity in a vacuum to that found in non-vacuum. The materials described by the constant are assumed isotropic, homogeneous and linear.

As the electric field is conservative ∴ $$\oint_c \mathbf{E}\cdot d\mathbf{l} = 0 \quad (5)$$

Returning to the idea of a test charge, move the charge from a great distance towards the principle charge. The integral of force with distance is the energy expended in moving the test charge to the resting position. This is the potential and can be described by a scalar quantity.

$$\phi(r) = -\int_\infty^r \mathbf{E}.\cdot d\mathbf{l} = \frac{-q}{4\pi\varepsilon_0 r} \text{ (V)} \qquad (6)$$

Generalising Eq. 6 $$\mathbf{E} = -grad\phi \qquad (7)$$

Consider a simple parallel plate capacitor with a charge on one plate of +Q, potential ϕ_+ and the other –Q, potential ϕ_-. Because the plates are equipotential surfaces, the potential difference V = $\phi_+ - \phi_-$. From Eq. 3 and Eq. 7

$$V = \left(\frac{d}{\varepsilon_0 A}\right)Q \quad \therefore V \propto Q \qquad (8)$$

Capacitance is defined as: $C = \dfrac{Q}{V}$ (9)

The energy stored in the capacitor can be derived as:

$$U = \frac{1}{2}CV^2 \text{ (J)} \qquad (10)$$

As the energy stored in the capacitor is due to the electric field, an attractive force exists between the two plates. Using the virtual work principle

Change in energy $\Delta U = -F \cdot \Delta x \quad \therefore F = \dfrac{1}{2}\varepsilon_0 E^2 A$ (N) (11)

It is possible to extend the idea of stored energy to the electric field and equipotentials.

Energy stored in field $U = \dfrac{1}{2}\varepsilon_0 \displaystyle\int_\tau \mathbf{D} \cdot \mathbf{E} d\tau$ (J) (12)

2.3 Elementary magnetostatics

This situation is when $\partial\mathbf{J}/\partial t = 0$ and $\partial\mathbf{B}/\partial t = 0$. A charge moving in a postulated magnetic field of flux density B is found to experience a force F and is defined as:

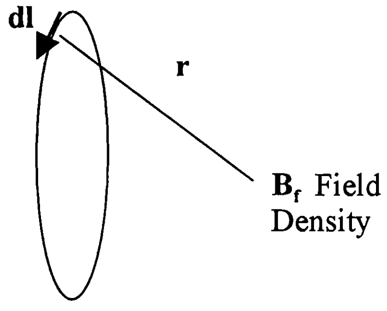
Lorentz force $\mathbf{F} \propto q\mathbf{v} \times \mathbf{B}$ (N) (13)

By use of an equivalent to electrostatics Eq. 1, the field density at a particular point in space can be related to the surrounding current densities. The physical constant that emerges is the permeability of free space.

Biot-Savart law $\mathbf{B}_f = \dfrac{\mu_0}{4\pi} \displaystyle\int_{\substack{\text{all}\\ \text{space}}} \frac{\mathbf{J}_s \times \hat{\mathbf{r}}_{12}}{r_{12}^2} d\tau$ (Tesla) (14)

For 'thin' wires $\mathbf{J}d\tau = \mathbf{J}Al = i\mathbf{dl}$ (15)

$\therefore$ $\mathbf{B}_\mathbf{f} = \dfrac{\mu_0}{4\pi} \displaystyle\oint \frac{i\mathbf{dl} \times \hat{\mathbf{r}}_{12}}{r_{12}^2} d\tau$ (16)



Current Loop

Figure 4: Calculation of field density at point $\mathbf{B}_f$ due to a current loop

By using Eq. 16 the magnetic field density at a point can be calculated (Figure 4). By applying this to the field surrounding a wire, then integrating the resulting equation over a closed path for the field density **B** results in Ampere's law:

Ampere's law $$\oint_C \mathbf{B} \cdot \mathbf{dl} = \mu_0 i \qquad (17)$$

Generalising $$\oint_C \mathbf{B} \cdot \mathbf{dl} = \mu_0 \int_S \mathbf{J} \cdot \mathbf{ds} \qquad (18)$$

Unlike the electrostatic field, there are no monopole sources and therefore the field is solenoidal.

i.e. closed surface flux $$\Phi = \int_S \mathbf{B} \cdot \mathbf{ds} = 0 \text{ (Weber)} \qquad (19)$$

The magnetic field is assumed to be in vacuum. To allow for the effect of situations other than vacuum the relative permeability is defined.

In the general case (isotropic materials) $\mu = \mu_r \mu_0$ ($\mathrm{Hm^{-1}}$) (20)

Similar to electrostatics a further auxiliary variable can be defined:

Field intensity (for isotropic materials) $$\mathbf{H} = \frac{\mathbf{B}}{\mu} \text{ (Am}^{-1}\text{)} \qquad (21)$$

There are three broad classes of materials in terms of magnetic relative permeability. Paramagnetic and diamagnetic substances have very small relative effect on the magnetic field. In most circumstances, the permeability can be assumed to be that of vacuum.

Ferromagnetic materials have a large relative permeability, which is the reason for their extensive use in electrical machines. The value of relative permeability is dependent on the value of flux density. When the flux density is sufficiently high, non-linear permeability results. The physical cause is an alignment of all the magnetic domains, with the external field, and hence saturation. The act of domain alignment is often referred to as magnetisation. Another consequence of the physical cause is the hysteresis effect, where a loss of energy is incurred due to energy needed for physical alignment of magnetic domains. When a flux density verses field intensity plot is made, different ferromagnetics show different initial slopes, saturation regions and hysteresis areas. The multi-valued nature of the characteristic is due to 'memory' of its previous state. This residual magnetism is the basis of permanent magnets.

2.4 Electromagnetism

The following relationship, Faraday's induction law, links the change in magnetic flux to the induced emf in a coupled electrical circuit.

Since *n* is constant $$emf = -n\frac{d\Phi}{dt} \text{ (V)} \qquad (22)$$

A useful definition often encountered is 'flux linkage': $\lambda = n\Phi$ (Weber) (23)

Eq. 23 is defined as the flux that links with a coil multiplied by the coil turns.

$\therefore$ $$emf = \frac{d\lambda}{dt} \qquad (24)$$

Inductance L is a measure of the effect upon the electrical terminals of a magnetic field that is set up by the current *i* flowing through the coil.

Self inductance $$L = n\frac{\Phi}{i} = \frac{\lambda}{i} \text{ (H)} \qquad (25)$$

If the field produced by one coil 'links' with another coil then the mutual inductance can be defined as:

$$M_{12} = \frac{\lambda_1}{i_2} \text{ (H)} \qquad (26)$$

The inductance is dependent upon the permeability of the surrounding magnetic material. If the material does not have a linear permeability (which is usually the case for machines operated at or near saturation) then the inductance is also non-linear.

Energy stored in a simple inductor can be related to the current through it and the emf developed across it (similar to electrostatic Eq. 10 for charge and voltage):

Energy stored in a coil is given by:

$$U = \frac{1}{2} L i^2 \text{ (J)} \qquad (27)$$

Generalising to the energy stored in the whole field:

$$U = \frac{1}{2} \int_{\tau} \mathbf{B} \cdot \mathbf{H} d\tau \text{ (J)} \qquad (28)$$

An important effect is that any conductor placed within a time varying magnetic field experiences an emf.

For example, in a solid copper wire a sinusoidal time varying current produces a time varying magnetic field. This induces an emf and therefore current within the conductor. The effect is to confine most of the current to the surface of the conductor. The skin depth is a measure of how far the current penetrates into the conductor. It is defined as the depth when the sinewave current drops to approx. 0.3679 of its peak value, and is frequency dependent.

Skin depth (sinewave frequency ϖ) $\delta = \sqrt{\dfrac{2}{\omega \mu \sigma}}$ (29)

A linked effect due to the time varying induced current is the reduction of the magnetic flux density at the centre of the wire. For a given average flux density the peak density is higher at the periphery, and so possibly leading to localised saturation (Mohan et al (7) pp.748) in laminations.

Further to the previous discussion of ferromagnetic materials, the non-linear DC magnetisation B/H curve plots the peak flux density verses cyclic flux intensity (neglecting hysteresis). Linearisation of the curve about an operating point, for small perturbations, leads to the differential permeability:

$$\mu_r = \frac{dB}{(\mu_0 dH)} \qquad (30)$$

For cyclic perturbations about an operating point, there is an incremental hysteresis; this particular feature of the magnetic response is often ignored. Another characteristic of real ferromagnetic materials is a preferential direction of magnetisation, this again is often ignored and the material assumed to be isotropic.

Manufactures of electrical steels often provide the following information about the materials magnetic properties (Fitzgerald et al (8)):

The DC magnetisation curve for the material is given as a plot of flux density verses magnetisation current, and is single valued monotonic.

Loss within the material is often given in terms of a curve of peak flux density verses watts per kilogram for a specific frequency and thickness of lamination.

A materials magnetisation characteristic is often given as a curve of peak flux density verses rms volt-amperes per kilogram at specified frequency and lamination thickness.

Simulation difficulties can arise if use is made of the real multi-valued hysteresis curve. It is therefore usual to neglect the hysteresis losses and approximate the magnetic response by a single valued monotonic function.

A related problem is also cross saturation, see Vagati et. al. (39). This is where a field component that links certain coils interacts with the field that links other coils, by changing the value of permeance of material that is common in the flux path of both.

2.5 A brief introduction to Maxwell's equations

The previous electromagnetic field equations can be reformulated in vector calculus differential and integral form (Eq. 31-Eq. 38). Maxwell added a displacement current,, to Faraday's law to correct for anomalies in the analysis (Eq. 37 & Eq. 38)(Dobbs (6) pp.104) of a capacitors time varying magnetic field.

Differential		Integral	
$\operatorname{div} \mathbf{E} = \rho/\varepsilon_0$	(31)	$\int_S \mathbf{E}\cdot d\mathbf{s} = \frac{1}{\varepsilon_0}\int_V \rho \, d\tau$	(32)
$\operatorname{div} \mathbf{B} = 0$	(33)	$\int_S \mathbf{B}\cdot d\mathbf{s} = 0$	(34)
$\operatorname{curl} \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial \mathrm{t}}$	(35)	$\oint_C \mathbf{E}\cdot d\mathbf{l} = -\int_S \frac{\partial \mathbf{B}}{\partial t}\cdot d\mathbf{s}$	(36)
$\operatorname{curl} \mathbf{B} = \mu_0\left(\mathbf{J} + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t}\right)$	(37)	$\oint_C \mathbf{B}\cdot d\mathbf{l} = \mu_0\left\{\int_S \mathbf{J}\cdot d\mathbf{s} + \int_S \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t}\cdot d\mathbf{s}\right\}$	(38)

The electromagnetic force law that becomes Coulomb and Lorentz forces is:

$$\mathbf{F} = q\mathbf{E} + q\mathbf{v}\times\mathbf{B} \qquad (39)$$

One of the assumptions made by nearly all models of electrical machines, is that a psuedo-magnetostatic analysis can be applied. This is usually valid because of the low frequencies that are predominant. For psuedo-magnetostatic situations the displacement term in Eq. 37 is negligible compared to that of current density.

$$\therefore \ \operatorname{curl} \mathbf{B} = \mu_0 \mathbf{J} \qquad (40)$$

By introducing field intensity $\mathbf{H} = \frac{\mathbf{B}}{\mu_0}$ (41)

From Eq. 40 and Eq. 41 $\operatorname{curl} \mathbf{H} = \mathbf{J}$ (42)

Given the two-dimensional case, and using the integral form of Eq. 40, also assuming that the surface integral of the current density **j** equals the number of turns n multiplied by the current i (in the direction of current density), the following results.

$$\oint H dl = ni \qquad (43)$$

2.6 Magnetic circuits

From $\Phi = \int_S \mathbf{B}\cdot d\mathbf{l}$ (44)

In the two dimensional case and using Eq. 20, Eq. 41, Eq. 43 and Eq. 44

$$\oint \frac{\Phi}{\mu(l)A(l)} dl = ni \qquad (45)$$

The permeability in most models is usually taken to be homogenous within the region of integration, therefore simplifying the integral. If it is not homogenous then the region of interest is split into several sub-regions.

Given the flux is assumed constant along the path of integration and defining Magneto-Motive Force (mmf):

$$mmf = S\Phi = \frac{\Phi}{\Lambda} \text{ (A)} \qquad (46)$$

$$\therefore \; mmf = ni \qquad (47)$$

From Eq. 45 $$S = \frac{1}{\Lambda} = \oint \frac{dl}{\mu(l)A(l)} \text{ (H}^{-1}\text{)} \qquad (48)$$

See Wilson et al (45) for another approach to the problem.

2.7 Reluctance calculations

Because of the relative permeability of the iron parts used in the construction of a machine, it is possible to make certain assumptions about the general flux paths. Assumptions made are that: the distributed field flux is contained with in the flux path and the flux is constant through out the path. By using Eq. 48 it is possible to find the flux where the mmf is known. If all the parameters are changed to peak or rms values then, provided the permeability and flux paths are constant, the calculation is valid for steady state alternating current. In the steady state it is possible to replace the differential operator in Eq. 22 by its complex phasor equivalent. This 'coarse lumping' of the magnetic circuit can be illustrated by the inductor and transformer examples given in Figure 5 and Figure 6 respectively.

2.7.1 Inductor example

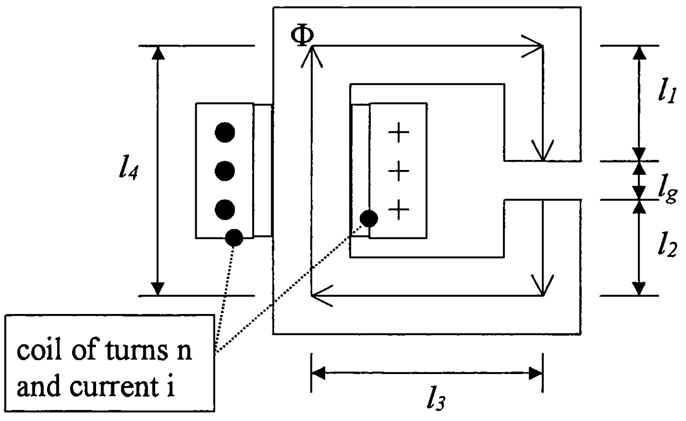


Figure 5: Section through a simple inductor example

If the flux is assumed to be totally contained with in the iron apart from when it flows in the air gap, the reluctance (S) can be calculated from:

Where A_c = area of core $S = \dfrac{l_1 + l_2 + 2l_3 + l_4}{A_c \mu_r \mu_0} + \dfrac{l_g}{A_g \mu_0}$ (49)

It can be seen that if $\mu_0\mu_r >> \mu_0$ then the total reluctance will be largely dominated by the reluctance of the air-gap.

Substituting Eq. 46 into Eq. 22 $emf = -\dfrac{nd\left(\dfrac{ni}{S}\right)}{dt}$ (50)

Assuming reluctance is constant:

$$emf = -\frac{n^2}{S}\frac{di}{dt} \qquad (51)$$

Using Eq. 23 gives the flux linkage as:

$$\lambda = \frac{n^2 i}{S} \qquad (52)$$

And the inductance is given from Eq. 25 :

$$L = \frac{n^2}{S} \qquad (53)$$

Also from Eq. 49 it can be seen that most of the reluctance will appear across the air-gap. The energy stored in the field has been shown to be proportional to the inductance (Eq. 27). As the inductance is largely determined by the air-gap (Eq.

53), it follows that most of the energy stored in the field will be in the air-gap. The accuracy of determining the inductance is therefore dependent on the accuracy of calculating the magnetic field in the air-gap.

An important aspect of the above calculations, is that they approximate a continuous distributed field by one that is confined to the iron. For the air-gap various empirical factors are available to correct for the field 'fringing' e.g. Carter "airgap extension factors" (9). The error very much depends on the length of the air-gap, from one iron surface to the other, verses the geometry of the air-gap area. A correction factor often used is to add the numerical value of the air-gap length to the dimensions that make up the air-gap area.

2.7.2 Transformer example

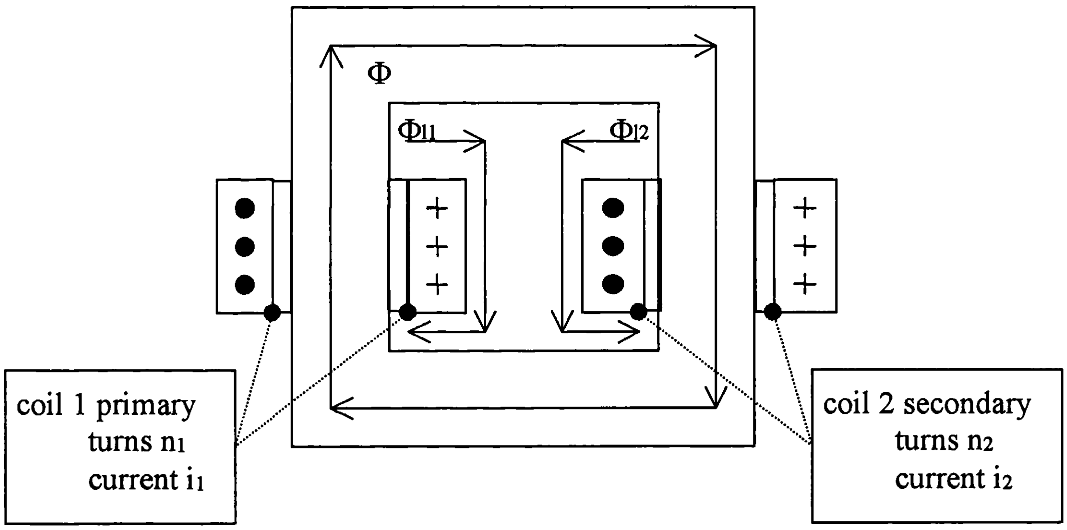

Φ
Φ_{l1}
Φ_{l2}
coil 1 primary turns n_1 current i_1
coil 2 secondary turns n_2 current i_2

Figure 6: Transformer example

The transformer is shown in Figure 6. Unlike the previous inductor it does not incorporate an air gap. The flux paths include a component of leakage shown as Φ_{l1} and Φ_{l2}. The leakage for coil one is that flux that does not link with coil two, the same is true for coil two. In some analysis an approximate value is obtained by calculating the mean path and hence reluctance through and around the coils.

In the following equations Φ is the mutual flux which links coil 1 to coil 2 after subtracting the opposing mmf of each coil. The reluctance (S) is for the flux path that links both coil one and coil two. The leakage reluctance's S_{l1} and S_{l2} are for the leakage flux paths taken by Φ_{l1} and Φ_{l1} respectively.

The primary coil emf (E_1) can be calculated from Eq. 22 & Eq. 46:

$$E_1 = -n_1 \frac{d}{dt}(\Phi_{l1} + \Phi) = -n_1 \frac{d}{dt}\left(\frac{n_1 i_1}{S_{l1}} + \frac{n_1 i_1}{S} - \frac{n_2 i_2}{S} \right) \qquad (54)$$

Therefore assuming n₁, n₂, S_{l1} & S constant:

$$E_1 = \frac{n_1^2}{S}\left(\frac{n_2}{n_1}\frac{di_2}{dt} - \frac{di_1}{dt} \right) + \frac{n_1^2}{S_{l1}}\frac{di_1}{dt} \qquad (55)$$

Applying Eq. 53 to each of the flux paths for both primary and secondary:

$$L_{l1} = \frac{n_1^2}{S_{l1}} ..(a),\ L_{l2} = \frac{n_2^2}{S_{l2}} ..(b),\ L_1 = \frac{n_1^2}{S} ..(c),\ L_2 = \frac{n_2^2}{S} ..(d) \qquad (56)$$

Substituting Eq. 56a and Eq. 56c into Eq. 55 gives:

$$E_1 = L_1\left(\frac{n_2}{n_1}\frac{di_2}{dt} - \frac{di_1}{dt} \right) + L_{l1}\frac{di_1}{dt} \qquad (57)$$

Rearranging Eq. 57 $\frac{di_1}{dt} = \left(\frac{L_1}{L_1 - L_{l1}} \right)\frac{n_2}{n_1}\frac{di_2}{dt} - \frac{E_1}{L_1 - L_{l1}}$ (58)

In Eq. 58 if L_{l1} is insignificant and the last term (magnetising current) is neglected, then the resulting equation is that of the ideal transformer (Eq. 59).

$$i_1 = \frac{n_2}{n_1} i_2 \qquad (59)$$

The magnetising current is supplied by the primary winding, it is the current needed to establish the magnetic field thus enabling power transfer. Extra current in the primary taken from the supply balances the current that flows in the secondary when a load is applied. For maximum power transfer this extra current must be in phase with the voltage.

Substituting Eq. 56c into Eq. 56d gives:

$$L_2 = \frac{n_2^2 L_1}{n_1^2} \qquad (60)$$

By substituting in the secondary variables into Eq. 54 gives:

$$\dot{E}_2 = -n_2 \frac{d}{dt}(\Phi_{l2} - \Phi) = -n_2 \frac{d}{dt}\left(\frac{n_2 i_2}{S_{l2}} + \frac{n_1 i_1}{S} - \frac{n_2 i_2}{S}\right) \tag{61}$$

Substituting Eq. 56b, Eq. 56d and Eq. 60 into Eq. 61 gives:

$$E_2 = L_1 \frac{n_2^2}{n_1^2}\left(\frac{n_1}{n_2}\frac{di_1}{dt} - \frac{di_2}{dt}\right) + L_{l2}\frac{di_2}{dt} \tag{62}$$

2.8 An introduction to electromagnetic potentials

In both electrostatic and magnetostatic analysis the potential is continuous at material boundaries, unlike the field quantities E, D and B, H respectively.

The electric potential has already been defined in Eq. 7 as a conservative scalar field.

From Eq. 7 & Eq. 31 $$\nabla^2 \phi = \frac{\rho}{\varepsilon_0} \tag{63}$$

A formulation of magnetic potential, known as scalar magnetic potential is possible (Hoole (9) pp.124).

The magnetic scalar potential *P* is defined such that the field strength is the gradient of the scalar potential:

$$\mathbf{H} = -\nabla P \tag{64}$$

The above is only valid for current source free magnetic fields. This is readily verified as the curl of a gradient is zero, which applied to Eq. 42 means the current density must be zero. The main advantage of scalar magnetic potential compared to that of vector potential is for 3 dimensional problems where the number field equations are reduced by a factor of 3.

One application of the scalar potential is in the boundary element method. The field is split into two components, one with source components and one with no sources i.e. in free space. The field is solved in free space, and the solution related to that with sources present.

The scalar magnetic potential can be related to the mmf of a magnetic circuit. For the line integral of mmf that starts at A and ends at B:

$$\Delta mmf = \int_A^B \mathbf{H} \cdot \mathbf{dl} = \int_A^B -\nabla P \cdot \mathbf{dl} = -\int_{P_A}^{P_B} dP = P_A - P_B \qquad (65)$$

Both the electrostatic and magnetostatic potentials above are scalar fields and therefore the boundary conditions are similar in both cases. Two principle types of boundary condition can be applied (see Figure 7):

Dirichlet (flux leaves normal to the surface) $\phi = \text{constant}$ (66)

Neumann (flux lies parallel to the surface) $\frac{\partial \phi}{\partial n} = 0$ (67)

In order to solve the above Poisson equation (Eq. 63) derived from Eq. 7 & Eq. 31 , it is necessary that at least one Dirichlet boundary condition be applied.

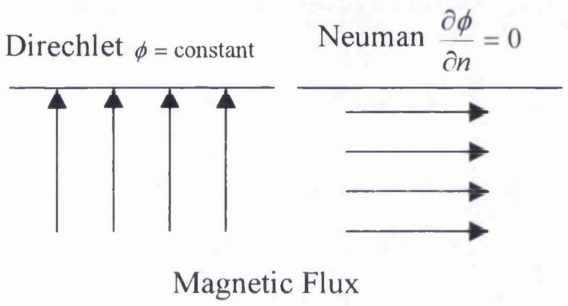

Direchlet ϕ = constant
Neuman $\frac{\partial \phi}{\partial n} = 0$
Magnetic Flux

Figure 7: Illustration of the effect of Direchlet and Neuman boundary conditions.

The magnetic vector potential is defined as:

$$\mathbf{B} = \text{curl } \mathbf{A} \qquad (68)$$

See Figure 8 for a diagram of the current through a coil and the resulting vector potential.

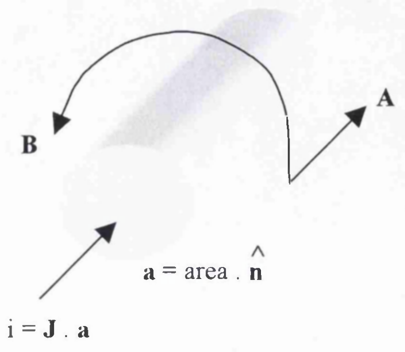


Figure 8: Relationship between A and B

From Eq. 40 & Eq. 68 the following Poisson equation results

$curl\frac{1}{\mu\mu_r}\cdot curl\mathbf{A} = \mathbf{J}$ or in the non-general case $\nabla^2\mathbf{A} = \mu\mu_r\,\mathbf{J}$ (69)

A vector quantity such as **A** is only fully defined if both curl and div are stated. The particular expression used for div **A** being known as the 'gauge'. Common choices for the 'gauge' being:

Coulomb gauge $\qquad \operatorname{div}\mathbf{A} = 0 \qquad$ (70)

Lorentz gauge $\qquad \operatorname{div}\mathbf{A} = -\mu\mu_r\varepsilon\frac{\partial\phi}{\partial t} \qquad$ (71)

In two dimensions, the vector magnetic potential reduces to a scalar that can be solved in much the same way as for a scalar electrostatic potential.

$$\nabla^2 A = \mu\mu_r\,J \qquad (72)$$

From Eq. 72 it can be shown for equipotential surfaces (Figure 9):

$$\Phi_{12} = A_2 - A_1 \qquad (73)$$

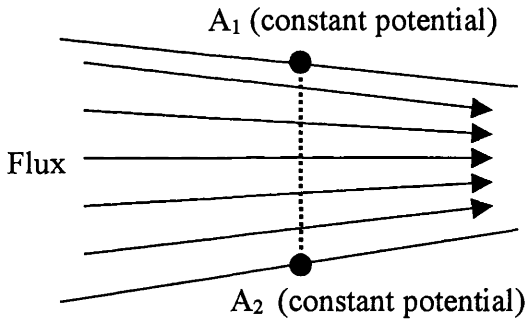


Figure 9: equipotential surfaces

See Figure 10 for the relationship between inductance and the vector potential of a coil.

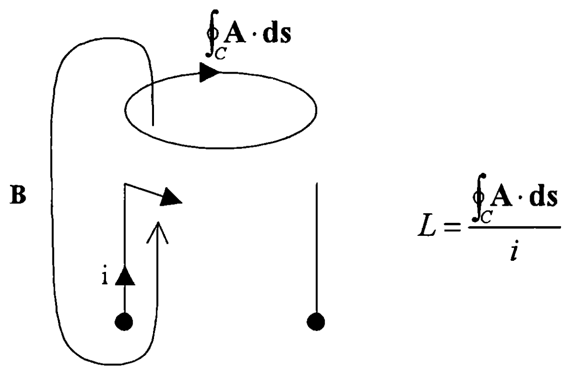

$\oint_C \mathbf{A}\cdot d\mathbf{s}$
B
i
$L = \frac{\oint_C \mathbf{A}\cdot d\mathbf{s}}{i}$

Figure 10: Calculating inductance of a coil from the vector potential

Carpenter (53) has proposed that electromagnetics be taught using the magnetic and electric potentials, instead of using the concept of flux. According to this view of magneto-static phenomena the inductance of a circuit can be interpreted as 'inertia' of the charges.

In contrast, Hammond et al (29) propose a 'duel energy' approach to solving electromagnetic field problems.

2.9 Finite difference

Finite difference, strictly a subtopic of finite elements, directly substitutes the derivatives of the field equations with difference equivalents. The resulting equation is a direct numerical solution of the field equations, where each field point is related to the four nearest itself as shown in Figure 11. An example of the finite difference method as applied to the two-dimensional Poisson equation of vector potential follows. Figure 11 shows an example set of five potentials within a finite difference mesh.

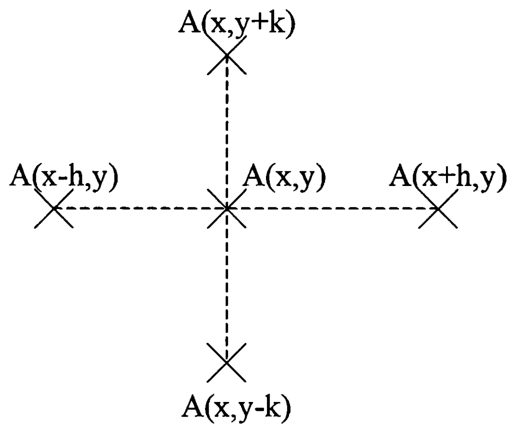


Figure 11: Finite difference potential grid

By expanding the potential term *A(x+h,y)* and using Taylor's series gives the following:

$$A(x+h,y) = A + \frac{h}{1!}\frac{\partial}{\partial x}A + \frac{h^2}{2!}\frac{\partial^2}{\partial x^2}A + \frac{h^3}{3!}\frac{\partial^3}{\partial x^3}A + \frac{h^4}{4!}\frac{\partial^4}{\partial x^4}A + O\left(h^5\right) \quad (74)$$

And similarly for *A(x-h,y)*:

$$A(x-h,y) = A - \frac{h}{1!}\frac{\partial}{\partial x}A + \frac{h^2}{2!}\frac{\partial^2}{\partial x^2}A - \frac{h^3}{3!}\frac{\partial^3}{\partial x^3}A + \frac{h^4}{4!}\frac{\partial^4}{\partial x^4}A + O\left(h^5\right) \quad (75)$$

Where in the above $A \equiv A(x,y)$

Summing Eq. 74 & Eq. 75, neglecting terms beyond second order gives:

$$\frac{\partial^2}{\partial x^2}A(x,y) = \frac{A(x-h,y) + A(x+h,y) - 2A(x,y)}{h^2} + O\left(h^2\right) \quad (76)$$

Similarly for *A(x,y-k)* and *A(x,y-k)*:

$$\frac{\partial^2}{\partial y^2}A(x,y) = \frac{A(x,y-k) + A(x,y+k) - 2A(x,y)}{k^2} + O\left(h^2\right) \quad (77)$$

The order of error in the above is denoted by the last term. Error is therefore proportional to the square of the distance between nodes. Substituting Eq. 76 and Eq. 77 in Eq. 72 gives the following approximation of the Poisson equation:

$$\frac{A(x-h,y) + A(x+h,y) - 2A(x,y)}{h^2} + \frac{A(x,y+k) + A(x,y-k) - 2A(x,y)}{k^2} = \mu J(x,y) \quad (78)$$

Problems with the method arise from the difficulty in substituting for the machines material boundaries with rectangular elements. When solving for the field, boundary conditions must be given, as an open boundary is not permitted. Methods to alleviate these problems do exist but lead to complex programs (Hoole (**9**)).

2.10 Finite element

In both of the following, it is usual to formulate electromagnetic problems in terms of the field potentials. For magnetic problems, scalar and vector potential are used. Two broad forms of finite element method can be described:

2.10.1 Integral

Integral methods are based on the integral formulations of electromagnetism, a typical example is the application to electrostatics. The potential at one point is the sum of the potential contributions from all the distributed charges, which are modelled by elemental volumes or equivalent surface charges. In non-homogenous media, the boundary element technique can be applied by replacing material of differing permittivity with an equivalent surface charge.

In magnetics a similar approach can be applied i.e. replacing the iron by an equivalent surface distribution of paired monopoles. This replacement allows a scalar magnetic representation to be used (Carpenter(15)).

The integral method provides the field at the points where the calculation has been performed, not a distribution through out the region (i.e. in contrast to differential solutions).

Al-Khayat (33) uses the boundary integral method to model the transformer windings, which is then used to derive the magnetic circuit.

The boundary element technique is also the method most often used within Printed Circuit Board parameter extraction packages. Also it is often used when there are complex 3D geometries and the inclusion of motion, Ramaswanny (62).

2.10.2 Differential

Differential methods are based on the differential forms of electromagnetics. A basis function is chosen which approximates the variation of the desired quantity in the solution region i.e. a linear variation of potential. A function is derived that is based on the field equations and boundary conditions e.g. energy, capacitance or inductance. The region is split into elements (for example triangular elements Figure 12) and the function is expressed in terms of the element (basis) approximation functions. The function is chosen such that its optimisation in each sub element leads to an optimisation as a whole, and hence the field equation solution. Methods used to derive this function include weighted residuals and calculus of variations.

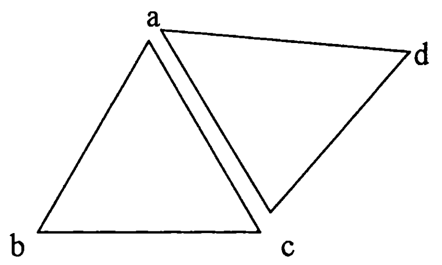


Figure 12: Finite elements

Finite elements by using approximation functions allow the field variables to be calculated throughout the solution region. Furthermore, because the function used in the solution is often energy based, quantities such as inductance and capacitance are known to a higher accuracy than derived quantities, such as electrostatic field strength and magnetic flux density.

Various adaptations have appeared that allows the incorporation of electric circuits and motion within both 2 dimensional and 3 dimensional FE formulations. An example of this includes work by Bedrosian (16). The incorporation of the constituent equation for current flow within conductive materials allows the eddy currents to be predicted in the solid iron sections.

Other work has included the coupling of the electrical, magnetic and thermal models of the machine, Drieden (60).

An interesting application of the differential FE technique is given by Demenko (17), where the similarity between FE equations and those of resistor & capacitor electrical networks is identified. A disadvantage of this approach is that a form of mutual capacitance is required for the model of each finite element. The combined field and electrical circuit equations are then solved numerically and the torque is derived via the use of the virtual work principle.

Various authors have compared the finite element method and magnetic equivalent circuits e.g. Wang et al (34). Others have combined the two modelling techniques e.g. Delforge et al (40) & McDermott et al (57).

It should also be noted that methods exist which allow the rotor skew of a typical machine to be taken into account within 2 dimensional finite element analysis (De Gersem et al (63)). These methods can also be adapted to other techniques to also account for the effect of rotor skew.

2.11 Magnetic equivalent circuits

2.11.1 Electrical and magnetic circuit similarities

Reluctance magnetic circuits use the analogy that the electric equivalent of flux is current and the equivalent of magnetic reluctance is resistance. This is the approach used by Laithwaite (18) and others. While this may allow analysis of the magnetic circuit, it does not give the correct power transfer between electrical and magnetic circuits, as pointed out by Carpenter (3) and Haydock (4). It was shown by Carpenter and Haydock that in magnetic circuits the equivalent of inductance is permeance or as described by the authors 'magnetic capacitance'.

Electrical: $$Q = CV \qquad (79)$$

Magnetic: $$\Phi = \Lambda mmf \qquad (80)$$

From Eq. 79 and Eq. 80 the similarities are obvious including the reason that Carpenter equates the permeance as a magnetic capacitance. It follows that the flow quantity is rate of change of flux. Magnetic circuits can be shown to obey the equivalents of Kirchoffs circuit laws. This extends to the replacement of a number of magnetic impedances by a single equivalent element (for linear impedances only), as noted by Carpenter (3). Also shown by Haydock is that the interface between the two types of circuit can be represented by a gyrator whereby the current in the electrical domain is transformed into mmf in the magnetic circuit. Haydock also demonstrated the ability to perform combined simulations of both electrical and magnetic circuits using the gyrator coupling. A complete explanation is given by Haydock (4) of the rules to transfer magnetic or electrical circuits across boundaries thus leading to a completely electrical or completely magnetic equivalent circuit.

An example magnetic circuit is shown in Figure 13 with C_m representing the magnetic permeance. The source mmf is controlled by current in the electrical circuit i.e. windings.

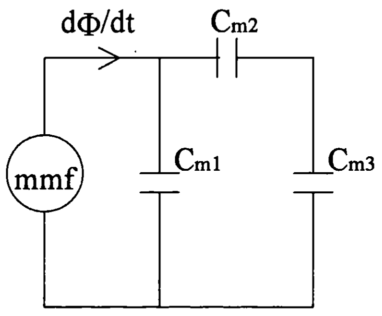


Figure 13: Example Magnetic Equivalent Circuit (MEC)

Magnetic equivalent circuits and permeance networks have found particular use in the analysis of permanent magnet machines e.g. Rasmuuen et al (36).

A variation on the use of magnetic circuit techniques, is the application of permeance networks to Bond-Graph models (Hecquet et al (56)). In the case of Delforge et al (19) Electrical, magnetic and mechanical elements are created that have the correct constituent equations and causal relationships. The rate of change of flux is the through variable and mmf the across variable. The resulting Bond-Graph network is then simulated using a package called 'Neptunix'.

A further explanation of the transformation between electrical and magnetic quantities, within combined electrical and magnetic equivalent circuits, is contained in appendix A.

2.12 Conclusion

The basic field quantities have been introduced, and the relationships between the various quantities have been presented. Some explanation as to the significance of the relationships has also been given. This has been only a very brief summary, but it will help provide a reference for the other chapters.

Finite element and finite difference methods can give accurate results for the field quantities. The accuracy of the results is increased by an increase in the number of nodes and elements. Finite elements are more flexible and widely used than finite difference in machine modelling. A problem with these methods is the computational effort needed for combining rotating electrical machines with power electronics.

3 Methods of Modelling Electrical Machines

3.1 Introduction

The purpose of this review is not to exhaustively catalogue all the relevant publications. It is to list those books and papers, which are thought to be significant and have formed the basis of the present work.

The work by Haydock (4) has a detailed comparison of the various two axis theories, and magnetic equivalent circuits, as applied to synchronous generators. Therefore a detailed review of the models themselves will not be undertaken, but a review of the modelling methods will be made.

Modelling of electromagnetic devices, specifically motors, generators and alternators, can be approached from several different viewpoints. These depend on the assumptions made in simplifying Maxwell's field equations for the complex geometry, coil configurations and the relative motions that are possible. Ultimately each modelling technique has its own particular advantages and disadvantages. The selection of the method used depends on the required use and purpose of the simulation.

Two broad classes of model can be described: idealised machine descriptions or field based approximations. In the former, various simplifying assumptions are made about the machine operation and an equivalent circuit is subsequently generated. In the latter, a geometrical model of the machine with the boundary conditions is used to set up field equations that are subsequently solved.

Another important simplification, often used in modelling machines, is to reduce what is a 3 dimensional problem to that of 2 dimensions. This simplification is usually valid due to machine construction and operating conditions, although important exceptions do occur e.g. where the field around the end turns of a complex winding is to be investigated.

An alternative method is that used by Gibson (31), where a variational solution is sought to lumped and distributed electrical circuits.

3.2 Machine magnetic circuits

In the examples of the inductor and transformer, relatively simple geometrical shapes were considered. Similar calculations for the magnetic circuit can also be performed for a machine. Difficulties arise because of the usually complicated shape and winding configurations.

From Eq. 27 it is evident that the torque is generated by the change in inductance of the circuit.

Energy stored in a pure inductor $U = \frac{1}{2}Li^2$ (27)

It should be noted that Eq. 27 only applies to a pure inductor and that in a machine the separate mutual and self-inductances need accounted for. The major element that changes the inductance with motion is the air-gap, therefore the air-gap needs careful modelling to correctly predict the torque.

A method of allowing for the effect of stator and rotor slots on the field in the air-gap was developed by Carter (10). This modifies the results obtained by the simple cylindrical model by applying a correction factor that takes into account the effect of the slot. This method was later modified by Neville (11) to take into account the effect of narrow teeth.

Another effect of varying the inductance is the generation of an emf in the motor or generator that is referred to as speed voltage. This emf can be defined by substituting Eq. 25 into Eq. 24 and splitting the derivative that gives:

$$emf = -\frac{d(Li)}{dt} = -\left(L\frac{di}{dt} + i\frac{dL}{dt}\right) \tag{81}$$

The first term in Eq. 81 is the transformer voltage and the second speed voltage.

An area of particular difficulty is the torque derived from the air gap, yet it can be shown by an analysis of the energy within the machine (Fitzgerald A.E. et al (8) pp. 83), and application of the virtual work principle, that the torque is given by:

$$T = -\frac{\partial U_{fld}}{\partial \theta} \tag{82}$$

In the above U_{fld} is the total stored energy in the field and θ is the angular displacement.

3.3 Electrical equivalent circuit

The T model is based on the analysis of flux paths and the measurable properties of the machine. Lumped values of inductances, which are based on the flux path analysis, can be converted to an equivalent electrical circuit. This allows electrical circuit solution methods to be applied, and aspects of the machine performance to be predicted.

A relatively simple example is provided by the transformer, where the device can be viewed as a system of coils linked by inductors. The values of the equivalent circuit components can be calculated from tests such as open and short circuit. In the open circuit test the secondary is left unconnected, and the voltage of the primary measured while varying the current. Conversely, for the short circuit test the secondary is short circuited, and a below rating voltage applied to the primary and resulting current is measured.

From the equations for the transformer, Eq. 57 and Eq. 62, it is possible to construct the equivalent circuit illustrated in Figure 14. Note that all the values have been transferred to the primary side of the transformer.

Figure 14: Transformer equivalent circuit

For three phase circuits, balanced conditions are assumed and the circuit parameters transformed to those of an equivalent single-phase circuit. Often for power system load analysis the values of circuit parameters are also converted to the per unit system (Fitzgerald A.E. et al (8) pp. 52). In the per unit system the turns ratio can be removed, this therefore allows simpler calculations.

In the case of the synchronous generator (Figure 15a), the simplest equivalent circuit consists of a synchronous impedance in series with the generator emf (Figure 15b).

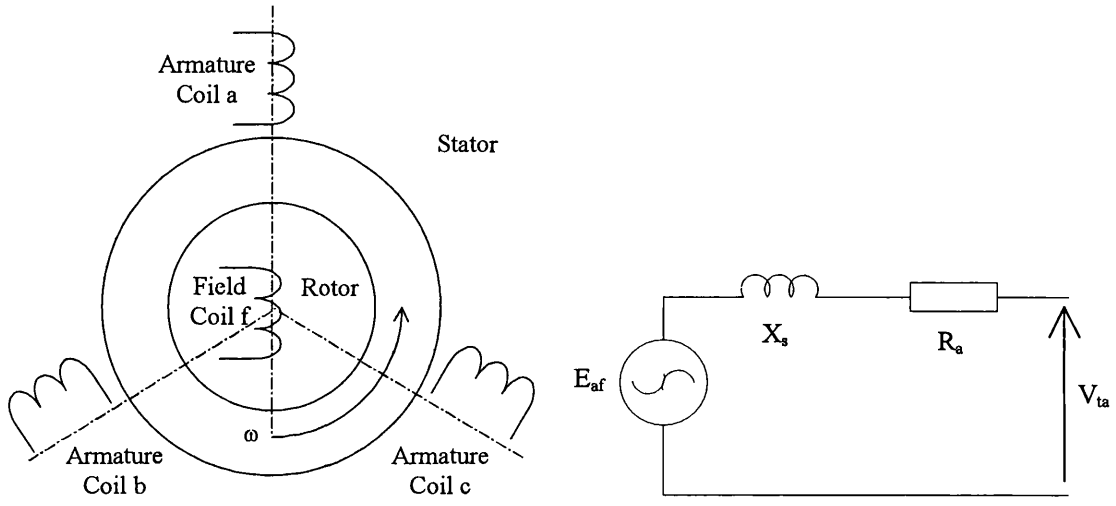


Figure 15: Simplified synchronous generator/motor (a) and equivalent circuit (b)

Another example of an equivalent circuit is that of the induction motor (Figure 16). The power lost to the mechanical load is represented by a resistive element (R_2(1-s)/s) that varies with the slip (*s*) (Fitzgerald A.E. et al (8) pp. 420). Slip is the ratio of difference in speed of the rotor and the stator field to that of the stator field. Note the similarity with the transformer model already derived (Figure 14.).

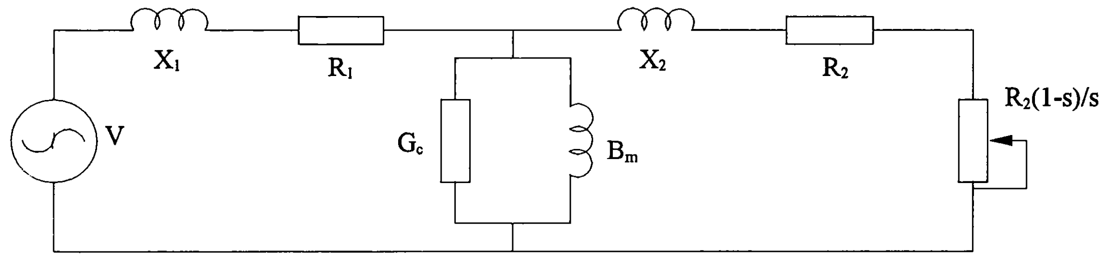


Figure 16: Equivalent circuit of a polyphase induction motor

The type and number of machine tests that are performed allow a certain fixed number of equation coefficients to be found. Therefore to determine the parameters of more complex models there needs to be more complex testing, unless simplifying assumptions are introduced into the model.

According to Slemon (12), in the specific case of induction machine models used in electric drives, the models are often too complex. By operating the machine at two speeds, four parameters can be measured, with five unknowns to be determined. The traditional resolution is to arbitrarily make stator leakage equal to rotor leakage ($L_{ls} = L_{lr}$). Slemons approach is to transform the T model (Figure 16) to a Γ model (Figure 17) that only requires four parameters.

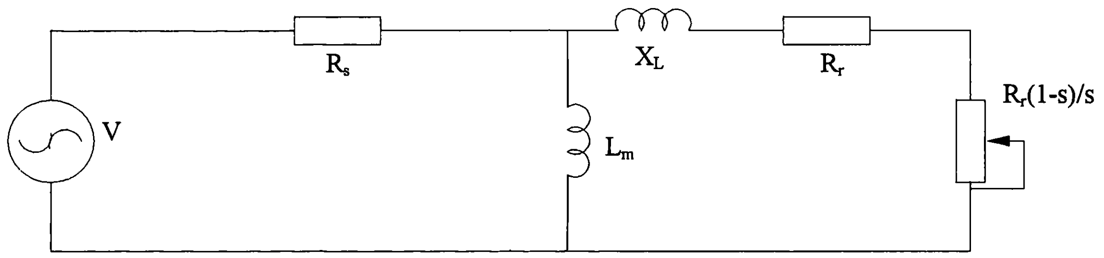


Figure 17: Induction motor equivalent circuit as suggested by Slemon

A method is presented by Freeman (30), which takes a basic field description of the machine and converts it to an equivalent circuit. This is based upon on a cylindrical model and 'T' circuit elements.

3.4 Two axis

The machine is idealised as a variation of a general-purpose machine with quadrature windings on the stator and rotor. Similar to the dc machine, one of the axes is attached to either the stator or rotor mmf. The mmf field in the air-gap is resolved as a result of two quadrature windings (Figure 18). An important assumption, that enables the separation of the flux into two orthogonal components, is that the field equations are linear. By the principle of superposition, the 3 phase windings can be replaced by their dq axis equivalents. This is shown in Eq. 83 where a_d, a_q and a_0 are the two axis components, while a_a, a_b and a_c are the 3 phase components (Adkins et al (13)).

$$\begin{bmatrix} a_d \\ a_q \\ a_0 \end{bmatrix} = \begin{bmatrix} \cos\theta & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ -\sin\theta & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} a_a \\ a_b \\ a_c \end{bmatrix} \qquad (83)$$

Note that for balanced conditions the a_0 term is zero (the same as for symmetrical component theory).

Use of two-axis theory is particularly appropriate in a salient machine, due to the difference in rotor permeance along the direct and quadrature axis. The dc motor and generator, synchronous motor, synchronous alternator and induction motor can be described by similar equations, when the rotating machine has been converted to the ideal form. By referring either stator or rotor quantities to the other side of the air gap, it is possible to simplify the equations. This is especially usefull for the salient synchronous machine as in balanced conditions the mutual inductance seen by the rotor is constant.

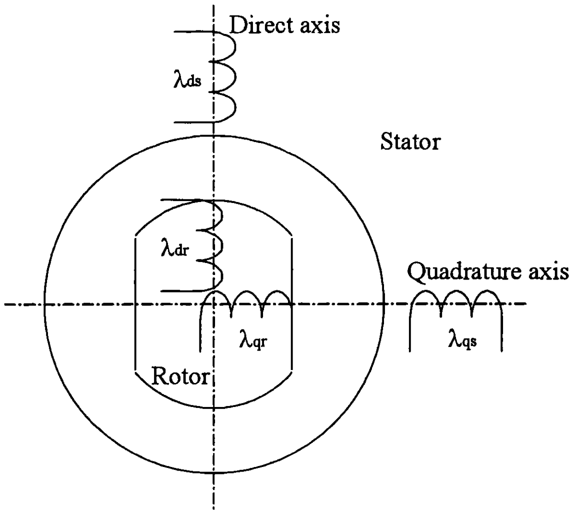


Figure 18: Two axis machine representation

In the case of the synchronous machine (Figure 18), the winding interactions are analysed with the direct axis aligned with the rotor field coil. This analysis gives the impedance matrix shown in Eq. 84. The d and q subscripts refer to ds and qs in the diagram, subscript f refers to subscript dr in the diagram. Note that the inductances are constant while the effect of rotor motion is included via the use of speed terms (ωL).

$$\begin{bmatrix} v_d \\ v_q \\ v_f \\ v_0 \end{bmatrix} = \begin{bmatrix} -R_a - L_d\frac{d}{dt} & \omega L_q & L_{af}\frac{d}{dt} & \\ -\omega L_d & -R_a - L_q\frac{d}{dt} & \omega L_f & \\ -\frac{3}{2}L_{af}\frac{d}{dt} & & R_f + L_{ff}\frac{d}{dt} & \\ & & & -R_a - L_0\frac{d}{dt} \end{bmatrix} \begin{bmatrix} i_d \\ i_q \\ i_f \\ i_0 \end{bmatrix} \quad (84)$$

Alternatively for the induction motor Parks transform results in Eq. 85.

$$\begin{bmatrix} V_{ds} \\ V_{dr} \\ V_{qr} \\ V_{qs} \end{bmatrix} = \begin{bmatrix} R_{ds} + L_s\frac{d}{dt} & M\frac{d}{dt} & & \\ M\frac{d}{dt} & R_{dr} + L_r\frac{d}{dt} & L_r\omega & M\omega \\ -M\omega & -L_r\omega & R_{qr} + L_r\frac{d}{dt} & M\frac{d}{dt} \\ & & M\frac{d}{dt} & R_{qs} + L_s\frac{d}{dt} \end{bmatrix} \begin{bmatrix} i_{ds} \\ i_{dr} \\ i_{qr} \\ i_{qs} \end{bmatrix} \quad (85)$$

In Eq. 84 and Eq. 85 the derivative terms are either included or assumed to be zero, depending on whether a transient or steady state solution is required.

A problem identified by Sudhoff (41) et al is that the *qd* induction motor model typically used in drive simulations can be very inaccurate in predicting machine performance. Another problem can be accurately identifying the model parameters (Jacobina (44)).

3.5 Space vector

The following is largely based on Vas (14). This method uses an analysis of mmf and flux around the periphery of the air gap of an ideal machine, as for the two axis model. A complex exponential is then used to represent both spatial distribution and time variation. The reference point for the spatial distribution, again similar to two axis theories, is any location that simplifies the calculations.

A good example of this being used in analysis is given by Suciu et al (42).

3.6 Calculation of permeance values for MEC

Two equivalent approaches exist for the calculation of permeance values. One is direct application of the definition of reluctance Eq. 48, or its permeance alternative.

Reluctance $$S = \frac{1}{\Lambda} = \oint \frac{dl}{\mu(l)A(l)} \qquad (48)$$

The other method that leads to the same result is the application of Eq. 46 (Ostovic ((20) pp.9).

$$mmf = S\Phi = \frac{\Phi}{\Lambda} \qquad (46)$$

Use of the equation requires rearranging the flux and mmf to give the permeance.

Alternatively, Delforge et al (19) use a finite element package to determine the flux and value of scalar magnetic potential difference. Thus it is possible to determine the 'flux tube' and hence the permeance network and element values.

3.6.1 Incorporation of saturation

Because the emf is the derivative of flux, therefore the first derivative of any model of saturation should be continuous. Accurate physical models of the magnetic response of ferromagnetic materials do exist. Whether these models can be used depends on the functions available within the modelling environment, and if the time taken to numerically evaluate the functions is prohibitive. Ostovic ((20) pp.54) lists several methods for modelling the B/H curve. The linear piecewise approximation being rejected due to its derivative discontinuity. The quadratic piecewise approximation allows a curve that has no discontinuities in the first derivative. If higher continuous derivatives are needed then a cubic approximation can be used. Other alternatives are exponential approximations such as Eq. (86) which depends on the curve fit of coefficients a, b and c. A problem with such a function is that the inverse cannot always be analytically found i.e. B is an implicit function of H.

$$H = aB + bB^c \qquad (86)$$

Where a curve fit is used for the inverse, simulation difficulties can arise due to curve fit error. If both normal and inverse equations are incorporated within a simulation then a multi-valued BH relationship exists.

Haydock uses a polynomial curve fit (SPICE 2G6 only allowed this option), this gives reasonable results for a certain operating range, but can be a poor fit at low or high mmf values.

3.6.2 Incorporation of motion.

A relatively simple method for incorporating of the effect of motion is that used by Ostovic (20) and others. This simulates relative movement of the stator and rotor by in effect altering the permeance values between rotor and stator. This relies on there being a permeance function between each possible stator peripheral flux path and the possible rotor peripheral flux paths. Thus for n stator flux paths and m rotor flux paths there are n×m permeance elements needed. Thus for a detailed sample of air-gap flux a very large number of flux path elements would be required. A problem then arises as how to derive the values of the air-gap permeance. This has two parts i.e. the permeance as a function of the motion and the peak value.

A method is that used by Chaudhry et al (2) where a finite element analysis is used to derive a uniformly-distributed samples of the change in permeance. These are then used as part of a conventional state-space technique to determine the open circuit characteristic.

Ostovic's approach is to use a piecewise sinusoidal approximation for the function. Skew of the air-gap is allowed for by suitable corrections of the function. Peak values of permeance are calculated for typical trapezoidal elements, with correction factors for large slot sizes and rotor non-concentricity.

Preston and Lyons (21) have used the MEC to model a switched reluctance motor. A finite element solution at rotor angles of 0, 90 and 180° is used to determine the values (α_1, α_2 & p) of the following function:

Air-gap reluctance $$Rg = \frac{1}{\alpha_1 + \alpha_2 \left|\cos(\theta/2)\right|^p} \qquad (87)$$

Good results were obtained for the multiply excited motor model when compared to a finite element model. As pointed out by Moallem et al (22) good results can be obtained from MEC models, provided the air-gap flux is accurately modelled. The model used by Moallem et al has multiple MEC elements in the air-gap, the elements being derived from a FE solution.

Another approach, where there was no restriction on the type of functions available, is that by used by Delforge et al (23)(24). While not describing the function used for the curve fit, it is stated that a transcendental function is curve fitted to a finite element analysis of the permeance variation.

Another method, as used by Haydock, is similar to that used in two axis methods. This depends on the observation that the rotor and stator are only concerned with the mmf, and not physical position. Therefore the phase currents are rotated around the stator, and the motional emfs are explicitly incorporated in the air-gap. The resulting circuit is smaller than the previous method, but the need to find phase symmetries means that automatic generation from geometrical data is difficult.

3.7 Transformer Optimisation Example using MECs

An increasingly important area of use for both Finite Element (FE) techniques and Magnetic Equivalent Circuits (MEC) is as part of optimisation studies. Of particular note is the work by Ratnajeevan et al (58) where the use of neural networks and gradient methods is explored. Often the there are device dimensions that need adjusting such that some design goal is satisfied. The designs performance according to certain criteria needs to be repeatedly evaluated, with usually minor changes to device dimensions. A risk with all such optimisation methods is that of false minima due to peculiarities of the device modelling method. Ranajeevan (54) in particular mentions the problems of mesh error when using FE solutions in the optimisation loop. The predominant effect is to give very slow optimisation, due to the interaction between the meshing/FE solution and the optimisation routine.

The design of the transformer depends on the optimisation of certain performance criteria. Another example of MEC models being used to optimise magnetic circuits is given by Hameyer (37). Optimisation can be defined as the maximising or minimising of certain performance criteria. These criteria are application dependent and include such things as magnetising current and secondary mass. The relative importance of each criterion can only be decided by engineering judgement.

Other restrictions upon the design include constraints on the range of values that certain variables are allowed. These constraints include for instance the maximum core width and height.

Two approaches to the design optimisation are available:

1. Plotting the variation of performance criteria with a change in independent variable, while holding the other variables constant. This assumes the independent variables have conjugate minimisation directions, which for the situation given has not been proved. If less than three independent variables are to be optimised then another strategy is to plot the variables as a multidimensional plot. Unfortunately in this situation, more than two independent variables need to be plotted.
2. Use can be made of numerical procedures to find the optimum. Several numerical methods exist for the minimisation of functions, two broad categories are methods that find an optimum and those that find the optimal solution. A definition of optimal is those values of independent variables that give the global maximum or minimum function value. This function is the objective and can incorporate various criteria via weightings. The weightings partially determine the criteria relative importance (partially as the criteria functions themselves will vary in scale). Determination of the global minimum can be guaranteed for linear objective functions. This is not the case for non-linear functions where local minima can give false optimal values. In such a situation, the best that can be achieved is that an optimal solution is determined i.e. it is nearly as good as the global optimal solution.

3.7.1 Optimisation

There are several numerical optimisation methods available. Principle differences include whether the algorithms assume a linear or non-linear function of the independent variables. Another differences is whether it is assumed that the independent variables have conjugate minimisation directions. In other words, the minimisation movement along one dimension does not spoil the minimisation achieved in one of the other directions.

Methods exist that can determine a suitable set of conjugate directions (basis vectors), these allow the application of single dimension minimisation routines to each dimension without the need to redo a previous minimisation. A problem with such methods is local minima that can give a false optimum. The degree to which the local minimum is different from the global minimum depends on the initial values of the minimisation routine. To guarantee for non-linear problems that the global minimum has been found would require finding every local minimum.

For the problem as presented here, it is sufficient that a solution that is near the global minimum is found. Methods that can provide such results are statistical in nature and have a certain probability of finding the global minimum. The particular method chosen is simulated annealing, due to its previously successful application on other problems.

3.7.2 Simulated annealing

This method is based on an analogy with the thermodynamic process where metals are slowly cooled from a high temperature. If the cooling is sufficiently slow then the atoms are ordered to form a pure crystal that can extend for several billions of atom sizes. The ordered crystal being the minimum energy state that the atoms can occupy. Conversely if the cooling is too fast then an amorphous state exists which has a far higher trapped energy state. From thermodynamics the probability of a atom having a certain energy can be approximated by the Boltzmann probability distribution:

$$\mathrm{Prob}(E) = e^{\frac{-E}{kT}} \qquad (88)$$

Where E is the energy, T is the temperature and k is Boltzmann's constant.

The application of the thermodynamic theory of annealing is by equating the objective function value with that of energy. The function independent variables are viewed as equivalent to the dimensions of the thermodynamic state space. During each iteration the values of the independent variables are altered in a random manner according to Boltzmann's equation. A slight modification known as the Metropolis algorithm always takes a down hill step but will sometimes take an uphill step. The best result is saved and is used as the starting point for a new set of iterations. From each set of iterations to the next, the value of notional temperature is decreased. The ultimate minimum is when the temperature gives random variations less than the machine floating point resolution.

The algorithm actually used also incorporates the downhill simplex method to improve on the convergence properties near the minimum. When the temperature is high the algorithm always accepts a downhill step, but will sometimes accept an uphill step. When the temperature is low the algorithm behaves exactly as the simplex method where the set of points (simplex vertices) are transformed such that the simplex always moves in the direction of the minimum.

3.7.3 Round transformer coupling equations

Shown in Figure 19 is a cross-section through the core of a round transformer, which is split through the centre at an angle to the horizontal. As stated earlier the objective is to optimise the design of the transformer given that certain of the dimensions have fixed limits, out of which they cannot be taken. k_1 & k_2 are variables which are to be varied, thus changing the physical dimensions. The calculation of the permeances is dependent on the values of k_1 & k_2.

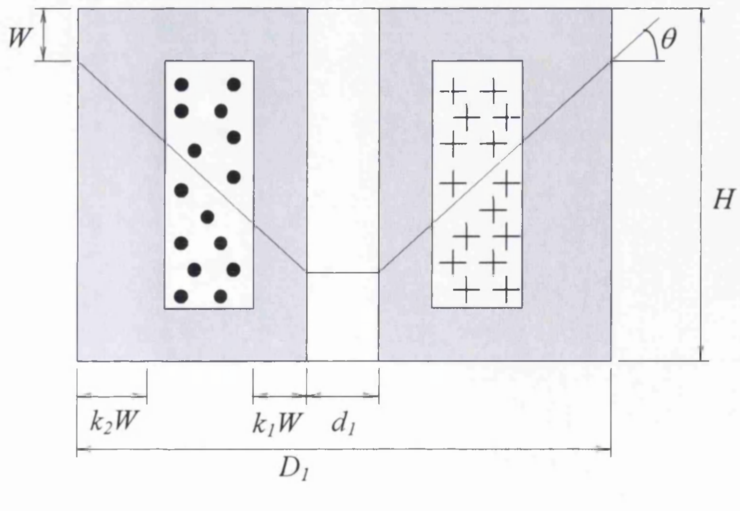


Figure 19: Cross-sectional view of round transformer

$$\cos\phi = p.f.\ \text{(power factor)} \tag{89}$$

Length of equivalent rectangular core (k_s lamination stacking factor):

$$l_{fe} = \pi k_s d_1 \tag{90}$$

Minimum area of core through which flux passes:

$$A_c = l_{fe} w \cdot \mathrm{Min}(k_1, \mathrm{k}_2, 1) \tag{91}$$

Number of turns to give voltage V_p (f frequency, $\hat{B}$ peak flux density):

$$N_p = \frac{V_p}{4.44 f \hat{B} A_c} \tag{92}$$

Reluctance of combined air-gap R_g with distance between cores of l_g:

$$R_g = \frac{l_g \cos^2\theta}{l_{fe} k_c \mu_0 w}\left(\frac{1}{k_1} + \frac{1}{k_2}\right) \tag{93}$$

Load current I_L as seen by primary: $I_l = \dfrac{S}{V_p}$ (94)

Magnetising current with contributions from the air-gap and core (S_{fe} core specific VA, P_{fe} core specific power loss):

$$I_m = \frac{\hat{B} A_c R_g}{N_p \sqrt{2}} + \frac{\sqrt{S_{fe}^2 - P_{fe}^2}}{V_p} W_{fe} \tag{95}$$

Total primary current including core losses:

$$I_p = \sqrt{\left(S\cos\phi + \frac{P_{fe}}{V_p} W_{fe}\right)^2 + \left(S\sin\phi + I_m\right)^2} \tag{96}$$

Copper area for specified current density δ:

$$A_{cu} = \frac{I_p N_p + I_l N_p}{\delta} \tag{97}$$

Height H (k_w winding window occupation factor):

$$H = \frac{A_{cu} / k_w}{(D_1 - d_1)/2 - (k_1 + k_2) w} + 2w \tag{98}$$

Value of θ given the copper area has the same ratio as secondary/primary current:

$$\tan\theta = \frac{2I_l(H-2w)}{((D_1-d_1)/2+(k_2-k_1)w)(I_p+I_l)} \tag{99}$$

Maximum value of $\theta \tan\theta_{\max} = \dfrac{2(H-2w)}{D_1-d_1}$ (100)

Copper volume: $V_{cu} = \pi A_{cu}\left((D_1+d_1)/2+(k_1-k_2)w\right)$ (101)

Area of iron: $A_{fe} = w\left((k_1+k_2)H+(D_1-d_1)-2(k_1+k_2)w\right)$ (102)

Volume of iron: $V_{fe} = l_{fe}A_{fe}$ (103)

Weight of copper (ρ_{cu} copper density):

$$W_{cu} = V_{cu}\rho_{cu} \tag{104}$$

Weight of iron (ρ_{fe} lamination density):

$$W_{fe} = V_{fe}\rho_{fe} \tag{105}$$

Losses within copper (ρ resistivity): $L_{cu} = \delta^2 \rho V_{cu}$ (106)

Losses within core (P_{fe} Iron specific power):

$$L_{fe} = P_{fe}W_{fe} \tag{107}$$

Area of secondary laminations:

$$A_{\sec.fe} = \frac{w(D_1-d_1)}{2} + \frac{k_2^2 w^2 \tan\theta}{2} + \frac{k_1^2 w^2 \tan\theta}{2} + k_1 w\left((D_1-d_1)/2+k_1 w\right)\tan\theta \tag{108}$$

Weight of secondary: $W_{\sec.} = W_{fe}\dfrac{A_{\sec.fe}}{A_{fe}} + W_{cu}\dfrac{I_l}{I_p+I_l}$ (109)

Weight of primary: $W_{prim.} = W_{fe} + W_{cu} - W_{\sec.}$ (110)

Maximum inter-winding voltage on the primary:

$$V_{\max.} = \frac{V_p}{N_p}\left(2\left(\frac{H-w(2+k_2\tan\theta)}{\sqrt{\dfrac{4I_p}{\pi\delta}}}\right)+1\right)$$

Efficiency: $\eta = \dfrac{S\cos\phi}{S\cos\phi + L_{cu} + L_{fe}}$ (111)

Permeance integral calculations:

Define: $$h_o = k_2 w \tag{112}$$

$$h_i = \left((D_1 - d_1)/2 - k_1 w\right)\tan\theta \tag{113}$$

Primary leakage: $$P_1 = \frac{\mu_0 l_{fe}\left(H - 2w - (h_i - h_o)/2\right)}{3\left((D_1 - d_1)/2 - (k_1 + k_2)w\right)} \tag{114}$$

Secondary leakage: $$P_2 = \frac{\mu_0 l_{fe}(h_i + h_o)/2}{3\left((D_1 - d_1)/2 - (k_1 + k_2)w\right)} \tag{115}$$

Integral 3 & 4 (lower and upper limb):

$$I_{3\&4} = \frac{-l_{fe}\ln\left(2((D_1 - d_1)/2 - (k_1 + k_2)w)/(D_1 - d_1)\right)}{k_1 + k_2} \tag{116}$$

Integral 5 (inside lower limb):

$$I_5 = \frac{-l_{fe}\ln\left((H - w - h_i - k_1 w\tan\theta)/(H - h_i - k_1 w\tan\theta)\right)k_1}{1 - k_1\tan\theta}$$

Integral 6 (inside upper limb): $$I_6 = \frac{-l_{fe}\ln(h_i)k_1}{1 + k_1\tan\theta} \tag{117}$$

Integral 7 (outer lower limb): $$I_7 = \frac{-l_{fe}\ln\left((H - 2w - h_o)/(H - w)\right)k_2 w}{h_o + w} \tag{118}$$

Integral 8 (outer upper limb): $$I_8 = \frac{-l_{fe}\ln(h_o / w)k_2 w}{w - h} \tag{119}$$

Inside air-gap permeance: $$P_9 = \frac{\mu_0 k_c l_{fe} k_1 w}{l_g \cos^2\theta} \tag{120}$$

Outside air-gap permeance: $$P_{10} = \frac{\mu_0 k_c l_{fe} k_2 w}{l_g \cos^2\theta} \tag{121}$$

3.7.4 Solving the design equations

To solve the transformer equations the root needs to be found, as θ is dependent on I_p and W_{fe} is dependent on $V_{fe.}$ The algorithm chosen is Broyden's method that is a multi-dimensional variation of the single dimension Secant method. The advantages of the method include no requirement for the first derivatives (Jacobian matrix) and it potentially converges faster than the Newton method.

3.7.5 Implementation of algorithms and design equations

The algorithms were implemented in 'C' using supplied routines for the simulated annealing and Broyden's method. Minor alterations were made to the Broyden routine. These alterations stopped the routine from exiting the program upon root finding failure. Instead, the point on the simplex where the failure occurred is given a value equal to the highest existing function value within the simplex. This failure could arise if there is no solution for the particular input values. The simplex will therefore tend to try a direction that is towards the lowest simplex point, and away from the problem point.

The constraints are incorporated by adjusting the objective function value by a cost factor. A suitable value for this cost factor is such that it forces acceptable compliance with the constraint while still minimising the objective function. The cost factor must therefore be less than the simulated annealing tolerance value.

3.7.6 Results of design optimisation

The values that are thought to be most important, in the design of the transformer, are the magnetising current and secondary mass. The program is therefore run with a range of weightings that alter the importance of one criterion with respect to the other. Figure 20 shows the results of running the optimisation program with a selection of weightings i.e. adjusting the relative importance of secondary magnetisation current versus secondary mass in the optimisation objective function.

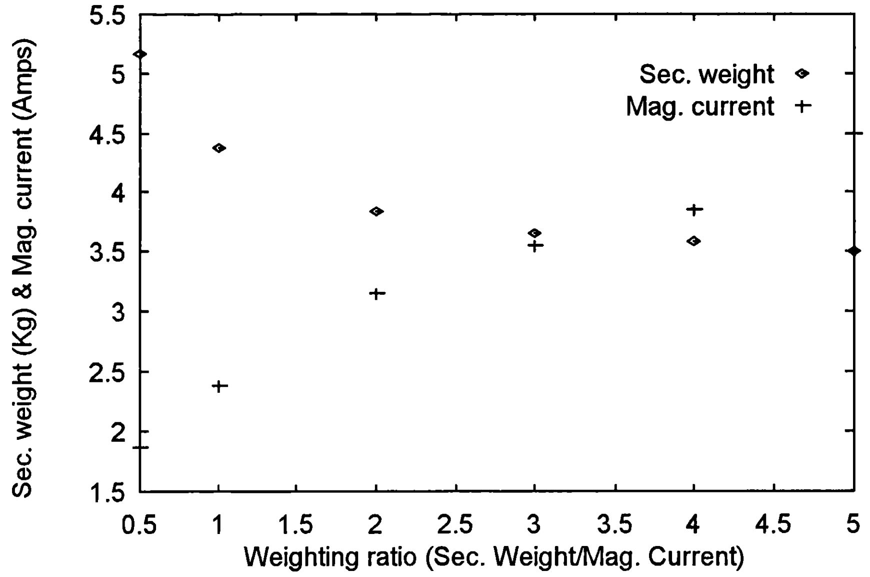


Figure 20: Plot of secondary weight and magnetising current verses weighting

It is important to note that the design equations used in the optimisation neglect any effect of leakage. The actual performance results from simulation and testing are therefore likely to be different from those presented here. However the above is a usefull example of how permeance elements can be used within an optimisation problem. The solution could be used as the basis for further design studies.

3.8 Conclusion

The broad background of various types of models has been presented. Reluctance based models allow electrical equivalent circuits to be derived, but these lead to models that are difficult to relate to the actual machine design. Certain field effects, such as fringing, are incorporated by the use of empirical correction factors. Models based on an ideal machine (such as two axis) often have quantities that are difficult to relate to the physical machine. When the machine is available for test then the model parameters can be determined, the resulting model being valid for operating conditions similar to those in which the tests were performed. A further problem is due to the inherent assumption that the machine can be described as a linear system.

Magnetic equivalent circuits have advantages and disadvantages compared with the previous methods. The topology of the magnetic circuit is dependent on the flux paths, which are either empirically deduced or found via field solutions. Non-linear properties are included directly in the magnetic circuit. Increased accuracy can be achieved by the inclusion of more elements in critical sections. The values of the lumped permeances can be related to the physical machine. It is also relatively easy, via use of the gyrator or linkage (see Appendix A), to connect electrical and magnetic circuits within one model.

Therefore, it is proposed to use the MEC to construct the machine models. Within this scheme the determination of permeance values will be via an application of the reluctance integral.

Another justification for the use of the MEC is that if very accurate field quantities are required then another scheme such as Finite Elements can be used. Wang et al (34) compare the results of using a finite element solution and a magnetic equivalent circuit for modelling a brushless DC motor and show that 65-90% accuracy can be achieved, with a relatively simple model. The MEC providing an intermediate complexity model.

4 Development of Multi-Physics SPICE models

4.1 Introduction

Machine types that will be modelled are restricted to rotational machines with one stator and a rotor. The magnetic field will be assumed to be uniform along the machine axis and end coil effects are neglected. The machine can therefore be approximated as a section through the middle of the rotor length with all currents being perpendicular to the plane of the section.

The main criteria are to provide models that are amenable to automatic generation from a single textual description. The various elements of the magnetic circuit will be implemented as either SPICE circuit components or sub circuits.

Non-linear capacitors will be defined and use will be made of these to represent the magnetic permeance.

The permeance of the air gap elements varies with the angle of the rotor relative to the stator. A suitable function therefore needs to be provided, which can model the variation of the air-gap permeance. A function is described that has a continuous first derivative and is compatible with SPICE.

The similarities between the electrical circuit equations and mechanical motion equations are shown.

Summation of the torque elements generated in the air-gap allows the total machine torque to be calculated. There is no analytical solution to the derivative of the permeance function, therefore a method of obtaining the derivative during the simulation run is derived and described. An alteration to reduce the inaccuracy of the simulation, for large values of accumulated rotation is developed.

A method of incorporating the BH curve characteristic is introduced allowing an approximation of the core loss to be simulated. This effect is illustrated with two examples.

Having defined Eq. 122 for the variation of the air-gap permeance with rotor angle, the coefficients need to be calculated from the machine information; an algorithm for this is described. Test results are also given with an example permeance verses angular position curve.

4.2 Non-linear capacitor

Using $Q = CV$, if the capacitor value is fixed then the charge (Q) on the capacitor can be varied by changing the voltage (*V*), this would have the same effect as varying the value of the capacitance. This effect is illustrated in Figure 21 where the voltage across the capacitor is modulated by a function denoted by fn. The charge on the capacitor (*C*) is therefore partially controlled by the function, a change in stored charge gives a current, which is transferred to the external circuit via the current controlled current source (*I*).

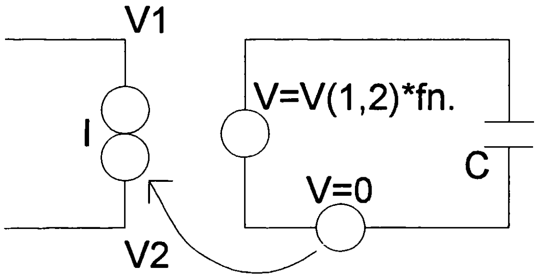


Figure 21: Non-linear capacitance

4.3 Permeability modulation from rotor motion

The technique used for the incorporation of relative motion, of stator and rotor, is the same as that used by Ostovic (20). This is a simple scheme, where there are n possible rotor air-gap flux paths and m possible stator air-gap flux paths, the resulting number of air-gap elements needed is n × m. This is far more than would be needed for a stationary rotor and stator while moving the field with explicit addition of motional emfs (Haydock (4)). The method allows relatively simple automatic generation of the air-gap elements, and has a major advantage that minor changes are needed to model non-concentric rotors. A disadvantage is the number of elements needed to model the air-gap is n × m compared to n + m when the field is moved.

A suitable function that allows a curve fit of the required permeance that can be implemented in SPICE version 3 is given by Eq. 122. This is one of many possible equations which can be implemented in spice, which could be used as the basis of a curve fit.

Permeance function $C_m(\theta) = P_{\max} \cdot e^{-a|\sin((\theta+b)/2)|^c}$ (122)

Note that Eq. 122 is first derivative continuous (Figure 22) as demanded by SPICE based analogue circuit simulators for proper convergence.

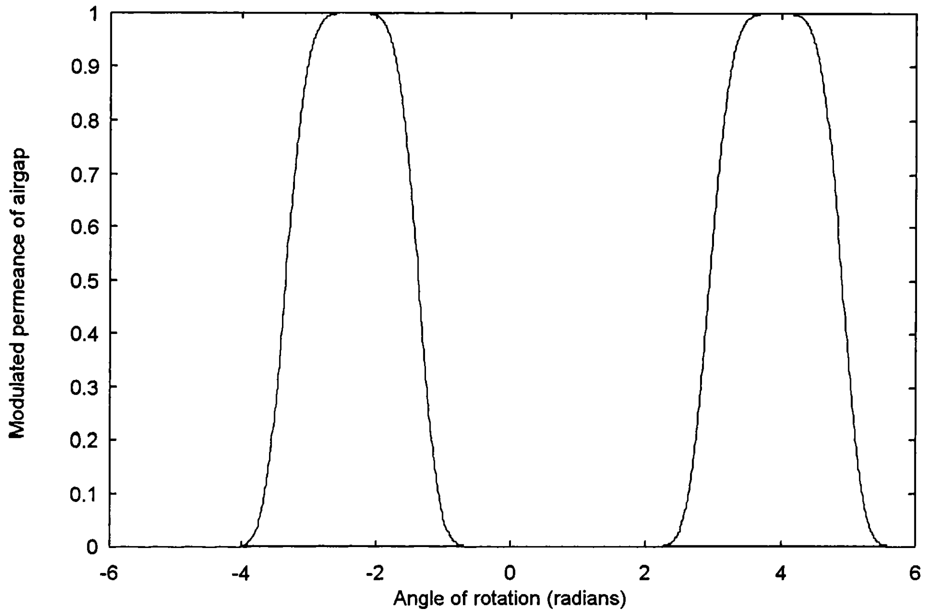


Figure 22: Plot of relative permeance function

4.4 Comparison of curve fitted permeance functions

As a check of the validity of the equation used as the basis for the curve fit, a comparison is made between the equation used by Preston and Lyons (21) and that used here. Due to the voltage developed in the coils being due to the derivative of the change in flux, the first derivative of the permeance function is very important for the successful modelling of the machine. For the comparison typical sets of coefficients values are used for the respective curves, the first derivatives are then compared.

$$f_1 = pe^{-a|\sin(x/2)|^2} \quad (123)$$

Using Mupad (52) the first derivative is:

$$f_1' = -\frac{\operatorname{sign}(\sin\frac{x}{2})\cdot acp\cos\frac{x}{2}\cdot\left|\sin x\right|^{c-1}\cdot e^{-a\left|\sin\frac{x}{2}\right|^{c}}}{2} \qquad (124)$$

Given the equation $f_1 = e^{-11|\sin x|^2}$ and the implied coefficients,

differentiating wrt x (using Mupad [52]):

$$f_1' = -22\cos x\cdot\left|\sin x\right|\cdot e^{-11|\sin x|^2}\cdot\operatorname{sign}(\sin x) \qquad (125)$$

The following is equivalent to the equation presented by Preston and Lyons (21) (Equation 120):

$$f_2 = a + b\cdot\left|\cos\frac{x}{2}\right|^{c} \qquad (126)$$

Using Mupad [52] the first derivative is:

$$f_2' = -\frac{\operatorname{sign}(\cos\frac{x}{2})\cdot bc\sin\frac{x}{2}\cdot\left|\cos\frac{x}{2}\right|^{c-1}}{2} \qquad (127)$$

if $f_2 = 0 + 1\cdot\left|\cos x\right|^{20}$ is the closest matching curve to that of f_1,

differentiating wrt x (using Mupad (52):

$$f_2' = -20\sin x\cdot\left|\cos x\right|^{19}\cdot\operatorname{sign}(\cos x) \qquad (128)$$

Shown in Figure 23 is a plot of the respective curves, as can be seen there is a very good match between the first derivatives for both equations.

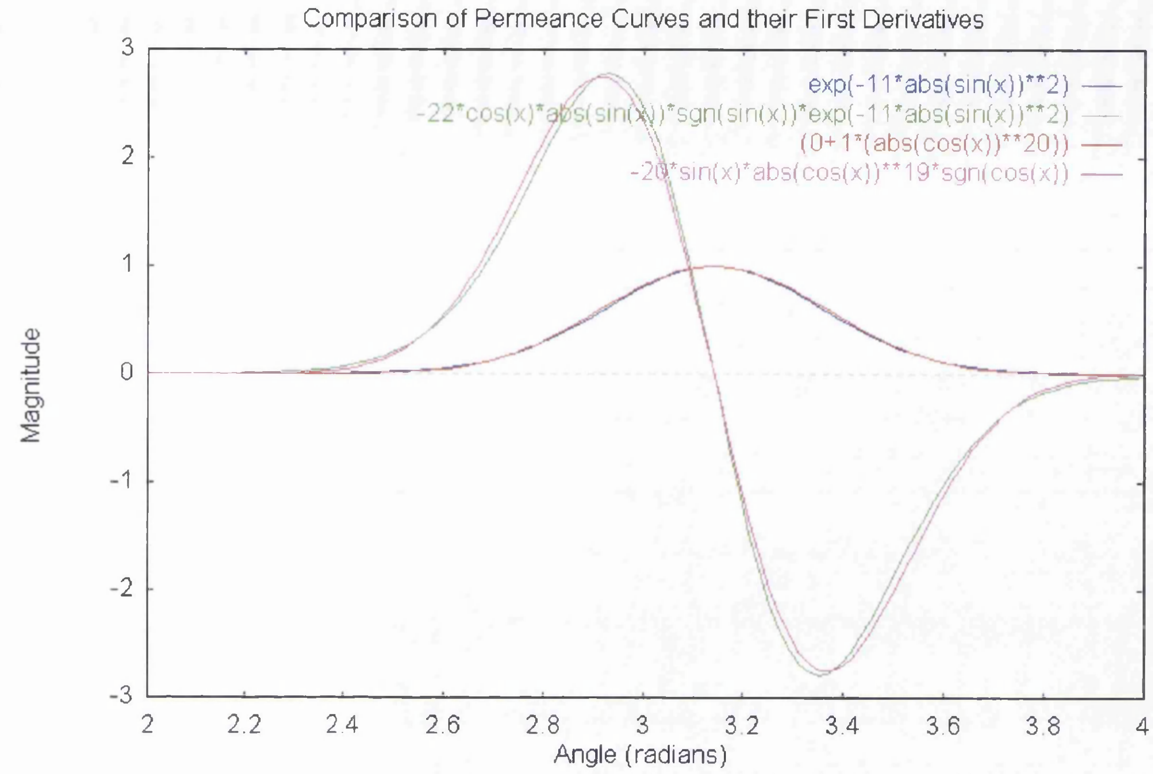


Figure 23: Comparison of permeance functions

In this particular case the equations f_1 & f_2 can used interchangeably. However in certain situations one or the other curve fit may not be possible.

The advantage in certain situations is that the equation, as presented in this thesis, is a better fit when there is a significant difference in rotor and stator pole area.

4.5 Representation of mechanical quantities

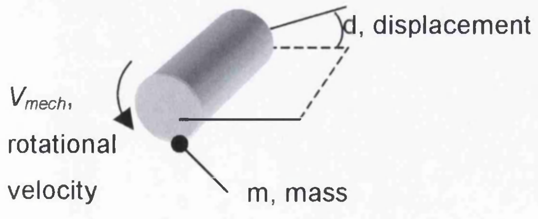


Figure 24: Typical mechanical quantities of a shaft.

The mechanical torque of a machine is given in Eq. 129 where position is obtained by integration of the velocity see Figure 24 for illustration. The inertia is proportional to the mass 'm', while the constant 'k' is proportional to torque divided by displacement. The voltage in an electrical LRC circuit is given by Eq. 130. The similarities between the two equations indicate how the mechanical quantities relate to their electrical equivalents.

Mechanical: $$T = R_{mech} v_{mech} + R_{wind} v_{mech}^{2} + J\frac{dv_{mech}}{dt} + kd \qquad (129)$$

Electrical: $$V = Ri + Ri^{2} + L\frac{di}{dt} + \frac{1}{C}\int i dt \qquad (130)$$

The torque generated by the magnetic circuit is calculated during the simulation and is used as a source voltage for the mechanical equivalent circuit. Current in the electrical equivalent is velocity in mechanical quantities; the integral of the velocity is fed back to the magnetic sub-circuit to allow the calculation of the dependent air-gap permeances.

4.6 Torque calculation

By application of the virtual work principle the torque can be calculated (Ostovic (20), Ratnajeevan et al (9), Demenko (17) uses:

torque $$T = \frac{\partial \mathrm{U}}{\partial \theta} \qquad (131)$$

From Eq. 28 energy in magnetic field is given by:

$$U = \frac{1}{2} mmf^{2} C_m(\theta) \qquad (132)$$

$\therefore$ Substituting Eq. 132 into Eq. 131 gives reluctance + Lorentz torque:

$$T = \frac{1}{2} mmf^{2} \frac{\partial\ C_m(\theta)}{\partial \theta} + mmf\ \frac{\partial\ mmf}{\partial \theta} C_m \qquad (133)$$

Assuming constant mmf gives:

$$T = \frac{1}{2} mmf^{2} \frac{\partial\ C_m(\theta)}{\partial \theta} \qquad (134)$$

To numerically calculate the partial differential of the magnetic permeance function, use is made of the central difference scheme:

$$\frac{\partial\ C_m(\theta)}{\partial \theta} \approx \frac{\left(C_m(\theta - \Delta\theta) - C_m(\theta + \Delta\theta)\right)}{2\Delta\theta} \qquad (135)$$

In the derivation of Eq. 134 it was assumed that the mmf is constant. In numerical calculations, the position perturbation must therefore be such that the change in mmf is as small as possible, while still producing numerically significant differences in permeance.

A numerical problem is that a small difference is added to, or subtracted from, the rotational position before the sine is taken. To improve accuracy for large numbers of revolutions or cycles the following minor change can be made to the sine function.

Using identity: $\sin(A \pm B) = \sin(A)\cos(B) \pm \cos(A)\sin(B)$

$$C'_m(\theta, \Delta\theta) = P_{\max} \cdot e^{-a\left|\sin(\theta/2)\cos((b+\Delta\theta)/2)+\cos(\theta/2)\sin((b+\Delta\theta)/2)\right|^c} \quad (136)$$

Provided the variation of permeance with respect to displacement is known, then applying a curve fit of Eq. 122 gives the plot illustrated in Figure 22. The numerical approximation to the derivative of the permeance can be found by applying the function in Eq. 136. This derivative curve is illustrated in Figure 25 and can be seen to be continuous, thus satisfying the non-discontinuity criterion.

An extension of the above technique can be made to find other forces within the system, provided the variation of permeance with displacement is known.

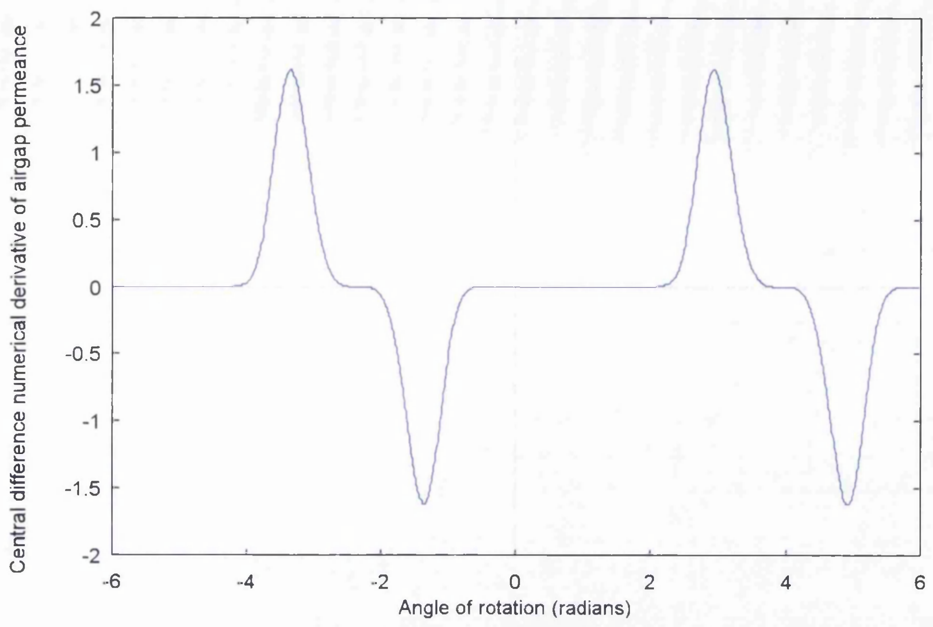


Figure 25: Central difference scheme used on relative permeance curve

4.7 BH curve fit for the non-linear magnetic permeance.

Some circuit simulators have a non-linear inductance model with hysteresis incorporated. A problem with using capacitors to model magnetic elements is that SPICE does not allow the transfer of non-linear curves belonging to inductors. A way to implement the required BH curve is therefore required.

Important criteria for the selection of an appropriate model of the BH curve were the continuity of first derivative, and compatibility with SPICE version 3. The curve used is an exponential function (Eq. 137) as described by Brauer (25).

Reluctivity $$\nu = k_1 e^{k_2 B^2} + k_3 \qquad (137)$$

The Brauer curve gives a closer fit, over a wider range, to the actual BH curve than does the polynomial curve fit. The implementation of the SPICE model of the Brauer exponential function is illustrated in Figure 26. This will impose higher processing time than a polynomial curve fit, but it would only be used where necessary in the model. Implementation of the Brauer function relies upon the iterative solution of the reluctivity. This could cause the simulator to give numerical errors and a lack of convergence but in practice, the particular simulator used for the test circuits, converged satisfactorily.

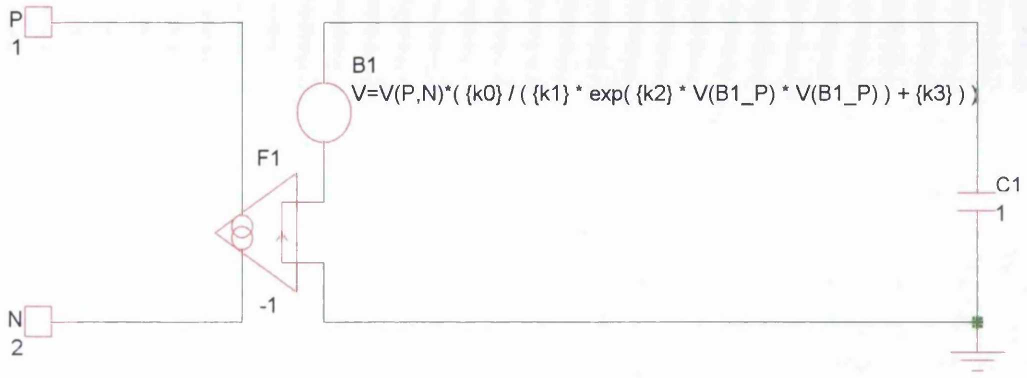

P
1
N
2
F1
-1
B1
V=V(P,N)*({k0} / ({k1} * exp({k2} * V(B1_P) * V(B1_P)) + {k3}))
C1
1

Figure 26: Implementation of non-linear permeance

The test circuit given in Figure 27 was used to obtain the results in Figure 28 with no hysteresis i.e. R2=0. This test circuit has no resistive losses and a BH curve of cold rolled steel was used. Note that in the following plots the ordinate is the voltage across the capacitor. This is the integral of current hence proportional to the magnetic circuit flux. The abscissa is the voltage across the non-linear capacitor, or in magnetic circuit terms the mmf.

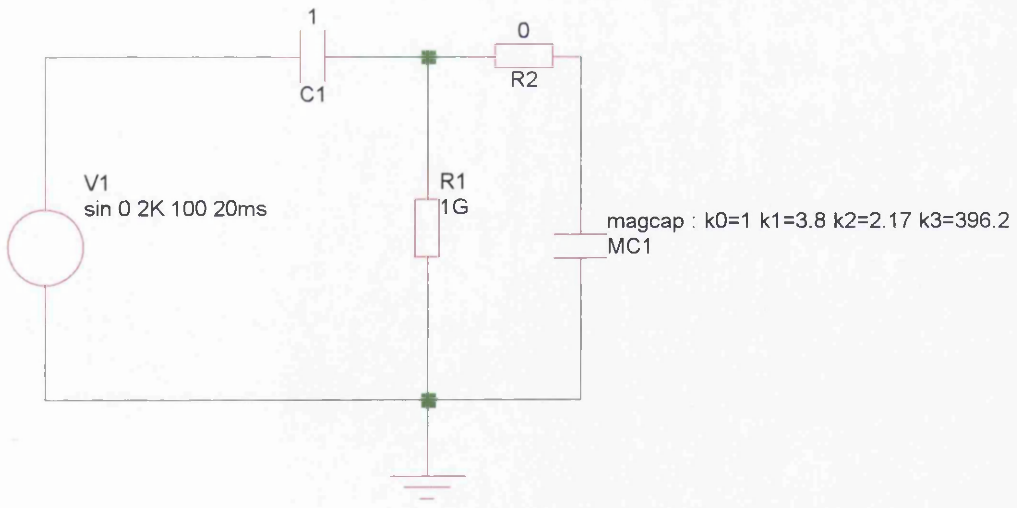


Figure 27: Test circuit for non-linear permeance

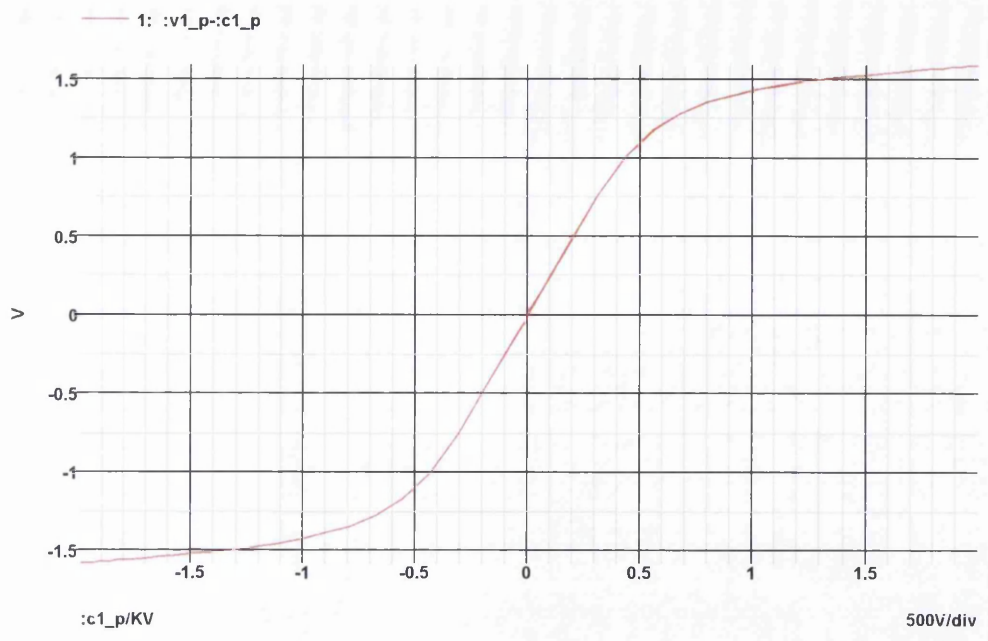


Figure 28: Graph of flux verses mmf, with no hysteresis

The effect of hysteresis and eddy current loss can be modelled by the incorporation an additional series resistive element (R_2). The method used to calculate the value of the loss resistance is to use the data available from the manufacturers who describe the loss per Kg at a fixed frequency. A typical value needs to be calculated at the expected operating point. This is not an ideal solution as the relationship is in fact non-linear, but gives an adequate approximation for simulation. Example curves with additional resistance are shown in Figure 29 (R=0.25), Figure 30 (R=0.5) and Figure 31 (R=1). It can be clearly seen that the additional resistance models the hysteresis effect of the magnetic circuit, thus validating the model. The greater the resistance the larger the hysteresis effect.

Figure 32 shows a circuit used to test the effect of an ac signal upon a dc value and the resulting incremental hysteresis curve. The curve in Figure 33 shows the mmf drop across the permeance rising to the fixed operating point. Figure 34 shows more clearly details of the incremental hysteresis loop.

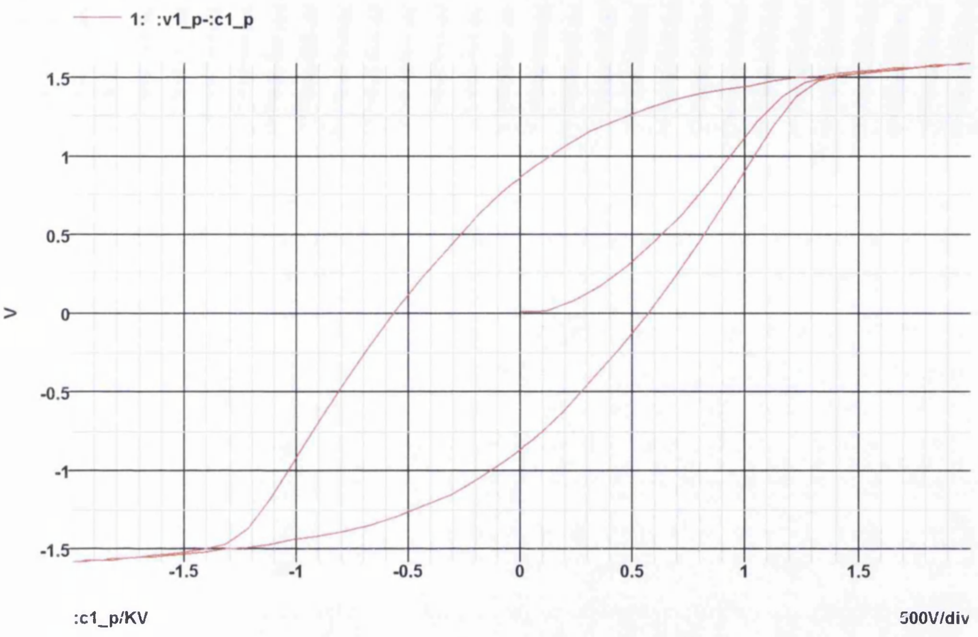


Figure 29: Graph of flux versus mmf with hysteresis (R2=0.25)

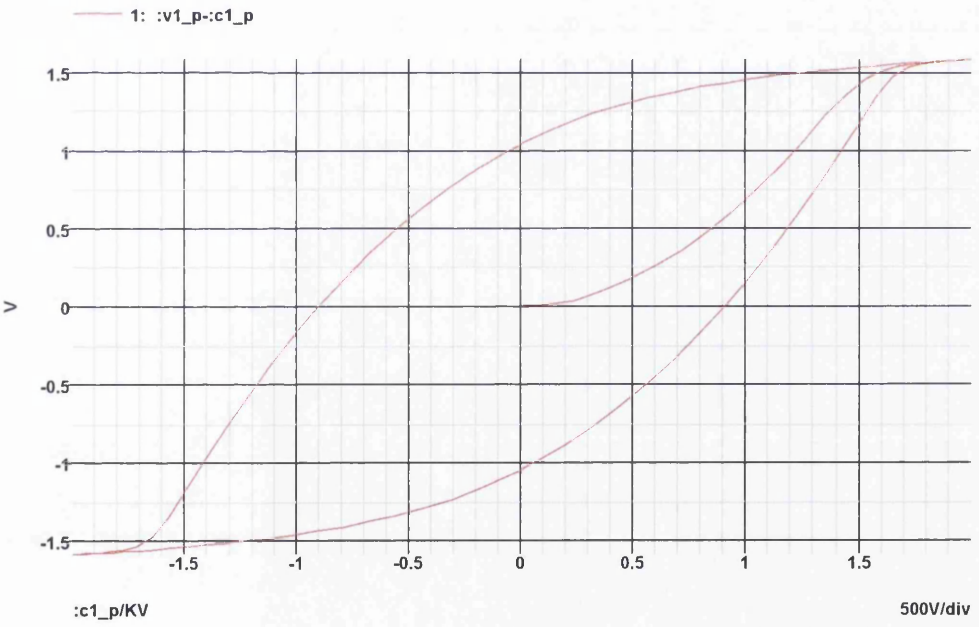


Figure 30: Graph of flux versus mmf with hysteresis (R2=0.5)

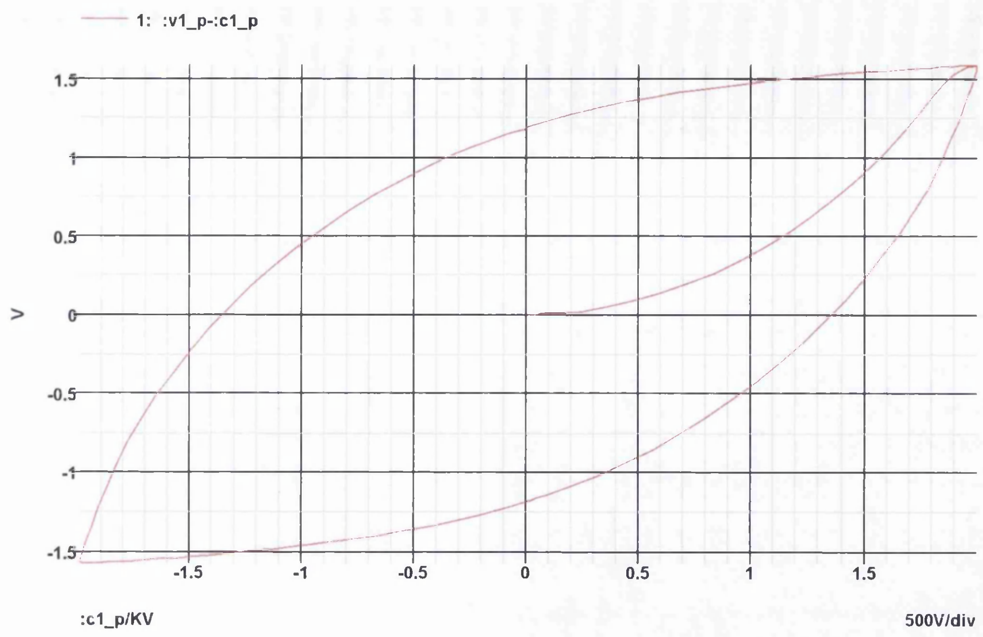


Figure 31: Graph of flux versus mmf with hysteresis (R2=1)

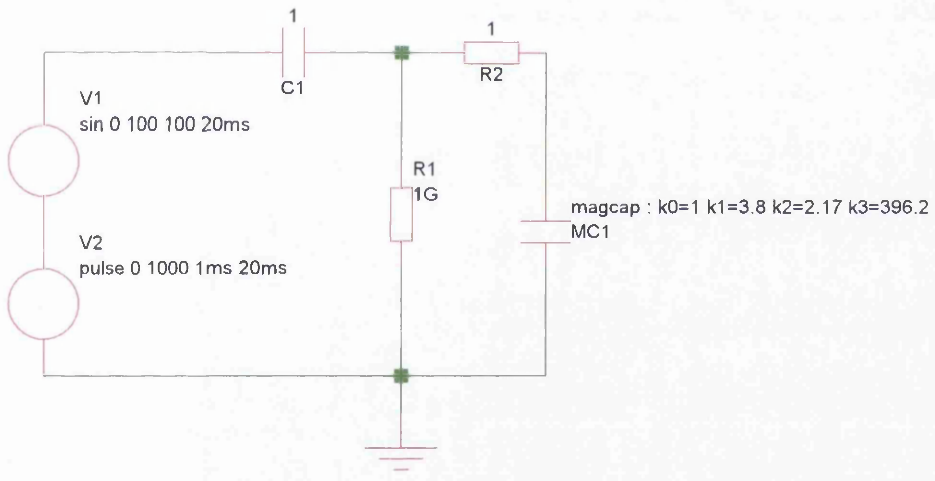


Figure 32: Incremental magnetic hysteresis loss test circuit

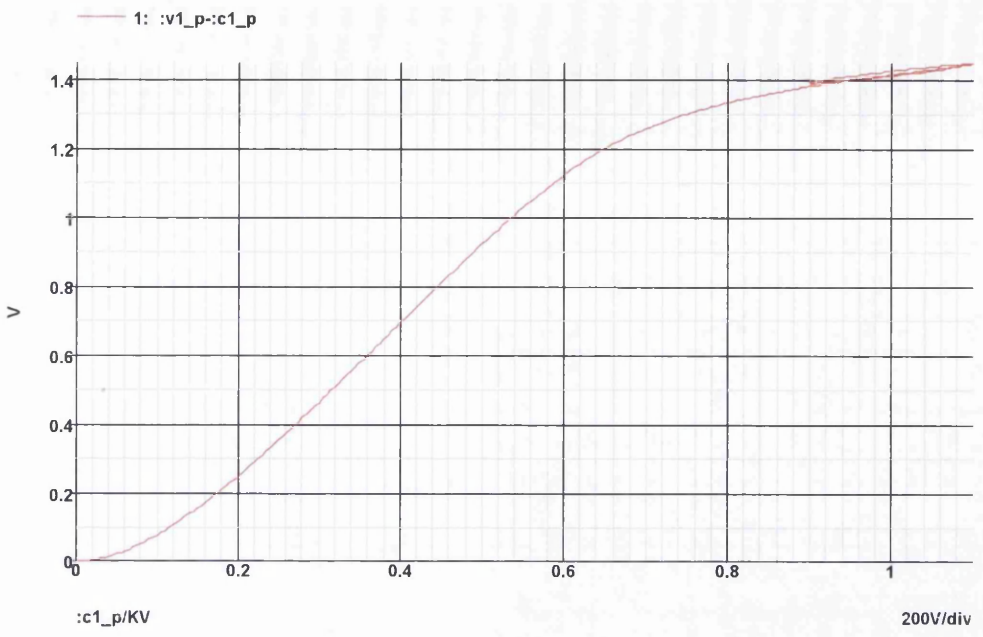


Figure 33: Graph of flux versus mmf with Incremental hysteresis

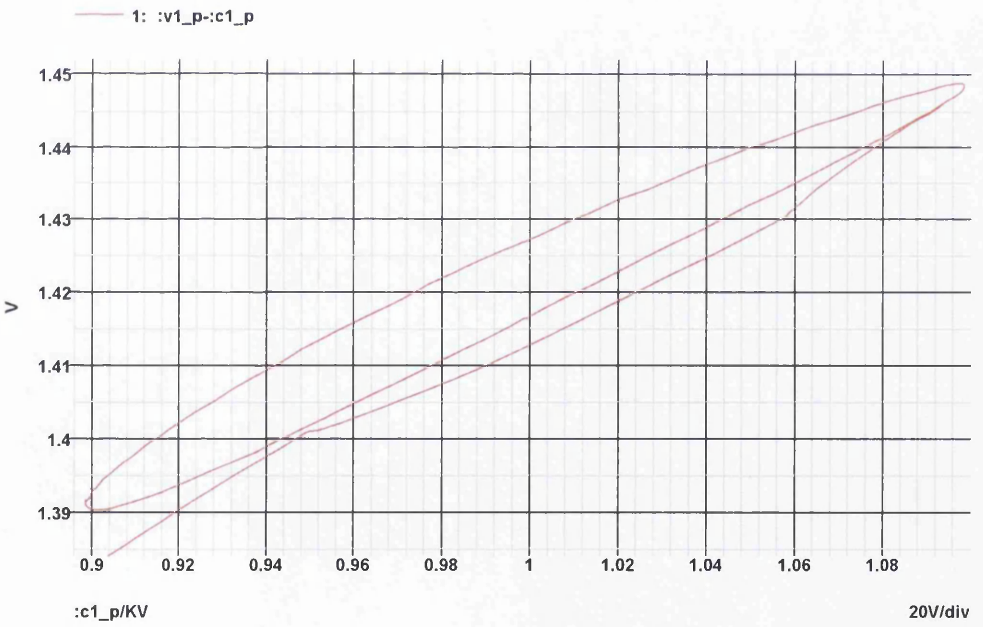


Figure 34: Detail of incremental hysteresis (flux versus mmf)

4.8 Calculation of the permeance

In Eq. 138 if the reluctance of a certain flux path can be calculated, the permeance by definition is known. The permeability of the area of integration is assumed uniform, permeance is therefore taken out of the integration and can be modelled separately.

$$\frac{1}{C_m} = \frac{1}{\Lambda} = S = \int_0^w \frac{dx}{\mu(x)A(x)}, \quad \therefore \frac{1}{C_m} = \frac{1}{\mu}\int_0^w \frac{dx}{A(x)} \qquad (138)$$

In Eq. 138 *w* is the length along the flux path and *A* is the area perpendicular to the flux path. By inspection of the above equation, a simple numerical algorithm is possible. Each flux path with in the machine is given boundary curves with designated flux entry and exit curves. In Figure 35, the flux entry and exit curves are designated as AB and EF respectively.

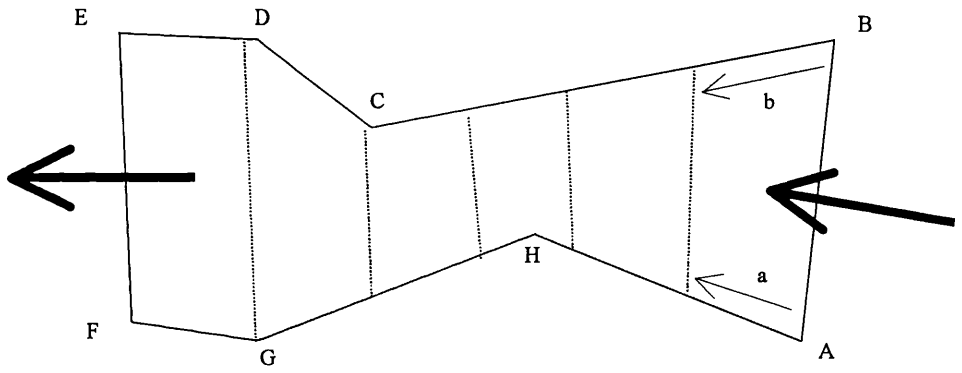


Figure 35: Example of integration path

The following procedure is performed on each part that represents a flux path:

1. Calculate the scalar length of each side of the path i.e. BE and AF.

2. Divide the scalar length by the number of sections required, to arrive at values a and b.

3. Step along each side of the path, starting with AB, and perform the sub integral.

4. Add all sub integrals to give final reluctance value (invert to give the permeance value used at simulation time).

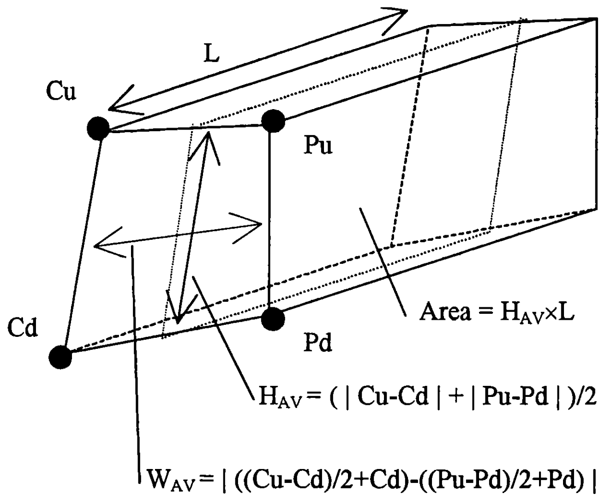

L
Cu
Pu
Cd
Pd
Area = H_{AV}×L
H_{AV}= (| Cu-Cd | + | Pu-Pd |)/2
W_{AV}= | ((Cu-Cd)/2+Cd)-((Pu-Pd)/2+Pd) |

Figure 36: Line integral of subsection

The sub-integral for each subsection of the part is illustrated in Figure 36 (relationship to the flux path for the part is shown in Figure 37). The nodes denoted by Cu, Cd, Pu and Pd are arrived at within the previous algorithm by stepping along the section sides. Cu and Cd are the current upper and lower nodes respectively. Pu and Pd are the previous upper and lower nodes respectively. The length L is the length of the rotor or stator. See Figure 37 for an example of finding the permeance of an arbitrary flux path.

From Eq. 138 $$\therefore \int_{PuPd}^{CuCd} \frac{dx}{A} \approx \frac{W_{AV}}{LH_{AV}} \qquad (139)$$

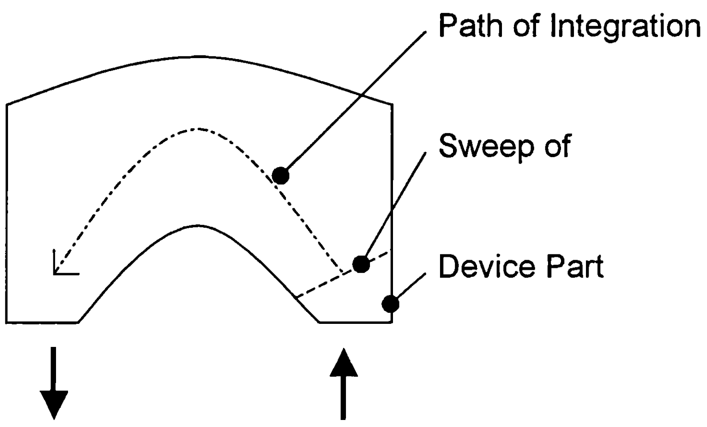


Figure 37: Example of arbitrary shape integration

A problem arises with start and end (flux entry and exit) curves that are not simple lines. The integration will miss out a section of the part that is on the start side, of the first integration line. The same problem arises also with the end curve. This needs to be taken into account when defining the flux path in terms of the shapes of the parts in the frame.

A possible solution to this problem is instead of sectioning the part along its length (based on the reluctance being summed for the part); the permeance is calculated directly. This would involve applying Eq. 140 and using sections that span the length of the part.

$$C_m = \Lambda = \int_0^h \mu(x)\frac{L(x)}{W(x)}dx, \quad \therefore C_m = \mu\int_0^h \frac{L(x)}{W(x)}dx \qquad (140)$$

In Eq. 140 *h* is the height of the part, *L* is the length perpendicular to the flux path and W is the length along the flux path.

4.9 Procedure to numerically find position equation coefficients

The permeance values for the air gap are dependent on the rotor angle relative to the stator. A problem therefore arises as to how to derive the equation coefficients from the machines geometrical data. Ostovic ((20) pp.36) uses a custom solver and is therefore able to use a piecewise function that implements a sine approximation. For the purposes of the present exercise, the calculation of correction factors for rotor eccentricity will be neglected. It will also be assumed that the air-gap is small and therefore the effect of fringing can be neglected. To accurately include the effect of fringing, on the flux path, a more sophisticated field solution would be needed e.g. Van den Bossche (38).

The optimum values of coefficients for use in Eq. 122 i.e. reduce execution time include that the exponent *c* must be an integer and as small as possible. The value of *a* should therefore be the larger value that gives a solution. Because of the complexity of the function, a relatively simple search scheme is used to find suitable coefficient values. This scheme is as follows:

1. Find the relative angles of rotor and stator curves that are designated as field entry or exit and are adjacent to the air-gap.

2. Find the mid point of the curves on both the rotor and the stator so that the relative offset values can be calculated.

3. Use the machines value of skew to give the lower and upper permeance curve positions, in the example given positions for 10% and 90% of maximum permeance.

4. Step through values for equation coefficients until suitable values found, or abort if coefficient limits reached. The search is performed by incrementing the integer *c* from 1 to the maximum allowable. After each increment of *c* the value of *a* is incremented by a suitable step size until the function lies within the 10% and 90% permeance limits, or the next value of c is needed

A test program of the coefficient search algorithm is shown below in Figure 38.

```
/* simple program to test solving of the rotional equation coefficients
   Author: D. Downes TNTU */
#include <math.h>
#include <stdio.h>
#include <conio.h>
#define MAX_POWER 13
#define MAX_EXP_MAG 13
#define STEP_FACTOR 100
#define SKEW1 0
#define SKEW2  6
#define CURVE1 3
#define CURVE2 6
int main (void)
{
   long c, b;
   double a, result, res_top, res_bot, change, bot, top;
   double skew1 = SKEW1;
   double skew2 = SKEW2;
   double curve1 = CURVE1;
   double curve2 = CURVE2;
   FILE *out;
   printf("open output file: %s\n","out.txt");
   if ((out = fopen("out.txt", "wt"))=="out.txt") {
     printf("Cannot open output file: %s\n","out.txt");
   return 1;
 }
   // find position equation coeff.
   change = ((curve1<curve2)?curve1:curve2)*2.0;
   change += (skew2-skew1);
   change = (change<(curve2+curve1))?change:(curve2+curve1);
   bot = (curve2+curve1+fabs(skew2-skew1));
   top = fabs(bot-change)/2.0; /* point at which 90% of max.*/
   bot /= 2.0; /* point at which 10% of max.*/
   fprintf(out,"Results of calc:\n");
   for (b=1; b<MAX_POWER; b++) {
      for (c=1; c<(MAX_EXP_MAG*STEP_FACTOR); c++) {
         a = powl(10,(double)c/STEP_FACTOR);
         res_top = exp(-1*a*pow(fabs(sin(M_PI*top/360.0)),b));
         if (res_top>0.9) {
            res_bot = exp(-1*a*pow(fabs(sin(M_PI*bot/360.0)),b));
            if (res_bot<0.1) {
                fprintf(out,"skew1: %2.4f, curve1: %2.4f, skew2: %2.4f, curve2:
%2.4f\n",skew1,curve1,skew2,curve2);
                fprintf(out,"therefore top: %2.4f, bottom: %2.4f\n",top,bot);
                fprintf(out,"result top: %1.3f, bottom: %1.3f\n",res_top,res_bot);
                fprintf(out,"b: %i, ",b);
                fprintf(out,"a: %3.3e\n\n",a);
               fclose(out);
                return 0;
             }
          }
       }
    }
    printf("Finished\n");
    fclose(out);
    return 0;
}
```

Figure 38: Program to test permeance coefficients algorithm

An example test run of the program is shown in Figure 39. The program has displayed values only when a valid result was found. Note that coefficient *c* in the equation above is referred to as coefficient *b* by the program. The resulting function is plotted in Figure 40.

```
Results of calc:
skew1: 0.0000, curve1: 3.0000, skew2: 6.0000, curve2: 6.0000
therefore top: 3.0000, bottom: 7.5000
result top: 0.943, bottom: 0.100
b: 4, a: 1.259e+05
```

Figure 39: Output of test program

A significant problem with the function, as described, is there is not always a satisfactory solution that for the 10-90% curve fit. Other curves would probably be better in certain situations, particularly if a finite element solution of the air gap flux is available.

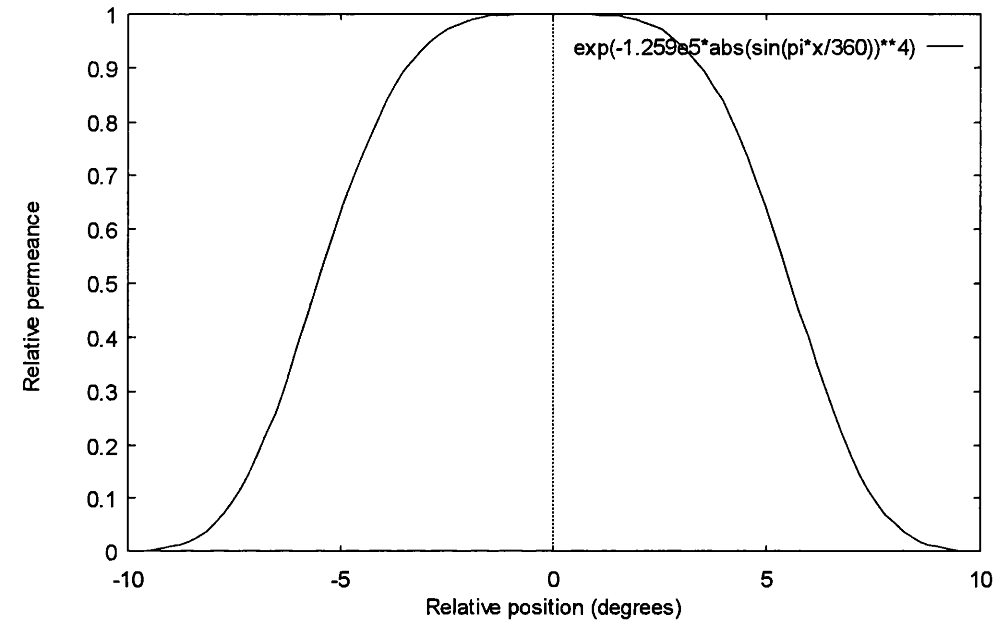


Figure 40: Plot of the resulting position equation

An improved version of the algorithm has also been developed. This relies on the principle that the value for the co-efficient ‘a’ can be directly calculated from an analytical solution of the equation. Therefore the exponential co-efficient ‘c’ is incremented and the value that most closely matches the desired curve is the one selected.

4.10 Conclusion

Equivalent circuits have been investigated that allow the implementation of MEC models within a standard circuit solver.

Non-linear capacitors can be implemented using voltage controlled voltage sources, a current controlled current source and a standard capacitor.

A new air-gap permeance modulating function (Eq. 122), which has not been used before, has been described and is shown to be compatible with SPICE version 3.

Permeance function $C_m(\theta) = P_{\max} \cdot e^{-a|\sin((\theta+b)/2)|^c}$ (122)

An additional improvement to the use of such functions (Eq. 136) is given which allows a greater simulation span, before numerical problems occur.

$$C'_m(\theta,\Delta\theta) = P_{\max}\cdot e^{-a\left|\sin(\theta/2)\cos((b+\Delta\theta)/2)+\cos(\theta/2)\sin((b+\Delta\theta)/2)\right|^c} \quad (136)$$

The similarity between the voltage equations for an electrical circuit and the torque equation for mechanics is noted. It is inferred that this allows the incorporation of the mechanical elements using electrical equivalents.

By use of the virtual work principle and central difference scheme, it is shown how the force or torque can be obtained numerically. This also allows additional forces within the machine to be obtained.

Due to the need to incorporate the magnetic BH characteristic within the simulation, use of the Brauer curve fit is investigated. This is implemented as a non-linear capacitor.

The determination of the permeance values via a numerical scheme is presented. This scheme although applied analytically to standard elements, it has not been found elsewhere applied numerically for the determination of the permeance. An improvement to the algorithm is suggested to remove the error at the start and end of the integral.

The air-gap permeance function already described, needs to be related to the geometry of the machine. Neglecting field effects such as fringing and slot width, a suitable algorithm for this is described. The inclusion of factors to allow for such effects could be incorporated into the algorithm, but comparison with field solution methods would be needed. A method of determining the numerical values for the air-gap permeance function is also given.

5 Software development

5.1 Introduction

The previous chapter described model elements that are used in the SPICE simulations. Figure 41 shows model generation, within the context of the overall simulation procedure. The input to the conversion program is a textual description, and output is the SPICE sub-circuit of the machine.

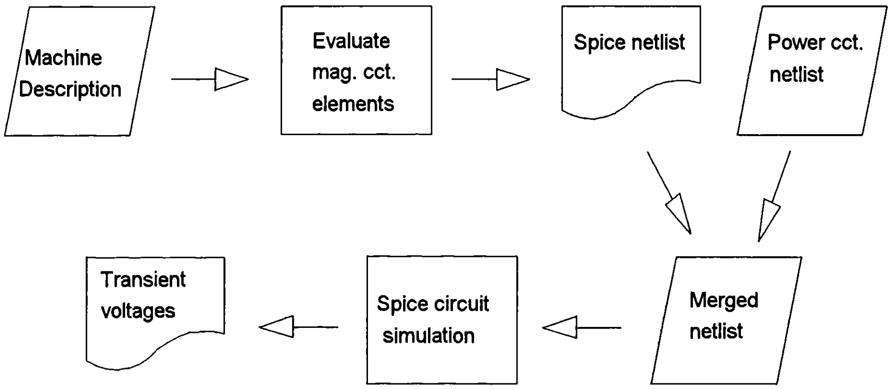


Figure 41: Relationship between GDL file and spice circuit

The description of the machine used by the computer needs a strict syntax and structure. In the following the particular syntax and structure is presented. The syntax includes a method for specifying repeating elements.

The conversion of the flux paths into the SPICE sub-circuit node numbers will be described. This is essential to the successful semiautomatic generation of the machine sub-circuit.

Development of the software is based on the decisions already taken in the previous chapter. The program, full listing is given in appendix H, takes as its input the geometry description of the flux paths. How the program fits into the overall simulation cycle is shown in Figure 41.

The way that the description of the flux paths is split into frames, parts, curves, etc. lends itself to a relatively simple structure with in the program. The requirements and structure of the program will be described in the following section. Descriptions of various important aspects will be expanded upon, but an exhaustive description of structure and operation is not given.

5.2 Representation of the machines physical geometry

For the machine equivalent circuits to be generated there needs to be some form of physical geometry and flux path representation. The representation chosen is that of a hierarchical set of geometry definitions. The base unit is the node that defines a co-ordinate point in space. All other definitions are built up from references to these, or higher definitions. The use of the method is restricted to that of two dimensions with two basic structures, the stator and rotor. The stator is fixed with respect to the global reference frame with the rotor rotating about the concentric centre.

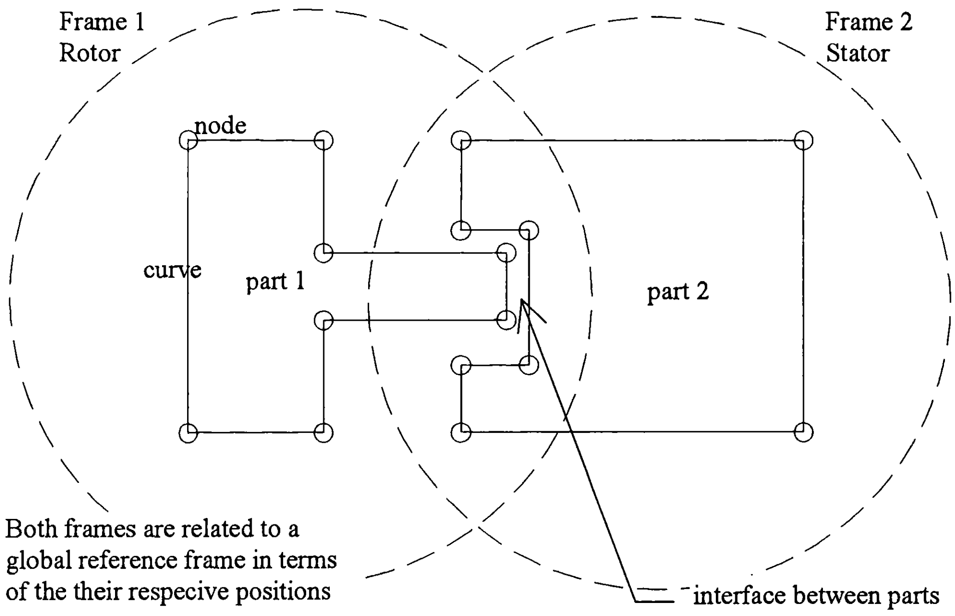

Frame 1
Rotor
Frame 2
Stator
node
curve
part 1
part 2
Both frames are related to a global reference frame in terms of the their respecive positions
interface between parts

Figure 42: Relationship between nodes, curves, parts and frames

Figure 42 shows how the various elements of the description relate to each other. Frame 1 and Frame 2 are the rotor and stator frames and these define how the parts contained within one frame move with respect to parts in the other frame.

The interface between the frames defines the air gap; this is where the elements that allow for the relative motion are situated.

In deciding upon the types of curves to be used, it was decided that lines and arcs would be used. These are the most predominant shapes used in rotating machine designs. The following summarises the description hierarchy:

- The frame is defined as containing nodes and parts.
- The parts are defined as consisting of curves. Each part has various attributes such as if it is a flux path, its BH characteristic and whether it is a mmf chord of a coil. Given a 2 dimensional cut is taken though a coil, the mmf chord is the line that connects one side of the coil and the other.
- The curves have a starting node, end node and information as to whether they are an arc or line.
- The nodes supply which of the associated reference frames (i.e. stator or rotor) they belong.

See Figure 43 below for the full syntax of the description used (this is an improved version of the syntax given in the paper included as Appendix B).

Syntax of Geometry Description Language (GDL) for Rotational Geometry.

```
bh { < name >  < linear | exp > < value [ value  value ] > }
coils { <coil_name> <value> <terminal_name> <terminal_name> }
< stator | rotor > { nodes { < node_name [ [ start_integer  no_of_repetitions  inc_integer  ] ] >
  < value | [ start_float  inc_float ] > [ d ]  < value | [ start_float  inc_float ] > }

  curves { < curve_name [ [ start_integer  no_of_repetitions  inc_integer ] ] >
  < line | circ > < node_name [ [ start_integer  inc_integer ] ] >
  <node_ name [ [ start_integer  inc_integer ] ] >
   [  < value | [ start_float  inc_float ] > ] }

  parts {< name > {
   bh < name >
   curves { < curve_name [ [ start_integer  no._of_repetitions  inc_integer ] ] > }
   flux {coils { <coil_name> }
   entry  { < curve_name [ [ start_integer  no._of_repetitions  inc_integer ] ] > }
   exit  { < curve_name [ [ start_integer  no._of_repetitions  inc_integer ] ] > } }
   }
  }
}
```

Key: < > Obligatory item[s] enclosed

[ ] Optional item[s] enclosed

| Alternatives separator

name First character must be a letter with any combination of alphanumeric characters can follow.

value Floating point numeric value.

no._of_repetitions Integer giving number of times that the item is to be repeated.

start_integer Integer starting value for naming sequence.

inc_integer Integer increment in naming sequence.

start_float Floating point start value for value sequence.

inc_float Floating point increment for value sequence.

Note:

- Non-italic characters are literal that is they are used exactly as shown.

- '{' and '}' signify that a list of items between the brackets is possible.
- A '#' character placed in the text file denotes a comment, and any characters after it up to the next newline character are ignored.

Figure 43: Syntax of machine description

The syntax allows for the repetition of nodes, curves etc. by an incremental numbering scheme. This means the specification of an arc of nodes a certain distance apart needs only one statement.

The descriptions of nodes, parts or curves can be in any order but if a curve references a node the node must already be defined. This restriction, although minor, does allow the implementation of a relatively simple parser.

5.3 Assignment of node numbers

To generate the spice circuits a procedure for circuit node number allocation needs to be defined. The node numbers are based on the flux path topology. A unique number needs to be assigned to each node, to uniquely define branch connections. Each number represents a set of parts that join at their flux entry or exit curves. The assigned numbers are therefore a property of the parts entry and exit curves. The procedure chosen is an iterative assignment and search:

1) Assign a number to the first part with a flux entry or exit curve.
2) Find all connected flux entry or exit curves that join, and assign the same number to the associated part. If a connected part already has a number assigned to it, then an error is reported and the procedure aborted.
3) Find flux part with flux entry or exit curve that has no number assigned. If no such part can be found then finish, else repeat step (2).

If a valid geometrical description has been entered, there should only be air gap nodes with unique numbers assigned. This therefore suggests a method of finding the air gap interface nodes. All nodes that are assigned to only one part are automatically attached to an air gap element. If more than one air gap interface are possible then an indication needs to be given as to how the air gap region relates to the air gap magnetic elements and the relative mechanical freedom of motion.

5.4 Program requirements and structure

The objective of the program is to read in the text file description and transform it into a SPICE sub-circuit. A list of the description language syntax rules have already been given, apart from these, another restriction is that standard ASCII text be used.

The program generates a spice model listing by using the following procedure:

1. Check that all the parts (flux paths) are valid connections of curves.
2. Assign circuit node numbers to the interfaces between flux paths.
3. Calculate line integrals of the flux paths.
4. Find suitable values for the air-gap permeance coefficients.
5. Output the circuit model to a text file with spice sub-circuit start and end text.

It was decided to use Object Orientated Program (OOP) design and programming. It was believed that such a technique would simplify the implementation of the program. This is mainly due to the complex relationship between the elements of the machine description. The traditional approach would be to separate the data sections from the related algorithms.

A penalty of using the object-orientated approach is that the execution time is probably longer, than would be the case for a more traditional approach. In this particular instance, this does not cause difficulty as the program is only run once to generate a model file for use by the simulator.

5.5.1 Overall program structure

With the use of OOP, the obvious choice is to make each element (frame, part, curve etc) of the description an object, with its own data and methods. The links between the various objects, i.e. what curves belong to a particular part, are stored as linked lists. The lists themselves are implemented as container objects with methods for searching through the linked list pointers.

The various linked lists and how they associate the various objects are illustrated in Figure 44, this shows the class structure and class/object relationships.

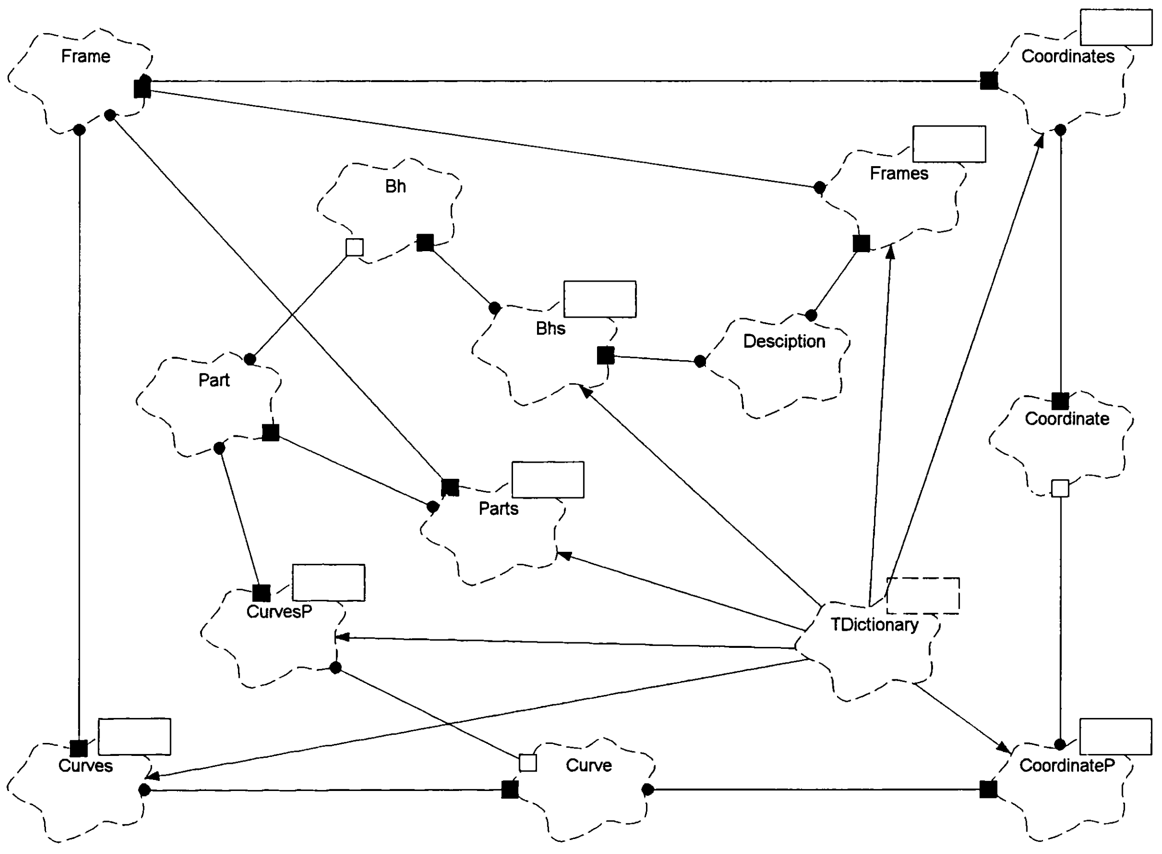


Figure 44: Relationships between the various objects used in program

5.5.2 Text conversion (parsing)

In order to convert the machine description into a sub-circuit, the entered text needs to be converted into a suitable internal computer representation. This is the purpose of the parser. The text parser is distributed through out the program with each element (frame, part, curve etc) having its own generic parser method, with each parser method implemented as a state machine particular to that class of description element. Use could have been made of parser generation tools (such as YACC or BISON), but it was thought it would complicate the program structure. A lexical analyser (recognition of keywords, characters and numbers) is shared between all the parser methods and is a defined as a separate object.

In operation, the top-level method for the application opens the text file. It then starts stepping through the text until it finds certain keywords. Once found it invokes the general parser for that object (frame, part, curve etc). The parser method adds the new object, with appropriate properties, to the relevant list. Parsing stops when the end of the file is reached.

5.5.3 Processing linked data

The lists of links to other objects, used by each object, are objects themselves and provide methods for list processing (Baase (27)). Thus there is an internal representation in memory of the machine description contained in the description file. Methods provided by the list objects include searching for a specific object and chaining or sequencing along the list.

As an example of the processing, the procedure of building the electrical sub-circuit will be examined. When the conversion method is called by the application, the various other object methods are called for specific calculations. These calculations are to do with aspects of the data that the object concerned has access. Generation of the sub-circuit starts with the description object. This calls the conversion procedure for the stator and rotor frames (Figure 45). These call the conversion procedures for the associated parts in the frames linked list (Figure 46). The parts conversion procedure converts its associated data into a circuit permeance element. In the process of performing this conversion, a call is made to its own line integral method. The line integral method calls the scalar length methods of the relevant part's listed curves.

```
cct << "************************************\n";
cct << "* Stator magnetic circuit elements *\n";
cct << "************************************\n";
ok &= stator.Convert(cct, sections);
cct << "***********************************\n";
cct << "* Rotor magnetic circuit elements *\n";
cct << "***********************************\n";
ok &= rotor.Convert(cct, sections);
cct << "******************************************************\n";
cct << "* Connection of stator coils to circuit terminations *\n";
cct << "******************************************************\n";
CoilsConvertElectrical(cct, stator.GetCoils()); // convert to coil circuit description
cct << "*****************************************************\n";
cct << "* Connection of rotor coils to circuit terminations *\n";
cct << "*****************************************************\n";
CoilsConvertElectrical(cct, rotor.GetCoils()); // convert to coil circuit description
cct << "*************************************\n";
cct << "* Magnetic to mechanical conversion *\n";
cct << "*************************************\n";
ok &= FrameGenerateMech(cct, stator, rotor, sections);
cct << ".ends\n\n\n\n";
```

Figure 45: Part of conversion method within description object

```
bool Frame::Convert(ofstream& cct, long sections)
{
  bool ok=true;
  PartsIterator partsIter(parts);
  while (partsIter != 0) {
    ok &= partsIter.Current()->Convert(cct, sections);
    partsIter++;
  }
  if (cctNodes.GetItemsInContainer()>0) {
    cct << "* Grounding resistors:\n";
    CctNodesIterator nodesIter(cctNodes);
    while (nodesIter != 0) {
      cct << "RGND" << (nodesIter.Current());
      cct << " " << (nodesIter++) << " 0 1G\n";
    }
  }
  return ok;
}
```

Figure 46: Frame conversion method

Generation of the air-gap elements is performed by a generic method called 'FrameGenerateMech', this again is listed in the appendix.

5.5.4 Output of sub-circuit

Once the sub-circuit nodes and flux path line integrals have been calculated, the SPICE compatible text file can be generated.

The permeance elements can either be linear or exponential functions. Depending on the type of assigned function the BH curve object either generates a constant permeance or substitutes the non-linear permeance sub-circuit.

5.5.5 Saving of linked data

A facility is incorporated that allows the internal data to be output as a text file. The parser can also read the resulting file. The file consists of the full description of the machine, but with no repetition syntax. This allows a check to be made on an input file that contains repetition syntax. Each object: frame, part, curve etc. has a method defined that will output a text description of its values. The application calls each object in turn and directs the output to a text file.

5.5.6 Visual check of geometrical data

Plotting of the stator and rotor curves results in a visual check of the description, as read in by the program. Screen captures of the program output are shown in the next chapter.

5.5.7 Interpretation of circuit simulation data

The program automatically generates the required instructions (Appendix E) to retrieve the simulation data needed to construct the flux density plots for the magnetic sub circuit. This data is then read in by the program and used in the generation of time stepped flux intensity plots (Sample circuit simulation data Appendix F).

5.6 Software implementation

The software was programmed using Borland C++ under Microsoft Windows and GCC 2.95 under Linux. The number of sections used in the line integral is alterable using an options dialogue box, the higher the number of sections then the higher the accuracy of the integration.

5.7 Conclusion

A method of entering the machine geometry and other details in text form was required. The solution used is a relatively simple description syntax that allows repeated items to be easily entered. The syntax of the description language is presented.

Before the conversion to the sub-circuit model, the circuit node numbers need to be assigned. A procedure is presented that allows this and determines which of the flux paths connect to an air gap element.

The program requirements have been given, where the main objective is the conversion of the text description to a SPICE sub-circuit. This has been achieved using the methods given in the previous chapter.

The program uses object orientated programming as this was thought to make design and analysis easier. The benefits of using this method of design and programming to some extent have been realised, but the benefits are not as great as the author had hoped.

Some of the design structure is given, with details of the way linked lists are used to store the relationships between the various elements of the machine description. An example of the program is given to illustrate the use of the linked lists. Comprehensive details are not given as the program implements the procedures and algorithms already described in the previous chapters.

6 Simulation Results

6.1 Generator simulation

In this section, an example generator was converted from a text description of its geometry and material to a spice sub-circuit. The sub-circuit will then be combined with a wiring description, to provide a combined sub-circuit for use within a test circuit. The simulation will provide a means to illustrate the use of the software and at the same time provide evidence of model consistency. By matching the power flows between the mechanical, magnetic equivalents and electrical circuits the model consistency was checked.

Using the conversion program generated sub-circuit the compatibility with a standard (SPICE) circuit simulation package will be tested. The package used is SIMETRIX (26) which implements the SPICE (version 3) circuit description syntax.

The resulting merged circuit description is suitable for use in any 'Spice 3f' compatible circuit analysis program. A flowchart showing the steps involved in the generation of the model, circuit simulation and capture of the transient data is shown in Figure 41.

6.1.1 Construction and wiring

The following generator design is a simple three phase synchronous machine. Leakage paths between stator teeth are included, as is leakage around the periphery of the rotor. The permeance of the magnetic circuit is assumed linear (i.e. no iron losses).

Mechanical sources are provided that provide torque and hence rotor rotation. Excitation voltage of the field coil will be constant dc. The load on the generator will be a set of three resistors in star configuration.

6.1.1.1 Stator construction

The length of both rotor and stator is 96mm, while the outside diameter of the stator is 76.45mm. Permeances on the periphery are connected together, thus allowing back iron flux. The permeances of the back iron also connect with the stator pole pieces. Pole pieces are placed between the coil slots. See Figure 47 for an illustration of the above. In the figure the light grey areas are the back iron sections. Dark grey the pole pieces and the light blue denotes the coil slots. The mmf sources for the coil chords are placed within the stator poles.

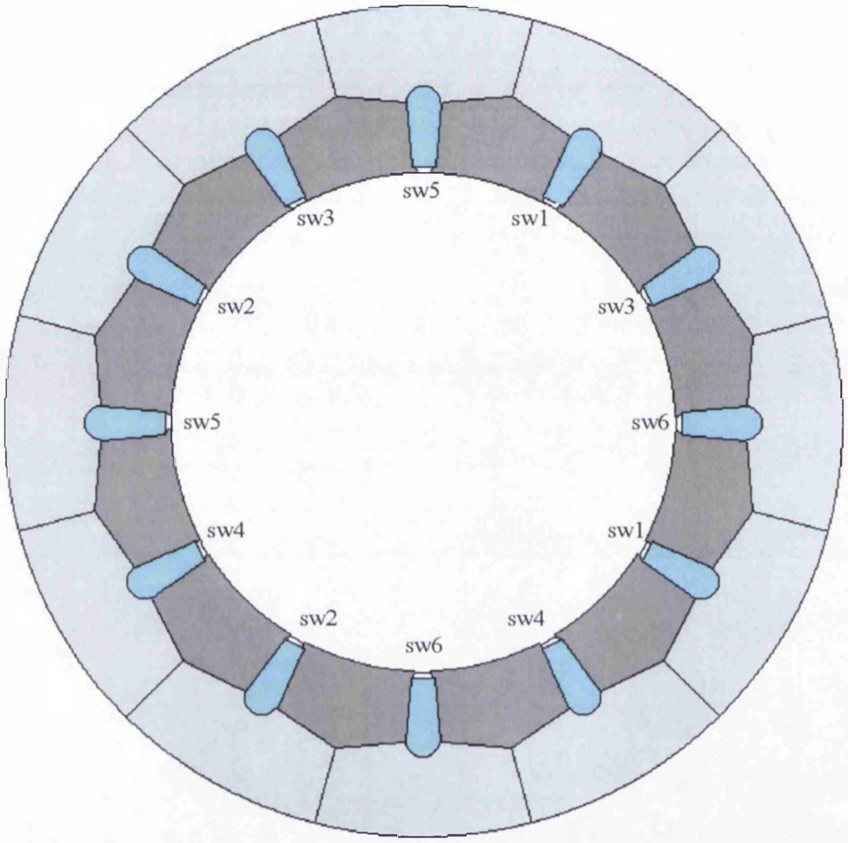


Figure 47: Stator construction and coil details

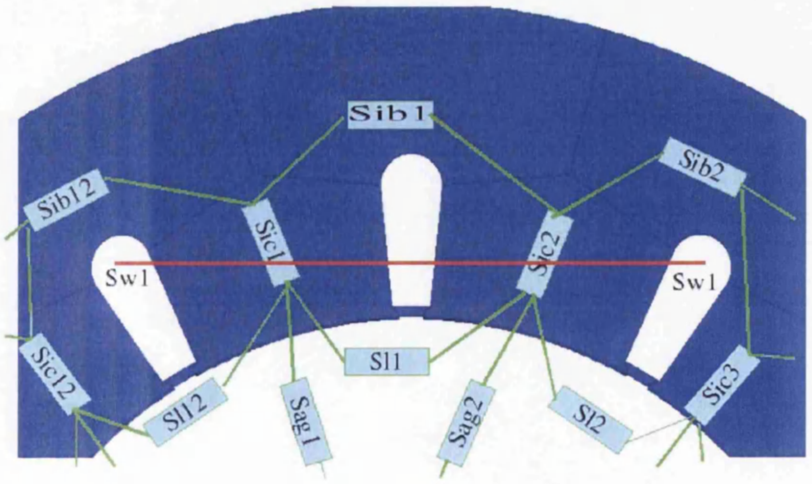


Figure 48: Flux paths (note coil sw1 not shown in actual position)

Figure 48 shows the details of the simplified flux paths chosen for the simple generator. Note that the coils 'Sw1' is not shown in its actual position but with a shortened length, for the purpose of illustration. 'Sib1' to 'Sib12' are the back iron flux paths. 'Sic1' to 'Sic12' are the iron parts, which intersect the coils contained in the slots ('Sw1' being an example coil which intersects parts 'Sic1' and 'Sic2'. 'Sl1' to 'Sl2' are the stator leakage paths and 'Sag1' to 'Sag24' are the airgap flux paths.

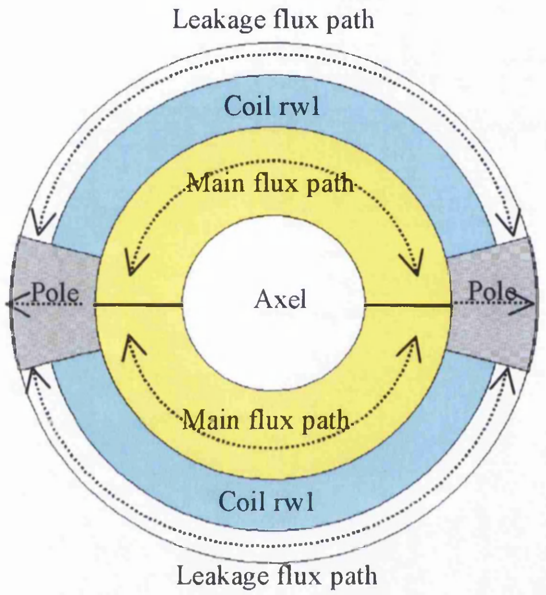


Figure 49: Rotor construction

6.1.1.2 Rotor construction

The rotor has a skew of 30° with pole pieces subtending 50° each of the circumference. The outside diameter of the rotor is 45.82mm. Leakage paths are allowed around the periphery of the rotor between the poles. See Figure 49 for an illustration of the rotor. The light blue denotes the coil winding areas, while the yellow areas are where the mmf sources are placed. Pole pieces are shown as grey while the leakage paths are shown as the light band between the pole pieces.

The air-gap elements connect each pole of the rotor with every pole piece of the stator, these are not shown but are generated by the program.

6.1.1.3 Wiring

The placement of the source terms with in the magnetic circuit is determined be the mmf chords of the coils. The gyrators for a particular coil are connected together by the program. The connections of the coils are bought out as nodes of the magnetic sub-circuit. Connection of the coils to form the three phase circuits is provided by the wiring sub-circuit (Figure 50). This also contains coil resistance elements. Thus the wiring configuration of the coils may be altered by changing only one small text file. The stator is arranged as a three phase star connection and connected to resistive loads in the test circuit.

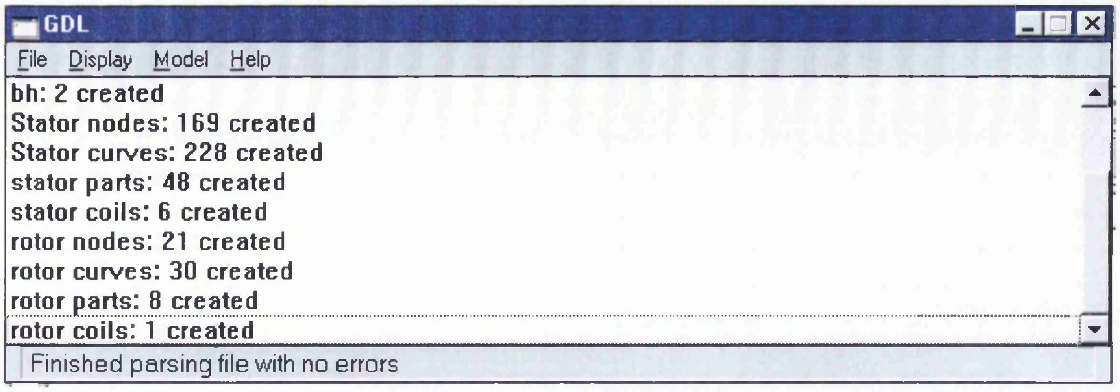
```
* definition of three phase windings

.subckt gen3ph torqp torqn posp posn u1 u2 v1 v2 w1 w2 r1 r2
x1 torqp torqn posp posn
+ s1a s1b s2a s2b s3a s3b s4a s4b s5a s5b s6a s6b
+ r1 r2 magnetic
R1 s3a u1 1
R2 s5a v2 1
R3 s2a w1 1
R4 s4a u2 1
R5 s6a v1 1
R6 s1a w2 1
R7 s1b s2b 1
R8 s4b s3b 1
R9 s6b s5b 1
.ends
```

Figure 50: Coil connection wiring sub circuit

See Appendix C for the text description file as given to the conversion program.

6.1.2 Generation of SPICE model

In the following, screen shots of the program running are shown. Figure 51 shows the result of loading a machine description file. The numbers of the various objects created are listed and whether there are any syntax errors. If there are any syntax errors then these are detailed in the scrolling text window.

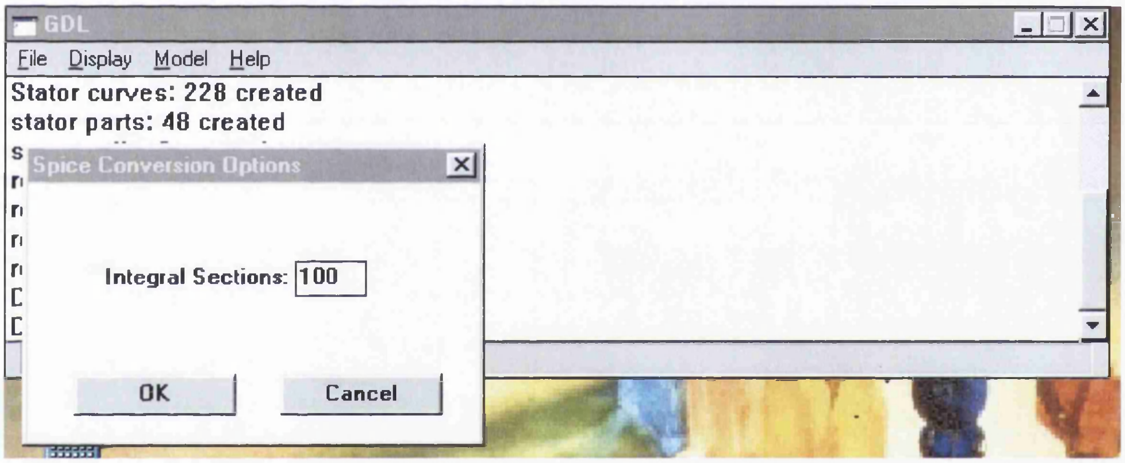


```
bh: 2 created
Stator nodes: 169 created
Stator curves: 228 created
stator parts: 48 created
stator coils: 6 created
rotor nodes: 21 created
rotor curves: 30 created
rotor parts: 8 created
rotor coils: 1 created
```

Finished parsing file with no errors

Figure 51: Messages informing numbers of objects created

In Figure 52 the options dialog for the number of line integral sections is shown. The more sections the greater the accuracy of the approximation. This does not however improve on any inaccuracy due to start and end curves.

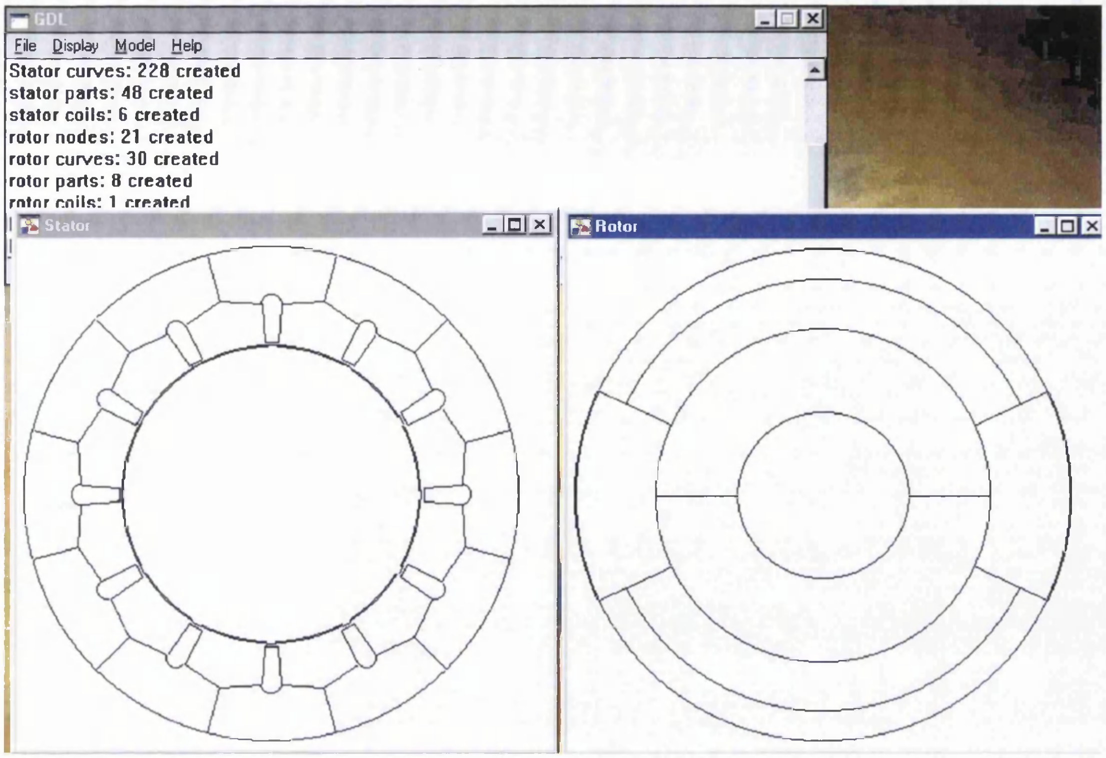


```
Stator curves: 228 created
stator parts: 48 created
```

Integral Sections: 100

Figure 52: Setting number of integral sections for integral

Figure 53 is the result of plotting the curves of both the rotor and stator.



```
Stator curves: 228 created
stator parts: 48 created
stator coils: 6 created
rotor nodes: 21 created
rotor curves: 30 created
rotor parts: 8 created
rotor coils: 1 created
```

Figure 53: Stator and rotor curve displays

Figure 54 gives a magnified detail of a coil slot, between the poles is the leakage element.

Figure 55 is a magnified section of the rotor showing an air gap element on the periphery.

See Appendix D for the SPICE sub-circuit output created by the program.

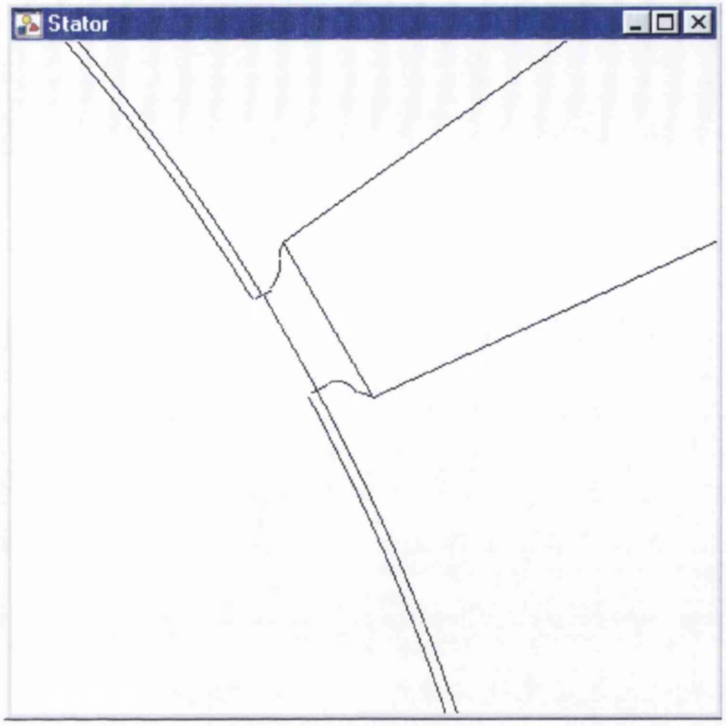


Figure 54: Stator detail

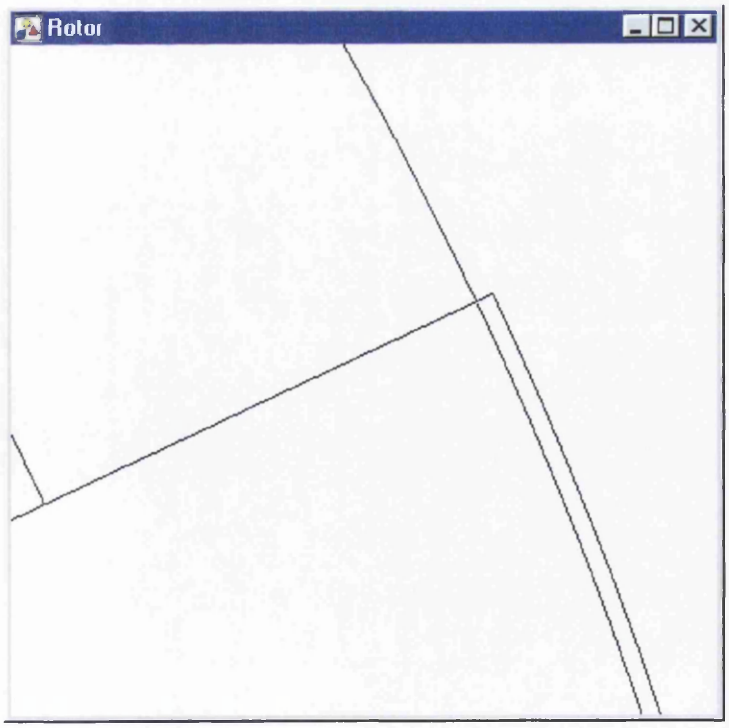


Figure 55: Rotor detail

6.1.3 Results of simulation

Shown in Figure 56 is the test circuit that was used for the spice simulations. Note that the voltage sources, in series with the resistors, are there for current sensing and do not affect the operation of the circuit.

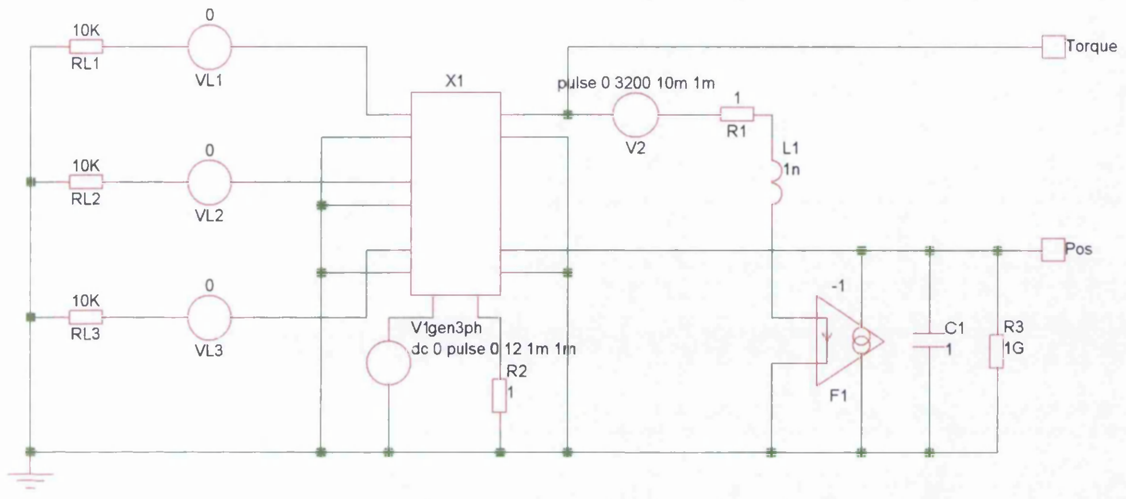


Figure 56: Test circuit for spice model

The value of mechanical load is very large compared to that of electrical and therefore no change in rotational velocity due to electrical load is expected. Therefore, the generator is working at constant speed with a purely resistive load. X1 in the diagram is a sub-circuit containing the lumped resistance of the coils and the magnetic equivalent circuit.

Figure 57 shows the variation of position with time. This is as expected a straight line, as the load on the mechanical source is mainly mechanical loss. The voltage is scaled by the mechanical conversion process such that it represents the angular position in radians.

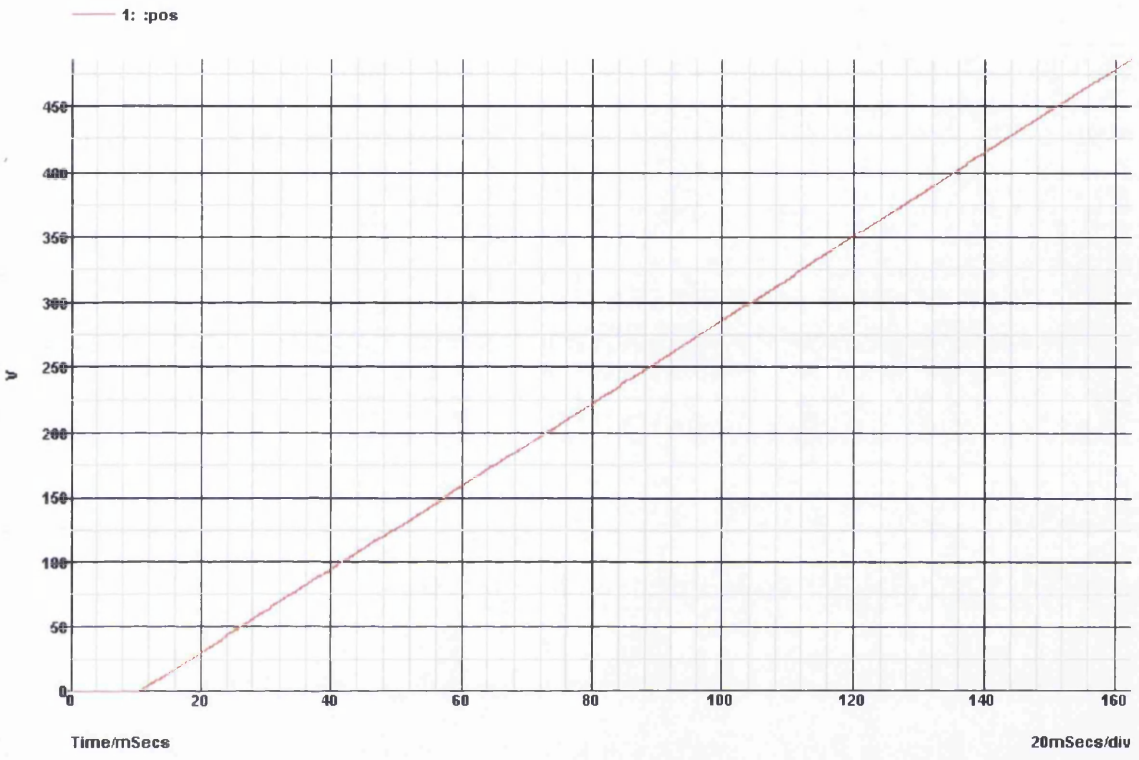


Figure 57: Plot of position terminal voltage

Figure 58 is a plot of the mechanical and electrical power input, electrical voltages and mechanical torque input into the mechanical terminals. Note that the mechanical and electrical powers closely agree. The upper green line is the mechanical power entering the magnetic sub-circuit. A blue line represents losses in the load resistors, while the orange line shows the total electrical losses, including the winding resistance. While middle green line represents the torque load presented to the mechanical side by the electrical load. The lower three traces are the voltages across the three load resistors. Peak torque and therefore peak mechanical load is seen to occur when the voltage across the load resistors is a peak, this is as expected for a real device.

Figure 59 shows the electrical and mechanical power in more detail. The upper green line is the mechanical power entering the magnetic sub-circuit. A blue line represents losses in the load resistors, while the orange line shows the total electrical losses, including the winding resistance. It is evident that the mechanical and electrical power flows closely agree. The power flow through the magnetic circuit must therefore be itself consistent for this to be so.

Figure 60 shows the generation of time-stepped plots (time separation = 1.375ms) of the magnetic circuit flux. The particular item of data being stepped through is the flux density in the air gap. The rotation of the rotor can be inferred from the change in flux in the stator.

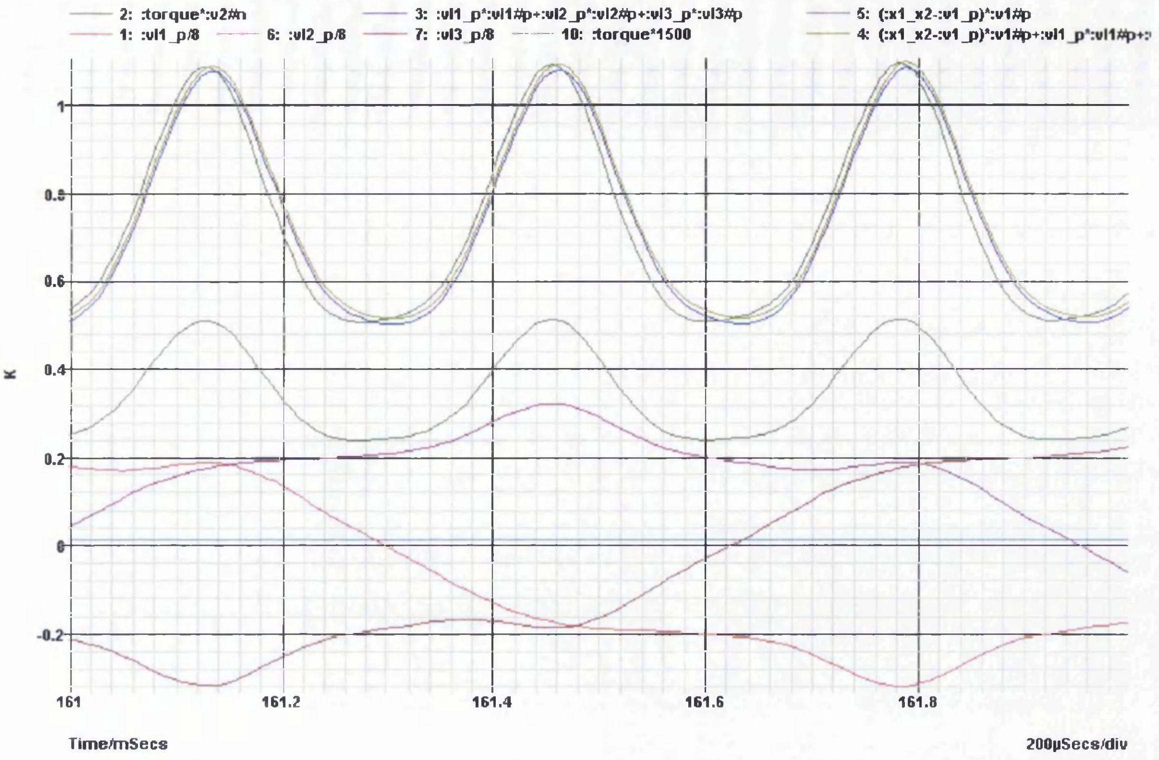


Figure 58: Power, Voltage and Torque waveforms for simple synchronous generator

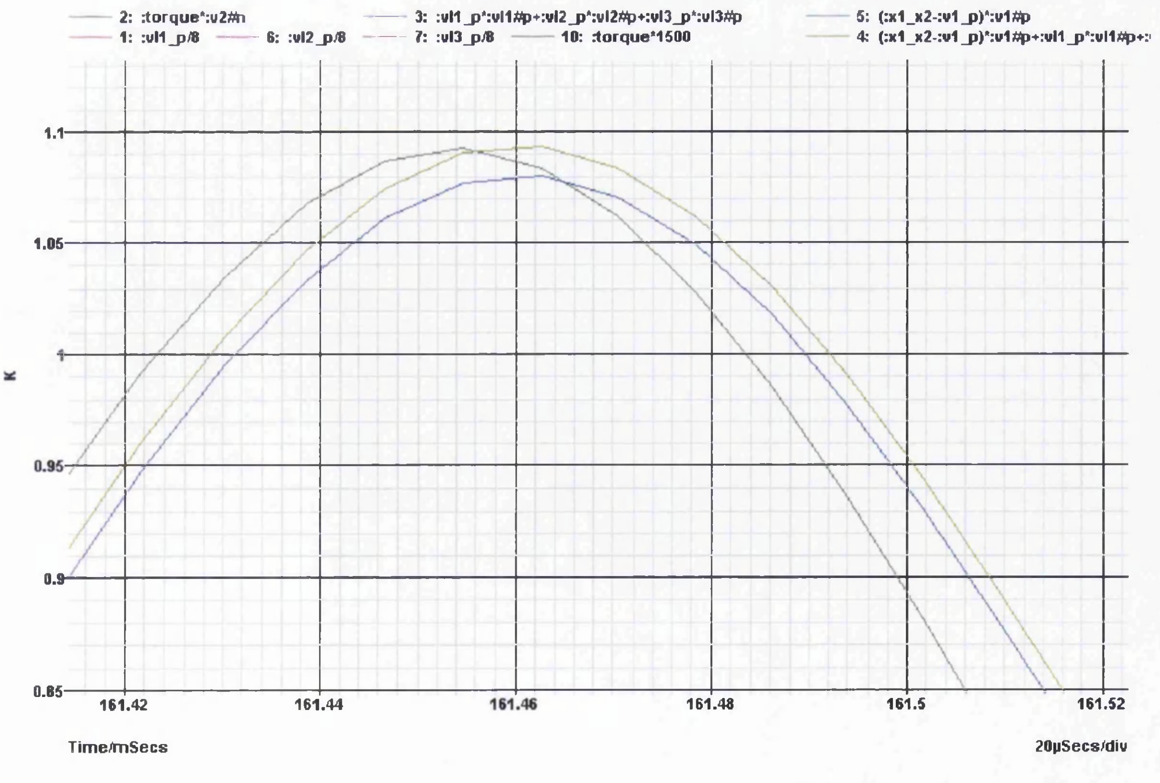


Figure 59: Detail of power waveforms for simple synchronous generator

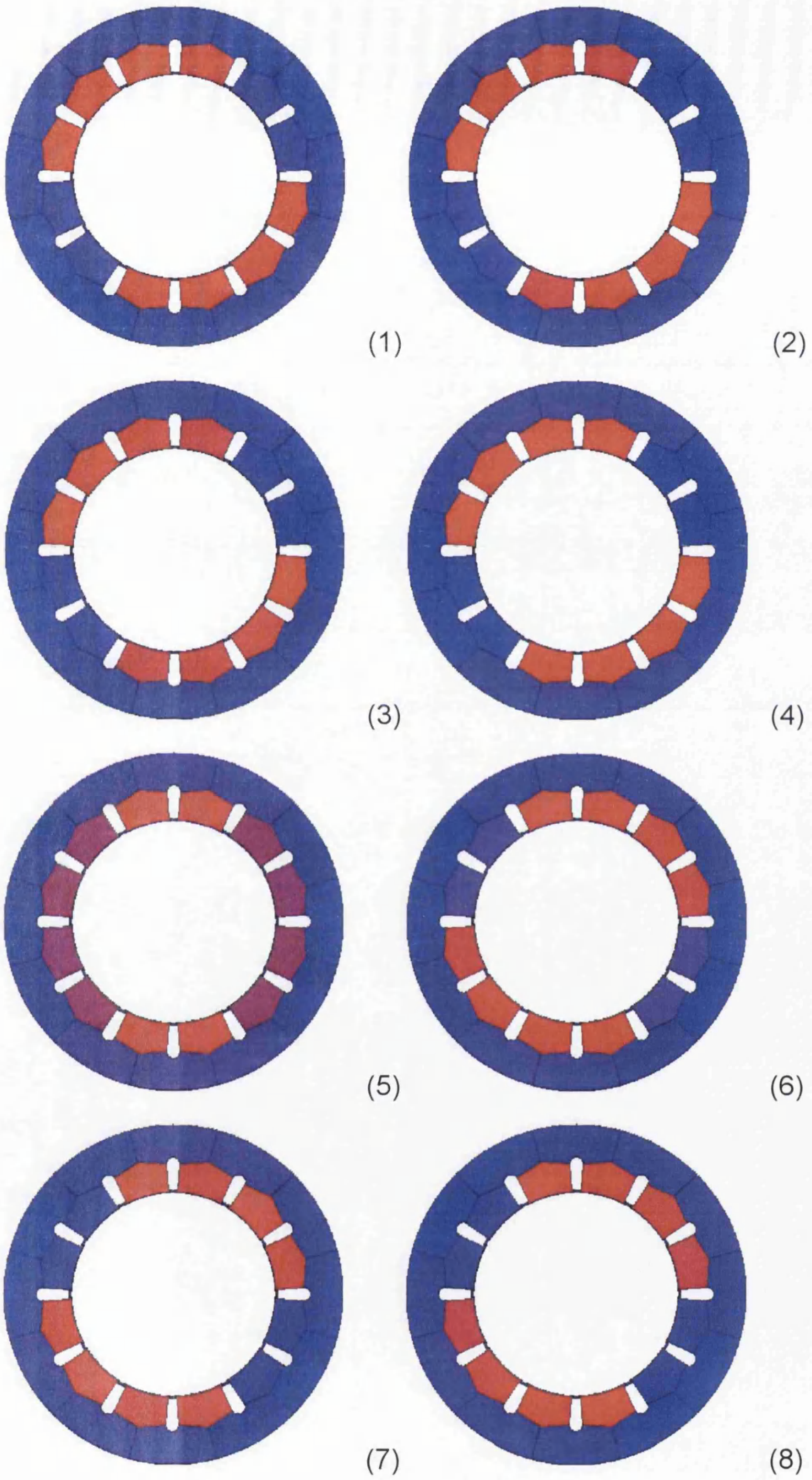
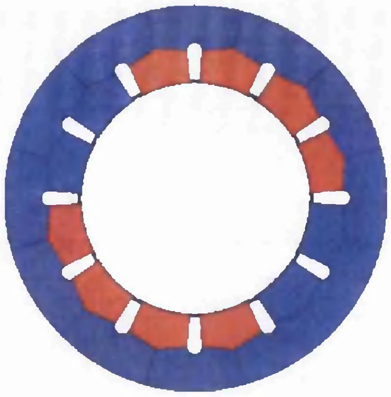
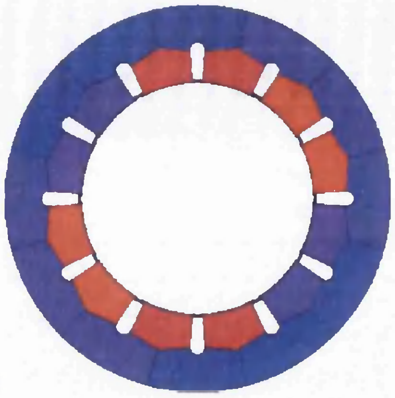
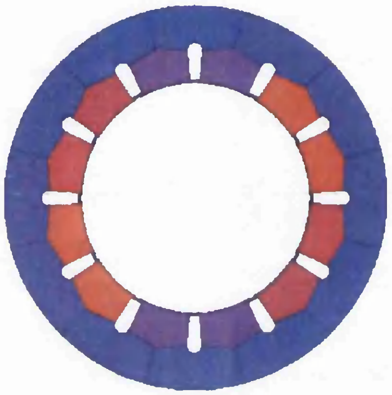
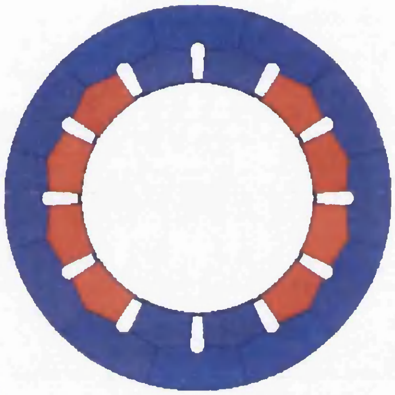


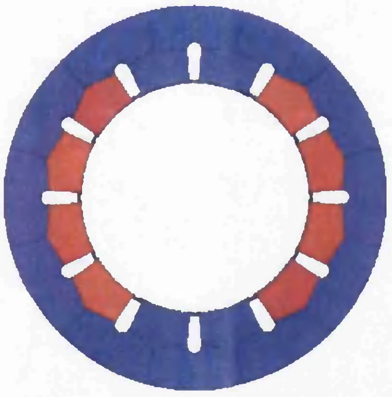
Figure 60: Stator flux plots for the simple generator

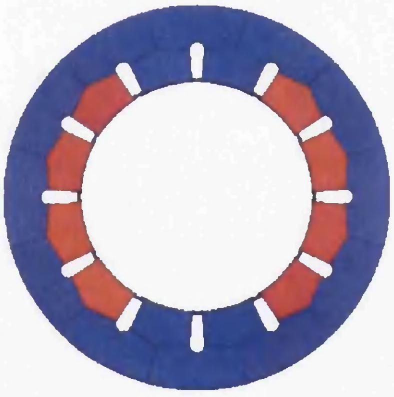

(9)

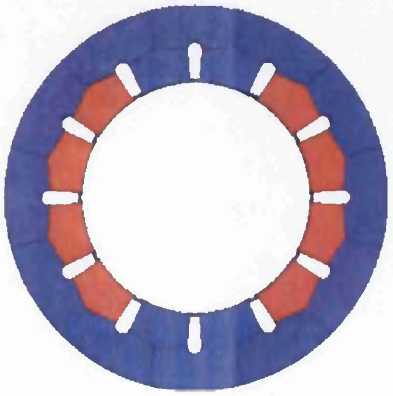

(10)


(11)


(12)


(13)


(14)


(15)

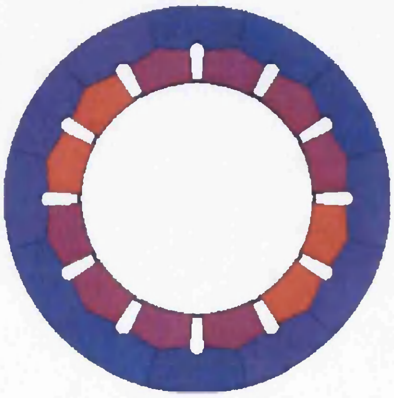

(16)

Figure 60: Stator flux plots for the simple generator (continued)

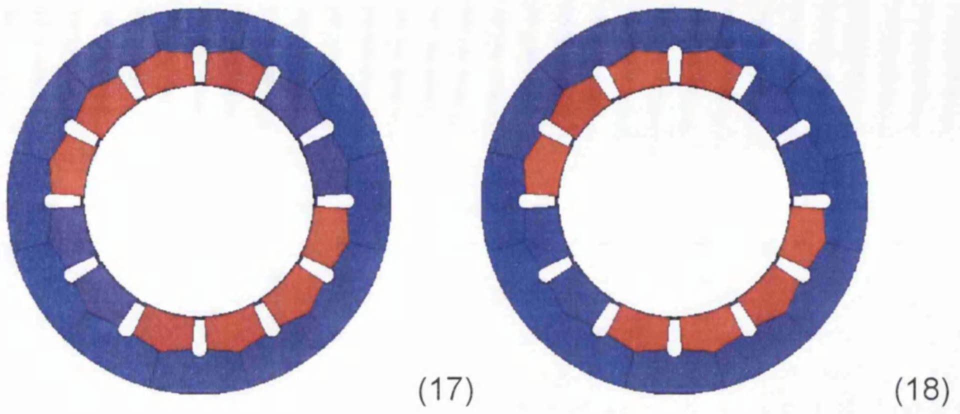


Figure 60: Stator flux plots for the simple generator (continued)

6.2 Generator simulation with power electronics

An additional simulation was performed with the addition of a power electronic rectifier with resistive and capacitive load, as shown in Figure 61. The configuration of the generator in terms of construction and wiring is exactly the same as before.

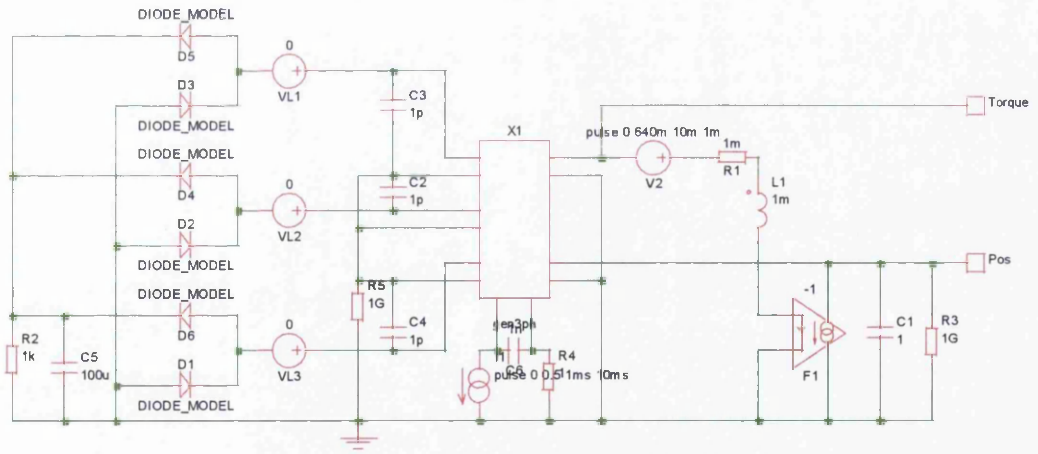


Figure 61: Generator with three-phase power electronic rectifier

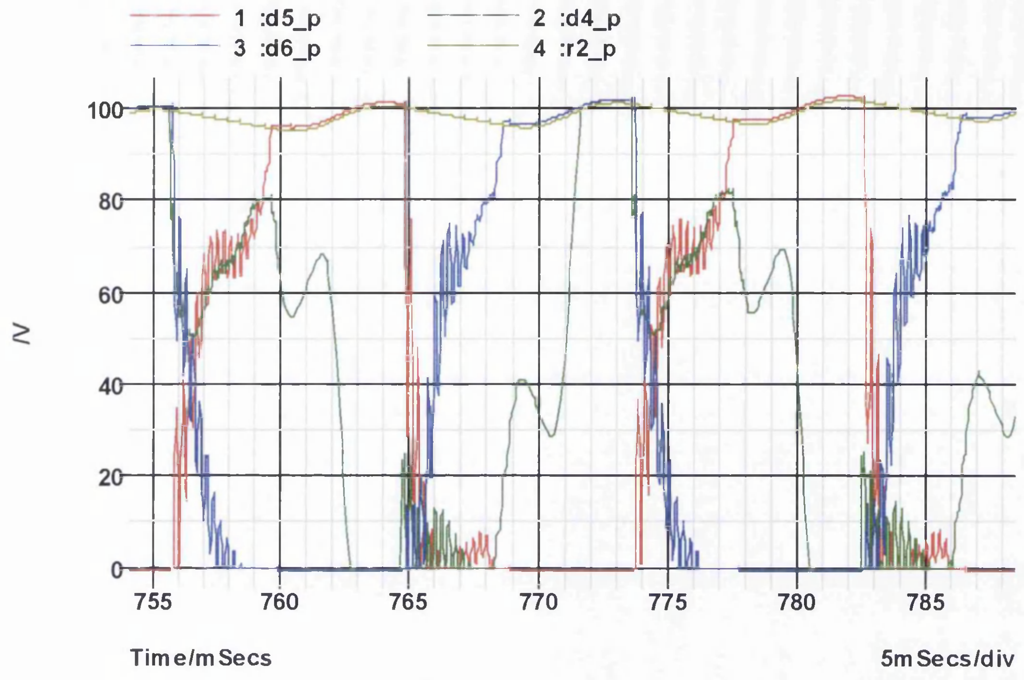


Figure 62: Voltage waveforms of the three phases before the rectifier

Figure 62 is set of curves which show the voltages on the input to the rectifier (red, blue and green curves). Voltage across the capacitive (100μF) and resistive (1000Ω) load is indicated by the brown line.

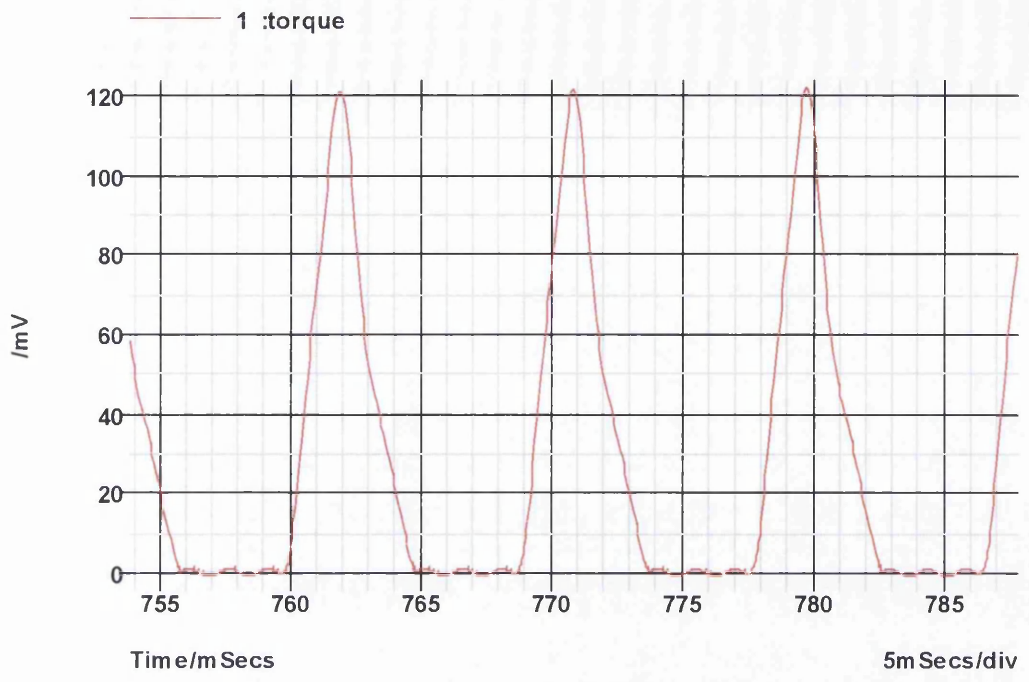


Figure 63: Torque curve on the mechanical power source

Figure 63 is a plot of the torque as seen by the constant velocity mechanical power source. Note the correspondence between the electrical rectifier load and that of the mechanical power source.

6.3 Synchronous Motor Simulation

The purpose of the following test is to use exactly the same model as has previously been demonstrated but instead alter the excitation and mechanical load, so that the device acts as a motor. The circuit used to test the device is shown in Figure 64.

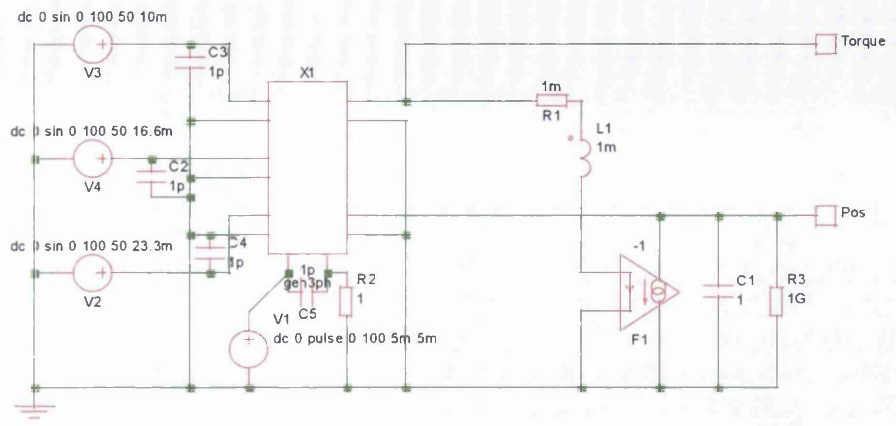


Figure 64: Test circuit used for synchonous motor simulation

The following figures (Figure 65 and Figure 66) show respectively the excitation voltages and the resultant flux in the stator. The first frame in Figure 66 starts at 80ms. The flux intensity is shown with the highest indicated by the red, blue the lowest. Notice that in Frame 5 the back iron is shown with higher flux intensity than some of the pole pieces.

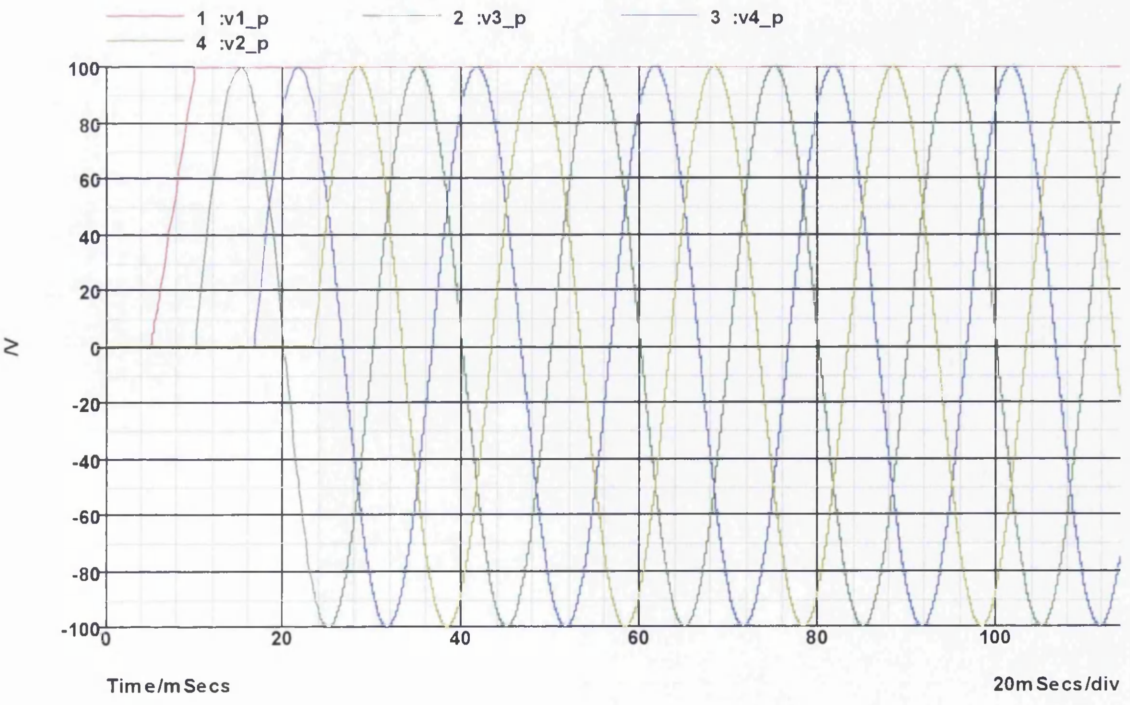


Figure 65: Plot of the 3-phase currents and the field voltage for simple motor

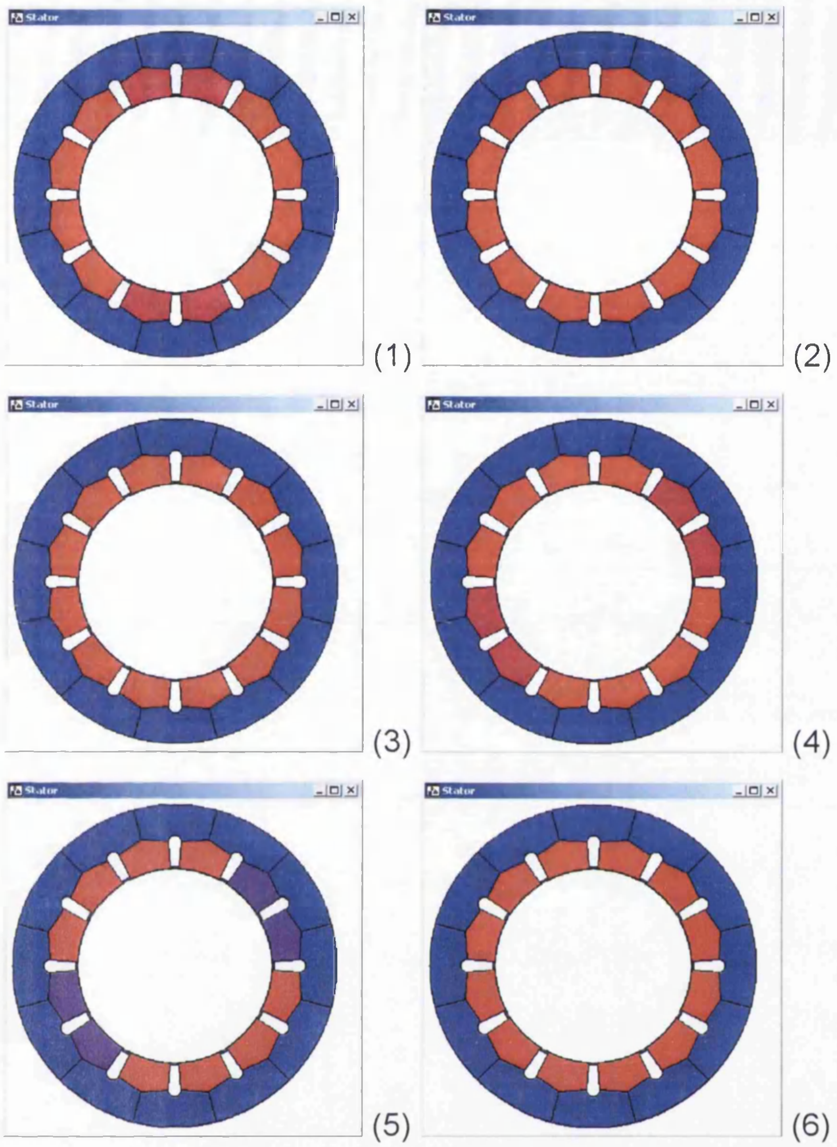


Figure 66: Change in stator flux with movement of rotor for simple motor

The resulting change in position is available via the mechanical equivalent circuit from the terminal marked 'pos', this is plotted in Figure 67. The plot clearly shows the rotor changing position and increasing in rotational velocity.

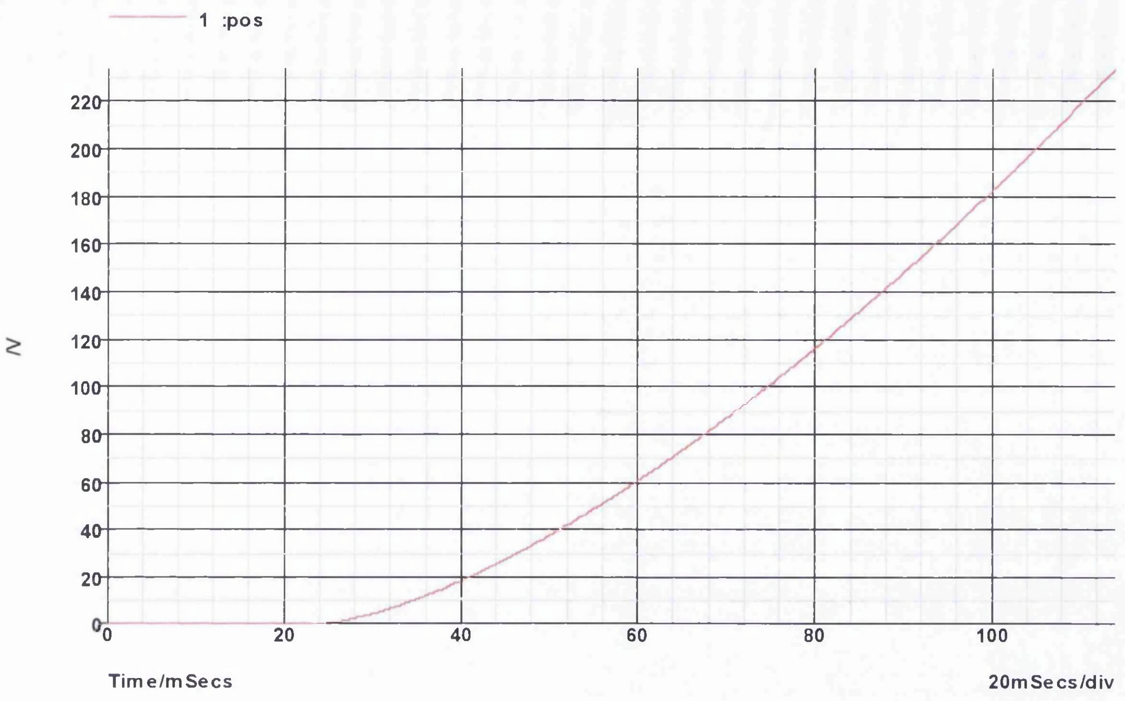


Figure 67: Plot of change in position for a simple synchronous motor

6.4 Induction motor simulation

This is a comparison between actual data obtained from a practical laboratory test and a simplified simulation model produced from the geometry information obtained from the manufacturer of the induction motor. The induction motor has the following specification:

- 4 pole, 3 phase 415V a.c.
- 2.2 KW power rating

The rotor contains 32 bars that are skewed with respect to the rotor axis, and the stator was concentric wound with 32 slots.

Test results were captured using a 'Keithly' high-speed data acquisition board (DAC 1600) with inputs from hall effect current probes having a bandwidth of 50KHz.

6.4.1 Simulation

The information supplied by the induction motor manufacturer was used to fill-in the machine description template (GDL file) developed as part of the project. The resulting spice network listing was automatically generated by the conversion program (gdl2spice). The power circuit model contained the simulated 3-phase voltage supply, which was merged with the machine equivalent circuit. Figure 69 shows the change of the coincident area between that of the rotor and stator teeth, modelled using the equation as given previously. The comparison is made with a numerical estimation of the actual change in coincident area with rotor rotation. The program used to derive the numerical values of coincident area is given in Appendix I. Note the high value in the exponential function; the effect is to cap the highest value to unity.

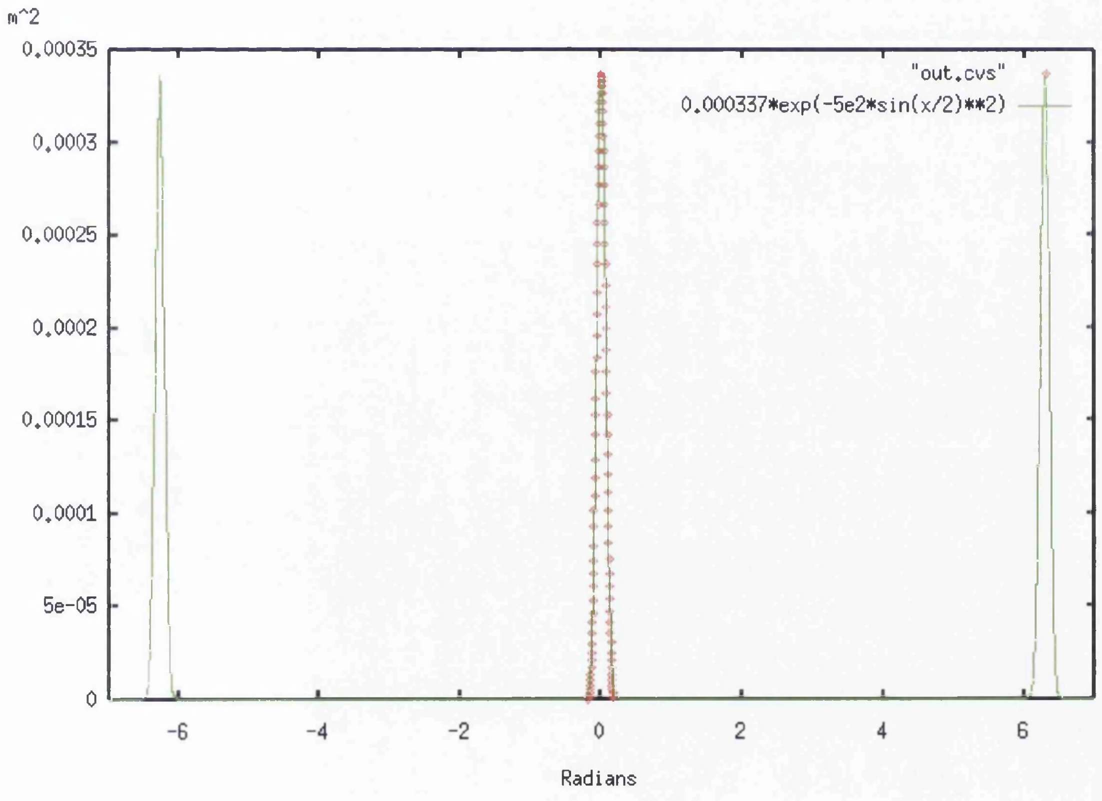


Figure 68: Change of area with rotor angle

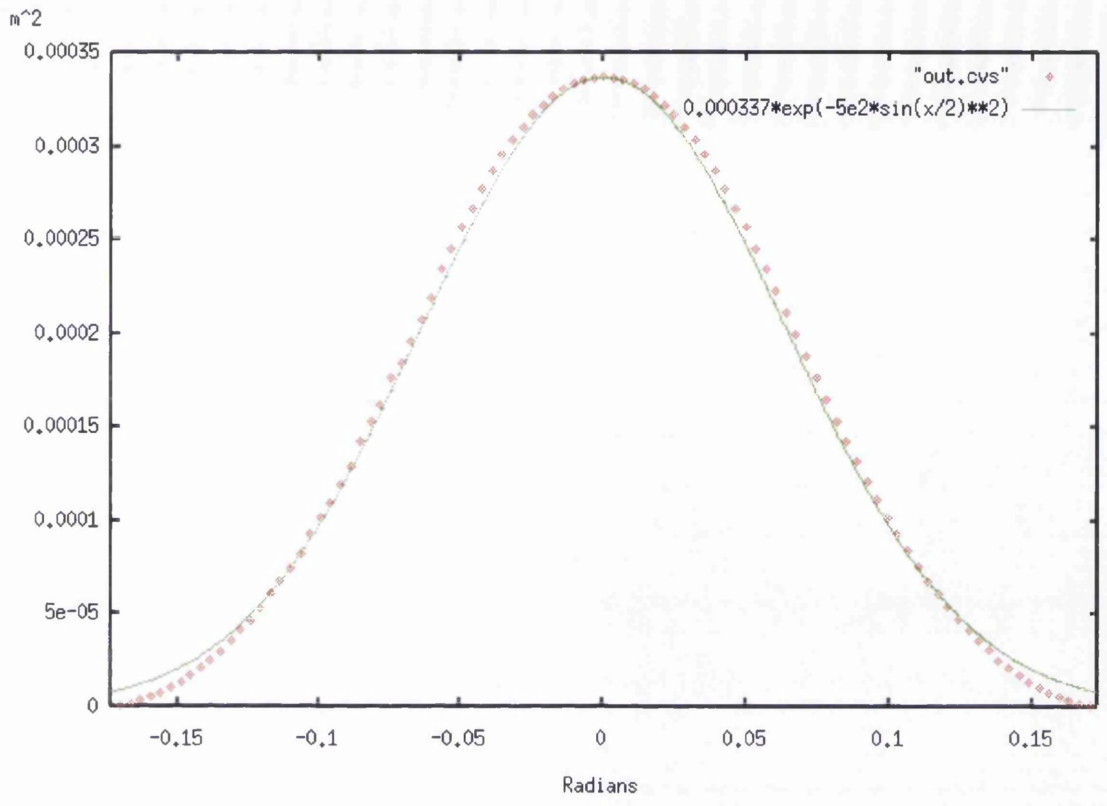


Figure 69: Change of area with rotor angle (enlarged)

The resulting model file was used in a SPICE 3 simulator. The plot shown in Figure 70 is of the steady state current for one of the phases with the rotor speed at 1497 rpm. Similar scaling has been used in both the simulation and experimental test result to allow qualitative comparison of the two waveforms.

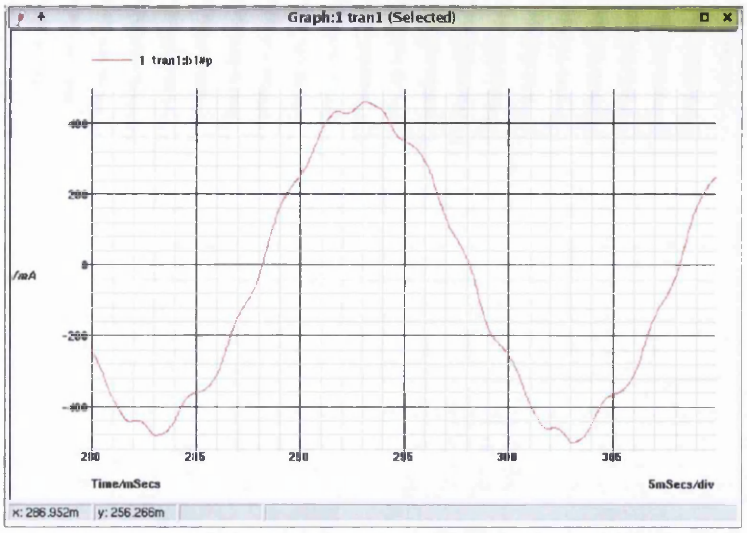


Figure 70: Simulation results (actual current multiply by 10)

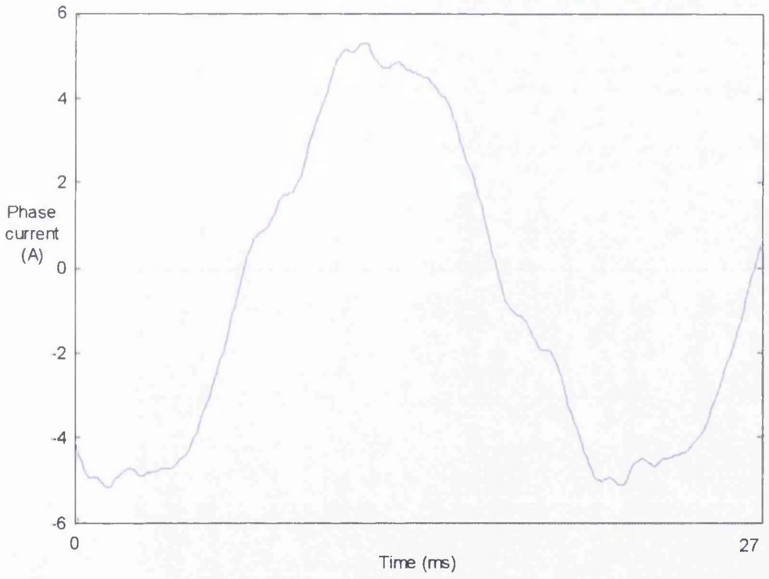


Figure 71: Test results (Light load)

6.4.2 Results

The experimental results produced in the laboratory for the steady state phase current are shown in Figure 71: Test results (Light load)

By comparing the results from the test and simulation correlation between the two curves can be seen.

6.5 Conclusion

An example design for a simple three phase synchronous machine has been given. The machine is assumed to have simple leakage paths and a linear magnetic circuit (i.e. no losses in the iron). It has been shown the steps needed to convert this into a SPICE subcircuit via the use of the conversion program. A user friendly interface is evidenced by the screen shots of the program, along with a demonstration of the information provided during the conversion process.

The example machine is shown operating in both in generator and motor modes. This was by changing whether the sources and loads were either on the electrical side or the mechanical, respectively.

The simulation results obtained using SIMETRIX are presented for the appropriate test circuits used. These show that there is consistency between the power flows for the electrical, magnetic and mechanical models for this simple model. Thus it is shown that the model method is reasonable in terms of conservation of energy between the electrical, magnetic and mechanical sections.

A further simulation provided is that of a 2.2 KW induction motor in the no-load situation. A correlation is evidenced between the graphs of the simulated and actual test data. While further tests are acknowledged to be necessary, to prove the model as accurate, it is concluded that it is reasonable in terms of the no load line current.

7 Discussion

7.0 Model

7.1 Motion

The model developed uses magnetic equivalent circuits (as exemplified in section 2.11). This method of modelling various types of machines has been explored by both by Ostovic (20) and Haydock (4). An alternative but similar approach is that by Azzerboni (59), specifically for cylindrical systems with conductors in motion. Perhaps one the most onerous problems when using these types of models is the relative motion of one MEC to the other e.g. stator and rotor. The technique used by Haydock (4) in the modelling of a synchronous machine was to separate the 'speed voltage' i.e. that due to relative motion of the coil. from the transformer component i.e. if:

$$emf = n\frac{d\Phi}{dt} \text{ (V)} \qquad (22)$$

In the above the flux Φ is dependent on both the current (transformer effect) and change in permeance (speed voltage). This technique lends itself to developing models for symmetrical periodic structures, but has problems when automatically generating models for non-periodic structures. Ostovic (20) developed several models of different machines and incorporated motion within the simulation. However the models were limited as they could not be used in standard circuit simulators and required custom matrix solution software. Of particular note is the way in which the motion is incorporated within the model, this is achieved by using a sparse 'connection' matrix, which depends on the rotor position and can directly modulate a periodic function. This solution requires the ability to implement arbitrary functions directly within the simulation software, for using standard circuit simulation packages this is not always possible, and an alternative solution is necessary. An alternative of using an arbitrary permeance function is to use functions available in standard circuit simulation packages. This was achieved by using a permeance function and curve fitting to a simple model of stator/rotor permeance variation:

$$C_m(\theta) = P_{\max} \cdot e^{-a\left|\sin((\theta+b)/2)\right|^c} \qquad (122)$$

A limitation with equation (122) occurs when the pole piece area of both the stator and/or rotor is small compared to the rotor or stator circumference, preventing suitable values for the coefficients *a*, *b* and *c* to be found. This model also assumes that the air gap depth is very small compared to the dimensions of the pole piece. This assumption simplifies the model eliminating the need to implement a full field solution to estimate the variation of air gap permeance with rotor rotation. On machines with a large air gap this assumption is not valid and a full field solution is required (Moallem et al (22)). The approach taken by Preston and Lyons (21) and Delforge et al (23)(24) is to curve fit a function to a full finite element solution. A finite element solution at rotor angles of 0, 90 and 180° is used to determine the values (α_1, α_2 & p) of the following function:

Air gap reluctance $$Rg = \frac{1}{\alpha_1 + \alpha_2\left|\cos(\theta/2)\right|^p} \qquad (120)$$

Equation 122 has been compared with equation 120, as used by Preston et al (21), for specific coefficient values and it has been shown that they can be used interchangeably. The results from the generator and motor simulations, in terms of energy conservation, match between the electrical, magnetic and mechanical sections. However, as far as checking the correctness of the air gap permeance function, this can only be verified via comparison to closed solutions, actual tests or finite element solutions. A simulation performed of an induction machine gave results comparable with that of an actual test. To investigate this further more practical tests would be required, but it is stated by Moallem et al (22) good results can be obtained from MEC models, provided the air gap flux is accurately modelled. The actual accuracy of the model also being dependent on the operating conditions being simulated.

7.2 Mechanical Torque

If an equivalent circuit is to be used to describe the full behaviour of a machine, whether generator or motor, then the mechanical effects need to be taken into account. The magnetic field in the machine will lead to forces being developed between the various components of the machine, due to the field stress. This will therefore lead to torque on the output shaft being developed on the rotor. Several methods exist for the calculation of the force between the components in the device: virtual work principle (Ostovic (20), Ratnajeevan et al (9), Demenko (17); magnetic shells (Carpenter (15)). The force is derived by the use of the virtual work principle (equation 125), as the others two methods partially require an analytical solution.

torque $$T = \frac{\partial \mathrm{U}}{\partial \theta} \quad (125)$$

In conventional lumped parameter machine models the torque is derived by assuming it can be derived with reasonable accuracy by just observing the state of the field in the air gap (Fitzgerald (9)). This assumption is also applied here i.e. the force between machine components is assumed to be calculable from state of the field in the magnetic elements, which model the air gap. In the magnetic equivalent circuit, as presented here (section 4.3), the value of the air gap permeance is purely dependent on rotational angle. The angle of the rotor is known and given the value of mmf (in the equivalent circuit the value of voltage across the magnetic capacitor) the energy of the air gap element can be calculated. Others have written custom software, or adapted existing software packages, to use a derivative function (Sulivan (46)). A derivative function is not available within standard circuit solvers, only standard spice functions. In order to calculate the derivative a simple first order approximation is used. The problem with a simple implementation of the function is that the delta change in position becomes insignificant compared to the absolute position value. The solution suggested and used in the work presented here is to change the calculation of the change in permeance to the following:

$$C'_m(\theta, \Delta\theta) = P_{\max} \cdot e^{-a\left|\sin(\theta/2)\cos((b+\Delta\theta)/2)+\cos(\theta/2)\sin((b+\Delta\theta)/2)\right|^{c}} \quad (129)$$

The advantage of this formulation is that the problem of numerical round off becomes significant much later in the simulation than would otherwise be the case. The disadvantage is that the position is stored as an absolute value and therefore the accuracy of the transcendental sin, cos etc. functions will become less as the absolute value increases, the increase of inaccuracy will depend on the floating point representation used in the simulation.

An interesting point is made by Wang et al in a comparison between a Finite Element method and MEC in that they found a substantial difference in the 'detent' torque values (30%) obtained. The conclusion reached was that the 'fringing' fields were not sufficiently well modelled by the MEC method and the solution was the inclusion of greater detail in the flux paths.

7.3 B-H Curve

The representation of the magnetic characteristics of the materials used in the machine i.e. B-H curve is dependent on the accuracy required and operating conditions of the model (Shamming et al (47), Lancarotte et al (43)). Many models such as traditional lumped parameter equivalent circuits (Adkins (13)) assume a linear relationship for the B-H curve. Alternative models of the magnetic hysteresis (Carpenter (61)) have been proposed which seem to realistically model the behaviour of actual material, these often require changes to the simulation engine for them to be incorporated. Attempts have been made at adapting the lumped parameter models to incorporate the non-linear characteristics of saturated electrical machines, particularly induction machines (Vagati et. al.(39)) The problem with this approach, as identified by Haydock (4) is that the models are adapted ad hoc, with little correspondence between the physical location or cause of non-linearity and the exact location for the incorporation of the non-linear element within the lumped parameter model. By adaptation of the model parameters to fit test data, a close fit between test data and simulation can be made for that particular machine operating under the exact same test and simulated conditions. The model also gives no indication of the internal magnetic state of the machine, e.g. where is the saturation occurring.

Finite elements methods can give very accurate results, even for machines with significant non-linear saturation of machine components (Jack (32)), for which no previous tests have been performed. The detailed field solutions available from FE methods give insight into the areas of the machine that are saturated. The problem of such solutions is the computational cost, particularly if non-linear elements are present. In order to ameliorate the effect of such components it is possible to only simulate as non-linear, those components that are likely to saturate. However, finite element solutions cannot be incorporated in to standard circuit simulation software, without fundamentally altering the software. Several formulas have been developed for use in FE solutions to model the B-H characteristic.

The approach taken by Ostovic (20) for incorporating non-linear elements within MEC models was to use a cubic formula to modulate the element permeance. Moallem et al (35) also modulate a permeance function, in this particular case to account for the saturation of parts in a switched reluctance motor. Haydock (4) used a polynomial curve fit, as this was the only available method of incorporating the curve within Spice 2g6. The method used here is to adapt a formula due to Brauer (25), originally used for finite element solutions, so that it can be used as an expression that modulates a capacitance value within Spice3f4. The capacitance value represents the magnetic permeance. This seems to work well, although it is computationally expensive compared to a linear equivalent or piecewise linear fit. The non-linear model used in the equivalent circuits here can be incorporated with in standard solvers, but could be simulated more efficiently if the simulation software was altered to incorporate it as a built in function.

Related to the modelling of the materials B-H characteristic is the inclusion of thermal effects, for which studies have already been performed e.g. Wilson et al (45) & Maxim et al (55).

7.4 Model Generation

A generic description of the machine geometry is used as an input to the program that creates the simulation file. The format of this file is specific to a certain type of machine, rotating rotor within a fixed stator. This format is unique to the application but is comparable in many respects to other geometry description languages and file formats. These other languages and file formats range from simple point lists, stored as binary numbers, to complex formats such as DXF (Drawing Exchange Format) (49) and languages such as SVG (Scalable Vector Graphics) (50). While these could have been extended (particularly SVG) the software would have been more complicated. To simplify the creation of the software a simple substitute and original construction format was created (GDL – Geometry Description Language).

Other modelling languages such as 'Modelica' (51), are designed to be generic in that they describe in a general manner the details of the system that is to be simulated. In Modelica the form of this description is that of mathematical equations which can be differential, algebraic or discrete. A possibility is that instead of targeting SPICE as the simulation engine that a Modelica simulation description could be generated from the machine geometry data. An advantage of this would be the already multi-physics nature of the Modelica environment.

7.5 Machine Description

The aforementioned description language (GDL) is relatively high level in that it directly describes the machine part geometry, the relationship between the parts and implicitly the flux path. The geometry consists of named nodes and the named curves that connect the nodes.

Compared to Drawing Exchange Format (DXF) (49) the purpose of the language is to provide a comparatively easy method of describing the typically repeating structure found in electrical machines. Hence the ability to describe the repeating pattern in terms of element names and sequences.

An alternative approach to the traditional method of finding the values of the lumped parameters in equivalent circuit models is that of creating an 'expert system' which contains knowledge in the field of electro-magnetic analysis. This is the approach taken by Kurumbalapitiya et al. (45).

7.6 A distributed Magnetic Equivalent Circuit

A very simple method of using the finite elements is to simply provide an equivalent permeance value for each of the elements, as shown in Figure 72 (a).

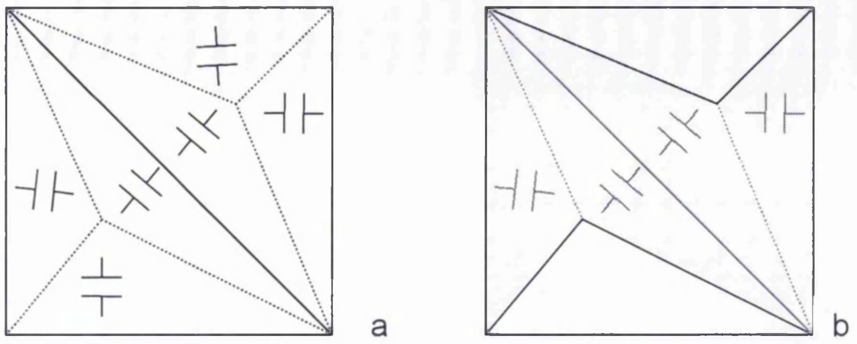


Figure 72: Equivalent permeance elements

Figure 72 (b) shows the error if the flux path is assumed to be from left to right. The more elements the better the fit and therefore the more accurate the representation.

An example of the discretization is shown on the following page. Figure 73 shows the physical layout of the core and coil. Figure 74 shows the equivalent circuit.

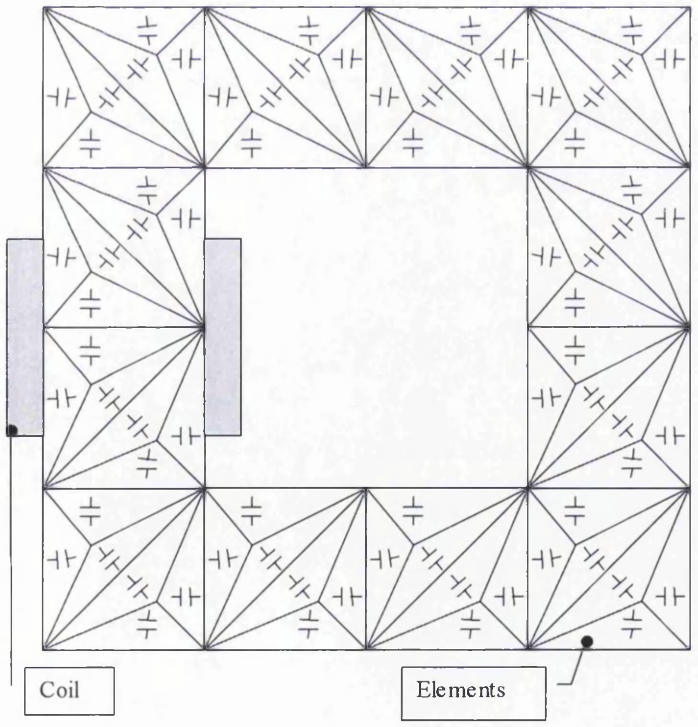


Figure 73: Discretization of example flux path into permeance elements

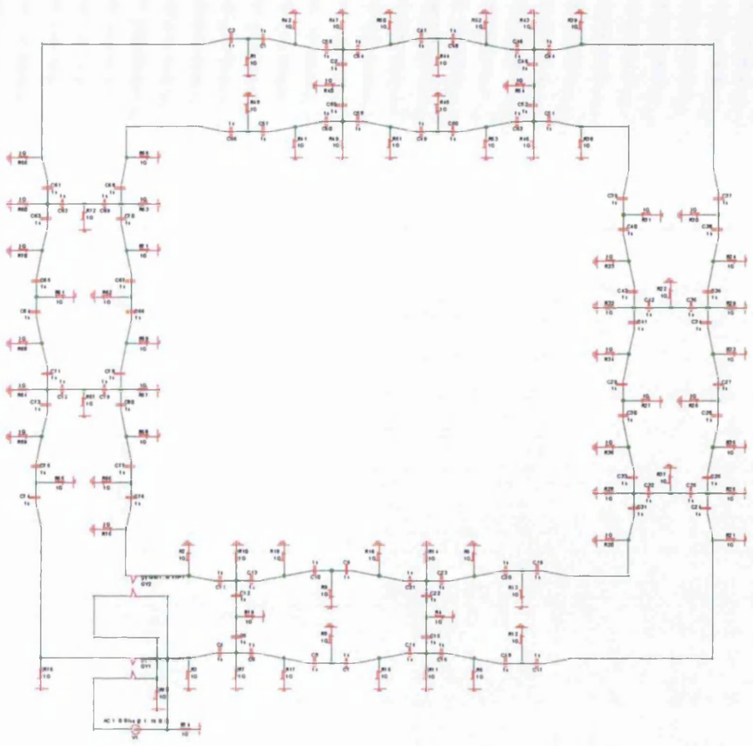


Figure 74: Equivalent circuit for the previous discretisation

Notice that there are simplifications, which could take place in terms of the permeance elements. This would be relatively easy to implement as any series permeances are simply combined. Also automatic discretisation of an arbitrary shape can be performed using Delauney triangulation.

However, a problem with this scheme is that the correctness, in terms of the approximation to the field solution, depends on the shape and orientation of the triangular elements.

A simple analysis of the problems, which arise with such simple elements, is shown in the following. Take the example of a simple triangular element, as shown in Figure 75. The calculation of the reluctance involves the following integral:

$$\int_0^y \frac{1}{x(y)\cdot z}dy \qquad (141)$$

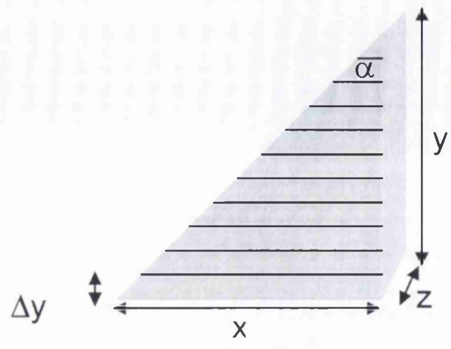


Figure 75: Calculation of reluctance

$$R = \frac{1}{\tan\alpha}\left[\ln y\right]_0^y \qquad (142)$$
$$\therefore R \to \infty$$

As can be seen from equation above the reluctance cannot be calculated using simple approximations where the integrating dimension disappears.

Several alternative approaches exist which would allow the value of the component to be calculated, one approach is to find the geometric mean value for the permeance, i.e. find the centroid of the shape and therefore the equivalent bounding box for the integral.

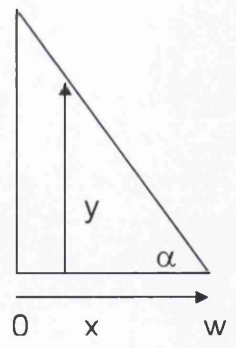


Figure 76: Calculation of geometric mean

Given the triangle in Figure 76, then the geometric mean can be calculated by the following equation:

$$GM = \sqrt{\frac{1}{w}\int_0^w y^2 dx} = \frac{w\tan\alpha}{\sqrt{3}}$$

$$Permeance \propto \frac{z \cdot w}{GM.} \qquad (143)$$

$$\therefore Permeance \propto \frac{z\sqrt{3}}{\tan\alpha}$$

Another approach is to split the overall triangle element into several sub elements, each of which has a simple approximation made of the equivalent permeance. This is shown in Figure 77. The area indicated as (1) denotes the area within the triangle of the combined area (1) & (2). The calculation of the overall permeance is that of areas ((1)+(2))/2. In the eventual magnetic equivalent circuit the combined value of permeance is therefore the average of that contributed from the first triangle, and that of the immediate neighbour. The value of permeance for the element indicated as (1) in the diagram is given by:

$$perm \propto \frac{2|z|}{\tan \angle XBA} \qquad (144)$$

Where X is centre of the triangle and ∠XBA is the angle of the element (1).

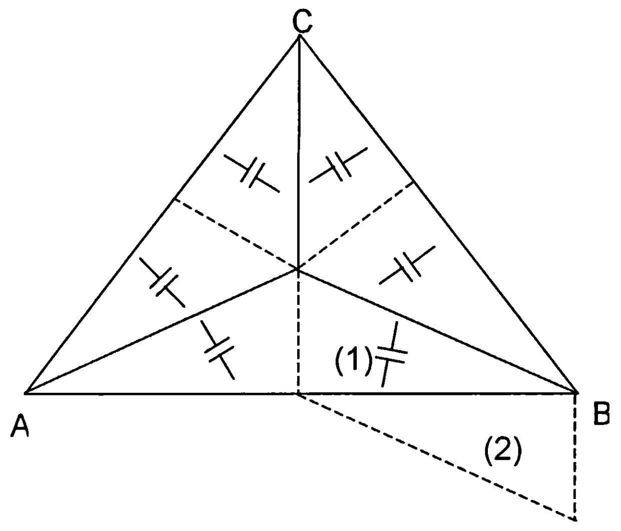


Figure 77: Alternative scheme for the calculation of permeance (Note that the depth of the element is denoted by the symbol z)

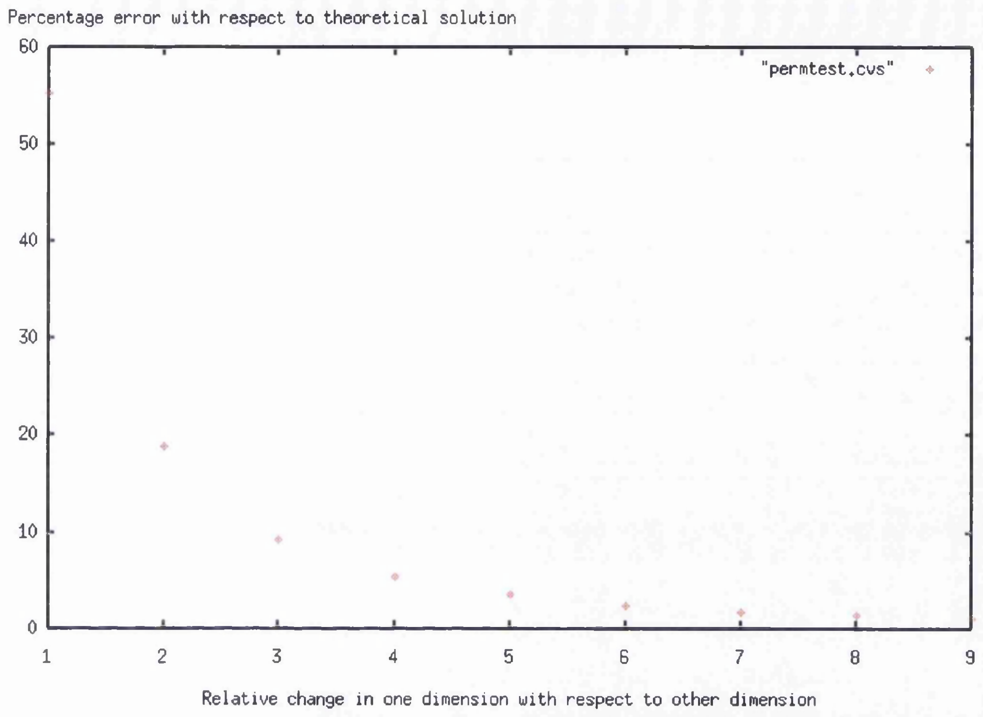


Figure 78: Variation of error of a two-element approximation compared to the closed solution

Shown in Figure 78 is a comparison between a simple two-element approximation and a simple rectangular element. As can be seen from the results the percentage error varies very significantly with the shape of the triangular elements. In conclusion therefore it can be assumed that any generic solution, which uses this method, would also be very strongly dependent upon the discretization algorithm in determining the overall correctness of the solution.

An interesting alternative to the MEC method described previously is that by Hammond et al (29). This involves a much more fundamental modelling of the electromagnetic field using differential forms. This gives upper and lower bounds to the solution but is mathematically complex.

8 Conclusions & Further Work

8.1 Conclusions

The simulation of power electronics systems has always suffered from unrealistic models of electric machines. Magnetic and mechanical effects cannot be taken into account when using a commercial simulation solver such as SPICE. The software package originated in the project allows these effects to be simulated using a standard circuit solver. The software enables the machines Magnetic Equivalent Circuit (MEC) simulation model to be produced from a textual description of the machine. This package allows designers with little knowledge of the electromagnetics to produce realistic models of machines for use in commercial simulators. The main advantage of the software is that a detailed knowledge of the machine operation is not required, similar to the method used by Integrated Circuit (I.C.) designers that do not require a detailed knowledge of I.C. fabrication. This new software package has been developed to take into account electric, magnetic and mechanical interaction with the potential of being exploited commercially.

8.1.1 Model development

To overcome the problem caused in machine models of modelling non-linear saturation and relative motion, the technique of using non-linear capacitors was implemented.

An original new function that models the changes in permeance with motion has been derived (Eq. 122) and successfully tested. A further enhancement of the function within a central difference scheme to numerically find the derivative, and hence the mechanical torque has been tested (Eq. 134). A solution to the problem of long simulation time where the position variable is continually increasing has been established and is described (Eq. 136). This has allowed longer simulation runs to be achieved before accumulative numerical errors, introduced by the numerical calculation of the derivative, become noticeable.

The method of integrating the mechanical system components with the electrical and magnetic elements has also been achieved. This combination of the new function, its derivative and the mechanical system components gives a simulation model with full interaction between electrical, magnetic and mechanical quantities. A search for a suitable function to represent the B-H curve, maintaining compatibility with SPICE type simulators, was undertaken and this resulted in the Brauer function that has been incorporated into the models. This was incorporated within the non-linear capacitor model and successfully tested using a SPICE based circuit solver. This improved non-linear capacitor model was successfully tested using a SPICE based circuit solver.

8.1.2 Model generation

A simple descriptive syntax has been derived for the entering of a machine physical geometry and material properties. The syntax created allows the machine to be described in plain language from design data which machine manufacturers would have readily available.

An algorithm that performs a numerical approximation of the permeance has been developed and is described (pp. 60). The language syntax was produced in conjunction with the algorithm and therefore makes it relatively easy to implement. An algorithm to determine the values of the permeance function coefficients was created (pp. 63) such that the least computationally expensive coefficients are selected. This algorithm is unique to the function described and takes into account the effects of skew on the coefficients. The algorithm was successfully implemented in software and its operation verified by the test carried out in section 5.5. The software will fit the curve between 10% and 90% of the maximum, where these points correspond to expected positions of the maximum and minimum permeance positions.

8.1.3 Software development

The software has been developed using Object Orientated Programming (OOP) and implemented in C++. This method has allowed the software to be more flexible and made model generation relatively easy. Additional effects (e.g. thermal) can easily be added to the general model. This allows models of different complexity to be used depending on the type of application required.

8.1.4 Simulation

The modelling technique has been verified by automatically generating and simulating a MEC model of a 3-phase induction motor; this was checked against practical results (pp 101) obtained from a test-bed consisting of an induction motor and associated load.

A further validation of the MEC modelling technique checks for consistency by ascertaining that the energy flow between the electrical, magnetic and mechanical sections balance (Figure 59) for a synchronous generator and synchronous motor.

The textual description of a simple synchronous electrical generator is given in Appendix C, with the automatically generated MEC model given in Appendix D.

The automatic production of complex Magnetic Equivalent Circuits of rotary machines produces a realistic model giving a high degree of confidence in the simulation results when combined with the power and control electronics in a SPICE type simulator.

8.2 Further Work

8.2.1 Expert system or 'heuristic' generation of flux paths

The flux paths must be manually entered for the calculation of the magnetic equivalent circuit elements to take place. Automatic calculation of these circuit elements would be a significant advantage. However as indicated earlier in the thesis this is dependent on knowledge of the operating characteristics of the machine. General heuristic rules can be found which in most cases give reasonable solutions. Similar in approach to that used by Kurumbalapitiya et al. [45] but instead of the emphasis being field solutions the magnetic equivalent circuit is generated.

8.2.2 A Magnetic Equivalent Circuit for use with finite element triangulation

A previously discussed very simple method of using the finite elements is to simply provide an equivalent permeance value for each of the elements, as shown in Figure 72 (a). Figure 72 (b) shows the error if the flux path is assumed to be from left to right. The more elements the better the fit and therefore the more accurate the representation.

Automatic triangulation of an arbitrary shape can be performed using Delauney triangulation. However, a problem with this scheme is that the correctness, in terms of the approximation to the field solution, depends on the shape and orientation of the triangular elements.
Further work would be the investigation of finite difference schemes rather than the adaptation of triangulation.

8.2.3 Computationally efficient air gap elements

Rotation is via the implementation of 'n to m' permeances i.e. if there are n stator pole pieces and m rotor pole pieces then there are n $\times$ m permeance elements required. This is computationally very expensive as the amount of detail in the design increases. Adding an extension to the SPICE simulation engine to make this more efficient would be a solution. This could be in the form of a special 'air gap' component with inputs of position, flux change and 'named air gap'. Thus the number of required components would be n + m and not n $\times$ m. This approach could be used for other situations where the modelling method is used.

8.2.4 Approximation of the permeance function

The function used for the variation of the permeance with rotation has a limited range of angles over which it can usefully be used. Alternatives could be found but it is probably the case that a piecewise linear fit would be more appropriate i.e. resorting to a finite element calculation of the air gap permeance variation. The further work would therefore consist of arriving at a measure that can be used to determine whether the approximation as used in this thesis is adequate or whether a finite element analysis needs to be performed.

8.2.5 Geometry description import from common computer aided design (CAD) packages

The language used to describe the physical geometry is custom and therefore non-standard. Use could be made of standard languages such as XML (specifically SVG Scalable Vector Graphics) or tagged DXF (Drawing eXchange Format) files to describe the machine geometry. This would allow the import of machine designs from packages such as AutoCAD. The requirement would then be the definition of the flux paths, this could be manual as in the present case or combined with the previous suggestion for automatic determination.

8.2.6 Generalised model generation

The requirements of the research are that the resultant magnetic equivalent circuit be usable in SPICE simulation packages. The trend for whole system simulation is towards general simulation description languages such as 'Modelica' (51). It would be of benefit if magnetic equivalent circuits could be used within simulation software that uses such general description languages. The benefit of this would also be the ability to more easily incorporate the general 'n+m' air-gap model.

References

[1] Carpenter J., “Understanding Electromagnetism”, IEE Engineering Science and Education Journal, December 1993

[2] Chaudhry S.R., Ahmed-Zaid S., Demerdash N.A. “Coupled finite-element/state-space modelling of turbo generators in the ABC frame of reference – the no-load case”, IEEE Transactions on Energy Conversion, vol. 10, no. 1, March 1995, pp. 56-62

[3] Carpenter C.J., 10/1968, “Magnetic Equivalent Circuits”, IEE Proceedings, vol. 115, no. 10, pp. 1503-1511

[4] Haydock, L., “Systematic Development of Equivalent Circuits for Synchronous Machines”, PhD Thesis, Imperial College, London, 1986

[5] Carpenter C.J., 10/1968, “A Circuit Approach to Field Computation”, IEE Proceedings, vol. 29, no. 2, pp. 1294-1300

[6] Dobbs E.R., “Basic Electromagnetism”, Chapman & Hall, 1993

[7] Mohan et al, “Power Electronics”, John Wiley & Sons, 2nd edition, 1995

[8] Fitzgerald A.E. et al, “Electric Machinery” Metric Edition, McGraw-Hill, Fourth Edition 1988

[9] Ratnajeevan S. Hoole H., “Computer-Aided Analysis and Design of Electromagnetic Devices”, Elsevier, New York, 1989

[10] Carter H.W., “Air-gap induction”, Electronics World, N.Y., 1901, 38, pp. 884-888

[11] Neville S., “Use of Carter’s coefficient with narrow teeth”, Proc. IEE, vol. 114, no. 9, September 1967, pp. 1245-1250

[12] Slemon G.R., “Modelling of induction machines for electric drives”, IEEE Transactions on Industry Applications, vol. 25, no. 6, November/December 1989, pp. 1126-1131

[13] Adkins B. Harley R.G., “The General Theory of Alternating Current Machines: Application to Practical Problems”, Chapman and Hall, London, 1975

[14] Vas P., "Electrical Machines and Drives: A space-vector theory approach", Clarendon Press, Oxford, 1992

[15] Carpenter C.J., "Theory and application of magnetic shells", Proc. IEE, vol. 114, no. 7, July 1967, pp. 995-1000

[16] Bedrosian G., "A New Method for Coupling Finite Element Field Solutions with External Circuits and Kinematics", IEEE Transactions on Magnetics, vol. 29, no.2, March 1993, pp. 1664-1668

[17] Demenko A., "Equivalent RC networks with mutual capacitances for electromagnetic field simulation of electrical machine transients", IEEE Transactions on Industry Applications, vol. 28, no.2, March 1992, pp. 1406-1409

[18] Laithwaite, E.R., 11/1967, "Magnetic equivalent circuits for electrical machines", IEE Proceedings, vol. 114, No. 11, November 1967, pp. 1805-1809

[19] Delforge C., Hecquet M., Brochet., "Bond-graph method applied to coupled electric and magnetic model of electrical devices", pp. 573-578

[20] Ostovic V., "Dynamics of Saturated Electric Machines", Springer-Verlag, New York, USA, 1989

[21] Preston M.A., Lyons J.P., "A Switched Reluctance Motor Model with Mutual Coupling and Multi-Phase Excitation", IEEE Transactions on Magnetics, vol. 27, no.6, March 1991, pp. 5423-5425

[22] Moallem M., Nikkhajoei H., Falahi M., "Predicting the performance of a switched reluctance machine using improved magnetic equivalent circuit method", IEEE catalogue no 95TH8025, 1995, pp. 198-201

[23] Delforge C., Hecquet M., Brochet., "Bond-graph method applied to coupled electric and magnetic model of electrical devices", pp. 573-578

[24] Delforge C., Lemaire-Semail B., "Induction machine modelling using finite element and permeance network methods", IEEE Transactions on Magnetics, vol. 31, no. 3, May 1995, pp. 2092-2095

[25] Brauer J.R., “Simple equations for the magnetisation and reluctivity curves of steel”, IEEE Transactions on Magnetics, 1975, pp. 81

[26] “SIMETRIX circuit simulator user’s manual”, Newbury Technology Ltd.

[27] Baase S., Van Gelder A., “Computer Algorithms”, Addison Wesley, 3rd Edition, 2000

[28] Blundell A.J. "Bond graphs for modelling engineering systems", Ellis Harwood, Chichester, 1982.

[29] Hammond P., Baldomir D., “Dual energy methods in electromagnetism using tubes and slices”, IEE Proceedings, vol. 135, Pt A, No. 3, March 1988

[30] Freeman E.M., “Equivalent circuits from electromagnetic theory: low-frequency induction devices”, IEE Proceedings, Vol. 121, No. 10, October 1974

[31] Gibson A.A.P., Dillon B.M., “Variational solution of lumped element and distributed electrical circuits”, IEE Proceedings Sci. Meas. Technol., Vol. 141, No. 5, September 1994

[32] Jack A.G., Mecrow B.C., “Methods for Magnetically Non-linear Problems Involving Significant Hysteresis and Eddy Currents”, IEEE Transactions on Magnetics, Vol. 26, No. 2, March 1990

[33] Al-Khayat N., “Power Transformer Simulation Models”, PhD Thesis, The Nottingham Trent University, 1994

[34] Wang J.P., Lie D.K., Lorimer W.L., Hartman A., “Comparison of Lumped Parameter and Finite Element Magnetic Modelling in a Brush less DC Motor”, IEEE Transactions on Magnetics, Vol. 33, No. 5 September 1997

[35] Moallem M., Dawson G.E, “An improved magnetic equivalent circuit method for predicting the characteristics of highly saturated electromagnetic devices”, IEEE Transactions on Magnetics, Vol. 34 5 1, September1998

[36] Rasmuuen C.B., Ritchie E., “A magnetic equivalent circuit approach for predicting PM motor performance”, Industry Applications Conference,

1997, Thirty-Second IAS Annual Meeting IAS '97., Conference Record of the 1997 IEEE, Vol. 1, 1997

[37] Hameyer K., Hanitish R., "Numerical optimisation of the electromagnetic field by stochastic search and MEC-model", IEE Transactions on Magnetics, Vol. 30 5 2, Sept. 1994

[38] Van den Bossche, A.; Valchev, V.; Filchev, T., "Improved approximation for fringing permeances in gapped inductors", Industry Applications Conference, 2002. 37th IAS Annual Meeting, vol.2, 932 -938

[39] Vagati A., Pastorelli M., Scapino F., Franceschini G., "Cross-saturation in synchronous reluctance motors of the transverse-laminated type", Industry Applications Conference, 1998, Thirty-Third IAS Annual Meeting, Vol. 1, 1998

[40] Delforge C., Lemare-Semail B., "Induction machine modelling using finite element and permeance network methods", IEEE Transactions on Magnetics, Vol. 31 3, May 1995

[41] Sudhoff S.D., Aliprantis D.C., Kuhn B.T., Chapman P.L. "A introduction Machine Model for Predicting Inverter-Machine Interaction", IEEE Transactions on Energy Conversion, Vol. 17, No. 2, June 2002, pp. 203-210

[42] Suciu C.,Kansara M., Holmes P., Szabo., "Performance Enhancment of an Induction Motor by Secondary Impedance Control", IEEE Transactions on Energy Conversion, Vol. 17, No. 2, June 2002, pp. 211-216

[43] Lancarotte M.S., Penteado, Jr. A.A., "Estimation of Core Losses under Sinusoidal or Non-Sinusoidal Induction by Analysis of Magnetization Rate", IEEE Transactions on Energy Conversion, Vol. 16, No. 2, June 2001, pp. 174-179

[44] Jacobina C.B., Chaves Filho J.E., Noguerira Lima A.M., "Estimating the Parameter of Induction Machines at Standstill", IEEE Transactions on Energy Conversion, Vol. 17, No. 1, March 2001, pp. 85-89

[45] Wilson P.R., Ross J.N., Brown A.D., "Simulation of Magnetic Component Models in Electric Circuits Including Dynamic Thermal Effects", IEEE Transactions on Power Electronics, Vol. 17, No. 1, January 2002, pp. 55-64

[46] Sulivan C.R., "Computationally Efficient Winding Loss Calculation with Multiple Windings, Arbitrary Waveforms, and Two-Dimensional or Three-Dimensional Field Geometry", IEEE Transactions on Energy Conversion, Vol. 16, No. 1, January 2001, pp. 143-150

[47] Shamming W., Xiangheng W., Yixiang L., Pengsheng S., Weiming W., Gaifan Z., "Steady-State Performance of Synchronous Generators with ac and dc Stator Connections Considering Saturation", IEEE Transactions on Energy Conversion, Vol. 17, No. 2, June 2002, pp. 85-89

[48] Kurumbalapitiya D., Ratnajeevan S. and Hoole H., 1993, "An Object-orientated Representation of Electromagnetic Knowledge", IEEE Trans. on Mag., 29 (2), 1939-1942

[49] http://astronomy.swin.edu.au/~pbourke/geomformats/dxf/ (Guide to the Drawing eXchange Format (DXF) graphics format)

[50] http://www.w3.org/Graphics/SVG/Overview.htm8 (World Wide Web Consortium website for extendable Meta Language (XML) derived Scalable Vector Graphics (SVG) language)

[51] http://www.modelica.org/index.shtml (Simulation language resource website)

[52] http://mupad.com (MuPAD computer algebra software website, SciFace Software GmbH & Co.)

[53] Carpenter C.J., "Electromagnetic theory without electric flux", IEE Proceedings-A, Vol. 139, No. 4, July 1992, 189-209

[54] Ratnajeevan S. et al., "Fictitious Minima of Object Functions, Finite Element Meshes, and Edge Elements in Electromagnetic Device Synthesis", IEEE Transactions on Magnetics, Vol. 27, No. 6, November 1991

[55] Maxim A. et al., "A Novel Behavioural Method of SPICE Macro-modelling of Magnetic Components Including the Temperature and Frequency Dependencies", IEEE 1998, 393-399.

[56] Hecquet M., "Modelling of a Claw-Pole Alternator using Permeance Network Coupled with Electric Circuits", IEEE Transactions on Magnetics, Vol. 31, No. 3, May 1995, 2131-2134

[57] McDermott T.E. et al., "Electromechanical System Simulation with Models Generated from Finite Element Solutions", IEEE Transactions on Magnetics, Vol. 33, No. 2, March 1997, 1682-1685

[58] Ratnajeevan S. et al., "Optimization of Electromagnetic Devices: Circuit Models, Neural Networks and Gradient Methods in Concert", IEEE Transactions on Magnetics, Vol31, No. 3, May 1995, 2016-2019

[59] Azzerboni B. et al., "Equivalent Network Modelling of Cylindrical Systems with Conductors in Motion", IEEE Transactions on Magnetics, Vol. 29, No. 2, March 1993,1689-1692

[60] Driesen J. et al. "Coupled Magneto-Thermal Simulation of Thermal Anisotropic Electrical Machines", IEEE 1999, 469-471

[61] Carpenter H., "Simple Models for Dynamic Hysteresis which add Frequency-Dependent Losses to Static Models", IEEE Transactions on Magnetics, Vol. 34, No. 3, May 1998, 619-622

[62] Ramswanny D. et al., "Fast Algorithms for 3-D Simulation", Journal of Modelling and Simulation of Microsystems, Vol. 1, No. 1, 1999, 77-82

[63] De Gersem, H.; Hameyer, K.; Weiland, T., "Skew interface conditions in 2-D finite-element machine models", IEEE Transactions on Magnetics, Volume: 39 Issue: 3 , May 2003, Page(s): 1452 -1455

Appendices

A

Magnetic equivalent circuits including charge and flux duality

The duality of electrical and magnetic circuits has been discussed already in section 2.11.1 where the simalarity between the electrical equation (79) (which relates charge (Q), capacitance (C) and voltage (V)) and the magnetic equation (80) (which relates flux (Φ), permeance (Λ) and magneomotive force (mmf)) has already been noted. The following differential equations (eq. 145 and eq. 146) describe the relationship between the electrical quantities and the magnetic quantities when transforming from the electrical to the magnetic and visa versa.

$$V = -n\frac{d\Phi}{dt} \qquad (145)$$

$$mmf = ni = n\frac{dQ}{dt} \qquad (146)$$

Given the above equations it is possible to use a 4 port electrical device known as a gyrator to connect the electrical and magnetic equivalent circuits. This device transforms a current on one side to a voltage multiplied by a constant (n) on the other.

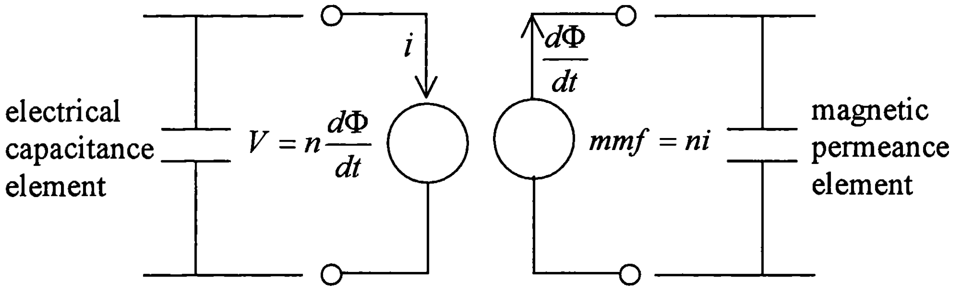


Figure 79: Transformation between electrical and magnetic equivalent circuits

Figure 79 illustrates the use of a gyrator to connect an electrical and magnetic equivalent circuit. The point within the simulation circuit where such devices are placed is where, if the magnetic circuit is pulled from the electrical circuit, the flux paths ‘cut’ through the electrical wiring or visa versa.

Appendices

B

'A novel technique to derive an analogue hardware description language model of a 2.2kw induction motor.'

Third International Conference on COMPUTATION IN ELECTROMAGNETICS, University of Bath, 10-12 April 1996

A NOVEL TECHNIQUE TO DERIVE AN ANALOGUE HARDWARE DESCRIPTION LANGUAGE MODEL OF A 2.2KW INDUCTION MOTOR.

D. Downes, P.G. Holmes, B. Patterson, J.M.K. Pratt, M. Kansara

The Nottingham Trent University

INTRODUCTION.

In machine design, especially where concurrent engineering is applied, the simulation of an electromagnetic device and its control electronics is becoming necessary. The problem therefore arises as to which domain does the simulation properly belong, a finite element package with circuit elements or a circuit simulation package with behavioural modelling of the electromagnetic device.

The view taken is that both approaches are necessary, depending on the particular aims of a simulation such as optimisation of the electromagnetic device or control circuit. An approach to allow the relative ease of movement between the two domains, without reformatting the data relating to machine geometry and configuration is part of the current research programme.

Haydock(1) and Ostovic(2) have shown that an approach based on magnetic circuits can lead to shorter solution times with a small detrimental effect on the accuracy of the solution. It has been shown (1) that the classic two axis equivalent circuits can be derived directly from the machines magnetic circuits by using a suitable transform from one side of the gyrator to the other. Haydock showed that a magnetic capacitor changes to an electrical inductance by the reciprocity relationship.

To be able to derive the values of the components in the magnetic circuit, the flux paths need to be known. The basis of the paper is not the calculation of the flux paths, but the automation of the calculation of the magnetic circuit. In the present paper the flux paths will be explicitly defined. The numerical calculation of the flux path is the subject of future work.

The intended application is to use the resulting network within an Analogue Hardware Description Language (AHDL) simulation package, so as to allow models of different complexity to be used within the same simulation. The simulation results are based on a standard 'SPICE' network description, and of course this does not fully describe the mechanical properties of the machine and its interaction with the electromagnetic field. This could be described by a behavioural model with the magnetic circuit still entered as a spice network within the model.

A LANGUAGE FOR MACHINE DESCRIPTION.

Previous work by Kurumbalaptitiya et al (3) has successfully applied Object Orientated Programming (OOP) concepts to the representation of electromagnetic knowledge. In order to allow implementation of the complex software in a reasonable time a similar approach is adopted for the representation of machine geometry and physical relationships. Initially only two dimensional design information is considered but the approach adopted is inherently scaleable to three dimensions.

The key concepts in terms of the machines representation, which translate directly into software objects are as follows:

- A global co-ordinate reference frame in which all other defined co-ordinate reference frames must exist.
- A reference FRAME that has a particular co-ordinate system and contains co-ordinate nodes. The location of the frame within a global frame and whether it has any degrees of movement with respect to the global reference frame.
- Co-ordinate NODES that can only belong to one particular frame and are defined by position according to the particular co-ordinate system used by the frame.
- CURVES must begin at one node and end at a second node.
- PARTS that are defined in terms of the CURVES to form closed areas. Physical information such as the materials electromagnetic properties are also associated with a particular part.
- Various other objects are also derived so as to embody other items of knowledge such as a particular materials B-H curve.

The relationships between the various objects mentioned above is shown in Fig. 1 and C++ has been used to implement the software models of the machine.

Part of the syntax for the description language used specifically for a rotational machine is shown in Fig. 2. The optional items provide a means to describe redundancy due to symmetry or repeated sections within the machines structure. Names used to denote particular parts, curves etc. are automatically cross-referenced when the input is parsed by the program.

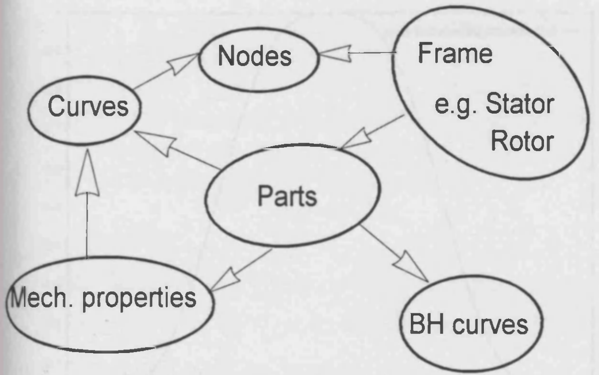


Figure 1 Software objects and reference links

```
bh < name >
{ < linear | exp > < value [ , value ,value ] > }

stator
{ node < name [ [ start_integer , no._of_repetitions ,
inc_integer ] ] > < value | [ start_float , inc_float ]
> , < value | [ start_float , inc_float ] > }

rotor
{ node < name [ [ start_integer , no._of_repetitions ,
inc_integer ] ] > < value | [ start_float , inc_float ] >
, < value | [ start_float , inc_float ] > }

curve < name [ [ start_integer , no._of_repetitions ,
                                   inc_integer ] ] >
{ < line | circ > < name [ [ start_integer , inc_integer
] ] > , < name [ [ start_integer , inc_integer ] ] > [ ,
< value | [ start_float , inc_float ] > ] }

part < name >
{ frame <stator | rotor > bh < name >
curve < name [ [ start_integer , no._of_repetitions ,
                                   inc_integer ] ] >
electrical                    <front_terminal_name>,
          <back_terminal_name> }
```

Figure 2 Syntax of rotationally constrained description

Checks are made on the machine description, to determine that it is consistent and complete. These checks include the following:

- every node must have at least two curves associated;
- each part has a closed set of curves;
- only one boundary set of curves exist for the device, no 'holes' exist in the space defined by the various parts;
- references are not made to non existent items;
- no parts overlap.

Application of the description to magnetic circuits.
It has been shown (2) that the reluctance of a flux path can be derived using the following equation:

$$Rm = \int_0^l \frac{dx}{\mu(x)A(x)} \qquad (1)$$

Where Rm = magnetic reluctance of path.
μ(x) = permeability along integration path.
A(x) = area normal to integration path.
l = length of path of integration.

It is assumed that for each part that has been defined, the flux enters and leaves through two curves associated with the part.
The direction of integration is therefore defined for each part, where each part is a flux path definition. The resulting magnetic circuit is therefore defined by the common curves shared by adjacent parts.
The value of μ is assumed to be homogeneous throughout any part, and has already been associated with a particular B-H curve via the machine description. The non-linear magnetic capacitance can therefore be derived, from the path integral of area and a non-linear function which approximates the materials magnetic saturation properties.
For present purpose, rotational velocity is assumed to be constant and the effect of motion is considered by varying the permanence between a point on the stator and on the rotor as in (2). The following function of the displacement angle is used to give the various values of stator to rotor permanence:

$$\Pr(\theta) = e^{-a\left|\sin((\theta+b)/2)\right|^{c}} \qquad (2)$$

where Pr(θ) = relative permeance of airgap.
θ = displacement angle.
b = displacement angle offset
a = 1st co-efficient of curve fit.
c = 2nd co-efficient of curve fit.

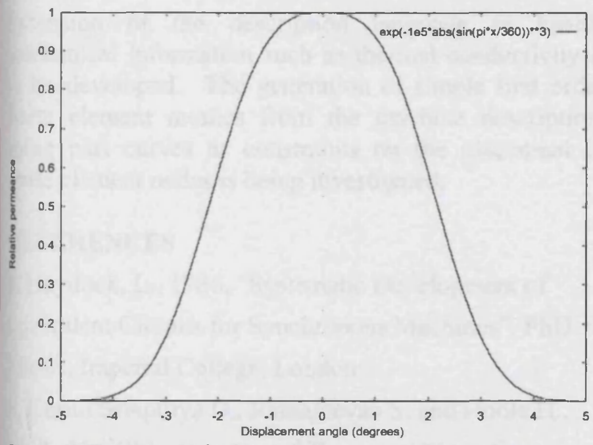


Figure 3 Variation of relative permeance with angle

The relative permeance as a function of displacement angle varies between 0 and 1 (see fig. 3 for example with maximum permeance centred on 10 degrees displacement). The function can be implemented using a standard non-linear voltage source in a spice simulator.

Experimental Testing

The methodology was verified by performing a physical test on a 4 pole 3 phase 415V a.c. 2.2KW induction motor with a skewed 32 bar rotor and 36 slot concentric wound stator.

Test results from the induction machine were captured using hall effect current probes and a high speed data acquisition board.

Simulation

A machine description was produced from the design data made available by the machine manufacturer of the tested machine. Then simple flux paths were defined and the resulting network listing obtained from the conversion program. A flowchart showing the steps involved is shown in fig. 4. The resulting model file was used in a SPICE 3 simulator and the simulation results obtained are illustrated in fig. 6.

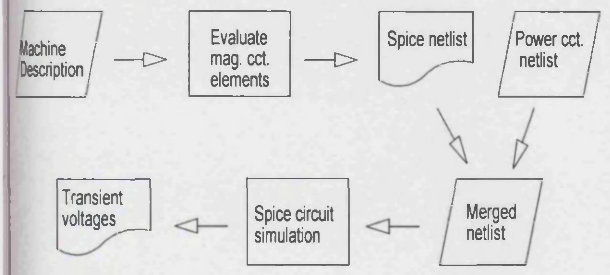


Figure 4 Flowchart

The plot is of the steady state current for one of the phases with the rotor speed at 1425 rpm. Similar scaling has been used in both the simulation and experimental test result to allow qualitative comparison of the two waveforms.

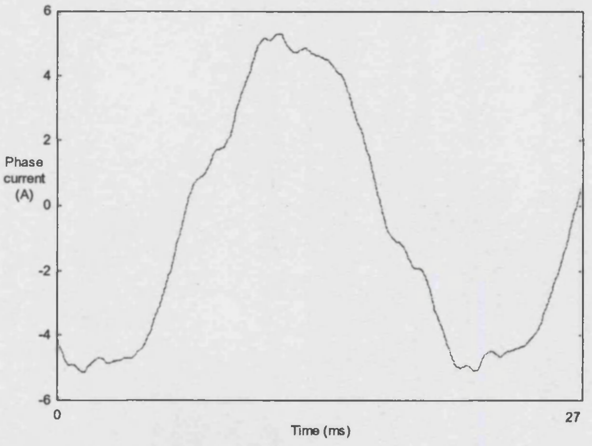


Figure 5 Test results (Light load)

Results

The experimental results produced in the laboratory for the steady state phase current are shown in fig. 6.

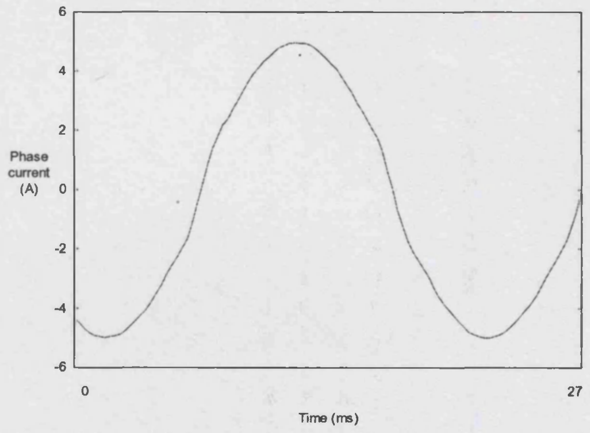


Figure 6 Simulation results (Light load)

By comparing the results from the test and simulation (Fig. 6) a correlation between the two curves can be seen.

CONCLUSION

A machine description language and interpreter has been developed that can be used as the basis for further work. Automation of the generation of equivalent circuits has been achieved by incorporating the suggested flux paths within the machine description. The comparison of the simulation results with the experimental results has produced similar waveforms. The difference between the experimental and simulation results is mainly due to the coarse model of the machine. This model will not take into account the effect of space harmonics, but the flat top on the waveform shows the presence of a 3rd harmonic component. Nevertheless this is a first attempt at synthesis of a model and the experimental software is now being refined to include more complex geometrical and electromagnetic detail.

FURTHER WORK

Appendices

C

GDL description file of machine

```
# simple machine

unitdistance 0.001

bh {
  newcor linear 2000 # exp 3.8 2.17 396.2 # linear 2000
  air linear 1
}

stator
{

  coils {  sw1 81 sw1a sw1b sw2 81 sw2a sw2b sw3 81 sw3a sw3b sw4 81 sw4a sw4b
      sw5 81 sw5a sw5b sw6 81 sw6a sw6b }

  nodes {
    sn0 0 d 0
    sn[1 12 20] [46.04 0] d [-1.56 30]
    sn[2 12 20] [46.294 0] d [-1.55 30]
    sn[3 12 20] [46.802 0] d [-2.15 30]
    sn[4 12 20] [47.074 0] d [-2.47 30]
    sn[5 12 20] [58.92 0] d [-3.09 30]
    sn[6 12 20] [46.04 0] d [1.56 30]
    sn[7 12 20] [46.294 0] d [1.55 30]
    sn[8 12 20] [46.802 0] d [2.15 30]
    sn[9 12 20] [47.074 0] d [2.47 30]
    sn[10 12 20] [58.92 0] d [3.09 30]
    sn[11 12 20] [62 0] d [15 30]
    sn[12 12 20] [76.45 0] d [15 30]
    sn[13 12 20] [45.82 0] d [-1.56 30]
    sn[14 12 20] [45.82 0] d [1.56 30]
  }

  curves {
    sc[1 11 20] circ sn[6 20] sn[21 20] [46.04 0]
    sc221 circ sn226 sn1 46.04
    sc[2 12 20] line sn[1 20] sn[2 20]
    sc[3 12 20] circ sn[3 20] sn[2 20] [0.508 0]
    sc[4 12 20] circ sn[3 20] sn[4 20] [0.84 0]
    sc[5 12 20] line sn[4 20] sn[5 20]
    sc[6 12 20] circ sn[5 20] sn[10 20] [3.18 0]
    sc[7 12 20] line sn[6 20] sn[7 20]
    sc[8 12 20] circ sn[7 20] sn[8 20] [0.84 0]
    sc[9 12 20] circ sn[9 20] sn[8 20] [0.508 0]
    sc[10 12 20] line sn[9 20] sn[10 20]
    sc[11 12 20] line sn[10 20] sn[11 20]
    sc[12 11 20] line sn[11 20] sn[25 20]
    sc232 line sn231 sn5
    sc[13 12 20] line sn[11 20] sn[12 20]
    sc[14 11 20] circ sn[12 20] sn[32 20] [76.45 0]
    sc234 circ sn232 sn12 76.45
    sc[15 12 20] line sn[1 20] sn[13 20]
    sc[16 12 20] line sn[6 20] sn[14 20]
    sc[17 11 20] circ sn[14 20] sn[33 20] [45.82 0]
    sc237 circ sn234 sn13 45.82
    sc[18 12 20] line sn[4 20] sn[9 20]
    sc[19 12 20] line sn[1 20] sn[6 20]
  }

  parts {
    sic1 {
      length 96
      bh newcor
      coils { sw1 }
      curves { sc1 sc7 sc8 sc9 sc10 sc11 sc12 sc22 sc23 sc24 sc25 }
      entry { sc11 sc12 }
      exit { sc1 sc7 sc8 sc22 sc23 }
    }
    sl1 {
      length 96
      bh air
      curves { sc2 sc3 sc4 sc18 sc19 sc7 sc8 sc9 }
      entry { sc7 sc8 }
      exit { sc2 sc3 }
    }
```

```
sib1 {
  length 96
  bh newcor
  curves { sc6 sc11 sc13 sc232 sc233 sc234 }
  entry { sc11 sc13 }
  exit { sc232 sc233 }
}
sag1 {
  length 96
  bh air
  curves { sc1 sc16 sc35 sc17 }
  entry { sc1 }
  exit { sc17 }
}

sic2 {
  length 96
  bh newcor
  coils { sw1 sw3 }
  curves { sc21 sc27 sc28 sc29 sc30 sc31 sc32 sc42 sc43 sc44 sc45 }
  entry { sc31 sc32 }
  exit { sc21 sc27 sc28 sc42 sc43 }
}
sl2 {
  length 96
  bh air
  curves { sc22 sc23 sc24 sc38 sc39 sc27 sc28 sc29 }
  entry { sc27 sc28 }
  exit { sc22 sc23 }
}
sib2 {
  length 96
  bh newcor
  curves { sc26 sc31 sc33 sc12 sc13 sc14 }
  entry { sc31 sc33 }
  exit { sc12 sc13 }
}
sag2 {
  length 96
  bh air
  curves { sc21 sc36 sc55 sc37 }
  entry { sc21 }
  exit { sc37 }
}

sic3 {
  length 96
  bh newcor
  coils { sw3 }
  curves { sc41 sc47 sc48 sc49 sc50 sc51 sc52 sc62 sc63 sc64 sc65 }
  entry { sc51 sc52 }
  exit { sc41 sc47 sc48 sc62 sc63 }
}
sl3 {
  length 96
  bh air
  curves { sc42 sc43 sc44 sc58 sc59 sc47 sc48 sc49 }
  entry { sc47 sc48 }
  exit { sc42 sc43 }
}
sib3 {
  length 96
  bh newcor
  curves { sc46 sc51 sc53 sc32 sc33 sc34 }
  entry { sc51 sc53 }
  exit { sc32 sc33 }
}
sag3 {
  length 96
  bh air
  curves { sc41 sc56 sc75 sc57 }
  entry { sc41 }
  exit { sc57 }
}
```

```
sic4 {
  length 96
  bh newcor
  coils { sw3 sw5 }
  curves { sc61 sc67 sc68 sc69 sc70 sc71 sc72 sc82 sc83 sc84 sc85 }
  entry { sc71 sc72 }
  exit { sc61 sc67 sc68 sc82 sc83 }
}
sl4 {
  length 96
  bh air
  curves { sc62 sc63 sc64 sc78 sc79 sc67 sc68 sc69 }
  entry { sc67 sc68 }
  exit { sc62 sc63 }
}
sib4 {
  length 96
  bh newcor
  curves { sc66 sc71 sc73 sc52 sc53 sc54 }
  entry { sc71 sc73 }
  exit { sc52 sc53 }
}
sag4 {
  length 96
  bh air
  curves { sc61 sc76 sc95 sc77 }
  entry { sc61 }
  exit { sc77 }
}

sic5 {
  length 96
  bh newcor
  coils { sw5 }
  curves { sc81 sc87 sc88 sc89 sc90 sc91 sc92 sc102 sc103 sc104 sc105 }
  entry { sc91 sc92 }
  exit { sc81 sc87 sc88 sc102 sc103 }
}
sl5 {
  length 96
  bh air
  curves { sc82 sc83 sc84 sc98 sc99 sc87 sc88 sc89 }
  entry { sc87 sc88 }
  exit { sc82 sc83 }
}
sib5 {
  length 96
  bh newcor
  curves { sc86 sc91 sc93 sc72 sc73 sc74 }
  entry { sc91 sc93 }
  exit { sc72 sc73 }
}
sag5 {
  length 96
  bh air
  curves { sc81 sc96 sc115 sc97 }
  entry { sc81 }
  exit { sc97 }
}

sic6 {
  length 96
  bh newcor
  coils { sw5 sw2 }
  curves { sc101 sc107 sc108 sc109 sc110 sc111 sc112 sc122 sc123 sc124 sc125 }
  entry { sc111 sc112 }
  exit { sc101 sc107 sc108 sc122 sc123 }
}
sl6 {
  length 96
  bh air
  curves { sc102 sc103 sc104 sc118 sc119 sc107 sc108 sc109 }
  entry { sc107 sc108 }
  exit { sc102 sc103 }
}
```

```
sib6 {
  length 96
  bh newcor
  curves { sc106 sc111 sc113 sc92 sc93 sc94 }
  entry { sc111 sc113 }
  exit { sc92 sc93 }
}
sag6 {
  length 96
  bh air
  curves { sc101 sc116 sc135 sc117 }
  entry { sc101 }
  exit { sc117 }
}

sic7 {
  length 96
  bh newcor
  coils { sw2 }
  curves { sc121 sc127 sc128 sc129 sc130 sc131 sc132 sc142 sc143 sc144 sc145 }
  entry { sc131 sc132 }
  exit { sc121 sc127 sc128 sc142 sc143 }
}
sl7 {
  length 96
  bh air
  curves { sc122 sc123 sc124 sc138 sc139 sc127 sc128 sc129 }
  entry { sc127 sc128 }
  exit { sc122 sc123 }
}
sib7 {
  length 96
  bh newcor
  curves { sc126 sc131 sc133 sc112 sc113 sc114 }
  entry { sc131 sc133 }
  exit { sc112 sc113 }
}
sag7 {
  length 96
  bh air
  curves { sc121 sc136 sc155 sc137 }
  entry { sc121 }
  exit { sc137 }
}

sic8 {
  length 96
  bh newcor
  coils { sw4 sw2 }
  curves { sc141 sc147 sc148 sc149 sc150 sc151 sc152 sc162 sc163 sc164 sc165 }
  entry { sc151 sc152 }
  exit { sc141 sc147 sc148 sc162 sc163 }
}
sl8 {
  length 96
  bh air
  curves { sc142 sc143 sc144 sc158 sc159 sc147 sc148 sc149 }
  entry { sc147 sc148 }
  exit { sc142 sc143 }
}
sib8 {
  length 96
  bh newcor
  curves { sc146 sc151 sc153 sc132 sc133 sc134 }
  entry { sc151 sc153 }
  exit { sc132 sc133 }
}
sag8 {
  length 96
  bh air
  curves { sc141 sc156 sc175 sc157 }
  entry { sc141 }
  exit { sc157 }
}
```

```
sic9 {
  length 96
  bh newcor
  coils { sw4 }
  curves { sc161 sc167 sc168 sc169 sc170 sc171 sc172 sc182 sc183 sc184 sc185 }
  entry { sc171 sc172 }
  exit { sc161 sc167 sc168 sc182 sc183 }
}
sl9 {
  length 96
  bh air
  curves { sc162 sc163 sc164 sc178 sc179 sc167 sc168 sc169 }
  entry { sc167 sc168 }
  exit { sc162 sc163 }
}
sib9 {
  length 96
  bh newcor
  curves { sc166 sc171 sc173 sc152 sc153 sc154 }
  entry { sc171 sc173 }
  exit { sc152 sc153 }
}
sag9 {
  length 96
  bh air
  curves { sc161 sc176 sc195 sc177 }
  entry { sc161 }
  exit { sc177 }
}

sic10 {
  length 96
  bh newcor
  coils { sw4 sw6 }
  curves { sc181 sc187 sc188 sc189 sc190 sc191 sc192 sc202 sc203 sc204 sc205 }
  entry { sc191 sc192 }
  exit { sc181 sc187 sc188 sc202 sc203 }
}
sl10 {
  length 96
  bh air
  curves { sc182 sc183 sc184 sc198 sc199 sc187 sc188 sc189 }
  entry { sc187 sc188 }
  exit { sc182 sc183 }
}
sib10 {
  length 96
  bh newcor
  curves { sc186 sc191 sc193 sc172 sc173 sc174 }
  entry { sc191 sc193 }
  exit { sc172 sc173 }
}
sag10 {
  length 96
  bh air
  curves { sc181 sc196 sc215 sc197 }
  entry { sc181 }
  exit { sc197 }
}

sic11 {
  length 96
  bh newcor
  coils { sw6 }
  curves { sc201 sc207 sc208 sc209 sc210 sc211 sc212 sc222 sc223 sc224 sc225 }
  entry { sc211 sc212 }
  exit { sc201 sc207 sc208 sc222 sc223 }
}
sl11 {
  length 96
  bh air
  curves { sc202 sc203 sc204 sc218 sc219 sc207 sc208 sc209 }
  entry { sc207 sc208 }
  exit { sc202 sc203 }
}
```

```
    sib11 {
      length 96
      bh newcor
      curves { sc206 sc211 sc213 sc192 sc193 sc194 }
      entry { sc211 sc213 }
      exit { sc192 sc193 }
    }
    sag11 {
      length 96
      bh air
      curves { sc201 sc216 sc235 sc217 }
      entry { sc201 }
      exit { sc217 }
    }

    sic12 {
      length 96
      bh newcor
      coils { sw6 sw1 }
      curves { sc221 sc227 sc228 sc229 sc230 sc231 sc232 sc2 sc3 sc4 sc5 }
      entry { sc231 sc232 }
      exit { sc221 sc227 sc228 sc2 sc3 }
    }
    sl12 {
      length 96
      bh air
      curves { sc222 sc223 sc224 sc238 sc239 sc227 sc228 sc229 }
      entry { sc227 sc228 }
      exit { sc222 sc223 }
    }
    sib12 {
      length 96
      bh newcor
      curves { sc226 sc231 sc233 sc212 sc213 sc214 }
      entry { sc231 sc233 }
      exit { sc212 sc213 }
    }
    sag12 {
      length 96
      bh air
      curves { sc221 sc236 sc15 sc237 }
      entry { sc221 }
      exit { sc237 }
    }

  }
}

rotor {

  skewangle d 30

  coils {  rw1 324 rb1a rb1b }

  nodes {
    rn0 0 d 0
    rn[1 2 20] [31 0] d [-25 180]
    rn[2 2 20] [45.6 0] d [-25 180]
    rn[3 2 20] [45.82 0] d [-25 180]
    rn[4 2 20] [31 0] d [25 180]
    rn[5 2 20] [45.6 0] d [25 180]
    rn[6 2 20] [45.82 0] d [25 180]
    rn[7 2 20] [15.875 0] d [0 180]
    rn[8 2 20] [31 0] d [0 180]
    rn[9 2 20] [40 0] d [-25 180]
    rn[10 2 20] [40 0] d [25 180]
  }

  curves {
    rc[1 2 20] circ rn[3 20] rn[6 20] [45.82 0]
    rc[2 2 20] circ rn[2 20] rn[5 20] [45.6 0]
    rc[3 2 20] line rn[1 20] rn[9 20]
    rc[4 2 20] line rn[2 20] rn[3 20]
    rc[5 2 20] line rn[4 20] rn[10 20]
    rc[6 2 20] line rn[5 20] rn[6 20]
  rc[7 2 20] circ rn[1 20] rn[8 20] [31 0]
  rc[8 2 20] circ rn[8 20] rn[4 20] [31 0]
  rc[9 2 20] line rn[7 20] rn[8 20]
  rc10 circ rn7 rn27 15.88
  rc30 circ rn27 rn7 15.88
  rc11 circ rn4 rn21 31
  rc31 circ rn24 rn1 31
  rc12 circ rn10 rn29 40
  rc32 circ rn30 rn9 40
  rc13 circ rn5 rn22 45.6
  rc33 circ rn25 rn2 45.6
  rc[14 2 20] line rn[9 20] rn[2 20]
  rc[15 2 20] line rn[10 20] rn[5 20]
}

parts {
  ric1 {
    length 98
    bh newcor
    coils { rw1 }
    curves { rc2 rc3 rc5 rc7 rc8 rc14 rc15 }
    entry { rc2 rc14 rc15 }
    exit { rc7 rc8 }
  }
  rip1 {
    length 98
    bh newcor
    curves { rc8 rc9 rc10 rc11 rc27 rc29 }
    entry { rc8 }
    exit { rc27 }
  }
  rag1 {
    length 98
    bh air
    curves { rc1 rc2 rc4 rc6 }
    entry { rc1 }
    exit { rc2 }
  }
  rl1 {
    length 98
    bh air
    curves { rc12 rc13 rc15 rc34 }
    entry { rc15 }
    exit { rc34 }
  }

  ric2 {
    length 98
    bh newcor
    curves { rc22 rc23 rc25 rc27 rc28 rc34 rc35 }
    entry { rc22 rc34 rc35 }
    exit { rc27 rc28 }
  }
  rip2 {
    length 98
    bh newcor
    curves { rc7 rc9 rc28 rc29 rc30 rc31 }
    entry { rc28 }
    exit { rc7 }
  }
  rag2 {
    length 98
    bh air
    curves { rc21 rc22 rc24 rc26 }
    entry { rc21 }
    exit { rc22 }
  }
  rl2 {
    length 98
    bh air
    curves { rc32 rc33 rc35 rc14 }
    entry { rc35 }
    exit { rc14 }
  }
}
```

Appendices

D

Spice circuit description from conversion program

```
* Circuit description generated by GDL program
.subckt magnetic TORQ_P TORQ_N POS_P POS_N
+ sw1a sw1b sw2a sw2b sw3a sw3b sw4a sw4b sw5a sw5b sw6a sw6b sw7a sw7b sw8a sw8b sw9a
sw9b sw10a sw10b sw11a sw11b sw12a sw12b
+ rb1a rb1b
************************************
* Stator magnetic circuit elements *
************************************
* part 'sib8' length: 0.096 integral: 14.673 entry cct node: 1 exit cct node: 2
* magnetic capacitance:
Csib8 1 2 0.000685121 IC=0
* coils:
* part 'sic2' length: 0.096 integral: 5.62713 entry cct node: 3 exit cct node: 4
* magnetic capacitance:
Csic2 3 43 0.00178649 IC=0
* coils:
Vsenssic2 43 44 0
Hsic2sw2 4 44 Vsenssw2 -81
* part 'sib2' length: 0.096 integral: 14.6818 entry cct node: 3 exit cct node: 5
* magnetic capacitance:
Csib2 3 5 0.000684709 IC=0
* coils:
* part 'sl11' length: 0.096 integral: 30.8315 entry cct node: 8 exit cct node: 9
* magnetic capacitance:
Csl11 8 9 4.0757e-08 IC=0
* coils:
* part 'sic1' length: 0.096 integral: 5.62713 entry cct node: 5 exit cct node: 10
* magnetic capacitance:
Csic1 5 45 0.00178649 IC=0
* coils:
Vsenssic1 45 46 0
Hsic1sw1 10 46 Vsenssw1 -81
* part 'sib1' length: 0.096 integral: 14.673 entry cct node: 5 exit cct node: 11
* magnetic capacitance:
Csib1 5 11 0.000685121 IC=0
* coils:
* part 'sic11' length: 0.096 integral: 5.62713 entry cct node: 16 exit cct node: 8
* magnetic capacitance:
Csic11 16 47 0.00178649 IC=0
* coils:
Vsenssic11 47 48 0
Hsic11sw11 8 48 Vsenssw11 -81
* part 'sib10' length: 0.096 integral: 14.673 entry cct node: 17 exit cct node: 18
* magnetic capacitance:
Csib10 17 18 0.000685121 IC=0
* coils:
* part 'sic6' length: 0.096 integral: 5.62713 entry cct node: 19 exit cct node: 20
* magnetic capacitance:
Csic6 19 49 0.00178649 IC=0
* coils:
Vsenssic6 49 50 0
Hsic6sw6 20 50 Vsenssw6 -81
* part 'sib11' length: 0.096 integral: 14.673 entry cct node: 16 exit cct node: 17
* magnetic capacitance:
Csib11 16 17 0.000685121 IC=0
* coils:
* part 'sic10' length: 0.096 integral: 5.62713 entry cct node: 17 exit cct node: 9
* magnetic capacitance:
Csic10 17 51 0.00178649 IC=0
* coils:
Vsenssic10 51 52 0
Hsic10sw10 9 52 Vsenssw10 -81
* part 'sib6' length: 0.096 integral: 14.673 entry cct node: 19 exit cct node: 21
* magnetic capacitance:
Csib6 19 21 0.000685121 IC=0
* coils:
* part 'sib12' length: 0.096 integral: 14.673 entry cct node: 11 exit cct node: 16
* magnetic capacitance:
Csib12 11 16 0.000685121 IC=0
* coils:
* part 'sic12' length: 0.096 integral: 5.62713 entry cct node: 11 exit cct node: 22
* magnetic capacitance:
Csic12 11 53 0.00178649 IC=0
* coils:
Vsenssic12 53 54 0
```

```
Hsic12sw12 22 54 Vsenssw12 -81
* part 'sic5' length: 0.096 integral: 5.62713 entry cct node: 21 exit cct node: 23
* magnetic capacitance:
Csic5 21 55 0.00178649 IC=0
* coils:
Vsenssic5 55 56 0
Hsic5sw5 23 56 Vsenssw5 -81
* part 'sib5' length: 0.096 integral: 14.673 entry cct node: 21 exit cct node: 24
* magnetic capacitance:
Csib5 21 24 0.000685121 IC=0
* coils:
* part 'sl12' length: 0.096 integral: 30.8315 entry cct node: 22 exit cct node: 8
* magnetic capacitance:
Csl12 22 8 4.0757e-08 IC=0
* coils:
* part 'sic4' length: 0.096 integral: 5.62713 entry cct node: 24 exit cct node: 12
* magnetic capacitance:
Csic4 24 57 0.00178649 IC=0
* coils:
Vsenssic4 57 58 0
Hsic4sw4 12 58 Vsenssw4 -81
* part 'sl1' length: 0.096 integral: 30.8315 entry cct node: 10 exit cct node: 22
* magnetic capacitance:
Csl1 10 22 4.0757e-08 IC=0
* coils:
* part 'sib4' length: 0.096 integral: 14.673 entry cct node: 24 exit cct node: 31
* magnetic capacitance:
Csib4 24 31 0.000685121 IC=0
* coils:
* part 'sl2' length: 0.096 integral: 30.8315 entry cct node: 4 exit cct node: 10
* magnetic capacitance:
Csl2 4 10 4.0757e-08 IC=0
* coils:
* part 'sic9' length: 0.096 integral: 5.62713 entry cct node: 18 exit cct node: 6
* magnetic capacitance:
Csic9 18 59 0.00178649 IC=0
* coils:
Vsenssic9 59 60 0
Hsic9sw9 6 60 Vsenssw9 -81
* part 'sl3' length: 0.096 integral: 30.8315 entry cct node: 14 exit cct node: 4
* magnetic capacitance:
Csl3 14 4 4.0757e-08 IC=0
* coils:
* part 'sib9' length: 0.096 integral: 14.673 entry cct node: 18 exit cct node: 1
* magnetic capacitance:
Csib9 18 1 0.000685121 IC=0
* coils:
* part 'sl4' length: 0.096 integral: 30.8315 entry cct node: 12 exit cct node: 14
* magnetic capacitance:
Csl4 12 14 4.0757e-08 IC=0
* coils:
* part 'sl5' length: 0.096 integral: 30.8315 entry cct node: 23 exit cct node: 12
* magnetic capacitance:
Csl5 23 12 4.0757e-08 IC=0
* coils:
* part 'sl6' length: 0.096 integral: 30.8315 entry cct node: 20 exit cct node: 23
* magnetic capacitance:
Csl6 20 23 4.0757e-08 IC=0
* coils:
* part 'sl7' length: 0.096 integral: 30.8315 entry cct node: 27 exit cct node: 20
* magnetic capacitance:
Csl7 27 20 4.0757e-08 IC=0
* coils:
* part 'sl8' length: 0.096 integral: 30.8315 entry cct node: 25 exit cct node: 27
* magnetic capacitance:
Csl8 25 27 4.0757e-08 IC=0
* coils:
* part 'sl9' length: 0.096 integral: 30.8315 entry cct node: 6 exit cct node: 25
* magnetic capacitance:
Csl9 6 25 4.0757e-08 IC=0
* coils:
* part 'sic3' length: 0.096 integral: 5.62713 entry cct node: 31 exit cct node: 14
* magnetic capacitance:
Csic3 31 61 0.00178649 IC=0
* coils:
```

```
Vsenssic3 61 62 0
Hsic3sw3 14 62 Vsenssw3 -81
* part 'sib3' length: 0.096 integral: 14.673 entry cct node: 31 exit cct node: 3
* magnetic capacitance:
Csib3 31 3 0.000685121 IC=0
* coils:
* part 'sl10' length: 0.096 integral: 30.8315 entry cct node: 9 exit cct node: 6
* magnetic capacitance:
Csl10 9 6 4.0757e-08 IC=0
* coils:
* part 'sic7' length: 0.096 integral: 5.62713 entry cct node: 2 exit cct node: 27
* magnetic capacitance:
Csic7 2 63 0.00178649 IC=0
* coils:
Vsenssic7 63 64 0
Hsic7sw7 27 64 Vsenssw7 -81
* part 'sib7' length: 0.096 integral: 14.673 entry cct node: 2 exit cct node: 19
* magnetic capacitance:
Csib7 2 19 0.000685121 IC=0
* coils:
* part 'sic8' length: 0.096 integral: 5.62713 entry cct node: 1 exit cct node: 25
* magnetic capacitance:
Csic8 1 65 0.00178649 IC=0
* coils:
Vsenssic8 65 66 0
Hsic8sw8 25 66 Vsenssw8 -81
* Grounding resistors:
RGND1 1 0 1G
RGND2 2 0 1G
RGND3 3 0 1G
RGND4 4 0 1G
RGND5 5 0 1G
RGND8 8 0 1G
RGND9 9 0 1G
RGND10 10 0 1G
RGND11 11 0 1G
RGND16 16 0 1G
RGND17 17 0 1G
RGND18 18 0 1G
RGND19 19 0 1G
RGND20 20 0 1G
RGND21 21 0 1G
RGND22 22 0 1G
RGND23 23 0 1G
RGND24 24 0 1G
RGND12 12 0 1G
RGND31 31 0 1G
RGND6 6 0 1G
RGND14 14 0 1G
RGND27 27 0 1G
RGND25 25 0 1G
***********************************
* Rotor magnetic circuit elements *
***********************************
* part 'rip1' length: 0.098 integral: 34.0226 entry cct node: 37 exit cct node: 38
* magnetic capacitance:
Crip1 37 38 0.000295474 IC=0
* coils:
* part 'ric2' length: 0.098 integral: 2.41523 entry cct node: 39 exit cct node: 38
* magnetic capacitance:
Cric2 39 38 0.00416225 IC=0
* coils:
* part 'ric1' length: 0.098 integral: 2.41523 entry cct node: 40 exit cct node: 37
* magnetic capacitance:
Cric1 40 67 0.00416225 IC=0
* coils:
Vsensric1 67 68 0
Hric1rw1 37 68 Vsensrw1 -324
* part 'rl1' length: 0.098 integral: 153.243 entry cct node: 40 exit cct node: 39
* magnetic capacitance:
Crl1 40 39 8.20004e-09 IC=0
* coils:
* part 'rl2' length: 0.098 integral: 153.243 entry cct node: 39 exit cct node: 40
* magnetic capacitance:
Crl2 39 40 8.20004e-09 IC=0
```

```
* coils:
* part 'rip2' length: 0.098 integral: 34.0226 entry cct node: 38 exit cct node: 37
* magnetic capacitance:
Crip2 38 37 0.000295474 IC=0
* coils:
* Grounding resistors:
RGND37 37 0 1G
RGND38 38 0 1G
RGND39 39 0 1G
RGND40 40 0 1G
******************************************************
* Connection of stator coils to circuit terminations *
******************************************************
Vsenssw1 sw1a 69 0
Hsw1sic1 sw1b 69 Vsenssic1 81
Vsenssw2 sw2a 70 0
Hsw2sic2 sw2b 70 Vsenssic2 81
Vsenssw3 sw3a 71 0
Hsw3sic3 sw3b 71 Vsenssic3 81
Vsenssw4 sw4a 72 0
Hsw4sic4 sw4b 72 Vsenssic4 81
Vsenssw11 sw11a 73 0
Hsw11sic11 sw11b 73 Vsenssic11 81
Vsenssw5 sw5a 74 0
Hsw5sic5 sw5b 74 Vsenssic5 81
Vsenssw6 sw6a 75 0
Hsw6sic6 sw6b 75 Vsenssic6 81
Vsenssw7 sw7a 76 0
Hsw7sic7 sw7b 76 Vsenssic7 81
Vsenssw8 sw8a 77 0
Hsw8sic8 sw8b 77 Vsenssic8 81
Vsenssw9 sw9a 78 0
Hsw9sic9 sw9b 78 Vsenssic9 81
Vsenssw10 sw10a 79 0
Hsw10sic10 sw10b 79 Vsenssic10 81
Vsenssw12 sw12a 80 0
Hsw12sic12 sw12b 80 Vsenssic12 81
*****************************************************
* Connection of rotor coils to circuit terminations *
*****************************************************
Vsensrw1 rb1a 81 0
Hrw1ric1 rb1b 81 Vsensric1 324
**************************************
* Magnetic to mechanical conversion *
**************************************
* frame: stator part: sag9 cct node: 6 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 7.59218 c: 5
B2sag9rag2 82 TORQ_P V=V(6,39)*V(6,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.79659)+cos(V(POS_P,
POS_N)/2)*sin(3.79659))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.79559)+cos(V(POS_P,
POS_N)/2)*sin(3.79559))^5) )/0.004
B1sag9rag2 B1sag9rag2_P B1sag9rag2_N V=V(6,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.79609)+cos(V(POS_P,
POS_N)/2)*sin(3.79609))^5)
C1sag9rag2 B1sag9rag2_P 0 8.37593e-06 IC=0
VF1sag9rag2 0 B1sag9rag2_N 0
FF1sag9rag2 39 6 VF1sag9rag2 -1
* frame: stator part: sag9 cct node: 6 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 4.45059 c: 5
B2sag9rag1 83 82 V=V(6,40)*V(6,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.22579)+cos(V(POS_P,
POS_N)/2)*sin(2.22579))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.22479)+cos(V(POS_P,
POS_N)/2)*sin(2.22479))^5) )/0.004
B1sag9rag1 B1sag9rag1_P B1sag9rag1_N V=V(6,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.22529)+cos(V(POS_P,
POS_N)/2)*sin(2.22529))^5)
C1sag9rag1 B1sag9rag1_P 0 8.37474e-06 IC=0
VF1sag9rag1 0 B1sag9rag1_N 0
FF1sag9rag1 40 6 VF1sag9rag1 -1
* frame: stator part: sag4 cct node: 12 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 4.97419 c: 5
B2sag4rag2 84 83 V=V(12,39)*V(12,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.48759)+cos(V(POS_P,
POS_N)/2)*sin(2.48759))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.48659)+cos(V(POS_P,
POS_N)/2)*sin(2.48659))^5) )/0.004
B1sag4rag2 B1sag4rag2_P B1sag4rag2_N V=V(12,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.48709)+cos(V(POS_P,
POS_N)/2)*sin(2.48709))^5)
C1sag4rag2 B1sag4rag2_P 0 8.37593e-06 IC=0
VF1sag4rag2 0 B1sag4rag2_N 0
FF1sag4rag2 39 12 VF1sag4rag2 -1
* frame: stator part: sag4 cct node: 12 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 1.8326 c: 5
B2sag4rag1 85 84 V=V(12,40)*V(12,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.916798)+cos(V(POS_P,
POS_N)/2)*sin(0.916798))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.915798)+cos(V(POS_P,
POS_N)/2)*sin(0.915798))^5) )/0.004
B1sag4rag1 B1sag4rag1_P B1sag4rag1_N V=V(12,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.916298)+cos(V(POS_P,
POS_N)/2)*sin(0.916298))^5)
C1sag4rag1 B1sag4rag1_P 0 8.37474e-06 IC=0
VF1sag4rag1 0 B1sag4rag1_N 0
FF1sag4rag1 40 12 VF1sag4rag1 -1
* frame: stator part: sag3 cct node: 14 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 4.45059 c: 5
B2sag3rag2 86 85 V=V(14,39)*V(14,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.22579)+cos(V(POS_P,
POS_N)/2)*sin(2.22579))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.22479)+cos(V(POS_P,
POS_N)/2)*sin(2.22479))^5) )/0.004
B1sag3rag2 B1sag3rag2_P B1sag3rag2_N V=V(14,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.22529)+cos(V(POS_P,
POS_N)/2)*sin(2.22529))^5)
C1sag3rag2 B1sag3rag2_P 0 8.37593e-06 IC=0
VF1sag3rag2 0 B1sag3rag2_N 0
FF1sag3rag2 39 14 VF1sag3rag2 -1
* frame: stator part: sag3 cct node: 14 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 1.309 c: 5
B2sag3rag1 87 86 V=V(14,40)*V(14,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.654998)+cos(V(POS_P,
POS_N)/2)*sin(0.654998))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.653998)+cos(V(POS_P,
POS_N)/2)*sin(0.653998))^5) )/0.004
B1sag3rag1 B1sag3rag1_P B1sag3rag1_N V=V(14,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.654498)+cos(V(POS_P,
POS_N)/2)*sin(0.654498))^5)
C1sag3rag1 B1sag3rag1_P 0 8.37474e-06 IC=0
VF1sag3rag1 0 B1sag3rag1_N 0
FF1sag3rag1 40 14 VF1sag3rag1 -1
* frame: stator part: sag8 cct node: 25 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 7.06858 c: 5
B2sag8rag2 88 87 V=V(25,39)*V(25,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.53479)+cos(V(POS_P,
POS_N)/2)*sin(3.53479))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.53379)+cos(V(POS_P,
POS_N)/2)*sin(3.53379))^5) )/0.004
B1sag8rag2 B1sag8rag2_P B1sag8rag2_N V=V(25,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.53429)+cos(V(POS_P,
POS_N)/2)*sin(3.53429))^5)
C1sag8rag2 B1sag8rag2_P 0 8.37593e-06 IC=0
VF1sag8rag2 0 B1sag8rag2_N 0
FF1sag8rag2 39 25 VF1sag8rag2 -1
* frame: stator part: sag8 cct node: 25 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 3.92699 c: 5
B2sag8rag1 89 88 V=V(25,40)*V(25,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.964)+cos(V(POS_P,
POS_N)/2)*sin(1.964))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.963)+cos(V(POS_P,
POS_N)/2)*sin(1.963))^5) )/0.004
B1sag8rag1 B1sag8rag1_P B1sag8rag1_N V=V(25,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.9635)+cos(V(POS_P,
POS_N)/2)*sin(1.9635))^5)
C1sag8rag1 B1sag8rag1_P 0 8.37474e-06 IC=0
VF1sag8rag1 0 B1sag8rag1_N 0
FF1sag8rag1 40 25 VF1sag8rag1 -1
* frame: stator part: sag7 cct node: 27 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 6.54498 c: 5
```

```
B2sag7rag2 90 89 V=V(27,39)*V(27,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.27299)+cos(V(POS_P,
POS_N)/2)*sin(3.27299))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.27199)+cos(V(POS_P,
POS_N)/2)*sin(3.27199))^5) )/0.004
B1sag7rag2 B1sag7rag2_P B1sag7rag2_N V=V(27,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.27249)+cos(V(POS_P,
POS_N)/2)*sin(3.27249))^5)
C1sag7rag2 B1sag7rag2_P 0 8.37593e-06 IC=0
VF1sag7rag2 0 B1sag7rag2_N 0
FF1sag7rag2 39 27 VF1sag7rag2 -1
* frame: stator part: sag7 cct node: 27 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 3.40339 c: 5
B2sag7rag1 91 90 V=V(27,40)*V(27,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.7022)+cos(V(POS_P,
POS_N)/2)*sin(1.7022))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.7012)+cos(V(POS_P,
POS_N)/2)*sin(1.7012))^5) )/0.004
B1sag7rag1 B1sag7rag1_P B1sag7rag1_N V=V(27,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.7017)+cos(V(POS_P,
POS_N)/2)*sin(1.7017))^5)
C1sag7rag1 B1sag7rag1_P 0 8.37474e-06 IC=0
VF1sag7rag1 0 B1sag7rag1_N 0
FF1sag7rag1 40 27 VF1sag7rag1 -1
* frame: stator part: sag2 cct node: 4 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 3.92699 c: 5
B2sag2rag2 92 91 V=V(4,39)*V(4,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.964)+cos(V(POS_P,
POS_N)/2)*sin(1.964))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.963)+cos(V(POS_P,
POS_N)/2)*sin(1.963))^5) )/0.004
B1sag2rag2 B1sag2rag2_P B1sag2rag2_N V=V(4,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.9635)+cos(V(POS_P,
POS_N)/2)*sin(1.9635))^5)
C1sag2rag2 B1sag2rag2_P 0 8.37593e-06 IC=0
VF1sag2rag2 0 B1sag2rag2_N 0
FF1sag2rag2 39 4 VF1sag2rag2 -1
* frame: stator part: sag2 cct node: 4 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 0.785398 c: 5
B2sag2rag1 93 92 V=V(4,40)*V(4,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.393199)+cos(V(POS_P,
POS_N)/2)*sin(0.393199))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.392199)+cos(V(POS_P,
POS_N)/2)*sin(0.392199))^5) )/0.004
B1sag2rag1 B1sag2rag1_P B1sag2rag1_N V=V(4,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.392699)+cos(V(POS_P,
POS_N)/2)*sin(0.392699))^5)
C1sag2rag1 B1sag2rag1_P 0 8.37474e-06 IC=0
VF1sag2rag1 0 B1sag2rag1_N 0
FF1sag2rag1 40 4 VF1sag2rag1 -1
* frame: stator part: sag1 cct node: 10 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 3.40339 c: 5
B2sag1rag2 94 93 V=V(10,39)*V(10,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.7022)+cos(V(POS_P,
POS_N)/2)*sin(1.7022))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.7012)+cos(V(POS_P,
POS_N)/2)*sin(1.7012))^5) )/0.004
B1sag1rag2 B1sag1rag2_P B1sag1rag2_N V=V(10,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.7017)+cos(V(POS_P,
POS_N)/2)*sin(1.7017))^5)
C1sag1rag2 B1sag1rag2_P 0 8.37593e-06 IC=0
VF1sag1rag2 0 B1sag1rag2_N 0
FF1sag1rag2 39 10 VF1sag1rag2 -1
* frame: stator part: sag1 cct node: 10 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 0.261799 c: 5
B2sag1rag1 95 94 V=V(10,40)*V(10,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.1314)+cos(V(POS_P,
POS_N)/2)*sin(0.1314))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.1304)+cos(V(POS_P,
POS_N)/2)*sin(0.1304))^5) )/0.004
B1sag1rag1 B1sag1rag1_P B1sag1rag1_N V=V(10,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(0.1309)+cos(V(POS_P,
POS_N)/2)*sin(0.1309))^5)
C1sag1rag1 B1sag1rag1_P 0 8.37474e-06 IC=0
VF1sag1rag1 0 B1sag1rag1_N 0
```

```
FF1sag1rag1 40 10 VF1sag1rag1 -1
* frame: stator part: sag11 cct node: 8 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 8.63938 c: 5
B2sag11rag2 96 95 V=V(8,39)*V(8,39)*8.37732e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.32019)+cos(V(POS_P,
POS_N)/2)*sin(4.32019))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.31919)+cos(V(POS_P,
POS_N)/2)*sin(4.31919))^5) )/0.004
B1sag11rag2 B1sag11rag2_P B1sag11rag2_N V=V(8,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.31969)+cos(V(POS_P,
POS_N)/2)*sin(4.31969))^5)
C1sag11rag2 B1sag11rag2_P 0 8.37732e-06 IC=0
VF1sag11rag2 0 B1sag11rag2_N 0
FF1sag11rag2 39 8 VF1sag11rag2 -1
* frame: stator part: sag11 cct node: 8 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 5.49779 c: 5
B2sag11rag1 97 96 V=V(8,40)*V(8,40)*8.37613e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.74939)+cos(V(POS_P,
POS_N)/2)*sin(2.74939))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.74839)+cos(V(POS_P,
POS_N)/2)*sin(2.74839))^5) )/0.004
B1sag11rag1 B1sag11rag1_P B1sag11rag1_N V=V(8,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.74889)+cos(V(POS_P,
POS_N)/2)*sin(2.74889))^5)
C1sag11rag1 B1sag11rag1_P 0 8.37613e-06 IC=0
VF1sag11rag1 0 B1sag11rag1_N 0
FF1sag11rag1 40 8 VF1sag11rag1 -1
* frame: stator part: sag6 cct node: 20 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 6.02139 c: 5
B2sag6rag2 98 97 V=V(20,39)*V(20,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.01119)+cos(V(POS_P,
POS_N)/2)*sin(3.01119))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.01019)+cos(V(POS_P,
POS_N)/2)*sin(3.01019))^5) )/0.004
B1sag6rag2 B1sag6rag2_P B1sag6rag2_N V=V(20,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.01069)+cos(V(POS_P,
POS_N)/2)*sin(3.01069))^5)
C1sag6rag2 B1sag6rag2_P 0 8.37593e-06 IC=0
VF1sag6rag2 0 B1sag6rag2_N 0
FF1sag6rag2 39 20 VF1sag6rag2 -1
* frame: stator part: sag6 cct node: 20 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 2.87979 c: 5
B2sag6rag1 99 98 V=V(20,40)*V(20,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.4404)+cos(V(POS_P,
POS_N)/2)*sin(1.4404))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.4394)+cos(V(POS_P,
POS_N)/2)*sin(1.4394))^5) )/0.004
B1sag6rag1 B1sag6rag1_P B1sag6rag1_N V=V(20,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.4399)+cos(V(POS_P,
POS_N)/2)*sin(1.4399))^5)
C1sag6rag1 B1sag6rag1_P 0 8.37474e-06 IC=0
VF1sag6rag1 0 B1sag6rag1_N 0
FF1sag6rag1 40 20 VF1sag6rag1 -1
* frame: stator part: sag10 cct node: 9 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 8.11578 c: 5
B2sag10rag2 100 99 V=V(9,39)*V(9,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.05839)+cos(V(POS_P,
POS_N)/2)*sin(4.05839))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.05739)+cos(V(POS_P,
POS_N)/2)*sin(4.05739))^5) )/0.004
B1sag10rag2 B1sag10rag2_P B1sag10rag2_N V=V(9,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.05789)+cos(V(POS_P,
POS_N)/2)*sin(4.05789))^5)
C1sag10rag2 B1sag10rag2_P 0 8.37593e-06 IC=0
VF1sag10rag2 0 B1sag10rag2_N 0
FF1sag10rag2 39 9 VF1sag10rag2 -1
* frame: stator part: sag10 cct node: 9 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 4.97419 c: 5
B2sag10rag1 101 100 V=V(9,40)*V(9,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.48759)+cos(V(POS_P,
POS_N)/2)*sin(2.48759))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.48659)+cos(V(POS_P,
POS_N)/2)*sin(2.48659))^5) )/0.004
B1sag10rag1 B1sag10rag1_P B1sag10rag1_N V=V(9,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.48709)+cos(V(POS_P,
POS_N)/2)*sin(2.48709))^5)
```

```
C1sag10rag1 B1sag10rag1_P 0 8.37474e-06 IC=0
VF1sag10rag1 0 B1sag10rag1_N 0
FF1sag10rag1 40 9 VF1sag10rag1 -1
* frame: stator part: sag12 cct node: 22 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 9.16298 c: 5
B2sag12rag2 102 101 V=V(22,39)*V(22,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.58199)+cos(V(POS_P,
POS_N)/2)*sin(4.58199))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.58099)+cos(V(POS_P,
POS_N)/2)*sin(4.58099))^5) )/0.004
B1sag12rag2 B1sag12rag2_P B1sag12rag2_N V=V(22,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(4.58149)+cos(V(POS_P,
POS_N)/2)*sin(4.58149))^5)
C1sag12rag2 B1sag12rag2_P 0 8.37593e-06 IC=0
VF1sag12rag2 0 B1sag12rag2_N 0
FF1sag12rag2 39 22 VF1sag12rag2 -1
* frame: stator part: sag12 cct node: 22 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 6.02139 c: 5
B2sag12rag1 103 102 V=V(22,40)*V(22,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.01119)+cos(V(POS_P,
POS_N)/2)*sin(3.01119))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.01019)+cos(V(POS_P,
POS_N)/2)*sin(3.01019))^5) )/0.004
B1sag12rag1 B1sag12rag1_P B1sag12rag1_N V=V(22,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(3.01069)+cos(V(POS_P,
POS_N)/2)*sin(3.01069))^5)
C1sag12rag1 B1sag12rag1_P 0 8.37474e-06 IC=0
VF1sag12rag1 0 B1sag12rag1_N 0
FF1sag12rag1 40 22 VF1sag12rag1 -1
* frame: stator part: sag5 cct node: 23 frame: rotor part: rag2 cct node: 39
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 5.49779 c: 5
B2sag5rag2 104 103 V=V(23,39)*V(23,39)*8.37593e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.74939)+cos(V(POS_P,
POS_N)/2)*sin(2.74939))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.74839)+cos(V(POS_P,
POS_N)/2)*sin(2.74839))^5) )/0.004
B1sag5rag2 B1sag5rag2_P B1sag5rag2_N V=V(23,39)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(2.74889)+cos(V(POS_P,
POS_N)/2)*sin(2.74889))^5)
C1sag5rag2 B1sag5rag2_P 0 8.37593e-06 IC=0
VF1sag5rag2 0 B1sag5rag2_N 0
FF1sag5rag2 39 23 VF1sag5rag2 -1
* frame: stator part: sag5 cct node: 23 frame: rotor part: rag1 cct node: 40
* top: 0.620398 bottom: 1.2913 a: 29.5121 b: 2.35619 c: 5
B2sag5rag1 105 104 V=V(23,40)*V(23,40)*8.37474e-06
+ *( exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.1786)+cos(V(POS_P,
POS_N)/2)*sin(1.1786))^5)
+ -exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.1776)+cos(V(POS_P,
POS_N)/2)*sin(1.1776))^5) )/0.004
B1sag5rag1 B1sag5rag1_P B1sag5rag1_N V=V(23,40)
+ *exp(-1*29.5121*abs(sin(V(POS_P,POS_N)/2)*cos(1.1781)+cos(V(POS_P,
POS_N)/2)*sin(1.1781))^5)
C1sag5rag1 B1sag5rag1_P 0 8.37474e-06 IC=0
VF1sag5rag1 0 B1sag5rag1_N 0
FF1sag5rag1 40 23 VF1sag5rag1 -1
VEND 105 TORQ_N 0
RPOS POS_P POS_N 1G
.ends
```

Appendices

E

Instructions to retrieve magnetic circuit simulation data

```
* rip1 0.00154833 37 38 34.0133 rip2 0.00154833 38 37 34.0133 rl1 0.000523809 39 40
153.361 ric1 0.00295584 39 37 2.42806 ric2 0.00295584 40 38 2.42806 rl2 0.000523809 40
39 153.361 rag1 0.0043613 41 39 0.0458184 rag2 0.00436163 42 40 0.0456052 sic5 0.0019629
1 2 5.62439 sl5 8.90547e-05 2 3 30.7718 sag3 0.00204481 4 5 0.104395 sic6 0.0019629 6 7
5.62439 sag4 0.00204481 3 8 0.104395 sic12 0.0019629 9 10 5.62439 sic7 0.0019629 11 12
5.62439 sl3 8.90547e-05 4 13 30.7718 sic10 0.0019629 14 15 5.62439 sic11 0.0019629 16 17
5.62439 sag5 0.00204481 2 18 0.104395 sib1 0.0014012 19 9 14.7614 sl8 8.90547e-05 20 12
30.7718 sic8 0.0019629 21 20 5.62439 sag6 0.00204481 7 22 0.104395 sib2 0.0014012 23 19
14.7614 sic9 0.0019629 24 25 5.62439 sl1 8.90547e-05 26 10 30.7718 sag12 0.00204481 10
27 0.104395 sag7 0.00204456 12 28 0.104645 sag10 0.00204456 15 29 0.104645 sib3
0.0014012 30 23 14.6724 sag11 0.00204481 17 31 0.104395 sl6 8.39771e-05 7 2 31.0886 sag8
0.00204481 20 32 0.104395 sib4 0.0014012 33 30 14.6724 sag9 0.00204481 25 34 0.104395
sib5 0.0014012 1 33 14.6724 sl4 8.90547e-05 3 4 30.7718 sic1 0.0019629 19 26 5.62439 sl9
8.90547e-05 25 20 30.7718 sib6 0.0014012 6 1 14.7614 sic2 0.0019629 23 13 5.62439 sl10
8.39771e-05 15 25 31.0886 sib12 0.0014012 9 16 14.6724 sib7 0.0014012 11 6 14.6724 sl2
8.90547e-05 13 26 30.7718 sib10 0.0014012 14 24 14.6724 sib11 0.0014012 16 14 14.6724
sic3 0.0019629 30 4 5.62439 sl11 8.39771e-05 17 15 31.0886 sl7 8.90547e-05 12 7 30.7718
sag1 0.00204481 26 35 0.104395 sib8 0.0014012 21 11 14.7614 sic4 0.0019629 33 3 5.62439
sl12 8.90547e-05 10 17 30.7718 sag2 0.00204481 13 36 0.104395 sib9 0.0014012 24 21
14.6724
* 1 2 3 6 7 9 10 11 12 4 13 14 15 16 17 19 20 21 23 24 25 26 30 33 37 38 39 40
*
show /file gen.dat x1.x1.1 x1.x1.2 x1.x1.3 x1.x1.6 x1.x1.7 x1.x1.9 x1.x1.10 x1.x1.11
x1.x1.12 x1.x1.4 x1.x1.13 x1.x1.14 x1.x1.15 x1.x1.16 x1.x1.17 x1.x1.19 x1.x1.20 x1.x1.21
x1.x1.23 x1.x1.24 x1.x1.25 x1.x1.26 x1.x1.30 x1.x1.33 x1.x1.37 x1.x1.38 x1.x1.39
x1.x1.40
```

Appendices

F

Circuit simualtion data

```
Time                    :x1.x1.2                :x1.x1.1
0                       0                       0
5e-05                   0                       0
0.0001                  0                       0
0.0002                  0                       0
0.0004                  0                       0
0.0008                  0                       0
0.0016                  0                       0
0.0026                  0                       0
0.0036                  0                       0
0.0046                  0                       0
0.005                   0                       0
0.0050056980132         9.26212690364e-05       9.26643579321e-05
0.00501709403961        0.000500445608639       0.000500678904266
0.00502643614368        0.00108952085605        0.00109003026845
0.00503607116367        0.00195104950067        0.00195196426523
0.00504979337793        0.00363185389973        0.00363356341251
0.00506638270339        0.00637858553845        0.00638160246974
0.00509006760344        0.0116534801598         0.0116590326019
0.0051207429074         0.0208498942108         0.0208599235928
0.00516311374512        0.0379343874731         0.037952875284
0.00521943569559        0.0684899927331         0.0685239514162
0.00529615118653        0.12445926723           0.124522415197
0.00539920484359        0.225559293492          0.225677244837
0.00550207219785        0.355942734346          0.356134280968
0.00559994243392        0.50715448735           0.507434622008
0.00569952381287        0.688028254461          0.68841820246
0.00581170903302        0.92421429739           0.924753143439
0.00593541836754        1.22420280005           1.22493860506
0.0060778983535         1.62067806763           1.62168596526
0.00623931341813        2.13516998649           2.13654848456
0.00642532872917        2.81324433456           2.81513778109
0.00663815111522        3.69957529232           3.70218177552
0.00688332797445        4.86489417391           4.86849870102
0.00716504042405        6.3915610552            6.39656481185
0.00748971391162        8.39553161452           8.40251138057
0.00786365837826        11.0213580853           11.0311386442
0.00829502425169        14.4628923413           14.476665724
0.00879265997372        18.9671472115           18.9866363813
0.00936712519421        24.855915177            24.8836231152
0.01                    32.1685140194           32.2074774557
0.0100097879765         32.2873558218           32.3265128555
0.0100293639296         32.5227060143           32.5622542566
0.0100685158358         32.9822780278           33.022627849
0.0101115261827         33.4697132534           33.5109737217
0.0101975468765         34.3894524746           34.4326282344
0.0103695882642         36.0100741084           36.0574465336
0.0105576690849         37.4523456499           37.5048439207
0.0107608801946         38.6314880612           38.6901248275
0.0109342913109         39.3330522305           39.3973950799
0.0111238590444         39.7879641454           39.8590066268
0.0113184586678         39.9265267726           40.0049149817
0.0115567377047         39.6617432325           39.7497170879
0.01185342376           38.7005507224           38.8012691955
0.0122122464159         36.6746450548           36.791796593
0.0126487127918         33.0798626244           33.2181810441
0.0131696740386         27.4361054542           27.6008230618
0.0137948890164         19.2274632144           19.4245574894
0.0145428602966         8.27340581457           8.50873790212
0.015                   1.35064467946           1.60853295855
0.0150903146325         -0.0251241714233        0.237113760877
0.0152709438974         -2.74751426273          -2.47669551641
0.0156322024272         -7.99362278624          -7.70627557552
0.0163547194868         -17.181459227           -16.8640982528
0.0166                  -19.832476945           -19.5059026044
0.016603615245          -19.8697143554          -19.5430097407
0.0166108457351         -19.9440764773          -19.6171158788
0.0166253067152         -20.0920014876          -19.7645508386
0.0166542286754         -20.3846556923          -20.0563155063
0.0167120725958         -20.9565837127          -20.6268287381
0.0168277604367         -22.0454569214          -21.7143281731
0.0170591361184         -23.9893755905          -23.6612876498
0.0175218874819         -26.8401721535          -26.5408675556
0.0181250949456         -28.2326718436          -28.0143368128
0.0186980255681         -26.9142030109          -26.8150291293
```

0.0193492960554	-22.1741735502	-22.2541872868
0.0197452691275	-17.6343387799	-17.8431544367
0.02	-14.0687247392	-14.3671669279
0.0200571464359	-13.1875019115	-13.5068522278
0.0201714393076	-11.3660335764	-11.7276660702
0.020400025051	-7.43301969825	-7.88117544827
0.0207391248861	-0.827349948706	-1.40756046961
0.0210563843774	6.28514504254	5.57905705599
0.021414257931	15.6334549763	14.7854486307
0.0216	21.1531267165	20.2323256319
0.0216351554386	22.2633922197	21.3289228489
0.0217054663157	24.5371225514	23.5754580715
0.02184608807	29.3540607402	28.3388040651
0.0221273315785	40.3305227988	39.2125714798
0.022586267285	63.9361626622	62.6711846646
0.0230123251614	96.7271880185	95.3629137371
0.0233	129.606558269	128.207505679
0.0233042239127	130.177579147	128.778240213
0.0233126717382	131.328915826	129.929021083
0.0233295673892	133.673911057	132.272983341
0.0233633586911	138.543227942	137.140573108
0.023430941295	149.069615308	147.665016666
0.0235661065027	173.95039213	172.549408845
0.0237417941865	216.904615089	215.529374976
0.0239306734224	284.534161342	283.228804454
0.0241007306623	378.148085272	376.968825702
0.0242329057745	486.715403342	485.702469874
0.0243448342406	616.119965595	615.320506512
0.0244433737	768.063520056	767.529598954
0.0245337353575	942.226317401	942.015507328
0.024626795622	1148.6215273	1148.82606226
0.0247123595199	1340.69305435	1341.33380579
0.0247976854496	1503.86361955	1504.95110298
0.0248829525194	1614.87289562	1616.37525491
0.0249484939455	1660.74239763	1662.52505573
0.025	1675.31148495	1677.28794639
0.0250075379979	1676.04367168	1678.04663806
0.0250226139938	1676.55275434	1678.6072512
0.0250527659855	1674.04501822	1676.19692054
0.025113069969	1656.91666617	1659.24254361
0.0251674651245	1629.79040785	1632.25235694
0.025216969391 8	1596.46433154	1599.03561591
0.0252658057473	1555.46669078	1558.13449256
0.0253152665879	1505.00871135	1507.76483896
0.0253646238537	1444.66116139	1447.4979275
0.0254140554371	1373.34851869	1376.26029901
0.0254637383631	1290.58220914	1293.56502434
0.0255137883857	1197.1142074	1200.16510421
0.0255642188409	1095.39037627	1098.50668996
0.0256137349697	991.970539967	995.147786573
0.0256618699182	892.134206303	895.367076348
0.0257093279356	797.945186121	801.228912287
0.0257563617144	711.421293854	714.751164425
0.0258032493919	633.594644643	636.966157924
0.0258502584534	564.787505376	568.196465757
0.0258976817004	504.808806703	508.251403302
0.0259457739317	453.261501123	456.734291812
0.0259947724776	409.659272133	413.159155599
0.0260448976825	373.533219078	377.057384716
0.0260963646558	344.501603848	348.047469643
0.0261493899983	322.33473364	325.899868225
0.0262042032212	307.022203587	310.604229257
0.0262610570837	298.865862208	302.462326286
0.0263202382592	298.625236228	302.233428499
0.0263820757339	307.764943317	311.381617533
0.0264469450611	328.89850473	332.519424986
0.0265152638047	366.622048423	370.241204728
0.0265874710613	429.14630592	432.754482173
0.0266	442.926288181	446.531396983
0.0266072888893	451.430000955	455.033128164
0.026621866668	469.504649842	473.10339309
0.0266510222253	510.447643917	514.035708904
0.0267030329774	602.65644977	606.217814554
0.0267567505058	731.329315823	734.849787219
0.0268115433771	909.617594574	913.077609303

0.0268643883111	1139.40663131	1142.78507808
0.0269144413191	1416.69700295	1419.97349194
0.0269621028106	1727.79930188	1730.95737943
0.0270079374704	2043.34552327	2046.37778772
0.02705300916	2326.29697233	2329.20702037
0.0270983899886	2541.72180643	2544.52250196
0.0271383030848	2657.29562732	2660.01481819
0.0271695083662	2701.38548445	2704.04809649
0.0271989940469	2711.6131301	2714.22438624
0.0272274815029	2698.02343739	2700.58327166
0.0272576883755	2663.23254148	2665.73159745
0.0272853870147	2615.51633327	2617.94998419
0.0273127568145	2554.31727434	2556.67333508
0.0273398248648	2479.46466399	2481.72762425
0.0273669263169	2388.7301796	2390.88020359
0.0273939816827	2280.57130731	2282.58629952
0.0274211259495	2153.10095218	2154.95708234
0.0274483761188	2006.20406921	2007.8786463
0.0274757550314	1841.9836153	1843.4581169
0.0275032465689	1665.36962034	1666.632979
0.0275297410549	1490.60632173	1491.66519993
0.0275555096265	1322.97891372	1323.84600043
0.027580733406	1166.74241487	1167.4346839
0.0276055557501	1024.62961604	1025.16624903
0.0276300797436	898.010833327	898.411538156
0.02765444471	786.857277367	787.140745005
0.0276787752444	690.46991193	690.653168335
0.0277031853706	607.785869394	607.88399268
0.0277277769512	537.615960952	537.642052829
0.027752647167	478.773994707	478.739271878
0.0277778916803	430.172890269	430.086848744
0.0278036106337	390.88180421	390.752408013
0.0278299144233	360.170272273	360.004145208
0.0278569304579	337.549810126	337.352416727
0.0278848102347	322.826670712	322.602480212
0.0279137370772	316.17950919	315.932139235
0.027943934234	318.283305446	318.015644601
0.0279756729678	330.510979233	330.225288282
0.0280092796542	355.264918666	354.962911269
0.0280451399736	396.517485891	396.200389419
0.0280836971413	460.665455618	460.3340531
0.0281254375832	557.745153782	557.399809142
0.0281660290066	685.041061246	684.683174586
0.0282064167469	848.319273276	847.949531888
0.0282460579406	1044.4933314	1044.11232049
0.0282853316834	1266.37999458	1265.98804321
0.0283	1352.87630424	1352.48031191
0.0283039367576	1376.17840467	1375.78133156
0.0283118102728	1422.64070343	1422.24147492
0.0283275573033	1514.2712928	1513.86777878
0.0283590513641	1685.59608177	1685.1841151
0.0283964334041	1852.19204528	1851.77030157
0.0284372451087	1972.93718077	1972.50514255
0.0284683967966	2021.06295237	2020.62333462
0.0284973060951	2036.07734275	2035.63089558
0.0285251689337	2028.67782604	2028.22496329
0.0285547570037	2002.39828592	2001.93876148
0.0285817244467	1965.2206798	1964.75518991
0.0286082610138	1918.07056257	1917.59927082
0.0286344167028	1861.80133737	1861.3243609
0.0286605372019	1795.41670968	1794.93405374
0.0286865448361	1718.09706787	1717.60871509
0.0287125719161	1628.22613594	1627.73199756
0.0287386320412	1524.84138075	1524.34132052
0.0287647396413	1408.02904449	1407.52287861
0.0287908621938	1279.75773343	1279.24524405
0.028816954932	1143.9815047	1143.46244717
0.0288416861069	1012.92851246	1012.40296396
0.0288656157249	888.642264185	888.110156497
0.0288887863149	774.597561095	774.058805438
0.0289113356621	672.395681306	671.850133679
0.0289333415839	582.714008044	582.161476264
0.0289549448249	505.091865225	504.532077228
0.0289762612626	438.675694892	438.108291288
0.0289974033518	382.384083657	381.808601315

0.0290184712811	335.109020936	334.524876443
0.0290395620697	295.796166742	295.202631722
0.0290607702232	263.508780657	262.904949644
0.0290821934773	237.454130251	236.838875461
0.0291039372831	217.002013991	216.373922061
0.0291261208089	201.698708554	201.05599006
0.0291488831898	191.287124697	190.627483076
0.0291723908614	185.740349324	185.060787362
0.0291968456627	185.318366103	184.614894933
0.0292224937225	190.665304338	189.932497379
0.0292496345592	202.967402549	202.197699707
0.0292786293482	224.205644069	223.388252102
0.0293099060293	257.535878639	256.655029863
0.0293439571352	307.786172961	306.818430146
0.0293789930269	376.613769558	375.532704236
0.0294140604865	464.086640685	462.863573421
0.0294495465974	570.501883316	569.103796978
0.029485540776	691.847365679	690.241964576
0.0295225624878	820.550293877	818.709902332
0.0295611538596	944.262866589	942.171449753
0.0296019127701	1048.40758957	1046.06612719
0.029638772907	1112.48007029	1109.93961809
0.0296704993658	1145.11533656	1142.42814539
0.0297004433382	1159.58174065	1156.77621516
0.0297197473229	1161.9639512	1159.09145824
0.0297391315472	1159.82247572	1156.88920829
0.0297594993381	1153.40471379	1150.41373178
0.0297894741082	1137.34759843	1134.28136859
0.0298183899636	1115.17691285	1112.04697341
0.0298457680835	1088.21873394	1085.03471113
0.0298729210861	1055.24411731	1052.01095545
0.029900314918	1014.785745	1011.50620353
0.0299277166161	966.064940788	962.741006724
0.0299551232766	908.314817536	904.947482797
0.0299825374865	841.487091324	838.076677198
0.03	794.637588953	791.199789525
0.0300027388493	787.032655375	783.590567292
0.030008216548	771.627021482	768.176363686
0.0300191719454	740.14154325	736.673793534
0.0300410827402	675.25106644	671.749476368
0.0300648180017	603.946739976	600.409435261
0.0300879666516	535.78154397	532.210941828
0.030112692475	466.974984744	463.371131415
0.0301354240306	409.139840145	405.507978237
0.030157718453	358.531460078	354.874763971
0.0301795029163	315.406363197	311.728004399
0.0302010915068	278.908117156	275.210792226
0.0302224904246	248.665640071	244.951859953
0.0302438869716	223.989524293	220.261469202
0.0302653980722	204.37675578	200.636374254
0.0302871712305	189.409990128	185.659031689
0.0303093480088	178.832742966	175.072825145
0.0303320961658	172.56099372	168.79366475
0.0303556041284	170.739171046	166.965994494
0.0303800942209	173.812679759	170.035343698
0.0304058272583	182.650843633	178.87131611
0.0304331108018	198.748011241	194.968771526
0.0304623027686	224.549070029	220.773465316
0.0304938115859	263.957974466	260.190775425
0.0305277516948	322.382857529	318.630914713
0.0305609132232	398.396994447	394.667338218
0.030594424672	496.739632817	493.040987924
0.0306278390744	616.99303013	613.334289795
0.03066156986	757.747164352	754.137055271
0.0306959784528	912.541440825	908.986945867
0.0307315310509	1069.24416615	1065.74880681
0.0307687819437	1211.26716965	1207.82984014
0.0308082460654	1321.80808359	1318.42351065
0.0308393341924	1377.02272079	1373.67314853
0.0308680569283	1405.47573908	1402.15489688
0.030895651253	1415.95549976	1412.6609761
0.0309249680124	1412.77500656	1409.50947058
0.0309516667783	1399.62711135	1396.39110938
0.03097791219	1378.61426903	1375.41231908
0.0310037550539	1350.52564472	1347.36414582

Appendices

G

Full Brook DL100 Induction Motor GDL Description File

```
# Brook Crompton D100L induction motor

unitdistance 0.001

airgap 1

bh {
        newcor exp 3.8 2.17 396.2 # linear 1989
        newcorril exp 3.8 2.17 396.2
        sl linear 1
}

stator
{
        length 96

        coils { sw1 81 sw1a sw1b sw2 81 sw2a sw2b sw3 81 sw3a sw3b sw4 81 sw4a sw4b
                sw5 81 sw5a sw5b sw6 81 sw6a sw6b sw7 81 sw7a sw7b sw8 81 sw8a sw8b
                sw9 81 sw9a sw9b sw10 81 sw10a sw10b sw11 81 sw11a sw11b sw12 81 sw12a
sw12b
                sw13 81 sw13a sw13b sw14 81 sw14a sw14b sw15 81 sw15a sw15b sw16 81
sw16a sw16b
                sw17 81 sw17a sw17b sw18 81 sw18a sw18b }

        nodes {
                sn0 0 d 0
                sn[1 36 20] [46.04 0] d [-1.56 10]
                sn[2 36 20] [46.294 0] d [-1.55 10]
                sn[3 36 20] [46.802 0] d [-2.15 10]
                sn[4 36 20] [47.074 0] d [-2.47 10]
                sn[5 36 20] [58.92 0] d [-3.09 10]
                sn[6 36 20] [46.04 0] d [1.56 10]
                sn[7 36 20] [46.294 0] d [1.55 10]
                sn[8 36 20] [46.802 0] d [2.15 10]
                sn[9 36 20] [47.074 0] d [2.47 10]
                sn[10 36 20] [58.92 0] d [3.09 10]
                sn[11 36 20] [62 0] d [5 10]
                sn[12 36 20] [76.45 0] d [5 10]
        }

        curves {
                sc[1 35 20] circ sn[6 20] sn[21 20] [46.04 0]
                sc701 circ sn706 sn1 46.04
                sc[2 36 20] line sn[1 20] sn[2 20]
                sc[3 36 20] circ sn[3 20] sn[2 20] [0.508 0]
                sc[4 36 20] circ sn[3 20] sn[4 20] [0.84 0]
                sc[5 36 20] line sn[4 20] sn[5 20]
                sc[6 36 20] circ sn[5 20] sn[10 20] [3.18 0]
                sc[7 36 20] line sn[6 20] sn[7 20]
                sc[8 36 20] circ sn[7 20] sn[8 20] [0.84 0]
                sc[9 36 20] circ sn[9 20] sn[8 20] [0.508 0]
                sc[10 36 20] line sn[9 20] sn[10 20]
                sc[11 36 20] line sn[10 20] sn[11 20]
                sc[12 35 20] line sn[11 20] sn[25 20]
                sc712 line sn711 sn5
                sc[13 36 20] line sn[11 20] sn[12 20]
                sc[14 35 20] circ sn[12 20] sn[32 20] [76.45 0]
                sc714 circ sn712 sn12 76.45
                sc[18 36 20] line sn[4 20] sn[9 20]
                sc[19 36 20] line sn[1 20] sn[6 20]
        }

        parts {
                sic1 {
                        bh newcor
                        coils { sw1 sw2 sw3 }
                        curves { sc1 sc7 sc8 sc9 sc10 sc11 sc12 sc22 sc23 sc24 sc25 }
                        entry { sc11 sc12 }
                        exit { sc1 sc7 sc8 sc22 sc23 }
                }
                sl1 {
                        bh sl
                        curves { sc2 sc3 sc4 sc18 sc19 sc7 sc8 sc9 }
                        entry { sc7 sc8 }
                        exit { sc2 sc3 }
                }
                sib1 {
                        bh newcor
                        curves { sc6 sc11 sc13 sc712 sc713 sc714 }
                        entry { sc11 sc13 }
                        exit { sc712 sc713 }
                }

                sic2 {
                        bh newcor
                        coils { sw1 sw2 sw3 sw4 }
                        curves { sc21 sc27 sc28 sc29 sc30 sc31 sc32 sc42 sc43 sc44 sc45 }
                        entry { sc31 sc32 }
                        exit { sc21 sc27 sc28 sc42 sc43 }
                }
                sl2 {
                        bh sl
                        curves { sc22 sc23 sc24 sc38 sc39 sc27 sc28 sc29 }
                        entry { sc27 sc28 }
                        exit { sc22 sc23 }
                }
                sib2 {
                        bh newcor
                        curves { sc26 sc31 sc33 sc12 sc13 sc14 }
                        entry { sc31 sc33 }
                        exit { sc12 sc13 }
                }

                sic3 {
                        bh newcor
                        coils { sw1 sw2 sw3 sw4 sw5 }
                        curves { sc41 sc47 sc48 sc49 sc50 sc51 sc52 sc62 sc63 sc64 sc65 }
                        entry { sc51 sc52 }
                        exit { sc41 sc47 sc48 sc62 sc63 }
                }
                sl3 {
                        bh sl
                        curves { sc42 sc43 sc44 sc58 sc59 sc47 sc48 sc49 }
                        entry { sc47 sc48 }
                        exit { sc42 sc43 }
                }
                sib3 {
                        bh newcor
                        curves { sc46 sc51 sc53 sc32 sc33 sc34 }
                        entry { sc51 sc53 }
                        exit { sc32 sc33 }
                }

                sic4 {
                        bh newcor
                        coils { sw1 sw2 sw3 sw4 sw5 sw6 }
                        curves { sc61 sc67 sc68 sc69 sc70 sc71 sc72 sc82 sc83 sc84 sc85 }
                        entry { sc71 sc72 }
                        exit { sc61 sc67 sc68 sc82 sc83 }
                }
                sl4 {
                        bh sl
                        curves { sc62 sc63 sc64 sc78 sc79 sc67 sc68 sc69 }
                        entry { sc67 sc68 }
                        exit { sc62 sc63 }
                }
                sib4 {
                        bh newcor
                        curves { sc66 sc71 sc73 sc52 sc53 sc54 }
                        entry { sc71 sc73 }
                        exit { sc52 sc53 }
                }

                sic5 {
                        bh newcor
                        coils { sw1 sw2 sw4 sw5 sw6 }
                        curves { sc81 sc87 sc88 sc89 sc90 sc91 sc92 sc102 sc103 sc104
sc105 }
                        entry { sc91 sc92 }
                        exit { sc81 sc87 sc88 sc102 sc103 }
                }
                sl5 {
                        bh sl
                        curves { sc82 sc83 sc84 sc98 sc99 sc87 sc88 sc89 }
                        entry { sc87 sc88 }
                        exit { sc82 sc83 }
                }
                sib5 {
                        bh newcor
                        curves { sc86 sc91 sc93 sc72 sc73 sc74 }
                        entry { sc91 sc93 }
                        exit { sc72 sc73 }
                }

                sic6 {
                        bh newcor
                        coils { sw1 sw4 sw5 sw6 }
                        curves { sc101 sc107 sc108 sc109 sc110 sc111 sc112 sc122 sc123
sc124 sc125 }
                        entry { sc111 sc112 }
                        exit { sc101 sc107 sc108 sc122 sc123 }
                }
                sl6 {
                        bh sl
                        curves { sc102 sc103 sc104 sc118 sc119 sc107 sc108 sc109 }
                        entry { sc107 sc108 }
                        exit { sc102 sc103 }
                }
                sib6 {
                        bh newcor
                        curves { sc106 sc111 sc113 sc92 sc93 sc94 }
                        entry { sc111 sc113 }
                        exit { sc92 sc93 }
                }

                sic7 {
                        bh newcor
                        coils { sw4 sw5 sw6 }
                        curves { sc121 sc127 sc128 sc129 sc130 sc131 sc132 sc142 sc143
sc144 sc145 }
                        entry { sc131 sc132 }
                        exit { sc121 sc127 sc128 sc142 sc143 }
                }
                sl7 {
                        bh sl
                        curves { sc122 sc123 sc124 sc138 sc139 sc127 sc128 sc129 }
                        entry { sc127 sc128 }
                        exit { sc122 sc123 }
                }
                sib7 {
                        bh newcor
                        curves { sc126 sc131 sc133 sc112 sc113 sc114 }
                        entry { sc131 sc133 }
                        exit { sc112 sc113 }
                }

                sic8 {
                        bh newcor
                        coils { sw4 sw5 sw6 sw7 }
                        curves { sc141 sc147 sc148 sc149 sc150 sc151 sc152 sc162 sc163
sc164 sc165 }
                        entry { sc151 sc152 }
                        exit { sc141 sc147 sc148 sc162 sc163 }
                }
                sl8 {
                        bh sl
                        curves { sc142 sc143 sc144 sc158 sc159 sc147 sc148 sc149 }
                        entry { sc147 sc148 }
                        exit { sc142 sc143 }
                }
                sib8 {
                        bh newcor
                        curves { sc146 sc151 sc153 sc132 sc133 sc134 }
                        entry { sc151 sc153 }
                        exit { sc132 sc133 }
                }

                sic9 {
                        bh newcor
                        coils { sw4 sw5 sw6 sw7 sw8 }
                        curves { sc161 sc167 sc168 sc169 sc170 sc171 sc172 sc182 sc183
sc184 sc185 }
                        entry { sc171 sc172 }
                        exit { sc161 sc167 sc168 sc182 sc183 }
                }
                sl9 {
                        bh sl
                        curves { sc162 sc163 sc164 sc178 sc179 sc167 sc168 sc169 }
                        entry { sc167 sc168 }
                        exit { sc162 sc163 }
                }
                sib9 {
                        bh newcor
                        curves { sc166 sc171 sc173 sc152 sc153 sc154 }
                        entry { sc171 sc173 }
                        exit { sc152 sc153 }
                }

                sic10 {
                        bh newcor
                        coils { sw4 sw5 sw6 sw7 sw8 sw9 }
                        curves { sc181 sc187 sc188 sc189 sc190 sc191 sc192 sc202 sc203
sc204 sc205 }
                        entry { sc191 sc192 }
                        exit { sc181 sc187 sc188 sc202 sc203 }
                }

                sl10 {
                        bh sl
                        curves { sc182 sc183 sc184 sc198 sc199 sc187 sc188 sc189 }
                        entry { sc187 sc188 }
                        exit { sc182 sc183 }
                }
                sib10 {
                        bh newcor
                        curves { sc186 sc191 sc193 sc172 sc173 sc174 }
                        entry { sc191 sc193 }
                        exit { sc172 sc173 }
                }

                sic11 {
                        bh newcor
                        coils { sw4 sw5 sw7 sw8 sw9 }
                        curves { sc201 sc207 sc208 sc209 sc210 sc211 sc212 sc222 sc223
sc224 sc225 }
                        entry { sc211 sc212 }
                        exit { sc201 sc207 sc208 sc222 sc223 }
                }
                sl11 {
                        bh sl
                        curves { sc202 sc203 sc204 sc218 sc219 sc207 sc208 sc209 }
                        entry { sc207 sc208 }
                        exit { sc202 sc203 }
                }
                sib11 {
                        bh newcor
                        curves { sc206 sc211 sc213 sc192 sc193 sc194 }
                        entry { sc211 sc213 }
                        exit { sc192 sc193 }
                }

                sic12 {
                        bh newcor
                        coils { sw4 sw7 sw8 sw9 }
                        curves { sc221 sc227 sc228 sc229 sc230 sc231 sc232 sc242 sc243
sc244 sc245 }
                        entry { sc231 sc232 }
                        exit { sc221 sc227 sc228 sc242 sc243 }
                }
                sl12 {
```

```
                bh sl
                curves { sc222 sc223 sc224 sc238 sc239 sc227 sc228 sc229 }
                entry { sc227 sc228 }
                exit { sc222 sc223 }
        }
        sib12 {
                bh newcor
                curves { sc226 sc231 sc233 sc212 sc213 sc214 }
                entry { sc231 sc233 }
                exit { sc212 sc213 }
        }

        sic13 {
                bh newcor
                coils { sw7 sw8 sw9 }
                curves { sc241 sc247 sc248 sc249 sc250 sc251 sc252 sc262 sc263
sc264 sc265 }
                entry { sc251 sc252 }
                exit { sc241 sc247 sc248 sc262 sc263 }
        }
        sl13 {
                bh sl
                curves { sc242 sc243 sc244 sc258 sc259 sc247 sc248 sc249 }
                entry { sc247 sc248 }
                exit { sc242 sc243 }
        }
        sib13 {
                bh newcor
                curves { sc246 sc251 sc253 sc232 sc233 sc234 }
                entry { sc251 sc253 }
                exit { sc232 sc233 }
        }

        sic14 {
                bh newcor
                coils { sw7 sw8 sw9 sw10 }
                curves { sc261 sc267 sc268 sc269 sc270 sc271 sc272 sc282 sc283
sc284 sc285 }
                entry { sc271 sc272 }
                exit { sc261 sc267 sc268 sc282 sc283 }
        }
        sl14 {
                bh sl
                curves { sc262 sc263 sc264 sc278 sc279 sc267 sc268 sc269 }
                entry { sc267 sc268 }
                exit { sc262 sc263 }
        }
        sib14 {
                bh newcor
                curves { sc266 sc271 sc273 sc252 sc253 sc254 }
                entry { sc271 sc273 }
                exit { sc252 sc253 }
        }

        sic15 {
                bh newcor
                coils { sw7 sw8 sw9 sw10 sw11 }
                curves { sc281 sc287 sc288 sc289 sc290 sc291 sc292 sc302 sc303
sc304 sc305 }
                entry { sc291 sc292 }
                exit { sc281 sc287 sc288 sc302 sc303 }
        }
        sl15 {
                bh sl
                curves { sc282 sc283 sc284 sc298 sc299 sc287 sc288 sc289 }
                entry { sc287 sc288 }
                exit { sc282 sc283 }
        }
        sib15 {
                bh newcor
                curves { sc286 sc291 sc293 sc272 sc273 sc274 }
                entry { sc291 sc293 }
                exit { sc272 sc273 }
        }
```

```
                sic16 {
                        bh newcor
                        coils { sw7 sw8 sw9 sw10 sw11 sw12 }
                        curves { sc301 sc307 sc308 sc309 sc310 sc311 sc312 sc322 sc323
sc324 sc325 }
                        entry { sc311 sc312 }
                        exit { sc301 sc307 sc308 sc322 sc323 }
                }
                sl16 {
                        bh sl
                        curves { sc302 sc303 sc304 sc318 sc319 sc307 sc308 sc309 }
                        entry { sc307 sc308 }
                        exit { sc302 sc303 }
                }
                sib16 {
                        bh newcor
                        curves { sc306 sc311 sc313 sc292 sc293 sc294 }
                        entry { sc311 sc313 }
                        exit { sc292 sc293 }
                }

                sic17 {
                        bh newcor
                        coils { sw7 sw8 sw10 sw11 sw12 }
                        curves { sc321 sc327 sc328 sc329 sc330 sc331 sc332 sc342 sc343
sc344 sc345 }
                        entry { sc331 sc332 }
                        exit { sc321 sc327 sc328 sc342 sc343 }
                }
                sl17 {
                        bh sl
                        curves { sc322 sc323 sc324 sc338 sc339 sc327 sc328 sc329 }
                        entry { sc327 sc328 }
                        exit { sc322 sc323 }
                }
                sib17 {
                        bh newcor
                        curves { sc326 sc331 sc333 sc312 sc313 sc314 }
                        entry { sc331 sc333 }
                        exit { sc312 sc313 }
                }

                sic18 {
                        bh newcor
                        coils { sw7 sw10 sw11 sw12 }
                        curves { sc341 sc347 sc348 sc349 sc350 sc351 sc352 sc362 sc363
sc364 sc365 }
                        entry { sc351 sc352 }
                        exit { sc341 sc347 sc348 sc362 sc363 }
                }
                sl18 {
                        bh sl
                        curves { sc342 sc343 sc344 sc358 sc359 sc347 sc348 sc349 }
                        entry { sc347 sc348 }
                        exit { sc342 sc343 }
                }
                sib18 {
                        bh newcor
                        curves { sc346 sc351 sc353 sc332 sc333 sc334 }
                        entry { sc351 sc353 }
                        exit { sc332 sc333 }
                }

                sic19 {
                        bh newcor
                        coils { sw10 sw11 sw12 }
                        curves { sc361 sc367 sc368 sc369 sc370 sc371 sc372 sc382 sc383
sc384 sc385 }
                        entry { sc371 sc372 }
                        exit { sc361 sc367 sc368 sc382 sc383 }
                }
                sl19 {
                        bh sl
                        curves { sc362 sc363 sc364 sc378 sc379 sc367 sc368 sc369 }
                        entry { sc367 sc368 }
                exit { sc362 sc363 }
        }
        sib19 {
                bh newcor
                curves { sc366 sc371 sc373 sc352 sc353 sc354 }
                entry { sc371 sc373 }
                exit { sc352 sc353 }
        }

        sic20 {
                bh newcor
                coils { sw10 sw11 sw12 sw13 }
                curves { sc381 sc387 sc388 sc389 sc390 sc391 sc392 sc402 sc403
sc404 sc405 }
                entry { sc391 sc392 }
                exit { sc381 sc387 sc388 sc402 sc403 }
        }
        sl20 {
                bh sl
                curves { sc382 sc383 sc384 sc398 sc399 sc387 sc388 sc389 }
                entry { sc387 sc388 }
                exit { sc382 sc383 }
        }
        sib20 {
                bh newcor
                curves { sc386 sc391 sc393 sc372 sc373 sc374 }
                entry { sc391 sc393 }
                exit { sc372 sc373 }
        }

        sic21 {
                bh newcor
                coils { sw10 sw11 sw12 sw13 sw14 }
                curves { sc401 sc407 sc408 sc409 sc410 sc411 sc412 sc422 sc423
sc424 sc425 }
                entry { sc411 sc412 }
                exit { sc401 sc407 sc408 sc422 sc423 }
        }
        sl21 {
                bh sl
                curves { sc402 sc403 sc404 sc418 sc419 sc407 sc408 sc409 }
                entry { sc407 sc408 }
                exit { sc402 sc403 }
        }
        sib21 {
                bh newcor
                curves { sc406 sc411 sc413 sc392 sc393 sc394 }
                entry { sc411 sc413 }
                exit { sc392 sc393 }
        }

        sic22 {
                bh newcor
                coils { sw10 sw11 sw12 sw13 sw14 sw15 }
                curves { sc421 sc427 sc428 sc429 sc430 sc431 sc432 sc442 sc443
sc444 sc445 }
                entry { sc431 sc432 }
                exit { sc421 sc427 sc428 sc442 sc443 }
        }
        sl22 {
                bh sl
                curves { sc422 sc423 sc424 sc438 sc439 sc427 sc428 sc429 }
                entry { sc427 sc428 }
                exit { sc422 sc423 }
        }
        sib22 {
                bh newcor
                curves { sc426 sc431 sc433 sc412 sc413 sc414 }
                entry { sc431 sc433 }
                exit { sc412 sc413 }
        }

        sic23 {
                bh newcor
                coils { sw10 sw11 sw13 sw14 sw15 }
                bh newcor
                curves { sc506 sc511 sc513 sc492 sc493 sc494 }
                entry { sc511 sc513 }
                exit { sc492 sc493 }
        }

        sic27 {
                bh newcor
                coils { sw13 sw14 sw15 sw16 sw17 }
                curves { sc521 sc527 sc528 sc529 sc530 sc531 sc532 sc542 sc543
sc544 sc545 }
                entry { sc531 sc532 }
                exit { sc521 sc527 sc528 sc542 sc543 }
        }
        sl27 {
                bh sl
                curves { sc522 sc523 sc524 sc538 sc539 sc527 sc528 sc529 }
                entry { sc527 sc528 }
                exit { sc522 sc523 }
        }
        sib27 {
                bh newcor
                curves { sc526 sc531 sc533 sc512 sc513 sc514 }
                entry { sc531 sc533 }
                exit { sc512 sc513 }
        }

        sic28 {
                bh newcor
                coils { sw13 sw14 sw15 sw16 sw17 sw18 }
                curves { sc541 sc547 sc548 sc549 sc550 sc551 sc552 sc562 sc563
sc564 sc565 }
                entry { sc551 sc552 }
                exit { sc541 sc547 sc548 sc562 sc563 }
        }
        sl28 {
                bh sl
                curves { sc542 sc543 sc544 sc558 sc559 sc547 sc548 sc549 }
                entry { sc547 sc548 }
                exit { sc542 sc543 }
        }
        sib28 {
                bh newcor
                curves { sc546 sc551 sc553 sc532 sc533 sc534 }
                entry { sc551 sc553 }
                exit { sc532 sc533 }
        }

        sic29 {
                bh newcor
                coils { sw13 sw14 sw16 sw17 sw18 }
                curves { sc561 sc567 sc568 sc569 sc570 sc571 sc572 sc582 sc583
sc584 sc585 }
                entry { sc571 sc572 }
                exit { sc561 sc567 sc568 sc582 sc583 }
        }
        sl29 {
                bh sl
                curves { sc562 sc563 sc564 sc578 sc579 sc567 sc568 sc569 }
                entry { sc567 sc568 }
                exit { sc562 sc563 }
        }
        sib29 {
                bh newcor
                curves { sc566 sc571 sc573 sc552 sc553 sc554 }
                entry { sc571 sc573 }
                exit { sc552 sc553 }
        }

        sic30 {
                bh newcor
                coils { sw13 sw16 sw17 sw18 }
                curves { sc581 sc587 sc588 sc589 sc590 sc591 sc592 sc602 sc603
sc604 sc605 }
                entry { sc591 sc592 }
        exit { sc581 sc587 sc588 sc602 sc603 }
}
sl30 {
        bh sl
        curv
```

```
# Brook Crompton D100L induction motor

unitdistance 0.001

airgap 1

bh {
  newcor exp 3.8 2.17 396.2 # linear 1989
  newcorril exp 3.8 2.17 396.2
  sl linear 1
}

stator
{
  length 96

  coils {  sw1 81 sw1a sw1b sw2 81 sw2a sw2b sw3 81 sw3a sw3b sw4 81 sw4a sw4b
    sw5 81 sw5a sw5b sw6 81 sw6a sw6b sw7 81 sw7a sw7b sw8 81 sw8a sw8b
    sw9 81 sw9a sw9b sw10 81 sw10a sw10b sw11 81 sw11a sw11b sw12 81 sw12a sw12b
    sw13 81 sw13a sw13b sw14 81 sw14a sw14b sw15 81 sw15a sw15b sw16 81 sw16a sw16b
    sw17 81 sw17a sw17b sw18 81 sw18a sw18b }

  nodes {
    sn0 0 d 0
    sn[1 36 20] [46.04 0] d [-1.56 10]
    sn[2 36 20] [46.294 0] d [-1.55 10]
    sn[3 36 20] [46.802 0] d [-2.15 10]
    sn[4 36 20] [47.074 0] d [-2.47 10]
    sn[5 36 20] [58.92 0] d [-3.09 10]
    sn[6 36 20] [46.04 0] d [1.56 10]
    sn[7 36 20] [46.294 0] d [1.55 10]
    sn[8 36 20] [46.802 0] d [2.15 10]
    sn[9 36 20] [47.074 0] d [2.47 10]
    sn[10 36 20] [58.92 0] d [3.09 10]
    sn[11 36 20] [62 0] d [5 10]
    sn[12 36 20] [76.45 0] d [5 10]
  }

  curves {
    sc[1 35 20] circ sn[6 20] sn[21 20] [46.04 0]
    sc701 circ sn706 sn1 46.04
    sc[2 36 20] line sn[1 20] sn[2 20]
    sc[3 36 20] circ sn[3 20] sn[2 20] [0.508 0]
    sc[4 36 20] circ sn[3 20] sn[4 20] [0.84 0]
    sc[5 36 20] line sn[4 20] sn[5 20]
    sc[6 36 20] circ sn[5 20] sn[10 20] [3.18 0]
    sc[7 36 20] line sn[6 20] sn[7 20]
    sc[8 36 20] circ sn[7 20] sn[8 20] [0.84 0]
    sc[9 36 20] circ sn[9 20] sn[8 20] [0.508 0]
    sc[10 36 20] line sn[9 20] sn[10 20]
    sc[11 36 20] line sn[10 20] sn[11 20]
    sc[12 35 20] line sn[11 20] sn[25 20]
    sc712 line sn711 sn5
    sc[13 36 20] line sn[11 20] sn[12 20]
    sc[14 35 20] circ sn[12 20] sn[32 20] [76.45 0]
    sc714 circ sn712 sn12 76.45
    sc[18 36 20] line sn[4 20] sn[9 20]
    sc[19 36 20] line sn[1 20] sn[6 20]
  }

  parts {
    sic1 {
      bh newcor
      coils { sw1 sw2 sw3 }
      curves { sc1 sc7 sc8 sc9 sc10 sc11 sc12 sc22 sc23 sc24 sc25 }
      entry { sc11 sc12 }
      exit { sc1 sc7 sc8 sc22 sc23 }
    }
    sl1 {
      bh sl
      curves { sc2 sc3 sc4 sc18 sc19 sc7 sc8 sc9 }
      entry { sc7 sc8 }
      exit { sc2 sc3 }
    }
    sib1 {
```

```
  bh newcor
  curves { sc6 sc11 sc13 sc712 sc713 sc714 }
  entry { sc11 sc13 }
  exit { sc712 sc713 }
}

sic2 {
  bh newcor
  coils { sw1 sw2 sw3 sw4 }
  curves { sc21 sc27 sc28 sc29 sc30 sc31 sc32 sc42 sc43 sc44 sc45 }
  entry { sc31 sc32 }
  exit { sc21 sc27 sc28 sc42 sc43 }
}
sl2 {
  bh sl
  curves { sc22 sc23 sc24 sc38 sc39 sc27 sc28 sc29 }
  entry { sc27 sc28 }
  exit { sc22 sc23 }
}
sib2 {
  bh newcor
  curves { sc26 sc31 sc33 sc12 sc13 sc14 }
  entry { sc31 sc33 }
  exit { sc12 sc13 }
}

sic3 {
  bh newcor
  coils { sw1 sw2 sw3 sw4 sw5 }
  curves { sc41 sc47 sc48 sc49 sc50 sc51 sc52 sc62 sc63 sc64 sc65 }
  entry { sc51 sc52 }
  exit { sc41 sc47 sc48 sc62 sc63 }
}
sl3 {
  bh sl
  curves { sc42 sc43 sc44 sc58 sc59 sc47 sc48 sc49 }
  entry { sc47 sc48 }
  exit { sc42 sc43 }
}
sib3 {
  bh newcor
  curves { sc46 sc51 sc53 sc32 sc33 sc34 }
  entry { sc51 sc53 }
  exit { sc32 sc33 }
}

sic4 {
  bh newcor
  coils { sw1 sw2 sw3 sw4 sw5 sw6 }
  curves { sc61 sc67 sc68 sc69 sc70 sc71 sc72 sc82 sc83 sc84 sc85 }
  entry { sc71 sc72 }
  exit { sc61 sc67 sc68 sc82 sc83 }
}
sl4 {
  bh sl
  curves { sc62 sc63 sc64 sc78 sc79 sc67 sc68 sc69 }
  entry { sc67 sc68 }
  exit { sc62 sc63 }
}
sib4 {
  bh newcor
  curves { sc66 sc71 sc73 sc52 sc53 sc54 }
  entry { sc71 sc73 }
  exit { sc52 sc53 }
}

sic5 {
  bh newcor
  coils { sw1 sw2 sw4 sw5 sw6 }
  curves { sc81 sc87 sc88 sc89 sc90 sc91 sc92 sc102 sc103 sc104 sc105 }
  entry { sc91 sc92 }
  exit { sc81 sc87 sc88 sc102 sc103 }
}
sl5 {
  bh sl
  curves { sc82 sc83 sc84 sc98 sc99 sc87 sc88 sc89 }
  entry { sc87 sc88 }
  exit { sc82 sc83 }
}
sib5 {
  bh newcor
  curves { sc86 sc91 sc93 sc72 sc73 sc74 }
  entry { sc91 sc93 }
  exit { sc72 sc73 }
}

sic6 {
  bh newcor
  coils { sw1 sw4 sw5 sw6 }
  curves { sc101 sc107 sc108 sc109 sc110 sc111 sc112 sc122 sc123 sc124 sc125 }
  entry { sc111 sc112 }
  exit { sc101 sc107 sc108 sc122 sc123 }
}
sl6 {
  bh sl
  curves { sc102 sc103 sc104 sc118 sc119 sc107 sc108 sc109 }
  entry { sc107 sc108 }
  exit { sc102 sc103 }
}
sib6 {
  bh newcor
  curves { sc106 sc111 sc113 sc92 sc93 sc94 }
  entry { sc111 sc113 }
  exit { sc92 sc93 }
}

sic7 {
  bh newcor
  coils { sw4 sw5 sw6 }
  curves { sc121 sc127 sc128 sc129 sc130 sc131 sc132 sc142 sc143 sc144 sc145 }
  entry { sc131 sc132 }
  exit { sc121 sc127 sc128 sc142 sc143 }
}
sl7 {
  bh sl
  curves { sc122 sc123 sc124 sc138 sc139 sc127 sc128 sc129 }
  entry { sc127 sc128 }
  exit { sc122 sc123 }
}
sib7 {
  bh newcor
  curves { sc126 sc131 sc133 sc112 sc113 sc114 }
  entry { sc131 sc133 }
  exit { sc112 sc113 }
}

sic8 {
  bh newcor
  coils { sw4 sw5 sw6 sw7 }
  curves { sc141 sc147 sc148 sc149 sc150 sc151 sc152 sc162 sc163 sc164 sc165 }
  entry { sc151 sc152 }
  exit { sc141 sc147 sc148 sc162 sc163 }
}
sl8 {
  bh sl
  curves { sc142 sc143 sc144 sc158 sc159 sc147 sc148 sc149 }
  entry { sc147 sc148 }
  exit { sc142 sc143 }
}
sib8 {
  bh newcor
  curves { sc146 sc151 sc153 sc132 sc133 sc134 }
  entry { sc151 sc153 }
  exit { sc132 sc133 }
}

sic9 {
  bh newcor
  coils { sw4 sw5 sw6 sw7 sw8 }
  curves { sc161 sc167 sc168 sc169 sc170 sc171 sc172 sc182 sc183 sc184 sc185 }
```

```
    entry { sc171 sc172 }
    exit { sc161 sc167 sc168 sc182 sc183 }
  }
  sl9 {
    bh sl
    curves { sc162 sc163 sc164 sc178 sc179 sc167 sc168 sc169 }
    entry { sc167 sc168 }
    exit { sc162 sc163 }
  }
  sib9 {
    bh newcor
    curves { sc166 sc171 sc173 sc152 sc153 sc154 }
    entry { sc171 sc173 }
    exit { sc152 sc153 }
  }

  sic10 {
    bh newcor
    coils { sw4 sw5 sw6 sw7 sw8 sw9 }
    curves { sc181 sc187 sc188 sc189 sc190 sc191 sc192 sc202 sc203 sc204 sc205 }
    entry { sc191 sc192 }
    exit { sc181 sc187 sc188 sc202 sc203 }
  }

  sl10 {
    bh sl
    curves { sc182 sc183 sc184 sc198 sc199 sc187 sc188 sc189 }
    entry { sc187 sc188 }
    exit { sc182 sc183 }
  }
  sib10 {
    bh newcor
    curves { sc186 sc191 sc193 sc172 sc173 sc174 }
    entry { sc191 sc193 }
    exit { sc172 sc173 }
  }

  sic11 {
    bh newcor
    coils { sw4 sw5 sw7 sw8 sw9 }
    curves { sc201 sc207 sc208 sc209 sc210 sc211 sc212 sc222 sc223 sc224 sc225 }
    entry { sc211 sc212 }
    exit { sc201 sc207 sc208 sc222 sc223 }
  }
  sl11 {
    bh sl
    curves { sc202 sc203 sc204 sc218 sc219 sc207 sc208 sc209 }
    entry { sc207 sc208 }
    exit { sc202 sc203 }
  }
  sib11 {
    bh newcor
    curves { sc206 sc211 sc213 sc192 sc193 sc194 }
    entry { sc211 sc213 }
    exit { sc192 sc193 }
  }

  sic12 {
    bh newcor
    coils { sw4 sw7 sw8 sw9 }
    curves { sc221 sc227 sc228 sc229 sc230 sc231 sc232 sc242 sc243 sc244 sc245 }
    entry { sc231 sc232 }
    exit { sc221 sc227 sc228 sc242 sc243 }
  }
  sl12 {
    bh sl
    curves { sc222 sc223 sc224 sc238 sc239 sc227 sc228 sc229 }
    entry { sc227 sc228 }
    exit { sc222 sc223 }
  }
  sib12 {
    bh newcor
    curves { sc226 sc231 sc233 sc212 sc213 sc214 }
    entry { sc231 sc233 }
    exit { sc212 sc213 }
```

```
}

sic13 {
  bh newcor
  coils { sw7 sw8 sw9 }
  curves { sc241 sc247 sc248 sc249 sc250 sc251 sc252 sc262 sc263 sc264 sc265 }
  entry { sc251 sc252 }
  exit { sc241 sc247 sc248 sc262 sc263 }
}
sl13 {
  bh sl
  curves { sc242 sc243 sc244 sc258 sc259 sc247 sc248 sc249 }
  entry { sc247 sc248 }
  exit { sc242 sc243 }
}
sib13 {
  bh newcor
  curves { sc246 sc251 sc253 sc232 sc233 sc234 }
  entry { sc251 sc253 }
  exit { sc232 sc233 }
}

sic14 {
  bh newcor
  coils { sw7 sw8 sw9 sw10 }
  curves { sc261 sc267 sc268 sc269 sc270 sc271 sc272 sc282 sc283 sc284 sc285 }
  entry { sc271 sc272 }
  exit { sc261 sc267 sc268 sc282 sc283 }
}
sl14 {
  bh sl
  curves { sc262 sc263 sc264 sc278 sc279 sc267 sc268 sc269 }
  entry { sc267 sc268 }
  exit { sc262 sc263 }
}
sib14 {
  bh newcor
  curves { sc266 sc271 sc273 sc252 sc253 sc254 }
  entry { sc271 sc273 }
  exit { sc252 sc253 }
}

sic15 {
  bh newcor
  coils { sw7 sw8 sw9 sw10 sw11 }
  curves { sc281 sc287 sc288 sc289 sc290 sc291 sc292 sc302 sc303 sc304 sc305 }
  entry { sc291 sc292 }
  exit { sc281 sc287 sc288 sc302 sc303 }
}
sl15 {
  bh sl
  curves { sc282 sc283 sc284 sc298 sc299 sc287 sc288 sc289 }
  entry { sc287 sc288 }
  exit { sc282 sc283 }
}
sib15 {
  bh newcor
  curves { sc286 sc291 sc293 sc272 sc273 sc274 }
  entry { sc291 sc293 }
  exit { sc272 sc273 }
}

sic16 {
  bh newcor
  coils { sw7 sw8 sw9 sw10 sw11 sw12 }
  curves { sc301 sc307 sc308 sc309 sc310 sc311 sc312 sc322 sc323 sc324 sc325 }
  entry { sc311 sc312 }
  exit { sc301 sc307 sc308 sc322 sc323 }
}
sl16 {
  bh sl
  curves { sc302 sc303 sc304 sc318 sc319 sc307 sc308 sc309 }
  entry { sc307 sc308 }
  exit { sc302 sc303 }
}
```

```
sib16 {
  bh newcor
  curves { sc306 sc311 sc313 sc292 sc293 sc294 }
  entry { sc311 sc313 }
  exit { sc292 sc293 }
}

sic17 {
  bh newcor
  coils { sw7 sw8 sw10 sw11 sw12 }
  curves { sc321 sc327 sc328 sc329 sc330 sc331 sc332 sc342 sc343 sc344 sc345 }
  entry { sc331 sc332 }
  exit { sc321 sc327 sc328 sc342 sc343 }
}
sl17 {
  bh sl
  curves { sc322 sc323 sc324 sc338 sc339 sc327 sc328 sc329 }
  entry { sc327 sc328 }
  exit { sc322 sc323 }
}
sib17 {
  bh newcor
  curves { sc326 sc331 sc333 sc312 sc313 sc314 }
  entry { sc331 sc333 }
  exit { sc312 sc313 }
}

sic18 {
  bh newcor
  coils { sw7 sw10 sw11 sw12 }
  curves { sc341 sc347 sc348 sc349 sc350 sc351 sc352 sc362 sc363 sc364 sc365 }
  entry { sc351 sc352 }
  exit { sc341 sc347 sc348 sc362 sc363 }
}
sl18 {
  bh sl
  curves { sc342 sc343 sc344 sc358 sc359 sc347 sc348 sc349 }
  entry { sc347 sc348 }
  exit { sc342 sc343 }
}
sib18 {
  bh newcor
  curves { sc346 sc351 sc353 sc332 sc333 sc334 }
  entry { sc351 sc353 }
  exit { sc332 sc333 }
}

sic19 {
  bh newcor
  coils { sw10 sw11 sw12 }
  curves { sc361 sc367 sc368 sc369 sc370 sc371 sc372 sc382 sc383 sc384 sc385 }
  entry { sc371 sc372 }
  exit { sc361 sc367 sc368 sc382 sc383 }
}
sl19 {
  bh sl
  curves { sc362 sc363 sc364 sc378 sc379 sc367 sc368 sc369 }
  entry { sc367 sc368 }
  exit { sc362 sc363 }
}
sib19 {
  bh newcor
  curves { sc366 sc371 sc373 sc352 sc353 sc354 }
  entry { sc371 sc373 }
  exit { sc352 sc353 }
}

sic20 {
  bh newcor
  coils { sw10 sw11 sw12 sw13 }
  curves { sc381 sc387 sc388 sc389 sc390 sc391 sc392 sc402 sc403 sc404 sc405 }
  entry { sc391 sc392 }
  exit { sc381 sc387 sc388 sc402 sc403 }
}
sl20 {
  bh sl
  curves { sc382 sc383 sc384 sc398 sc399 sc387 sc388 sc389 }
  entry { sc387 sc388 }
  exit { sc382 sc383 }
}
sib20 {
  bh newcor
  curves { sc386 sc391 sc393 sc372 sc373 sc374 }
  entry { sc391 sc393 }
  exit { sc372 sc373 }
}

sic21 {
  bh newcor
  coils { sw10 sw11 sw12 sw13 sw14 }
  curves { sc401 sc407 sc408 sc409 sc410 sc411 sc412 sc422 sc423 sc424 sc425 }
  entry { sc411 sc412 }
  exit { sc401 sc407 sc408 sc422 sc423 }
}
sl21 {
  bh sl
  curves { sc402 sc403 sc404 sc418 sc419 sc407 sc408 sc409 }
  entry { sc407 sc408 }
  exit { sc402 sc403 }
}
sib21 {
  bh newcor
  curves { sc406 sc411 sc413 sc392 sc393 sc394 }
  entry { sc411 sc413 }
  exit { sc392 sc393 }
}

sic22 {
  bh newcor
  coils { sw10 sw11 sw12 sw13 sw14 sw15 }
  curves { sc421 sc427 sc428 sc429 sc430 sc431 sc432 sc442 sc443 sc444 sc445 }
  entry { sc431 sc432 }
  exit { sc421 sc427 sc428 sc442 sc443 }
}
sl22 {
  bh sl
  curves { sc422 sc423 sc424 sc438 sc439 sc427 sc428 sc429 }
  entry { sc427 sc428 }
  exit { sc422 sc423 }
}
sib22 {
  bh newcor
  curves { sc426 sc431 sc433 sc412 sc413 sc414 }
  entry { sc431 sc433 }
  exit { sc412 sc413 }
}

sic23 {
  bh newcor
  coils { sw10 sw11 sw13 sw14 sw15 }
  curves { sc441 sc447 sc448 sc449 sc450 sc451 sc452 sc462 sc463 sc464 sc465 }
  entry { sc451 sc452 }
  exit { sc441 sc447 sc448 sc462 sc463 }
}
sl23 {
  bh sl
  curves { sc442 sc443 sc444 sc458 sc459 sc447 sc448 sc449 }
  entry { sc447 sc448 }
  exit { sc442 sc443 }
}
sib23 {
  bh newcor
  curves { sc446 sc451 sc453 sc432 sc433 sc434 }
  entry { sc451 sc453 }
  exit { sc432 sc433 }
}

sic24 {
  bh newcor
  coils { sw10 sw13 sw14 sw15 }
```

```
    curves { sc461 sc467 sc468 sc469 sc470 sc471 sc472 sc482 sc483 sc484 sc485 }
    entry { sc471 sc472 }
    exit { sc461 sc467 sc468 sc482 sc483 }
  }
  sl24 {
    bh sl
    curves { sc462 sc463 sc464 sc478 sc479 sc467 sc468 sc469 }
    entry { sc467 sc468 }
    exit { sc462 sc463 }
  }
  sib24 {
    bh newcor
    curves { sc466 sc471 sc473 sc452 sc453 sc454 }
    entry { sc471 sc473 }
    exit { sc452 sc453 }
  }

  sic25 {
    bh newcor
    coils { sw13 sw14 sw15 }
    curves { sc481 sc487 sc488 sc489 sc490 sc491 sc492 sc502 sc503 sc504 sc505 }
    entry { sc491 sc492 }
    exit { sc481 sc487 sc488 sc502 sc503}
  }
  sl25 {
    bh sl
    curves { sc482 sc483 sc484 sc498 sc499 sc487 sc488 sc489 }
    entry { sc487 sc488 }
    exit { sc482 sc483 }
  }
  sib25 {
    bh newcor
    curves { sc486 sc491 sc493 sc472 sc473 sc474 }
    entry { sc491 sc493 }
    exit { sc472 sc473 }
  }

  sic26 {
    bh newcor
    coils { sw13 sw14 sw15 sw16 }
    curves { sc501 sc507 sc508 sc509 sc510 sc511 sc512 sc522 sc523 sc524 sc525 }
    entry { sc511 sc512 }
    exit { sc501 sc507 sc508 sc522 sc523 }
  }
  sl26 {
    bh sl
    curves { sc502 sc503 sc504 sc518 sc519 sc507 sc508 sc509 }
    entry { sc507 sc508 }
    exit { sc502 sc503 }
  }
  sib26 {
    bh newcor
    curves { sc506 sc511 sc513 sc492 sc493 sc494 }
    entry { sc511 sc513 }
    exit { sc492 sc493 }
  }

  sic27 {
    bh newcor
    coils { sw13 sw14 sw15 sw16 sw17 }
    curves { sc521 sc527 sc528 sc529 sc530 sc531 sc532 sc542 sc543 sc544 sc545 }
    entry { sc531 sc532 }
    exit { sc521 sc527 sc528 sc542 sc543 }
  }
  sl27 {
    bh sl
    curves { sc522 sc523 sc524 sc538 sc539 sc527 sc528 sc529 }
    entry { sc527 sc528 }
    exit { sc522 sc523 }
  }
  sib27 {
    bh newcor
    curves { sc526 sc531 sc533 sc512 sc513 sc514 }
    entry { sc531 sc533 }
    exit { sc512 sc513 }
```

```
}

sic28 {
  bh newcor
  coils { sw13 sw14 sw15 sw16 sw17 sw18 }
  curves { sc541 sc547 sc548 sc549 sc550 sc551 sc552 sc562 sc563 sc564 sc565 }
  entry { sc551 sc552 }
  exit { sc541 sc547 sc548 sc562 sc563 }
}
sl28 {
  bh sl
  curves { sc542 sc543 sc544 sc558 sc559 sc547 sc548 sc549 }
  entry { sc547 sc548 }
  exit { sc542 sc543 }
}
sib28 {
  bh newcor
  curves { sc546 sc551 sc553 sc532 sc533 sc534 }
  entry { sc551 sc553 }
  exit { sc532 sc533 }
}

sic29 {
  bh newcor
  coils { sw13 sw14 sw16 sw17 sw18 }
  curves { sc561 sc567 sc568 sc569 sc570 sc571 sc572 sc582 sc583 sc584 sc585 }
  entry { sc571 sc572 }
  exit { sc561 sc567 sc568 sc582 sc583 }
}
sl29 {
  bh sl
  curves { sc562 sc563 sc564 sc578 sc579 sc567 sc568 sc569 }
  entry { sc567 sc568 }
  exit { sc562 sc563 }
}
sib29 {
  bh newcor
  curves { sc566 sc571 sc573 sc552 sc553 sc554 }
  entry { sc571 sc573 }
  exit { sc552 sc553 }
}

sic30 {
  bh newcor
  coils { sw13 sw16 sw17 sw18 }
  curves { sc581 sc587 sc588 sc589 sc590 sc591 sc592 sc602 sc603 sc604 sc605 }
  entry { sc591 sc592 }
  exit { sc581 sc587 sc588 sc602 sc603 }
}
sl30 {
  bh sl
  curves { sc582 sc583 sc584 sc598 sc599 sc587 sc588 sc589 }
  entry { sc587 sc588 }
  exit { sc582 sc583 }
}
sib30 {
  bh newcor
  curves { sc586 sc591 sc593 sc572 sc573 sc574 }
  entry { sc591 sc593 }
  exit { sc572 sc573 }
}

sic31 {
  bh newcor
  coils { sw16 sw17 sw18 }
  curves { sc601 sc607 sc608 sc609 sc610 sc611 sc612 sc622 sc623 sc624 sc625 }
  entry { sc611 sc612 }
  exit { sc601 sc607 sc608 sc622 sc623 }
}
sl31 {
  bh sl
  curves { sc602 sc603 sc604 sc618 sc619 sc607 sc608 sc609 }
  entry { sc607 sc608 }
  exit { sc602 sc603 }
}
```

```
sib31 {
  bh newcor
  curves { sc606 sc611 sc613 sc592 sc593 sc594 }
  entry { sc611 sc613 }
  exit { sc592 sc593 }
}

sic32 {
  bh newcor
  coils { sw1 sw16 sw17 sw18 }
  curves { sc621 sc627 sc628 sc629 sc630 sc631 sc632 sc642 sc643 sc644 sc645 }
  entry { sc631 sc632 }
  exit { sc621 sc627 sc628 sc642 sc643 }
}
sl32 {
  bh sl
  curves { sc622 sc623 sc624 sc638 sc639 sc627 sc628 sc629 }
  entry { sc627 sc628 }
  exit { sc622 sc623 }
}
sib32 {
  bh newcor
  curves { sc626 sc631 sc633 sc612 sc613 sc614 }
  entry { sc631 sc633 }
  exit { sc612 sc613 }
}

sic33 {
  bh newcor
  coils { sw1 sw2 sw16 sw17 sw18 }
  curves { sc641 sc647 sc648 sc649 sc650 sc651 sc652 sc662 sc663 sc664 sc665 }
  entry { sc651 sc652 }
  exit { sc641 sc647 sc648 sc662 sc663 }
}
sl33 {
  bh sl
  curves { sc642 sc643 sc644 sc658 sc659 sc647 sc648 sc649 }
  entry { sc647 sc648 }
  exit { sc642 sc643 }
}
sib33 {
  bh newcor
  curves { sc646 sc651 sc653 sc632 sc633 sc634 }
  entry { sc651 sc653 }
  exit { sc632 sc633 }
}

sic34 {
  bh newcor
  coils { sw1 sw2 sw3 sw16 sw17 sw18 }
  curves { sc661 sc667 sc668 sc669 sc670 sc671 sc672 sc682 sc683 sc684 sc685 }
  entry { sc671 sc672 }
  exit { sc661 sc667 sc668 sc682 sc683 }
}
sl34 {
  bh sl
  curves { sc662 sc663 sc664 sc678 sc679 sc667 sc668 sc669 }
  entry { sc667 sc668 }
  exit { sc662 sc663 }
}
sib34 {
  bh newcor
  curves { sc666 sc671 sc673 sc652 sc653 sc654 }
  entry { sc671 sc673 }
  exit { sc652 sc653 }
}

sic35 {
  bh newcor
  coils { sw1 sw2 sw3 sw17 sw18 }
  curves { sc681 sc687 sc688 sc689 sc690 sc691 sc692 sc702 sc703 sc704 sc705 }
  entry { sc691 sc692 }
  exit { sc681 sc687 sc688 sc702 sc703 }
}
sl35 {
      bh sl
      curves { sc682 sc683 sc684 sc698 sc699 sc687 sc688 sc689 }
      entry { sc687 sc688 }
      exit { sc682 sc683 }
    }
    sib35 {
      bh newcor
      curves { sc686 sc691 sc693 sc672 sc673 sc674 }
      entry { sc691 sc693 }
      exit { sc672 sc673 }
    }

    sic36 {
      bh newcor
      coils { sw1 sw2 sw3 sw18 }
      curves { sc701 sc707 sc708 sc709 sc710 sc711 sc712 sc2 sc3 sc4 sc5 }
      entry { sc711 sc712 }
      exit { sc701 sc707 sc708 sc2 sc3 }
    }
    sl36 {
      bh sl
      curves { sc702 sc703 sc704 sc718 sc719 sc707 sc708 sc709 }
      entry { sc707 sc708 }
      exit { sc702 sc703 }
    }
    sib36 {
      bh newcor
      curves { sc706 sc711 sc713 sc692 sc693 sc694 }
      entry { sc711 sc713 }
      exit { sc692 sc693 }
    }
  }
}

rotor {
  length 98

  skewangle d 3.5

  coils {  rb1 1 rb1a rb1b rb2 1 rb2a rb2b rb3 1 rb3a rb3b rb4 1 rb4a rb4b
        rb5 1 rb5a rb5b rb6 1 rb6a rb6b rb7 1 rb7a rb7b rb8 1 rb8a rb8b
        rb9 1 rb9a rb9b rb10 1 rb10a rb10b rb11 1 rb11a rb11b rb12 1 rb12a rb12b
        rb13 1 rb13a rb13b rb14 1 rb14a rb14b rb15 1 rb15a rb15b rb16 1 rb16a rb16b
        rb17 1 rb17a rb17b rb18 1 rb18a rb18b rb19 1 rb19a rb19b rb20 1 rb20a rb20b
        rb21 1 rb21a rb21b rb22 1 rb22a rb22b rb23 1 rb23a rb23b rb24 1 rb24a rb24b
        rb25 1 rb25a rb25b rb26 1 rb26a rb26b rb27 1 rb27a rb27b rb28 1 rb28a rb28b
        rb29 1 rb29a rb29b rb30 1 rb30a rb30b rb31 1 rb31a rb31b rb32 1 rb32a rb32b }

  nodes {
    rn0 0 d 0
    rn[1 32 20] [32.39 0] d [-1.88 11.25]
    rn[2 32 20] [43.26 0] d [-2.82 11.25]
    rn[3 32 20] [32.39 0] d [1.88 11.25]
    rn[4 32 20] [43.26 0] d [2.82 11.25]
    rn[5 32 20] [15.875 0] d [5.625 11.25]
    rn[6 32 20] [31 0] d [5.625 11.25]
    rn[7 32 20] [45.6 0] d [-2.82 11.25]
    rn[9 32 20] [45.6 0] d [2.82 11.25]
  }

  curves {
    rc[1 32 20] circ rn[3 20] rn[1 20] [1.1 0]
    rc[2 32 20] circ rn[2 20] rn[4 20] [2.2 0]
    rc[3 32 20] line rn[1 20] rn[2 20]
    rc[4 32 20] line rn[4 20] rn[3 20]
    rc[5 31 20] circ rn[5 20] rn[25 20] [15.875 0]
    rc625 circ rn625 rn5 15.875
    rc[6 32 20] line rn[5 20] rn[6 20]
    rc[7 32 20] line rn[3 20] rn[6 20]
    rc[8 31 20] line rn[6 20] rn[21 20]
    rc628 line rn626 rn1
    rc[9 32 20] line rn[2 20] rn[7 20]
    rc[11 32 20] line rn[4 20] rn[9 20]
    rc[13 31 20] circ rn[27 20] rn[9 20] [45.6 0]
    rc633 circ rn7 rn629 45.6
```

```
   rc[15 32 20] circ rn[9 20] rn[7 20] [45.6 0]
 }

 parts {
   ric1 {
     bh newcor
     coils { rb1 }
     curves { rc4 rc7 rc8 rc11 rc13 rc23 rc29 }
     entry { rc7 rc8 }
     exit { rc11 rc13 rc29 }
   }
   ril1 {
     bh newcorril
     curves { rc2 rc9 rc11 rc15 }
     entry { rc11 }
     exit { rc9 }
   }
   rib1 {
     bh newcor
     curves { rc1 rc625 rc6 rc7 rc626 rc628 }
     entry { rc6 rc7 }
     exit { rc626 rc628 }
   }

   ric2 {
     bh newcor
     coils { rb2 }
     curves { rc24 rc27 rc28 rc31 rc33 rc43 rc49 }
     entry { rc27 rc28 }
     exit { rc31 rc33 rc49 }
   }
   ril2 {
     bh newcorril
     curves { rc22 rc29 rc31 rc35 }
     entry { rc31 }
     exit { rc29 }
   }
   rib2 {
     bh newcor
     curves { rc21 rc5 rc26 rc27 rc6 rc8 }
     entry { rc26 rc27 }
     exit { rc6 rc8 }
   }

   ric3 {
     bh newcor
     coils { rb3 }
     curves { rc44 rc47 rc48 rc51 rc53 rc63 rc69 }
     entry { rc47 rc48 }
     exit { rc51 rc53 rc69 }
   }
   ril3 {
     bh newcorril
     curves { rc42 rc49 rc51 rc55 }
     entry { rc51 }
     exit { rc49 }
   }
   rib3 {
     bh newcor
     curves { rc41 rc25 rc46 rc47 rc26 rc28 }
     entry { rc46 rc47 }
     exit { rc26 rc28 }
   }

   ric4 {
     bh newcor
     coils { rb4 }
     curves { rc64 rc67 rc68 rc71 rc73 rc83 rc89 }
     entry { rc67 rc68 }
     exit { rc71 rc73 rc89 }
   }
   ril4 {
     bh newcorril
     curves { rc62 rc69 rc71 rc75 }
     entry { rc71 }
```

```
  exit { rc69 }
}
rib4 {
  bh newcor
  curves { rc61 rc45 rc66 rc67 rc46 rc48 }
  entry { rc66 rc67 }
  exit { rc46 rc48 }
}

ric5 {
  bh newcor
  coils { rb5 }
  curves { rc84 rc87 rc88 rc91 rc93 rc103 rc109 }
  entry { rc87 rc88 }
  exit { rc91 rc93 rc109 }
}
ril5 {
  bh newcorril
  curves { rc82 rc89 rc91 rc95 }
  entry { rc91 }
  exit { rc89 }
}
rib5 {
  bh newcor
  curves { rc81 rc65 rc86 rc87 rc66 rc68 }
  entry { rc86 rc87 }
  exit { rc66 rc68 }
}

ric6 {
  bh newcor
  coils { rb6 }
  curves { rc104 rc107 rc108 rc111 rc113 rc123 rc129 }
  entry { rc107 rc108 }
  exit { rc111 rc113 rc129}
}
ril6 {
  bh newcorril
  curves { rc102 rc109 rc111 rc115 }
  entry { rc111 }
  exit { rc109 }
}
rib6 {
  bh newcor
  curves { rc101 rc85 rc106 rc107 rc86 rc88 }
  entry { rc106 rc107 }
  exit { rc86 rc88 }
}

ric7 {
  bh newcor
  coils { rb7 }
  curves { rc124 rc127 rc128 rc131 rc133 rc143 rc149 }
  entry { rc127 rc128 }
  exit { rc131 rc133 rc149 }
}
ril7 {
  bh newcorril
  curves { rc122 rc129 rc131 rc135 }
  entry { rc131 }
  exit { rc129 }
}
rib7 {
  bh newcor
  curves { rc121 rc105 rc126 rc127 rc106 rc108 }
  entry { rc126 rc127 }
  exit { rc106 rc108 }
}

ric8 {
  bh newcor
  coils { rb8 }
  curves { rc144 rc147 rc148 rc151 rc153 rc163 rc169 }
  entry { rc147 rc148 }
  exit { rc151 rc153 rc169 }
```

```
}
ril8 {
  bh newcorril
  curves { rc142 rc149 rc151 rc155 }
  entry { rc151 }
  exit { rc149 }
}
rib8 {
  bh newcor
  curves { rc141 rc125 rc146 rc147 rc126 rc128 }
  entry { rc146 rc147 }
  exit { rc126 rc128 }
}

ric9 {
  bh newcor
  coils { rb9 }
  curves { rc164 rc167 rc168 rc171 rc173 rc183 rc189 }
  entry { rc167 rc168 }
  exit { rc171 rc173 rc189 }
}
ril9 {
  bh newcorril
  curves { rc162 rc169 rc171 rc175 }
  entry { rc171 }
  exit { rc169 }
}
rib9 {
  bh newcor
  curves { rc161 rc145 rc166 rc167 rc146 rc148 }
  entry { rc166 rc167 }
  exit { rc146 rc148 }
}

ric10 {
  bh newcor
  coils { rb10 }
  curves { rc184 rc187 rc188 rc191 rc193 rc203 rc209 }
  entry { rc187 rc188 }
  exit { rc191 rc193 rc209 }
}
ril10 {
  bh newcorril
  curves { rc182 rc189 rc191 rc195 }
  entry { rc191 }
  exit { rc189 }
}
rib10 {
  bh newcor
  curves { rc181 rc165 rc186 rc187 rc166 rc168 }
  entry { rc186 rc187 }
  exit { rc166 rc168 }
}

ric11 {
  bh newcor
  coils { rb11 }
  curves { rc204 rc207 rc208 rc211 rc213 rc223 rc229 }
  entry { rc207 rc208 }
  exit { rc211 rc213 rc229 }
}
ril11 {
  bh newcorril
  curves { rc202 rc209 rc211 rc215 }
  entry { rc211 }
  exit { rc209 }
}
rib11 {
  bh newcor
  curves { rc201 rc185 rc206 rc207 rc186 rc188 }
  entry { rc206 rc207 }
  exit { rc186 rc188 }
}

ric12 {
  bh newcor
  coils { rb12 }
  curves { rc224 rc227 rc228 rc231 rc233 rc243 rc249 }
  entry { rc227 rc228 }
  exit { rc231 rc233 rc249 }
}
ril12 {
  bh newcorril
  curves { rc222 rc229 rc231 rc235 }
  entry { rc231 }
  exit { rc229 }
}
rib12 {
  bh newcor
  curves { rc221 rc205 rc226 rc227 rc206 rc208 }
  entry { rc226 rc227 }
  exit { rc206 rc208 }
}

ric13 {
  bh newcor
  coils { rb13 }
  curves { rc244 rc247 rc248 rc251 rc253 rc263 rc269 }
  entry { rc247 rc248 }
  exit { rc251 rc253 rc269 }
}
ril13 {
  bh newcorril
  curves { rc242 rc249 rc251 rc255 }
  entry { rc251 }
  exit { rc249 }
}
rib13 {
  bh newcor
  curves { rc241 rc225 rc246 rc247 rc226 rc228 }
  entry { rc246 rc247 }
  exit { rc226 rc228 }
}

ric14 {
  bh newcor
  coils { rb14 }
  curves { rc264 rc267 rc268 rc271 rc273 rc283 rc289 }
  entry { rc267 rc268 }
  exit { rc271 rc273 rc289 }
}
ril14 {
  bh newcorril
  curves { rc262 rc269 rc271 rc275 }
  entry { rc271 }
  exit { rc269 }
}
rib14 {
  bh newcor
  curves { rc261 rc245 rc266 rc267 rc246 rc248 }
  entry { rc266 rc267 }
  exit { rc246 rc248 }
}

ric15 {
  bh newcor
  coils { rb15 }
  curves { rc284 rc287 rc288 rc291 rc293 rc303 rc309 }
  entry { rc287 rc288 }
  exit { rc291 rc293 rc309 }
}
ril15 {
  bh newcorril
  curves { rc282 rc289 rc291 rc295 }
  entry { rc291 }
  exit { rc289 }
}
rib15 {
  bh newcor
  curves { rc281 rc265 rc286 rc287 rc266 rc268 }
  entry { rc286 rc287 }
  exit { rc266 rc268 }
}

ric16 {
  bh newcor
  coils { rb16 }
  curves { rc304 rc307 rc308 rc311 rc313 rc323 rc329 }
  entry { rc307 rc308 }
  exit { rc311 rc313 rc329 }
}
ril16 {
  bh newcorril
  curves { rc302 rc309 rc311 rc315 }
  entry { rc311 }
  exit { rc309 }
}
rib16 {
  bh newcor
  curves { rc301 rc285 rc306 rc307 rc286 rc288 }
  entry { rc306 rc307 }
  exit { rc286 rc288 }
}

ric17 {
  bh newcor
  coils { rb17 }
  curves { rc324 rc327 rc328 rc331 rc333 rc343 rc349 }
  entry { rc327 rc328 }
  exit { rc331 rc333 rc349 }
}
ril17 {
  bh newcorril
  curves { rc322 rc329 rc331 rc335 }
  entry { rc331 }
  exit { rc329 }
}
rib17 {
  bh newcor
  curves { rc321 rc305 rc326 rc327 rc306 rc308 }
  entry { rc326 rc327 }
  exit { rc306 rc308 }
}

ric18 {
  bh newcor
  coils { rb18 }
  curves { rc344 rc347 rc348 rc351 rc353 rc363 rc369 }
  entry { rc347 rc348 }
  exit { rc351 rc353 rc369 }
}
ril18 {
  bh newcorril
  curves { rc342 rc349 rc351 rc355 }
  entry { rc351 }
  exit { rc349 }
}
rib18 {
  bh newcor
  curves { rc341 rc325 rc346 rc347 rc326 rc328 }
  entry { rc346 rc347 }
  exit { rc326 rc328 }
}

ric19 {
  bh newcor
  coils { rb19 }
  curves { rc364 rc367 rc368 rc371 rc373 rc383 rc389 }
  entry { rc367 rc368 }
  exit { rc371 rc373 rc389 }
}
ril19 {
  bh newcorril
  curves { rc362 rc369 rc371 rc375 }
  entry { rc371 }
  exit { rc369 }
}
rib19 {
  bh newcor
  curves { rc361 rc345 rc366 rc367 rc346 rc348 }
  entry { rc366 rc367 }
  exit { rc346 rc348 }
}

ric20 {
  bh newcor
  coils { rb20 }
  curves { rc384 rc387 rc388 rc391 rc393 rc403 rc409 }
  entry { rc387 rc388 }
  exit { rc391 rc393 rc409 }
}
ril20 {
  bh newcorril
  curves { rc382 rc389 rc391 rc395 }
  entry { rc391 }
  exit { rc389 }
}
rib20 {
  bh newcor
  curves { rc381 rc365 rc386 rc387 rc366 rc368 }
  entry { rc386 rc387 }
  exit { rc366 rc368 }
}

ric21 {
  bh newcor
  coils { rb21 }
  curves { rc404 rc407 rc408 rc411 rc413 rc423 rc429 }
  entry { rc407 rc408 }
  exit { rc411 rc413 rc429 }
}
ril21 {
  bh newcorril
  curves { rc402 rc409 rc411 rc415 }
  entry { rc411 }
  exit { rc409 }
}
rib21 {
  bh newcor
  curves { rc401 rc385 rc406 rc407 rc386 rc388 }
  entry { rc406 rc407 }
  exit { rc386 rc388 }
}

ric22 {
  bh newcor
  coils { rb22 }
  curves { rc424 rc427 rc428 rc431 rc433 rc443 rc449 }
  entry { rc427 rc428 }
  exit { rc431 rc433 rc449 }
}
ril22 {
  bh newcorril
  curves { rc422 rc429 rc431 rc435 }
  entry { rc431 }
  exit { rc429 }
}
rib22 {
  bh newcor
  curves { rc421 rc405 rc426 rc427 rc406 rc408 }
  entry { rc426 rc427 }
  exit { rc406 rc408 }
}

ric23 {
  bh newcor
  coils { rb23 }
  curves { rc444 rc447 rc448 rc451 rc453 rc463 rc469 }
  entry { rc447 rc448 }
  exit { rc451 rc453 rc469 }
```

```
}
ril23 {
  bh newcorril
  curves { rc442 rc449 rc451 rc455 }
  entry { rc451 }
  exit { rc449 }
}
rib23 {
  bh newcor
  curves { rc441 rc425 rc446 rc447 rc426 rc428 }
  entry { rc446 rc447 }
  exit { rc426 rc428 }
}

ric24 {
  bh newcor
  coils { rb24 }
  curves { rc464 rc467 rc468 rc471 rc473 rc483 rc489 }
  entry { rc467 rc468 }
  exit { rc471 rc473 rc489 }
}
ril24 {
  bh newcorril
  curves { rc462 rc469 rc471 rc475 }
  entry { rc471 }
  exit { rc469 }
}
rib24 {
  bh newcor
  curves { rc461 rc445 rc466 rc467 rc446 rc448 }
  entry { rc466 rc467 }
  exit { rc446 rc448 }
}

ric25 {
  bh newcor
  coils { rb25 }
  curves { rc484 rc487 rc488 rc491 rc493 rc503 rc509 }
  entry { rc487 rc488 }
  exit { rc491 rc493 rc509 }
}
ril25 {
  bh newcorril
  curves { rc482 rc489 rc491 rc495 }
  entry { rc491 }
  exit { rc489 }
}
rib25 {
  bh newcor
  curves { rc481 rc465 rc486 rc487 rc466 rc468 }
  entry { rc486 rc487 }
  exit { rc466 rc468 }
}

ric26 {
  bh newcor
  coils { rb26 }
  curves { rc504 rc507 rc508 rc511 rc513 rc523 rc529 }
  entry { rc507 rc508 }
  exit { rc511 rc513 rc529 }
}
ril26 {
  bh newcorril
  curves { rc502 rc509 rc511 rc515 }
  entry { rc511 }
  exit { rc509 }
}
rib26 {
  bh newcor
  curves { rc501 rc485 rc506 rc507 rc486 rc488 }
  entry { rc506 rc507 }
  exit { rc486 rc488 }
}

ric27 {
    bh newcor
    coils { rb27 }
    curves { rc524 rc527 rc528 rc531 rc533 rc543 rc549 }
    entry { rc527 rc528 }
    exit { rc531 rc533 rc549 }
  }
  ril27 {
    bh newcorril
    curves { rc522 rc529 rc531 rc535 }
    entry { rc531 }
    exit { rc529 }
  }
  rib27 {
    bh newcor
    curves { rc521 rc505 rc526 rc527 rc506 rc508 }
    entry { rc526 rc527 }
    exit { rc506 rc508 }
  }

  ric28 {
    bh newcor
    coils { rb28 }
    curves { rc544 rc547 rc548 rc551 rc553 rc563 rc569 }
    entry { rc547 rc548 }
    exit { rc551 rc553 rc569 }
  }
  ril28 {
    bh newcorril
    curves { rc542 rc549 rc551 rc555 }
    entry { rc551 }
    exit { rc549 }
  }
  rib28 {
    bh newcor
    curves { rc541 rc525 rc546 rc547 rc526 rc528 }
    entry { rc546 rc547 }
    exit { rc526 rc528 }
  }

  ric29 {
    bh newcor
    coils { rb29 }
    curves { rc564 rc567 rc568 rc571 rc573 rc583 rc589 }
    entry { rc567 rc568 }
    exit { rc571 rc573 rc589 }
  }
  ril29 {
    bh newcorril
    curves { rc562 rc569 rc571 rc575 }
    entry { rc571 }
    exit { rc569 }
  }
  rib29 {
    bh newcor
    curves { rc561 rc545 rc566 rc567 rc546 rc548 }
    entry { rc566 rc567 }
    exit { rc546 rc548 }
  }

  ric30 {
    bh newcor
    coils { rb30 }
    curves { rc584 rc587 rc588 rc591 rc593 rc603 rc609 }
    entry { rc587 rc588 }
    exit { rc591 rc593 rc609 }
  }
  ril30 {
    bh newcorril
    curves { rc582 rc589 rc591 rc595 }
    entry { rc591 }
    exit { rc589 }
  }
  rib30 {
    bh newcor
    curves { rc581 rc565 rc586 rc587 rc566 rc568 }
```

```
      entry { rc586 rc587 }
      exit { rc566 rc568 }
    }

    ric31 {
      bh newcor
      coils { rb31 }
      curves { rc604 rc607 rc608 rc611 rc613 rc623 rc629 }
      entry { rc607 rc608 }
      exit { rc611 rc613 rc629 }
    }
    ril31 {
      bh newcorril
      curves { rc602 rc609 rc611 rc615 }
      entry { rc611 }
      exit { rc609 }
    }
    rib31 {
      bh newcor
      curves { rc601 rc585 rc606 rc607 rc586 rc588 }
      entry { rc606 rc607 }
      exit { rc586 rc588 }
    }

    ric32 {
      bh newcor
      coils { rb32 }
      curves { rc624 rc627 rc628 rc631 rc633 rc3 rc9 }
      entry { rc627 rc628 }
      exit { rc631 rc633 rc9 }
    }
    ril32 {
      bh newcorril
      curves { rc622 rc629 rc631 rc635 }
      entry { rc631 }
      exit { rc629 }
    }
    rib32 {
      bh newcor
      curves { rc621 rc605 rc626 rc627 rc606 rc608 }
      entry { rc626 rc627 }
      exit { rc606 rc608 }
    }
  }
}
```

Appendices

H

'gdl2spice' program listing

```
#if !defined(bh_h)              // Sentry, use file only if it's not already included.
#define bh_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         bh.h
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Class definitions for bh.
*/

#include <string>

#include "unique.h"
#include "descobj.h"
#include "part.h"

class Description;

// Bh
class Bh : public DescriptionObject
{
public:
    enum BhType {UNKNOWN, LINEAR, EXP};
private:
    BhType type;
    Description* description;
    double k1, k2, k3;
public:
    Bh(Description* d, const string& n);
    Bh(Description* d, const char* n = "unknown");
    ~Bh();

    BhType GetType();
    double GetK1();
    double GetK2();
    double GetK3();
    Description* GetDescription();
    Bh::BhType SetType(Bh::BhType);
    double SetK1(double);
    double SetK2(double);
    double SetK3(double);
    void SetK(double, double, double);
    string GetCapNode(Part&);
    double GetCapValue(double);

    void WriteProperties(ofstream&);
    static bool Parse(LexGDL&, Description*, bool inform=true);
    bool ConvertLinear(double&, double);
    bool Convert(ofstream&, Part&, long, long, double, double); // needs reference to
part to build voltage source
};

typedef Unique<Bh *> Bhs;
typedef Unique<Bh *>::iterator BhsIterator;

void BhsWrite(ofstream&, Bhs&);
#endif                                      // bh_h sentry.
```

```
/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         bh.cpp
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Source file for implementation of bh.
*/

#include "bh.h"
#include "desc.h"

const double BH_PERM = 1.2566E-6; // 4*M_PI*1E-7 H/m

void BhsWrite(ofstream& out, Bhs& bhs)
{
    out << "bh { ";
    BhsIterator iter=bhs.begin();
    while (iter != bhs.end()) {
        (*iter)->WriteProperties(out);
        out << " ";
        iter++;
    }
    out << "}";
}

void Bh::WriteProperties(ofstream& out)
{
    out << GetName() << " ";
    if (type==Bh::LINEAR) out << "linear " << k1;
    if (type==Bh::EXP) out << "exp " << k1 << " " << k2 << " " << k3;
}

bool Bh::ConvertLinear(double& value, double integral)
{
    if (type==Bh::LINEAR) {
        value = ((1.0/integral)*k1*BH_PERM);
        return true;
    }
    else {
        return false;
    }
}

string Bh::GetCapNode(Part& part)
{
  return string("B"+part.GetName()+"_P");
}

double Bh::GetCapValue(double integral)
{
    if (type==Bh::LINEAR) return ((k1*BH_PERM)/integral);
    if (type==Bh::EXP) return (1.0/integral);
    return 0;
}

bool Bh::Convert(ofstream& cct,Part& part, long fNode, long sNode, double integral,
double volume)
{
    bool ok=false;
    if (type==Bh::LINEAR) {
        cct << "* magnetic capacitance:\n";
        cct << "C" << part.GetName() << " " << fNode << " " << sNode << " " <<
((k1*BH_PERM)/integral) << " ic=0\n";
//          cct << "RMC" << part.GetName() << " " << fNode << " " << sNode << " 1e12\n";
        ok = true;
    }
    if (type==Bh::EXP) {
        long rNode=GetDescription()->GetCctNode(); // midpoint node for flux loss
equivalent resistance
```

```
        cct << "* flux loss equivalent resistance:\n"; // needs volume calculation added
        cct << "RL" << part.GetName() << " " << fNode << " " <<  rNode << " 1n\n";
        cct << "* nonlinear magnetic capacitance:\n";
        cct << "B" << part.GetName() << " B" << part.GetName() << "_P 0  v=V(" << rNode
<< ", " << sNode << ")";
        cct << "/(" << k1 << "*exp(" << k2 << "*V(B" << part.GetName() << "_P)*V(B" <<
part.GetName() << "_P))+" << k3 << ")";
        cct << "\nC" << part.GetName() << " B" << part.GetName() << "_P  V" <<
part.GetName() << "_P " << (1.0/integral) << " ic=0\n";
        cct << "V" << part.GetName() << " V" << part.GetName() << "_P  0 0\n";
        cct << "F" << part.GetName() << " " << rNode << " " << sNode << " V" <<
part.GetName() << " 1\n";
//        cct << "RMC" << part.GetName() << " B" << part.GetName() << "_P 0 1e12\n";
        ok = true;
    }
    return ok;
}

Bh::Bh(Description* d, const string& n):DescriptionObject(n)
{
    description=d;
    type=Bh::UNKNOWN;
    k1 = k2 = k3 = 0;
}

Bh::Bh(Description* d, const char* n):DescriptionObject(n)
{
    description=d;
    type=Bh::UNKNOWN;
    k1 = k2 = k3 = 0;
}

Bh::~Bh()
{
}

Description* Bh::GetDescription()
{
    return description;
}

Bh::BhType Bh::GetType()
{
    return type;
}

double Bh::GetK1()
{
    return k1;
}

double Bh::GetK2()
{
    return k2;
}

double Bh::GetK3()
{
    return k3;
}

Bh::BhType Bh::SetType(Bh::BhType t)
{
    return (type=t);
}

double Bh::SetK1(double s)
{
    return (k1=s);
}

double Bh::SetK2(double s)
{
    return (k2=s);
}
```

```
double Bh::SetK3(double s)
{
    return (k3=s);
}

void Bh::SetK(double s1, double s2, double s3)
{
    k1=s1;
    k2=s2;
    k3=s3;
}

bool Bh::Parse(LexGDL& lex, Description* desc, bool inform) // expects to find a new
token
{
    int state = 0;
    bool ok = true;
    bool currok = true;
    string nameId;
    Bh::BhType type;
    double k1=0;
    double k2=0;
    double k3=0;
    do {
        switch (state) {
            case 0 : {
                if (lex.CurrStr()=="{") {
                    lex.GetToken();  // set up for parse
                    state++;
                }
                else {
                    cerr << "ERROR (line: " << lex.LineCount() << ") '{' expected" <<
endl;
                    state = 8; // get out of parser
                    currok = false;
                }
            }
            break;
            case 1 : {
                if (lex.CurrToken()==LexGDL::UNKNOWN) {
                    nameId = lex.CurrStr();
                    k1 = k2 = k3 = 0;
                    lex.GetToken();  // new token
                    currok = true; // reset ok flag as new bh
                    state++;
                }
                else {
                    lex.GetToken();  // get rid of token
                    currok = false;
                    state = 7; // find unknown
                }
            }
            break;
            case 2 : {
                switch (lex.CurrToken()) {
                    case LexGDL::LINEAR : type=Bh::LINEAR; state++; break;
                    case LexGDL::EXP : type=Bh::EXP; state++; break;
                    default : currok = false; state = 7; // find unknown
                }
                lex.GetToken();  // new token
            }
            break;
            case 3 : {
                currok &= lex.CurrVar(k1);
                lex.GetToken();  // new token
                if (type==Bh::EXP)
                    state++;
                else
                    state = 6; // create new bh objcet
            }
            break;
            case 4 : {
                currok &= lex.CurrVar(k2);
                state++;
                lex.GetToken();
            }
            break;
            case 5 : {
                currok &= lex.CurrVar(k3);
                state++;
                lex.GetToken();
            }
            break;
            case 6 : {
                if (currok) {
                    Bh* bh = new Bh(desc, nameId);
                    if (desc->Add(bh)) {
                        bh->SetType(type);
                        bh->SetK(k1,k2,k3);
                        if (inform) {
                            cout << "BH: " << bh->GetName() << " #" << desc-
>GetBhs().size() << " added" << endl;
                        }
                    }
                    else {
                        cerr << "ERROR (bh: " << bh->GetName() << ") duplicate" << endl;
                        ok = false;
                        delete bh;
                    }
                }
                else {
                    cerr << "ERROR (line: " << lex.LineCount() << " bh: " << nameId
                         << ") syntax '{ <name> linear|exp <float> [<float> <float>] }'
expected" << endl;
                    ok = false;
                }
                state++;
            }
            break;
            case 7 : { // check for closing bracket
                if (lex.CurrStr()=="}") {
                    lex.GetToken();  // new token
                    state++;
                }
                else {
                    state = 1; // get unknown
                }
            }
            break;
            default : {
                currok = false;
                cerr << "Bh parser illegal state";
                state = 7; // get unknown
            }
        }
        ok &= currok;
/*        if (inform) {
            cout << "Bh parse state: " << state << endl;
        } */
    } while (lex.CurrToken()!=LexGDL::END && state<8);
    if (lex.PrevStr()!="}") {
        cerr << "ERROR (line: " << lex.LineCount()-1 << ") '}' expected" << endl;
        lex.GetToken();  // set up for parse
    }
    return ok;
}
```

```
#if !defined(coil_h)              // Sentry, use file only if it's not already included.
#define coil_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         coil.h
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Class definitions for coil.
*/

#include <string>

#include "unique.h"
#include "descobj.h"

class Frame;

// Coil
class Coil : public DescriptionObject
{
private:
    Frame* frame;
    string firstTerminal;
    string secondTerminal;
    double turns;

public:
    Coil(Frame*, const string&);
    Coil(Frame*, const char* n = "unknown");
    ~Coil();

    Frame* GetFrame();
    string& SetFirst(string&);
    string& SetSecond(string&);
    double SetTurns(const double&);
    string& GetFirst();
    string& GetSecond();
    double GetTurns();

    void WriteProperties(ofstream&);
    static bool Parse(LexGDL&, Frame*, bool inform=true);
    bool Convert(ofstream&);
};

typedef Unique<Coil *> Coils;
typedef Unique<Coil *>::iterator CoilsIterator;
typedef Unique<Coil *>::reverse_iterator CoilsReverseIterator;

void CoilsWrite(ofstream&, Coils&);
void CoilsConvertElectrical(ofstream&, Coils&); // convert coils to circuit description
void CoilListConvertTerminals(ofstream&, Coils&); // output a list of the circuit
terminals

#endif                                        // coil_h sentry.
```

```
/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         coil.cpp
    AUTHOR:       D. Downes

    OVERVIEW
    ========
    Source file for implementation of coil.
*/

#include <strstream>

#include "coil.h"
#include "desc.h"

void CoilsWrite(ofstream& out, Coils& coils)
{
    out << "coils { ";
    CoilsIterator iter=coils.begin();
    while (iter != coils.end()) {
        (*iter)->WriteProperties(out);
        out << " ";
        iter++;
    }
    out << "}";
}

void CoilsConvertElectrical(ofstream& cct, Coils& coils) // convert coil list to circuit
description
{
    CoilsIterator iter=coils.begin();
    while (iter != coils.end()) {
        (*iter)->Convert(cct);
        iter++;
    }
}

void CoilListConvertTerminals(ofstream& cct, Coils& coils) // output a list of the
circuit terminals
{
    CoilsReverseIterator iter=coils.rbegin();
    while (iter != coils.rend()) { // list coils
        cct << (*iter)->GetFirst() << " " << (*iter)->GetSecond();
        if (++iter != coils.rend()) cct << " ";
    }
}

void Coil::WriteProperties(ofstream& out)
{
    out << GetName() << " " << turns << " " << firstTerminal << " " << secondTerminal;
}

bool Coil::Convert(ofstream& cct)
{
    // build list of parts that are linked to this coil
    Parts parts;  // new list of parts that are linked to coil
    PartsIterator iter=(frame->GetParts()).begin(); // parts contained in frame
    while (iter != (frame->GetParts()).end()) { // step through part list
        if ((*iter)->Find(this)) // part contains this coil in its coil list?
            parts.Add(*iter); // if it does then add the part to the coils part list
        iter++;
    }
    // generate gyrators
    PartsIterator iterParts=parts.begin();
    if (iterParts != parts.end()) { // check coil belongs to at least one part
        string cctNode;
        Part* partptr;
        long prevCctNode = frame->GetDescription()->GetCctNode();
        long midCctNode=prevCctNode; // mid point connection of src pair, always numeric
        cct << "Vsens" << GetName() << " " << firstTerminal << " " << midCctNode  << "
0\n";
        do {
```

```
//              frame->AddCctNode(midCctNode); // provide ground point for mid node
            partptr = *(iterParts++); // set pointer so as to find out if its the last
part in the list
            if (iterParts == parts.end()) {// last coil
                cctNode=secondTerminal; // last coil so use the second coil terminal
            }
            else {
                strstream num; // used to convert numeric to string
                prevCctNode = frame->GetDescription()->GetCctNode();
                num << prevCctNode; // needs a new cct node
                cctNode=num.str();
            }
            cct << "H" << GetName() << partptr->GetName() << " " << cctNode << " " <<
midCctNode
                << " Vsens" << partptr->GetName() << " " << (turns) << "\n";
            midCctNode=prevCctNode;
        } while (iterParts != parts.end());
        return true;
    }
    else {
        return false;
    }
}

Coil::Coil(Frame* f, const string& n):DescriptionObject(n)
{
    frame=f;
    turns=1;
}

Coil::Coil(Frame* f, const char* n):DescriptionObject(n)
{
    frame=f;
    turns=1;
}

Coil::~Coil()
{
}

Frame* Coil::GetFrame()
{
    return frame;
}

string& Coil::SetFirst(string& first)
{
    return (firstTerminal=first);
}

string& Coil::SetSecond(string& second)
{
  return (secondTerminal=second);
}

double Coil::SetTurns(const double& t)
{
    return (turns=t);
}

string& Coil::GetFirst()
{
    return firstTerminal;
}

string& Coil::GetSecond()
{
    return secondTerminal;
}

double Coil::GetTurns()
{
    return turns;
}
```

```
bool Coil::Parse(LexGDL& lex, Frame * frame, bool inform) // expects to find two new
tokens
{
    int state = 0;
    bool ok = true;
    bool currok = true;
    string nameId, firstTerm, secTerm;
    double turns;
    do {
        switch (state) {
            case 0 : {
                if (lex.CurrStr()=="{") {
                    lex.GetToken();  // set up for parse
                    state++;
                }
                else {
                    cerr << "ERROR (line: " << lex.LineCount() << ") '{' expected" <<
endl;
                    state = 7; // get ot of parser
                    currok = false;
                }
            }
            break;
            case 1 : {
                if (lex.CurrToken()==LexGDL::UNKNOWN) {
                    nameId = lex.CurrStr();
                    lex.GetToken();  // new token
                    currok = true; // reset ok flag as new bh
                    state++;
                }
                else {
                    lex.GetToken();  // get rid of token
                    currok = false;
                    state = 6; // find unknown
                }
            }
            break;
            case 2 : {
                if (currok &= lex.CurrVar(turns)) {
                    lex.GetToken();
                    state++;
                }
                else {
                    currok = false;
                    state=6;
                }
            }
            break;
            case 3 : {
                if (currok &= lex.CurrVar(firstTerm)) {
                    lex.GetToken();
                    state++;
                }
                else {
                    currok = false;
                    state=6;
                }
            }
            break;
            case 4 : {
                if (currok &= lex.CurrVar(secTerm)) {
                    lex.GetToken();
                    state++;
                }
                else {
                    currok = false;
                    state=6;
                }
            }
            break;
            case 5 : {
                if (currok) {
                    Coil* coil = new Coil(frame, nameId);
                    if (frame->Add(coil)) {
                        frame->AddCoilList(coil);
                        coil->SetFirst(firstTerm);
                        coil->SetSecond(secTerm);
                        coil->SetTurns(turns);
                        if (inform) {
                            cout << "TERMINAL: " << coil->GetName() << " #" << frame-
>GetCoils().size() << " added" << endl;
                        }
                    }
                    else {
                        cerr << "ERROR (coil: " << coil->GetName() << ") duplicate" <<
endl;
                        ok = false;
                        delete coil;
                    }
                }
                else {
                    cerr << "ERROR (line: " << lex.LineCount() << " coil: " << nameId
                         << ") syntax '{ <coil_name> <terminal_name> <terminal_name> }'
expected" << endl;
                    ok = false;
                }
                state++;
            }
            break;
            case 6 : { // check for closing bracket
                if (lex.CurrStr()=="}") {
                    lex.GetToken();  // new token
                    state++;
                }
                else {
                    state = 1; // get unknown
                }
            }
            break;
            default : {
                currok = false;
                cerr << "Coil parser illegal state" << endl;
                state = 6; // get unknown
            }
        }
        ok &= currok;
/*        if (inform) {
          cout << "Coil parse state: " << state << endl;
        } */
    } while (lex.CurrToken()!=LexGDL::END && state<7);
    if (lex.PrevStr()!="}") {
        cerr << "ERROR (line: " << lex.LineCount()-1 << ") '}' expected" << endl;
        lex.GetToken();  // set up for parse
    }
    return ok;
}
```

```
#if !defined(coord_h)              // Sentry, use file only if it's not already included.
#define coord_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         coord.h
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Class definitions for cooordinate.
*/

#include <string>

#include "unique.h"
#include "vector2d.h"
#include "descobj.h"

class Description;
class Frame;

// Coordinate
class Coord : public DescriptionObject
{
private:
    Frame* frame;
    Vector2d pos;
public:
    Coord();
    Coord(Frame* f, const string& n);
    Coord(Frame* f, const char* n = "unknown");
    ~Coord();

    Vector2d& SetPos(Vector2d&);
    Vector2d& GetPos();
    Frame* GetFrame();
    Description* GetDescription();

    void WriteProperties(ofstream&);
    static bool Parse(LexGDL&, Frame*, bool inform=true);
};

typedef Unique<Coord *> Coords;
typedef Unique<Coord *>::iterator CoordsIterator;

void CoordsWrite(ofstream&, Coords&);
#endif                                        // coord_h sentry.
```

```
/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         coord.cpp
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Source file for implementation of coordinate.
*/

#include <cmath>
#include <strstream>

#include "coord.h"
#include "frame.h"
#include "desc.h"

void CoordsWrite(ofstream& out, Coords& coords)
{
    out << "nodes { ";
    CoordsIterator iter=coords.begin();
    while (iter != coords.end()) {
        (*iter)->WriteProperties(out);
        out << " ";
        iter++;
    }
    out << "}";
}

void Coord::WriteProperties(ofstream& out)
{
    out << GetName() << " ";
    out << (Mag(pos)/GetDescription()->GetUnitDistance()) << " ";
    out << Angle(pos);
}

Coord::Coord(Frame* f, const string& n):DescriptionObject(n)
{
    frame=f;
}

Coord::Coord(Frame* f, const char* n):DescriptionObject(n)
{
    frame=f;
}

Coord::~Coord()
{
}

Frame* Coord::GetFrame()
{
    return frame;
}

Description* Coord::GetDescription()
{
    return (frame?frame->GetDescription():NULL);
}

Vector2d& Coord::SetPos(Vector2d& p)
{
    return (pos=p);
}

Vector2d& Coord::GetPos()
{
    return pos;
}

bool Coord::Parse(LexGDL& lex, Frame* frame, bool inform) // expects to find new token
already current
```

```
{
    string nameId;
    double p, q;
    bool degrees = false;
    long start, rep, inc;
    double startp, incp;
    double startq, incq;
    bool pattern = false;
    bool currok = true;
    bool ok = true;
    int state = 0;
    do {
        switch (state) {
            case 0 : {
                if (lex.CurrStr()=="{") {
                    lex.GetToken();  // set up for parse
                    state++;
                }
                else {
                    cerr << "ERROR (line: " << lex.LineCount() << ") '{' expected" <<
endl;
                    state = 10; // get out of parser
                    currok = false;
                }
            }
            break;
            case 1 : {
                if (lex.CurrToken()==LexGDL::UNKNOWN) {
                    nameId = lex.CurrStr();
                    start = inc = 0;
                    rep = 1; // by default, if a plain name, then repeat just once
                    startp = incp = 0;
                    startq = incq = 0;
                    p = q = 0;
                    currok = true; // reset ok flag as new coord
                    lex.GetToken();
                    state++;
                }
                else {
                    currok = false;
                    lex.GetToken();
                    state = 9; // find unknown
                }
            }
            break;
            case 2 : {
                if (lex.CurrStr()=="[") {
                    currok &= lex.ParseBracket(start, rep, inc);
                    pattern = true;
                    state = 4;
                }
                else {
                    pattern = false;
                    state = 3;
                }
            }
            break;
            case 3 : {
                if (currok &= lex.CurrVar(p)) {
                    lex.GetToken();
                    state=5;
                }
                else {
                    currok = false;
                    state=9;
                }
            }
            break;
            case 4 : {
                if (currok &= lex.ParseBracket(startp, incp)) {
                    state++;
                }
                else {
                    currok = false;
                    state=9;
                }
            }
            break;
            case 5 : {
                if (degrees = (lex.CurrToken()==LexGDL::DEG)) lex.GetToken();
                state = (pattern?7:6);
            }
            break;
            case 6 : {
                if (currok &= lex.CurrVar(q)) {
                    lex.GetToken();
                    state=8;
                }
                else {
                    currok = false;
                    state=9;
                }
            }
            break;
            case 7 : {
                if (currok &= lex.ParseBracket(startq, incq)) {
                    state++;
                }
                else {
                    currok = false;
                    state=9;
                }
            }
            break;
            case 8 : {
                if (currok) {
                    long curr = start;
                    long c = 0;
                    double currp = startp;
                    double currq = startq;
                    Vector2d pos;
                    Coord* coord;
                    while (c<rep) {
                        if (pattern) {
                            strstream number;
                            number << curr; // change integer to char string
                            coord = new Coord(frame, nameId + number.str());
                            p = currp;
                            q = currq;
                        }
                        else {
                            coord = new Coord(frame, nameId);
                        }
                        if (frame->Add(coord)) {
                            if (degrees) q*=M_PI/180;
                            pos(Vector2d::POLAR,p*frame->GetDescription()-
>GetUnitDistance(),q);
                            // cout << "unit distance: " << frame->GetDescription()-
>GetUnitDistance() << "\n";
                            coord->SetPos(pos);
                            if (inform) {
                                cout << "Coord: " << coord->GetName() << " #" << frame-
>GetCoords().size() << " added" << endl;
                            }
                        }
                        else {
                            cerr << "ERROR (coord: " << coord->GetName() << ")
duplicate" << endl;
                            ok = false;
                            delete coord;
                        }
                        currp += incp;
                        currq += incq;
                        curr += inc;
                        c++;
                    }
                }
                else {
                    cerr << "ERROR (line: " << lex.LineCount() << " coord: "
                        << nameId << ") syntax '<name|[int int int]> <float|[float
float]> [d] <float|[float float]>' expected" << endl;
```

```
                    ok = false;
                }
                state++;
            }
            break;
            case 9 : {
                if (lex.CurrStr()=="}") { // check for closing bracket
                    lex.GetToken();
                    state++;
                }
                else {
                    state = 1; // get unknown
                }
            }
            break;
            default : {
                currok = false;
                cerr << "Coord parser illegal state" << endl;
                state = 9; // get unknown
            }
        }
        ok &= currok;
/*          if (inform) {
            cout << "Coord parse state: " << state << endl;
        } */
    } while (lex.CurrToken()!=LexGDL::END && state<10);
    if (lex.PrevStr()!="}") {
        cerr << "ERROR (line: " << lex.LineCount()-1 << ") '}' expected" << endl;
    }
    return ok;
}
```

```
#if !defined(curve_h)              // Sentry, use file only if it's not already included.
#define curve_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:     gdl.exe Application
    FILE:          curve.h
    AUTHOR:        D. Downes


    OVERVIEW
    ========
    Class definitions for curve.
*/

#include <string>

#include "unique.h"
#include "vector2d.h"
#include "descobj.h"

class Description;
class Frame;
class Coord;

// Curve
class Curve : public DescriptionObject
{
public:
    enum CurveType {UNKNOWN, LINE, CIRC};
private:
    CurveType type;
    Frame* frame;
    Coord* begin;
    Coord* end;
    double radius;
    bool boundary;

public:
    Curve();
    Curve(Frame* f, const string& n);
    Curve(Frame* f, const char* n = "unknown");
    ~Curve();

    CurveType GetType();
    Coord* GetBegin();
    Coord* GetEnd();
    double GetRadius();
    Frame* GetFrame();
    Description* GetDescription();
    CurveType SetType(CurveType);
    Coord* SetBegin(Coord*);
    Coord* SetEnd(Coord*);
    double SetRadius(double);
    bool IsBoundary(bool);
    bool IsBoundary();

    bool Centre(Vector2d&);
    bool GetMinMax(Vector2d&, Vector2d&);
    double Length();
    Coord* OtherIfValid(Coord*);
    Vector2d FractionPos(Coord*, double);

    void WriteProperties(ofstream&);
    static bool Parse(LexGDL&, Frame*, bool inform=true);

    friend Coord* SharedCoord(Curve*, Curve*);
};

typedef Unique<Curve *> Curves;
typedef Unique<Curve *>::iterator CurvesIterator;
typedef Unique<Curve *>::reverse_iterator CurvesReverseIterator;
```

```
void CurvesWrite(ofstream&, Curves&);
void CurvesSetBoundary(Curves& curves);
Coord* SharedCoord(Curve*, Curve*);
#endif                                          // curve_h sentry.
```

```
/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         curve.cpp
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Source file for implementation of curve.
*/

#include <strstream>

#include "coord.h"
#include "frame.h"
#include "curve.h"
#include "desc.h"

void CurvesWrite(ofstream& out, Curves& curves)
{
    out << "curves { ";
    CurvesIterator iter=curves.begin();
    while (iter != curves.end()) {
        (*iter)->WriteProperties(out);
        out << " ";
        iter++;
    }
    out << "}";
}

void CurvesSetBoundary(Curves& curves)
{
    CurvesIterator iter=curves.begin();
    while (iter != curves.end()) {
        (*(iter++))->IsBoundary(true);
    }
}

void Curve::WriteProperties(ofstream& out)
{
    out << GetName() << " ";
    if (type==Curve::LINE) out << "line " << begin->GetName() << " " << end->GetName();
    if (type==Curve::CIRC) out << "circ " << begin->GetName() << " " << end->GetName()
                           << " " << (radius/GetDescription()->GetUnitDistance());
}

Curve::Curve(Frame* f, const string& n):DescriptionObject(n)
{
    frame=f;
    begin=NULL;
    end=NULL;
    radius=0;
    boundary=true;
}

Curve::Curve(Frame* f, const char* n):DescriptionObject(n)
{
    frame=f;
    begin=NULL;
    end=NULL;
    radius=0;
    boundary=true;
}

Curve::~Curve()
{
}

Frame* Curve::GetFrame()
{
    return frame;
}
```

```
Description* Curve::GetDescription()
{
    return (frame?frame->GetDescription():NULL);
}

Curve::CurveType Curve::GetType()
{
    return type;
}

Coord* Curve::GetBegin()
{
    return begin;
}

Coord* Curve::GetEnd()
{
    return end;
}

double Curve::GetRadius()
{
    return radius;
}

Curve::CurveType Curve::SetType(Curve::CurveType t)
{
    return (type=t);
}

Coord* Curve::SetBegin(Coord* c)
{
    return (begin=c);
}

Coord* Curve::SetEnd(Coord* c)
{
    return (end=c);
}

double Curve::SetRadius(double r)
{
    return (radius=r);
}

bool Curve::IsBoundary(bool b)
{
    return (boundary=b);
}

bool Curve::IsBoundary()
{
    return (boundary);
}

bool Curve::Centre(Vector2d& g)
{
    double r = fabs(radius);
    Vector2d b, e;
    b=begin->GetPos();
    e=end->GetPos();
    bool ok = true;
    if (type==Curve::CIRC) {
        if (b.X()==e.X() && b.Y()==e.Y()) {
            g=b;
        }
        else {
            Vector2d c;
            c = e - b;
            if (Mag(c)>2*r)  {
                cerr << "ERROR (curve: " << nameId << ") Curve radius " << radius
                     << " specified is too small for nodes: "
                     << b.X() << " , " << b.Y() << " & " << e.X() << " , " << e.Y() <<
endl;
```

```
                ok = false;
            }
            else {
                if (Mag(c)==2*r)
                    g = b + (c/2);
                else
                    g = b+c/2+UnitNorm(c)*sqrt(pow(r,2)-pow(Mag(c)/2,2));
            }
        }
    }
    else {
        ok = false;
    }
    return ok;
}

bool Curve::Parse(LexGDL& lex, Frame* frame, bool inform) // expects to find new token
already current
{
    string nameId;
    string p,q,pname,qname;
    double r = 0.0;
    double startr, incr;
    Curve::CurveType type;
    bool circ = false;
    long start, rep, inc;
    long startp, incp;
    long startq, incq;
    bool pattern = false;
    bool currok = true;
    bool ok = true;
    int state = 0;
    do {
        switch (state) {
            case 0 : {
                if (lex.CurrStr()=="{") {
                    lex.GetToken();  // set up for parse
                    state++;
                }
                else {
                    cerr << "ERROR (line: " << lex.LineCount() << ") '{' expected" <<
endl;
                    state = 12; // get ot of parser
                    currok = false;
                }
            }
            break;
            case 1 : {
                if (lex.CurrToken()==LexGDL::UNKNOWN) {
                    nameId = lex.CurrStr();
                    start = inc = 0;
                    rep = 1; // by default if parsed a plain name then repeat just once
                    startp = incp = 0;
                    startq = incq = 0;
                    currok = true; // reset ok flag as curve bh
                    lex.GetToken();
                    state++;
                }
                else {
                    currok = false;
                    lex.GetToken();
                    state = 11; // find unknown
                }
            }
            break;
            case 2 : {
                if (lex.CurrStr()=="[") {
                    currok &= lex.ParseBracket(start, rep, inc);
                    pattern = true;
                    state++;
                }
                else {
                    pattern = false;
                    state++;
                }
            }
            break;
            case 3 : {
                switch (lex.CurrToken()) {
                    case LexGDL::LINE : type=Curve::LINE; circ=false; lex.GetToken();
state++; break;
                    case LexGDL::CIRC : type=Curve::CIRC; circ=true; lex.GetToken();
state++; break;
                    default : currok = false;  state = 1; // find next unknown
                }
            }
            break;
            case 4 : {
                if (currok &= lex.CurrVar(p)) {
                    lex.GetToken();
                    state = (pattern?5:6);
                }
                else {
                    currok = false;
                    state=11;
                }
            }
            break;
            case 5 : {
                if (currok &= lex.ParseBracket(startp, incp)) {
                    state++;
                }
                else {
                    currok = false;
                    state=11;
                }
            }
            break;
            case 6 : {
                if (currok &= lex.CurrVar(q)) {
                    lex.GetToken();
                    state = (pattern?7:(circ?8:10));
                }
                else {
                    currok = false;
                    state=11;
                }
            }
            break;
            case 7 : {
                if (currok &= lex.ParseBracket(startq, incq)) {
                    state=(circ?9:10);
                }
                else {
                    currok = false;
                    state=11;
                }
            }
            break;
            case 8 : {
                if (currok &= lex.CurrVar(r)) {
                    lex.GetToken();
                    state=10;
                }
                else {
                    currok = false;
                    state=11;
                }
            }
            break;
            case 9 : {
                if (currok &= lex.ParseBracket(startr, incr)) {
                    state=10;
                }
                else {
                    currok = false;
                    state=11;
                }
            }
            break;
            case 10 : {
                if (currok) {

                    long curr = start;

                    long c = 0;
                    long currp = startp;
                    long currq = startq;
                    double currr = startr;
                    Curve* curve;
                    while (c<rep) {
                        if (pattern) {
                            strstream number;
                            strstream numberp;
                            strstream numberq;
                            number << curr; // change integer to char string
                            curve = new Curve(frame, nameId + number.str());
                            numberp << currp;
                            pname = p + numberp.str();
                            numberq << currq;
                            qname = q + numberq.str();
                            r = currr;
                            // reset strstream !!!
                        }
                        else {
                            curve = new Curve(frame, nameId);
                            pname = p;
                            qname = q;
                        }
                        if (!(curve->SetBegin(frame->Find(&Coord(frame,pname))))) {
                            cerr << "ERROR (curve: " << curve->GetName() << ") node: "
<< pname << " doesn't exist" << endl;
                            currok = false;
                        }

                        if (!(curve->SetEnd(frame->Find(&Coord(frame,qname))))) {
                            cerr << "ERROR (curve: " << curve->GetName() << ") node: "
<< qname << " doesn't exist" << endl;
                            currok = false;
                        }
                        if (currok) {
                            if (frame->Add(curve)) {
                                curve->SetType(type);
                                curve->SetRadius(r*frame->GetDescription()-
>GetUnitDistance());

                                if (inform) {
                                    cout << "Curve: " << curve->GetName() << " #" <<
frame->GetCurves().size() << " added" << endl;
                                }
                            }
                            else {
                                cerr << "ERROR (curve: " << curve->GetName() << ")
duplicate" << endl;

                                ok = false;
                                delete curve;
                            }
                        }
                        currp += incp;
                        currq += incq;
                        currr += incr;
                        curr += inc;
                        c++;
                    }
                }
                else {
                    cerr << "ERROR (line: " << lex.LineCount() << " coord: "  << nameId
                         << ") syntax '<name|[int int int]> <line|circ> <name|[int int]>
<name|[int int]> <float|[float float]>' expected" << endl;
                    ok = false;
                }
                state++;
            }
            break;
            case 11 : {
                if (lex.CurrStr()=="}") { // check for closing bracket
```

```
                    lex.GetToken();
                    state++;
                }
                else {
                    state = 1; // get unknown
                }
            }
            break;
            default : {
                currok = false;
                cerr << "Curve parser illegal state" << endl;
                state = 11; // get unknown
            }
        }
        ok &= currok;
/*        if (inform) {
          cout << "Curve parse state: " << state << endl;
        } */
    } while (lex.CurrToken()!=LexGDL::END && state<12);
    if (lex.PrevStr()!="}") {
        cerr << "ERROR (line: " << lex.LineCount()-1 << ") '}' expected" << endl;
    }
    return ok;
}

bool Curve::GetMinMax(Vector2d& cmin, Vector2d& cmax)
{
    if (begin!=NULL && end!=NULL)  {
        Vector2d b(begin->GetPos());
        Vector2d e(end->GetPos());
        if (type==Curve::CIRC) {
            Vector2d r(radius,radius);
            Vector2d origin;
            if (Centre(origin)) {
                b=origin+r;
                e=origin-r;
            }
            else {
                b+=r;
                e-=r;
            }
        }
        cmin((b.X()<e.X()?b.X():e.X()),(b.Y()<e.Y()?b.Y():e.Y()));
        cmax((b.X()>e.X()?b.X():e.X()),(b.Y()>e.Y()?b.Y():e.Y()));
        return true;
    }
    else {
        return false;
    }
}

double Curve::Length()
{
    double length = 0.0;
    if (type==Curve::LINE)
        length = fabs(Mag(end->GetPos()-begin->GetPos()));
    if (type==Curve::CIRC) {
        if (begin->GetPos()==end->GetPos())
            length = fabs(2*M_PI*radius);
        else
            length = fabs(2*radius*asin(Mag(end->GetPos()-begin->GetPos())/(2*radius)));
//        length = fabs(Mag(end->GetPos()-begin->GetPos()));
    }
    return length;
}

Coord* Curve::OtherIfValid(Coord* coord)
{
    Coord* c=NULL;
    if (coord==begin) c=end;
    if (coord==end) c=begin;
    return c;
}

Vector2d Curve::FractionPos(Coord* coord, double fraction)
```

```
{
    Vector2d pos;
    if (begin && end && begin!=end && (coord==begin || coord==end)) {
        Coord* start=coord;
        Coord* finish=(coord==begin?end:begin);
        if (type==Curve::LINE) {
            pos=(finish->GetPos()-start->GetPos())*fraction+start->GetPos();
        }
        if (type==Curve::CIRC) {
            Vector2d origin;
            if (Centre(origin)) {
                double newAngle = 2*asin(Mag(finish->GetPos()-start-
>GetPos())/(2*radius))*fraction;
                Vector2d pos_origin;
                if (start==begin) {
                    pos_origin(Vector2d::POLAR,radius,Angle(start->GetPos()-
origin)+newAngle);
                }
                else {
                    pos_origin(Vector2d::POLAR,radius,Angle(start->GetPos()-origin)-
newAngle);
                }
                pos=origin+pos_origin;
            }
            else {
                // use an aproximation of straight line if origin calc. failed
                pos=(finish->GetPos()-start->GetPos())*fraction+start->GetPos();
            }
        }
    }
    return pos;
}

Coord* SharedCoord(Curve* c1, Curve* c2)
{
    Coord* c=NULL;
    if (c1->begin==c2->begin || c1->begin==c2->end) c=c1->begin;
    if (c1->end==c2->begin || c1->end==c2->end) c=c1->end;
    return c;
}
```

```
#if !defined(objdesc_h)                 // Sentry, use file only if it's not already
included.
#define objdesc_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         descobj.h
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Class definitions for description type objects.
*/

#include <string>

#include "lex.h"

// descobject
class DescriptionObject
{
protected:
    string nameId;
public:
    DescriptionObject() {nameId="unknown";}
    DescriptionObject(const string& n) {nameId=n;}
    DescriptionObject(const char* n) {nameId=n;}
    virtual ~DescriptionObject() {}

    const string GetName() {return nameId;}
    virtual int operator==(const DescriptionObject& other) const {return
other.nameId==nameId;}
//    virtual unsigned HashValue() const {return nameId.hash();}

    friend ostream& operator << (ostream& os, const DescriptionObject obj)
        {return os << "No proper stream function for " << obj.nameId;}
};

#endif                                          // objdesc_h sentry.
```

```
#if !defined(desc_h)              // Sentry, use file only if it's not already included.
#define desc_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         desc.h
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Class definitions for description.
*/

#include <string>

#include "descobj.h"
#include "lex.h"
#include "vector2d.h"
#include "bh.h"
#include "frame.h"

// Description
class Description : public DescriptionObject
{
private:
    Frame stator;
    Frame rotor;
    Bhs bhs;
    long cctNodeCount;
    double unitDistance;
    double torqueModifier;
    double airgap;

public:
    Description(const string& n);
    Description(const char* n = "unknown");
    ~Description();

    Frame& GetStator();
    Frame& GetRotor();
    Bhs& GetBhs();
    long GetCctNode();
    long SetCctNode(const long& cctNodeCount=1);
    Bh* Find(Bh*);
    bool Add(Bh*);
    double GetUnitDistance();
    double GetTorqueModifier();
    double SetUnitDistance(double);
    double SetTorqueModifier(double);

    bool Convert(ofstream&, ofstream&, const char *name, const char *prefix, long
sections, bool verbose, bool torq);
    void Write(ofstream&);
    bool Parse(ifstream*, bool inform=true);
    void Check(long);
    void Info();
};

#endif                                        // desc_h sentry.
```

```
/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         desc.cpp
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Source file for implementation of description.
*/

#include "desc.h"

Description::Description( const string& n):stator(this, "stator"),rotor(this,"rotor"),
DescriptionObject(n)
{
    cctNodeCount=1; // cct node count set to 1
    unitDistance=1;
    torqueModifier=1;
    airgap=1;
}

Description::Description( const char* n):stator(this, "stator"),rotor(this,"rotor"),
DescriptionObject(n)
{
    cctNodeCount=1; // cct node count set to 1
    unitDistance=1;
    torqueModifier=1;
    airgap=1;
}

Description::~Description()
{
//    bhs.Flush(true);
}

double Description::GetUnitDistance()
{
    return (unitDistance);
}

double Description::GetTorqueModifier()
{
    return (torqueModifier);
}

double Description::SetUnitDistance(double d)
{
    return (unitDistance=d);
}

double Description::SetTorqueModifier(double m)
{
    return (torqueModifier=m);
}

long Description::GetCctNode()
{
    return (cctNodeCount++);
}

long Description::SetCctNode(const long& cctNode)
{
    return (cctNodeCount=cctNode);
}

Frame& Description::GetStator()
{
    return stator;
}

Frame& Description::GetRotor()
```

```
{
    return rotor;
}

Bhs& Description::GetBhs()
{
    return bhs;
}

Bh* Description::Find(Bh* b)
{
    return bhs.Find(b);
}

bool Description::Add(Bh* b)
{
    return bhs.Add(b);
}

void Description::Write(ofstream& outstream)
{
  outstream << "# file generated by GDL program\n\n";
  outstream << "unitdistance " << unitDistance << "\n";
  outstream << "torquemodifier " << torqueModifier << "\n\n";
  BhsWrite(outstream, bhs);
  outstream << "\n\n";
  FrameWrite(outstream, stator);
  outstream << "\n\n";
  FrameWrite(outstream, rotor);
  outstream << "\n\n";
}

bool Description::Convert(ofstream& cct, ofstream& lst, const char *name, const char
*prefix, long sections, bool verbose, bool torq)
{
    bool  ok;
    SetCctNode(); // set cct node count to initial value
    cct << "* Circuit description generated by GDL program\n";
    if (ok = (stator.GenerateCctNodes(verbose) && rotor.GenerateCctNodes(verbose)
//generate messages = true
          && stator.FindBoundaryCurves() && rotor.FindBoundaryCurves())) {
        // carry on and generate the subcircuit file
        cct << ".subckt magnetic";
        if (!stator.GetCoilList().empty() && !rotor.GetCoilList().empty()) {
            if (torq) {
              cct << " TORQ_P TORQ_N";
            }
            cct << " POS_P POS_N";
        }
        cct << "\n";
        if (!stator.GetCoilList().empty()) {
            cct << "+ ";
            CoilListConvertTerminals(cct, stator.GetCoilList()); // output a list of the
stator circuit terminals
            cct << "\n";
        }
        if (!rotor.GetCoilList().empty()) {
            cct << "+ ";
            CoilListConvertTerminals(cct, rotor.GetCoilList()); // output a list of the
rotor circuit terminals
            cct << "\n";
        }
        cct << "*************************************\n";
        cct << "* Stator magnetic circuit elements *\n";
        cct << "*************************************\n";
        ok &= stator.Convert(cct, sections);
        cct << "************************************\n";
        cct << "* Rotor magnetic circuit elements *\n";
        cct << "************************************\n";
        ok &= rotor.Convert(cct, sections);
        cct << "*****************************************************\n";
        cct << "* Connection of stator coils to circuit terminations *\n";
        cct << "*****************************************************\n";
        CoilsConvertElectrical(cct, stator.GetCoils()); // convert to coil circuit
description
```

```
        cct << "*******************************************************\n";
        cct << "* Connection of rotor coils to circuit terminations *\n";
        cct << "*******************************************************\n";
        CoilsConvertElectrical(cct, rotor.GetCoils()); // convert to coil circuit
description
        cct << "****************************************\n";
        cct << "* Magnetic to mechanical conversion *\n";
        cct << "****************************************\n";
        ok &= FrameGenerateMech(cct, stator, rotor, sections, torq, verbose);
        cct << ".ends\n\n\n\n"; // extra three returns as simetrix seems to need them
        // generate the node list file

        lst << "* ";
        PartsIterator rPartsIter=rotor.GetParts().begin();
        while (rPartsIter != rotor.GetParts().end()) {
            if ((*rPartsIter)->IsMagPart()) {
              lst << (*rPartsIter)->GetName() << " " << (*rPartsIter)->GetMinArea() << "
";
              lst << (*rPartsIter)->GetEntryCctNode() << " " << (*rPartsIter)-
>GetExitCctNode() << " ";
              lst << (*rPartsIter)->GetLineIntegral() << " ";
            }
            rPartsIter++;
        }
        PartsIterator sPartsIter=stator.GetParts().begin();
        while (sPartsIter != stator.GetParts().end()) {
            if ((*sPartsIter)->IsMagPart()) {
              lst << (*sPartsIter)->GetName() << " " << (*sPartsIter)->GetMinArea() << "
";
              lst << (*sPartsIter)->GetEntryCctNode() << " " << (*sPartsIter)-
>GetExitCctNode() << " ";
              lst << (*sPartsIter)->GetLineIntegral() << " ";
            }
            sPartsIter++;
        }
        DiffNodesIterator sDiffNodesIter=stator.GetDiffNodes().begin();
        DiffNodesIterator rDiffNodesIter=rotor.GetDiffNodes().begin();
        lst << "\n* ";
        while (sDiffNodesIter != stator.GetDiffNodes().end()) {
            lst << (*sDiffNodesIter) << " ";
            sDiffNodesIter++;
        }
        while (rDiffNodesIter != rotor.GetDiffNodes().end()) {
            lst << (*rDiffNodesIter) << " ";
            rDiffNodesIter++;
        }
        lst << "\n";
        sDiffNodesIter=stator.GetDiffNodes().begin();
        rDiffNodesIter=rotor.GetDiffNodes().begin();
        lst << "*\n.control\nwrite " << name << " ";
        while (sDiffNodesIter != stator.GetDiffNodes().end()) {
            lst << prefix << (*sDiffNodesIter) << " ";
            sDiffNodesIter++;
        }
        while (rDiffNodesIter != rotor.GetDiffNodes().end()) {
            lst << prefix << (*rDiffNodesIter) << " ";
            rDiffNodesIter++;
        }
        lst << "\n.endc\n";
    }
    else {
        cct << "Error occured when generating circuit nodes and boundary curves\n";
    }
    return ok;
}

bool Description::Parse(ifstream* instream, bool inform)
{
    LexGDL lex(instream);
    lex.GetToken(); // prepare for first token check
    bool ok = true;
    do {
        switch (lex.CurrToken()) {
            case LexGDL::UNIT :
                {     lex.GetToken();  // get new token
```

```
                    if (!lex.CurrVar(unitDistance)) {
                        cerr << "ERROR (line: " << lex.LineCount() << ") value for unit
distance expected" << endl;
                        ok = false;
                    }
                    else {
                        lex.GetToken();  // get new token
                    }
                }
                break;
            case LexGDL::AIRGAP :
                {    lex.GetToken();  // get new token
                    if (!lex.CurrVar(airgap)) {
                        cerr << "ERROR (line: " << lex.LineCount() << ") value for
airgap distance expected" << endl;
                        ok = false;
                    }
                    else {
                        lex.GetToken();  // get new token
                    }
                }
                break;
            case LexGDL::TORQUE :
                {    lex.GetToken();  // get new token
                    if (!lex.CurrVar(torqueModifier)) {
                        cerr << "ERROR (line: " << lex.LineCount() << ") value for
torque modifier expected" << endl;
                        ok = false;
                    }
                    else {
                        lex.GetToken();  // get new token
                    }
                }
                break;
            case LexGDL::STATOR : {
                cout << "found STATOR" << endl;
                lex.GetToken(); // get new token
                ok  &= stator.Parse(lex,inform); // expects a new token and returns a
new token
            }
            break;
            case LexGDL::ROTOR : {
                cout << "found ROTOR" << endl;
                lex.GetToken(); // get new token
                ok &= rotor.Parse(lex,inform); // expects a new token and returns a new
token
            }
            break;
            case LexGDL::BH : {
                lex.GetToken();  // get new token
                ok &= Bh::Parse(lex, this, inform);
            }
            break;
            default : {
                cerr << "ERROR (line: " << lex.LineCount() << ") '" << lex.CurrStr() <<
"' not valid" << endl;
                lex.GetToken(); // get a new token
                ok = false;
            }
        }
    } while (lex.CurrToken()!=LexGDL::END && !instream->eof());
    cout << "PARSER SUMMARY:" << endl;
    cout << "bh: " << bhs.size() << " created" << endl;
    cout << "Stator nodes: " << stator.GetCoords().size() << " created" << endl;
    cout << "Stator curves: " << stator.GetCurves().size() << " created" << endl;
    cout << "stator parts: " << stator.GetParts().size() << " created" << endl;
    cout << "stator coils: " << stator.GetCoils().size() << " created" << endl;
    cout << "rotor nodes: " << rotor.GetCoords().size() << " created" << endl;
    cout << "rotor curves: " << rotor.GetCurves().size() << " created" << endl;
    cout << "rotor parts: " << rotor.GetParts().size() << " created" << endl;
    cout << "rotor coils: " << rotor.GetCoils().size() << " created" << endl;
    return ok;
}

void Description::Check(long sections)
```

```
{
    SetCctNode(); // set cct node count to initial value

    if (!stator.FindBoundaryCurves()) {
        cerr << "Error occured while finding stator boundary curves" << endl;
    }
    if (!rotor.FindBoundaryCurves()) {
        cerr << "Error occured while finding rotor boundary curves" << endl;
    }
    if (!stator.GenerateCctNodes(true)) {
        cerr << "Error occured while generating stator nodes" << endl;
    }
    if (!rotor.GenerateCctNodes(true)) {
        cerr << "Error occured while generating rotor nodes" << endl;
    }
    cout << "Stator check:" << endl;
    stator.Check(sections);
    cout  << "Rotor check:" << endl;
    rotor.Check(sections);
}

void Description::Info()
{
    cout << "summary:" << endl;
    cout << "bh: " << bhs.size() << " created" << endl;
    cout << "Stator nodes: " << stator.GetCoords().size() << " created" << endl;
    cout << "Stator curves: " << stator.GetCurves().size() << " created" << endl;
    cout << "stator parts: " << stator.GetParts().size() << " created" << endl;
    cout << "stator coils: " << stator.GetCoils().size() << " created" << endl;
    cout << "rotor nodes: " << rotor.GetCoords().size() << " created" << endl;
    cout << "rotor curves: " << rotor.GetCurves().size() << " created" << endl;
    cout << "rotor parts: " << rotor.GetParts().size() << " created" << endl;
    cout << "rotor coils: " << rotor.GetCoils().size() << " created" << endl;
}
```

```
#if !defined(frame_h)              // Sentry, use file only if it's not already included.
#define frame_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         frame.h
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Class definitions for frame.
*/

#include <string>

#include "unique.h"
#include "descobj.h"
#include "curve.h"
#include "coord.h"
#include "part.h"
#include "coil.h"

class Description;

typedef list<long> CctNodes;
typedef list<long>::iterator CctNodesIterator;
typedef list<string> DiffNodes;
typedef list<string>::iterator DiffNodesIterator;

// Frame
class Frame : public DescriptionObject
{
private:
    Description* description;
    Parts parts;
    Curves curves;
    Coords coords;
    Coils coils;
    Coils coilList;
    double skew;
    double length;
    CctNodes cctNodes;
    DiffNodes diffNodes;

public:
    Frame(Description* d, const string& n);
    Frame(Description* d, const char* n = "unknown");
    ~Frame();

    Parts& GetParts();
    Curves& GetCurves();
    Coords& GetCoords();
    Description* GetDescription();
    Part* Find(Part*);
    Curve* Find(Curve*);
    Coord* Find(Coord*);
    bool Add(Part*);
    bool Add(Curve*);
    bool Add(Coord*);
    Coils& GetCoils();
    Coil* Find(Coil*);
    bool Add(Coil*);
    Coils& GetCoilList();
    bool AddCoilList(Coil*);
    double GetSkew();
    double SetSkew(double);
    double GetLength();
    double SetLength(double);
    bool AddCctNode(long);
    CctNodes& GetCctNodes();
    bool AddDiffNode(string);
    DiffNodes& GetDiffNodes();

    bool CurvesMinMax(Vector2d&, Vector2d&);
    void Info();
    void Check(long);
    void WriteProperties(ofstream&);
    bool Parse(LexGDL&, bool inform=true);
    bool Convert(ofstream&, long);
    bool GenerateCctNodes(bool messages = true);
    bool FindBoundaryCurves();
};

void FrameWrite(ofstream&, Frame&);
bool FrameGenerateMech(ofstream&, Frame&, Frame&, long sections, bool torq, bool
messages=true);
#endif                                      // frame_h sentry.
```

```
/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         frame.cpp
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Source file for implementation of frame.
*/

const int MIN_COEFF_POW = 0 ; // position curve fit max. values
const int MAX_COEFF_POW = 12;
const float TOP_LIMIT = 0.2;
const float BOT_LIMIT = 0.8;

const int ARC_STEPS = 100; // number of steps for each arc

const float TYPICAL_LENGTH=0.1; // typical length of frame, inherited by parts

#include <strstream>
#include <algorithm>

#include "frame.h"
#include "desc.h"
#include "part.h"

// given a value pos, which is a fraction of the total tooth pithch, calculate the high
boundary value
float calcHi(float skew, float length, float radius, float angle, float pos)
{
  float x=pos*(radius*angle+fabs(tan(skew)*length));
  float y=fabs(x/tan(skew));
  return (y>length?length:y);
}

// given a value pos, which is a fraction of the total tooth pithch, calculate the low
boundary value
float calcLo(float skew, float length, float radius, float angle, float pos)
{
  float x=(1-pos)*(radius*angle+fabs(tan(skew)*length));
  float y=length-fabs(x/tan(skew));
  return (y<0?0:y);
}

double triangleArea(double x1, double y1, double x2, double y2, double x3, double y3) {
  double a=sqrt(pow(x1-x2,2)+pow(y1-y2,2));
  double b=sqrt(pow(x2-x3,2)+pow(y2-y3,2));
  double c=sqrt(pow(x3-x1,2)+pow(y3-y1,2));
  double s=(a+b+c)/2.0;
  return sqrt(s*(s-a)*(s-b)*(s-c));
}

void FrameWrite(ofstream& out, Frame& frame)
{
    frame.WriteProperties(out);
}

bool FrameGenerateMech(ofstream& cct, Frame& fFrame, Frame& sFrame, long sections, bool
torq, bool messages)
{
    double diff = 1e-3;
    bool ok=true;
    bool needsEndSource = false;
    PartsIterator fPIter=fFrame.GetParts().begin();
    PartsIterator sPIter=sFrame.GetParts().begin();
    string cctTorqueP = "TORQ_P"; // needs initialising for the first pass
    string cctTorqueN;
    while (fPIter != fFrame.GetParts().end()) {
        sPIter=sFrame.GetParts().begin();
        while (sPIter != sFrame.GetParts().end()) {
            Part* fPart=(*fPIter);
            Part* sPart=(*sPIter);
            long fNode;
            long sNode;
            Curves* fCurves=NULL;
            Curves* sCurves=NULL;
            if (fPart->IsEntryBoundary()) {
                fNode=fPart->GetEntryCctNode();
                fCurves = &(fPart->GetEntry());
            }
            if (fPart->IsExitBoundary()) {
                fNode=fPart->GetExitCctNode();
                fCurves = &(fPart->GetExit());
            }
            if (sPart->IsEntryBoundary()) {
                sNode=sPart->GetEntryCctNode();
                sCurves = &(sPart->GetEntry());
            }
            if (sPart->IsExitBoundary()) {
                sNode=sPart->GetExitCctNode();
                sCurves = &(sPart->GetExit());
            }
            if (fCurves && sCurves) {
                cct << "* frame: " << fFrame.GetName() << " part: " << fPart->GetName()
<< " cct node: " << fNode
                    << " frame: " << sFrame.GetName() << " part: " << sPart->GetName()
<< " cct node: " << sNode << " ";
                    CurvesIterator fCurvesIter=fCurves->begin();
                    CurvesIterator sCurvesIter=sCurves->begin();
                    CurvesIterator fCurvesIter2=fCurves->begin();
                    CurvesIterator sCurvesIter2=sCurves->begin();
                    // find the extent of the boundary and its centre
                    Coord* fBegin = (*fCurvesIter)->GetBegin();
                    Coord* fEnd = (*fCurvesIter)->GetEnd();
                    Coord* sBegin = (*sCurvesIter)->GetBegin();
                    Coord* sEnd = (*sCurvesIter)->GetEnd();
                    double fCurveAngle = 0;
                    double fAvgBoundaryMag = 0;
                    if (fCurves->size()>1) {
                        // find begin and end curves
                        double dist = Mag(fEnd->GetPos()-fBegin->GetPos());
                        double avgTotal = 0;
                        double avgCount = 0;
                        while (fCurvesIter!=fCurves->end()) {
                            Curve* fCurve=(*fCurvesIter);
                            if  (fCurve->IsBoundary()) {
                                double fAngle = fabs(Angle(fCurve->GetEnd()->GetPos())-
Angle(fCurve->GetBegin()->GetPos()));
                                fCurveAngle += (fAngle>M_PI)?(2*M_PI-fAngle):fAngle; //
add all the curve angles to give the total boundary extent
                                avgTotal += (Mag(fCurve->GetEnd()->GetPos())+Mag(fCurve-
>GetBegin()->GetPos()))/2;
                                avgCount++;
                                fCurvesIter2=fCurves->begin();
                                while (fCurvesIter2!=fCurves->end()) {
                                    Curve* fCurve2=(*fCurvesIter2);
                                    if (fCurve!=fCurve2 && fCurve2->IsBoundary()) {
                                        if (Mag((fCurve->GetEnd()->GetPos())-(fCurve2-
>GetBegin()->GetPos()))>dist) {
                                            fEnd = fCurve->GetEnd();
                                            fBegin = fCurve2->GetBegin();
                                            dist = Mag(fEnd->GetPos()-fBegin->GetPos());
                                        }
                                        if (Mag((fCurve->GetEnd()->GetPos())-(fCurve2-
>GetEnd()->GetPos()))>dist) {
                                            fEnd = fCurve->GetEnd();
                                            fBegin = fCurve2->GetEnd();
                                            dist = Mag(fEnd->GetPos()-fBegin->GetPos());
                                        }
                                        if (Mag((fCurve->GetBegin()->GetPos())-(fCurve2-
>GetEnd()->GetPos()))>dist) {
                                            fEnd = fCurve->GetBegin();
                                            fBegin = fCurve2->GetEnd();
                                            dist = Mag(fEnd->GetPos()-fBegin->GetPos());
                                        }
                                        if (Mag((fCurve->GetBegin()->GetPos())-(fCurve2-
>GetBegin()->GetPos()))>dist) {
```

```
                                        fEnd = fCurve->GetBegin();
                                        fBegin = fCurve2->GetBegin();
                                        dist = Mag(fEnd->GetPos()-fBegin->GetPos());
                                    }
                                }
                                fCurvesIter2++;
                            }
                        }
                        fCurvesIter++;
                    }
                    if (avgCount) fAvgBoundaryMag = avgTotal/avgCount;
                }
                else {
                    double fAngle = fabs(Angle(fEnd->GetPos())-Angle(fBegin-
>GetPos()));
                    fCurveAngle = (fAngle>M_PI)?(2*M_PI-fAngle):fAngle;
                    fAvgBoundaryMag = (Mag(fEnd->GetPos())+Mag(fBegin->GetPos()))/2;
                }
                double sCurveAngle = 0;
                double sAvgBoundaryMag = 0;
                if (sCurves->size()>1) {
                    // find begin and end curves
                    double dist = Mag(sEnd->GetPos()-sBegin->GetPos());
                    double avgTotal = 0;
                    double avgCount = 0;
                    while (sCurvesIter!=sCurves->end()) {
                        Curve* sCurve=(*sCurvesIter);
                        if  (sCurve->IsBoundary()) {
                            double sAngle = fabs(Angle(sCurve->GetEnd()->GetPos())-
Angle(sCurve->GetBegin()->GetPos()));
                            sCurveAngle += (sAngle>M_PI)?(2*M_PI-sAngle):sAngle;
                            avgTotal += (Mag(sCurve->GetEnd()->GetPos())+Mag(sCurve-
>GetBegin()->GetPos()))/2;
                            avgCount++;
                            sCurvesIter2=sCurves->begin();
                            while (sCurvesIter2!=sCurves->end()) {
                                Curve* sCurve2=(*sCurvesIter2);
                                if (sCurve!=sCurve2 && sCurve2->IsBoundary()) {
                                    if (Mag((sCurve->GetEnd()->GetPos())-(sCurve2-
>GetBegin()->GetPos()))>dist) {
                                        sEnd = sCurve->GetEnd();
                                        sBegin = sCurve2->GetBegin();
                                        dist = Mag(sEnd->GetPos()-sBegin->GetPos());
                                    }
                                    if (Mag((sCurve->GetEnd()->GetPos())-(sCurve2-
>GetEnd()->GetPos()))>dist) {
                                        sEnd = sCurve->GetEnd();
                                        sBegin = sCurve2->GetEnd();
                                        dist = Mag(sEnd->GetPos()-sBegin->GetPos());
                                    }
                                    if (Mag((sCurve->GetBegin()->GetPos())-(sCurve2-
>GetEnd()->GetPos()))>dist) {
                                        sEnd = sCurve->GetBegin();
                                        sBegin = sCurve2->GetEnd();
                                        dist = Mag(sEnd->GetPos()-sBegin->GetPos());
                                    }
                                    if (Mag((sCurve->GetBegin()->GetPos())-(sCurve2-
>GetBegin()->GetPos()))>dist) {
                                        sEnd = sCurve->GetBegin();
                                        sBegin = sCurve2->GetBegin();
                                        dist = Mag(sEnd->GetPos()-sBegin->GetPos());
                                    }
                                }
                                sCurvesIter2++;
                            }
                        }
                        sCurvesIter++;
                    }
                    if (avgCount) sAvgBoundaryMag = avgTotal/avgCount;
                }
                else {
                    double sAngle = fabs(Angle(sEnd->GetPos())-Angle(sBegin-
>GetPos()));
                    sCurveAngle = (sAngle>M_PI)?(2*M_PI-sAngle):sAngle;
                    sAvgBoundaryMag = (Mag(sEnd->GetPos())+Mag(sBegin->GetPos()))/2;
                    }
                    double l=(fPart->GetLength()<sPart->GetLength())?fPart-
>GetLength():sPart->GetLength();
                    double fSkew = (tan(fFrame.GetSkew())*l)/fAvgBoundaryMag;
                    double sSkew = (tan(sFrame.GetSkew())*l)/sAvgBoundaryMag;
                    double fCurveCentre = Angle(((fEnd->GetPos()-fBegin-
>GetPos())/2.0)+fBegin->GetPos())+fSkew/2;
                    double sCurveCentre = Angle(((sEnd->GetPos()-sBegin-
>GetPos())/2.0)+sBegin->GetPos())+sSkew/2;
                    // angle of one curve relative to the other for position of
permeance change
                    double coeff_b = fCurveCentre-sCurveCentre;
                    // cout << "Stator: " << fPart->GetName() << " Rotor: " << sPart-
>GetName() << " Coeff_b: "<< coeff_b << endl;
                    double bot = (sCurveAngle+fCurveAngle+fabs(sSkew-fSkew))/2.0; /*
point at which bot_limit% of max.*/
                    double fPoleWidth = fCurveAngle*fAvgBoundaryMag;
                    double fSkewWidth = fSkew*fAvgBoundaryMag;
                    double sPoleWidth = sCurveAngle*sAvgBoundaryMag;
                    double sSkewWidth = sSkew*sAvgBoundaryMag;

                    int sections=100; // number of sections to use in the numerical
aproximation
                    double maxarea=0;
                    double topx=0;
                    double botx=0;
                    double lr=fPoleWidth+fSkewWidth;  // total length of rotor tooth
                    double ls=sPoleWidth+sSkewWidth;  // total length of stator tooth
                    double incx=(lr+ls)/sections; // increment of rotor past stator
//                    for (double x=-ls; x<=((lr-ls)/2.0); x+=incx) {  // loop for the
full motion of the rotor
                    for (double x=-ls; x<=lr; x+=incx) {  // loop for the full motion of
the rotor
                      double area=0; // reset area to zero
                      double xs=x>0?x:0; // find coincident start of rotor and stator
                      double xe=x+ls<lr?x+ls:lr;  // find coincident end of rotor and
stator
                      double incz=(xe-xs)/sections;  // setup increment for area
                      if (incz>0 && incz*10.0>=incx/sections) {  // only perform
calculation if valid coincident area (needs to trap corner case of xe==xs)
                        for (double z=xs; z<=xe; z+=incz) { // integrate over the
coincident area
                            double pr=z/lr; // current point on rotor as ratio of length
                            double ps=xs>0?z/ls:(z-x)/ls; // current point on stator as
ratio of length
                            double rhi=calcHi(fSkew,l,fAvgBoundaryMag,fCurveAngle,pr); //
calculate the current values
                            double rlo=calcLo(fSkew,l,fAvgBoundaryMag,fCurveAngle,pr); //
for the position of the rotor teeth
                            double shi=calcHi(sSkew,l,sAvgBoundaryMag,sCurveAngle,ps);
                            double slo=calcLo(sSkew,l,sAvgBoundaryMag,sCurveAngle,ps);
                            if (rhi>slo && shi>rlo) {  // check teeth are coincident
                              area+=((rhi<shi?rhi:shi)-(rlo>slo?rlo:slo))*incz;  //
approximate coincident area
                            }
                        }
                      }
                      double angle = (x+(ls-
lr)/2.0)/((fAvgBoundaryMag+sAvgBoundaryMag)/2.0);  // position
                      if (fabs(angle)>fabs(bot*BOT_LIMIT) && area>botx) botx=area;
                      if (fabs(angle)>fabs(bot*TOP_LIMIT) && area>topx) topx=area;
                      if (area>maxarea) maxarea=area;
                    }
                    double magCap = 4*M_PI*1E-7*maxarea/fabs(fAvgBoundaryMag-
sAvgBoundaryMag)*2;
                    // find position equation coeff.
                    int coeff_c;
                    double coeff_a;
                    int c = MIN_COEFF_POW;
                    double a = 0.0;
                    double lowest_diff=0;
                    bool first=true;
                    while (c<MAX_COEFF_POW) {
                      c+=2; // must start at 2
                      a=(log(botx/maxarea)/(-1.0*powl(sin(BOT_LIMIT*bot/2.0),
c))+log(topx/maxarea)/(-1.0*powl(sin(TOP_LIMIT*bot/2.0),c)))/2.0;
```

```
                      diff=sqrt(pow(botx/maxarea-exp(-1*a*pow(sin(BOT_LIMIT*bot/2.0),c)),
2)+pow(topx/maxarea-exp(-1*a*pow(sin(TOP_LIMIT*bot/2.0),c)),2));
                      if (first) {
                         first=false;
                        coeff_c=c;
                        coeff_a=a;
                        lowest_diff=diff;
                      } else {
                        if (diff<lowest_diff) {
                          coeff_c=c;
                          coeff_a=a;
                          lowest_diff=diff;
                        }
                      }
                    }
                    if (messages) {
                      cout << "mech: " << fFrame.GetName() << "(" << fPart->GetName() <<
") " << sFrame.GetName() << "(" << sPart->GetName() << ")";
                      cout << " b: " << coeff_b << " c: " << coeff_c << "\n";
                    }
                    cct << " a: " << coeff_a << " b: " << coeff_b << " c: " << coeff_c
<< "\n";
                    // torque capacitor
                    strstream num; // used to convert numeric to string
                    num << fFrame.GetDescription()->GetCctNode(); // needs a new cct node
                    cctTorqueN=num.str();
                    if (torq) {
                      cct << "B2" << fPart->GetName() << sPart->GetName() << " " <<
cctTorqueN << " " << cctTorqueP
                        << " v=V(" << fNode << "," << sNode << ")*V(" << fNode << "," <<
sNode << ")*"<< magCap << "\n";
                      cct  << "+ *( exp(" << -1*coeff_a << "*(sin(V(POS_P,
POS_N)/2)*cos(" << (coeff_b/2+diff)
                        << ")+cos(V(POS_P,POS_N)/2)*sin(" << (coeff_b/2+diff) << "))^"
<< coeff_c << ")\n";
                      cct  << "+ -exp(" << -1*coeff_a << "*(sin(V(POS_P,POS_N)/2)*cos("
<< (coeff_b/2-diff)
                        << ")+cos(V(POS_P,POS_N)/2)*sin(" << (coeff_b/2-diff) << "))^"
<< coeff_c << ") )/"
                        << (4.0*diff) << "\n";
                    }
                    cct << "B1" << fPart->GetName() << sPart->GetName()
                      << " B1" << fPart->GetName() << sPart->GetName() << "_P 0"
                      << " v=V(" << fNode << "," << sNode << ")" << "\n";
                    cct << "+ *" << magCap*1e3 << "*exp(" << -1*coeff_a <<
"*(sin(v(POS_P,POS_N)/2)*" << cos(coeff_b/2) << "+cos(v(POS_P,POS_N)/2)*" <<
sin(coeff_b/2) << ")^2)\n";
                    cct << "C1" << fPart->GetName() << sPart->GetName()
                      << " B1" << fPart->GetName() << sPart->GetName() << "_P  VF1" <<
fPart->GetName() << sPart->GetName() << "_P 1E-3 ic=0\n"; // << magCap << " ic=0\n";
                    cct << "VF1" << fPart->GetName() << sPart->GetName() << "  VF1" <<
fPart->GetName() << sPart->GetName() << "_P  0 0\n";
                    cct << "FF1" << fPart->GetName() << sPart->GetName() << " " << fNode
<< " " << sNode
                      << " VF1" << fPart->GetName() << sPart->GetName() << " 1\n";
                    cctTorqueP=cctTorqueN; // next pass connects to previous
                    needsEndSource = true;
                  }
            sPIter++;
        }
        fPIter++;
    }
    if (needsEndSource) {
        if (torq) {
          cct << "VEND " << cctTorqueP << " TORQ_N 0\n";
        }
        cct << "RPOS POS_P POS_N 1G\n";
    }
    return ok;
}

bool Frame::AddCctNode(long node)
{
    if (find(cctNodes.begin(),cctNodes.end(),node)==cctNodes.end()) {
      cctNodes.push_front(node);
```

```
        return true;
      } else {
        return false;
      }
}

CctNodes& Frame::GetCctNodes()
{
      return (cctNodes);
}

bool Frame::AddDiffNode(string node)
{
      if (find(diffNodes.begin(),diffNodes.end(),node)==diffNodes.end()) {
        diffNodes.push_front(node);
        return true;
      } else {
        return false;
      }
}

DiffNodes& Frame::GetDiffNodes()
{
      return (diffNodes);
}

bool Frame::GenerateCctNodes(bool messages)
{
      if (messages) {
          cout << "Generation of cct nodes for " << GetName() << ":" << endl;
      }
      PartsIterator partsIter=parts.begin();
      bool ok=true;
      bool changeNode=true;
      while (partsIter != parts.end()) { // clear all existing cct node allocations
           (*partsIter)->ClearCctNodes();
           partsIter++;
      }
      while (ok && changeNode) {
           changeNode = false;
           partsIter=parts.begin();
           while (partsIter != parts.end() && !changeNode) { // only change one node before
checking for propagation
                Part* part=(*partsIter);
                // check that are flux entry and exit curves already defned for current part
                if (!part->GetEntry().empty() && !part->GetExit().empty()) {
                    // flux entry cct node
                    if (part->GetEntryCctNode()<0 && !changeNode) { // assume if node is
assigned then all boundary curves must have been found
                        part->SetEntryCctNode(GetDescription()->GetCctNode()); // get a new
cct node
                        if (messages) {
                            cout << part->GetName() << " exit -> " << part-
>GetEntryCctNode() << " new" << endl;
                        }
                        changeNode = true;
                    }
                    // flux exit cct node
                    if (part->GetExitCctNode()<0 && !changeNode) {
                        part->SetExitCctNode(GetDescription()->GetCctNode()); // get a new
cct node
                        if (messages) {
                            cout << part->GetName() << " entry -> " << part-
>GetExitCctNode() << " new" << endl;
                        }
                        changeNode = true;
                    }
                }
                partsIter++;
           }
           bool spread = changeNode;
           while (ok && spread) {
                spread = false;
                partsIter=parts.begin();
                while (partsIter != parts.end()) {
```

```
                Part* part=(*partsIter);
                // check that are flux entry and exit curves already defned for current
part
                if (!part->GetEntry().empty() && !part->GetExit().empty()) {
                    // flux entry cct node
                    if (part->GetEntryCctNode()<0) { // assume if node is assigned then
all boundary curves must have been found
                        CurvesIterator entryIter=part->GetEntry().begin(); // assign an
iterator to the flux entry curves
                        long node = -1;
                        while (entryIter != part->GetEntry().end() && node<0) { // step
through all curves until cct node found
                            Curve* entry=(*entryIter);
                            // step through parts until curve found in other parts entry
or exit curve list
                            PartsIterator findPartIter=parts.begin();
                            while (findPartIter != parts.end()  && node<0) {
                                Part* findPart=(*findPartIter);
                                // carry on if not self test and flux entry/exit curves
exist
                                if (findPart!=part && !findPart->GetEntry().empty() &&
!findPart->GetExit().empty()) {
                                    // test if current entry curve is in other parts
entry curves
                                    if (findPart->FindEntry(entry) && findPart-
>GetEntryCctNode()>=0) {

                                        node = findPart->GetEntryCctNode();
                                    }
                                    if (findPart->FindExit(entry) && findPart-
>GetExitCctNode()>=0) {

                                        node = findPart->GetExitCctNode();
                                    }
                                }
                                findPartIter++;
                            }
                            entryIter++;
                        }
                        if (node>=0) {
                            part->SetEntryCctNode(node); // use existing cct node number
                            if (messages) {
                                cout << part->GetName() << " entry -> " << part-
>GetEntryCctNode() << endl;
                            }
                            spread = true;
                        }
                    }
                    // flux exit cct node
                    if (part->GetExitCctNode()<0) {
                        CurvesIterator exitIter=part->GetExit().begin(); // assign an
iterator to the flux exit curves
                        long node = -1;
                        while (exitIter != part->GetExit().end() && node<0) { // step
through all curves until cct node found
                            Curve* exit=(*exitIter);
                            // step through parts until curve found in other parts entry
or exit curve list
                            PartsIterator findPartIter=parts.begin();
                            while (findPartIter != parts.end() && node<0) {
                                Part* findPart=(*findPartIter);
                                // carry on if not self test and flux entry/exit curves
exist
                                if (findPart!=part && !findPart->GetEntry().empty() &&
!findPart->GetExit().empty()) {
                                    // test if current entry curve is in other parts
entry curves
                                    if (findPart->FindEntry(exit) && findPart-
>GetEntryCctNode()>=0) {

                                        node = findPart->GetEntryCctNode();
                                    }
                                    if (findPart->FindExit(exit) && findPart-
>GetExitCctNode()>=0) {

                                        node = findPart->GetExitCctNode();
                                    }
                                }
                                findPartIter++;
                                }
                                exitIter++;
                            }
                            if (node>=0) {
                                part->SetExitCctNode(node); // use existing cct node number
                                if (messages) {
                                    cout << part->GetName() << " exit -> " << part-
>GetExitCctNode() << endl;
                                }
                                spread = true;
                            }
                        }
                    }
                    partsIter++;
                }
            }
    }
    return ok;
}

bool Frame::FindBoundaryCurves()
{
    PartsIterator partsIter=parts.begin();
    PartsIterator findPartIter=parts.begin();
    bool ok=true;
    CurvesSetBoundary(curves); // set all curves to default of boundary indicated
    while (partsIter != parts.end()) {
        Part* part=(*partsIter);
        CurvesIterator curveIter=part->GetCurves().begin(); // assign an iterator to the
part curves
        while (curveIter != part->GetCurves().end()) { // step through all curves
            Curve* curve=(*curveIter);
            if (curve->IsBoundary()) { // only perform check if its not already been
found to share parts
                // step through parts until curve found in other parts curve list
                findPartIter=parts.begin();
                bool found=false;
                while (findPartIter != parts.end() && !found) {
                    if ((*findPartIter)!=part) found=(((*findPartIter)-
>Find(curve))!=NULL);
                    findPartIter++;
                }
                curve->IsBoundary(!found);
            }
            curveIter++;
        }
        partsIter++;
    }
    return ok;
}

void Frame::WriteProperties(ofstream& out)
{
    out << GetName() << " {\n";
    out << "skewangle " << skew << "\n";
    CoilsWrite(out, coils);
    out << "\n";
    CoordsWrite(out, coords);
    out << "\n";
    CurvesWrite(out, curves);
    out << "\n";
    PartsWrite(out, parts);
    out << " }";
}

Frame::Frame(Description* d, const string& n):DescriptionObject(n)
{
    description=d;
    skew = 0;
    length = TYPICAL_LENGTH;
}

Frame::Frame(Description* d, const char* n):DescriptionObject(n)
{
    description=d;
    skew = 0;
    length = TYPICAL_LENGTH;
}

Frame::~Frame()
{
//    coords.Flush(true);
//    curves.Flush(true);
//    parts.Flush(true);
}

Description* Frame::GetDescription()
{
    return description;
}

Part* Frame::Find(Part* p)
{
    return parts.Find(p);
}

Curve* Frame::Find(Curve* c)
{
    return curves.Find(c);
}

Coord* Frame::Find(Coord* c)
{
    return coords.Find(c);
}

bool Frame::Add(Part* p)
{
    return parts.Add(p);
}

bool Frame::Add(Curve* c)
{
    return curves.Add(c);
}

bool Frame::Add(Coord* c)
{
    return coords.Add(c);
}

Parts& Frame::GetParts()
{
    return parts;
}

Curves& Frame::GetCurves()
{
    return curves;
}

Coords& Frame::GetCoords()
{
    return coords;
}

Coils& Frame::GetCoils()
{
    return coils;
}

Coil* Frame::Find(Coil* t)
{
    return coils.Find(t);
}

bool Frame::Add(Coil* t)
{
    return coils.Add(t);
}
```

```
Coils& Frame::GetCoilList()
{
    return coilList;
}

bool Frame::AddCoilList(Coil* t)
{
    return coilList.Add(t);
}

double Frame::GetSkew()
{
    return (skew);
}

double Frame::SetSkew(double s)
{
    return (skew=s);
}

double Frame::GetLength()
{
    return (length);
}

double Frame::SetLength(double l)
{
    return (length=l);
}

bool Frame::Parse(LexGDL& lex, bool inform) // expects to find two new tokens
{
    bool ok = true;
    if (lex.CurrStr()=="{") {
        lex.GetToken(); // get new token
        do {
            switch (lex.CurrToken()) {
                case LexGDL::SKEW :
                    {   bool deg;
                        double s;
                        if (deg=(lex.GetToken()==LexGDL::DEG))  lex.GetToken(); // get
new token
                        if (lex.CurrVar(s)) {
                            skew = (deg?s*M_PI/180.0:s);
                            lex.GetToken();  // get new token
                        }
                        else {
                            cerr << "ERROR (line: " << lex.LineCount() << " frame: " <<
nameId << ") value for skew angle expected" << endl;
                            ok = false;
                        }
                    }
                    break;
                 case LexGDL::LENGTH :
                    {    double l;
                        lex.GetToken();  // get new token
                        if (!lex.CurrVar(l)) {
                            cerr << "ERROR (line: " << lex.LineCount() << ") value for
length expected" << endl;
                            ok = false;
                        }
                        else {
                            length=l*(GetDescription()->GetUnitDistance());
                            lex.GetToken();  // get new token
                        }
                    }
                break;
                case LexGDL::COIL :
                    {    lex.GetToken();  // get new token
                        ok &= Coil::Parse(lex, this, inform);
                    }
                    break;
                case LexGDL::NODE :
                    {    lex.GetToken();  // set up for parse
```

```
                         ok &= Coord::Parse(lex, this, inform);
                     }
                     break;
                 case LexGDL::CURVE :
                     {    lex.GetToken();  // set up for parse
                         ok &= Curve::Parse(lex, this, inform);
                     }
                     break;
                 case LexGDL::PART :
                     {    lex.GetToken();  // set up for parse
                         ok &= Part::Parse(lex, this, inform);
                     }
                     break;
                 default :
                     {    cerr << "ERROR (line: " << lex.LineCount() << ") '" <<
lex.CurrStr() << "' not valid" << endl;
                         ok = false;
                         lex.GetToken();
                     }
             }
         } while (lex.CurrToken()!=LexGDL::END && lex.CurrStr()!="}");
         if (lex.CurrStr()=="}") {
             lex.GetToken();
         }
         else {
             cerr << "ERROR (line: " << lex.LineCount() << ") '}' expected" << endl;
             ok = false;
         }
     }
     else {
         cerr << "ERROR (line: " << lex.LineCount() << " frame:" << nameId << ") '{'
expected" << endl;
         ok = false;
     }
     return ok;
}

bool Frame::CurvesMinMax(Vector2d& cmin, Vector2d& cmax)
{
    CurvesIterator iter=curves.begin();
    if (iter!=curves.end()) {
        Vector2d currmin, currmax;
        (*(iter++))->GetMinMax(cmin, cmax);
        while (iter != curves.end()) {
            (*(iter++))->GetMinMax(currmin, currmax);
            cmin((cmin.X()<currmin.X()?cmin.X():currmin.X()),
(cmin.Y()<currmin.Y()?cmin.Y():currmin.Y()));
            cmax((cmax.X()>currmax.X()?cmax.X():currmax.X()),
(cmax.Y()>currmax.Y()?cmax.Y():currmax.Y()));
        }
        return true;
    }
    else {
        return false;
    }
}

void Frame::Info()
{
    CurvesIterator iter=curves.begin();
    while (iter != curves.end()) {
        cout << "curve '" << (*iter)->GetName() << "' length: " << (*iter)->Length() <<
endl;
        iter++;
    }
}

void Frame::Check(long sections)
{
    double integral;
    bool ok;
    PartsIterator iter=parts.begin();
    while (iter != parts.end()) {
        ok = (*iter)->LineIntegral(integral,sections);
        cout << "part '" << (*iter)->GetName();
```

```
#if !defined(lex_h)              // Sentry, use file only if it's not already included.
#define lex_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         desc.h
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Class definitions for description.
*/

#include <string>
#include <fstream>

// lexical analysis of text file returning tokens
class LexGDL
{
public:
    enum Token {END, UNKNOWN, CHAR, NUM, IDENT, LENGTH, STATOR, ROTOR, PART, BH,  COIL,
CURVE,
                LINE, CIRC, LINEAR, EXP, NODE, ENTRY, EXIT, INTERFACE, UNIT, TORQUE,
SKEW, DEG, AIRGAP};

    LexGDL(ifstream* in);
    ~LexGDL();
    long LineCount();
    Token CurrToken();
    bool CurrVar(string&);
    bool CurrVar(double&);
    bool CurrVar(long&);
    string& CurrStr();
    double CurrNum();
    Token PrevToken();
    bool PrevVar(string&);
    bool PrevVar(double&);
    bool PrevVar(long&);
    string& PrevStr();
    double PrevNum();
    void DelCurrVar();
    void DelPrevVar();
    Token GetToken();

    // various constructs used by other routines
    bool ParseBracket(long&, long&, long&);
    bool ParseBracket(long&, long&);
    bool ParseBracket(double&, double&);
private:
    ifstream* instream;
    union LexVar {
        double *num;
        string *str;
    };
    long lineCount;
    Token prevToken;
    LexVar prevVar;
    Token currToken;
    LexVar currVar;
};

#endif                                  // lex_h sentry.
```

```
/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         lex.cpp
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Source file for implementation of lexical analyer.
*/

#include "lex.h"

LexGDL::LexGDL(ifstream* in)
{
    lineCount=1;
    currToken=NUM;
    currVar.num=0;
    prevToken=NUM;
    prevVar.num=0;
    instream=in;
}

LexGDL::~LexGDL()
{
    DelCurrVar();
    DelPrevVar();
}

long LexGDL::LineCount()
{
    return lineCount;
}

LexGDL::Token LexGDL::CurrToken()
{
    return currToken;
}

bool LexGDL::CurrVar(string& str)
{
    if (currToken==LexGDL::UNKNOWN||currToken==LexGDL::CHAR){
        str=*(currVar.str);
        return true;
    }
    else {
        str="";
        return false;
    }
}

bool LexGDL::CurrVar(double& num)
{
    if (currToken==LexGDL::NUM){
        num=*(currVar.num);
        return true;
    }
    else {
        num=0.0;
        return false;
    }
}

bool LexGDL::CurrVar(long& num)
{
    if (currToken==LexGDL::NUM){
        num=(long)(*(currVar.num));
        return true;
    }
    else {
        num=0;
        return false;
    }
}

string& LexGDL::CurrStr()
{
    if (currToken==LexGDL::UNKNOWN||currToken==LexGDL::CHAR)
        return *(currVar.str);
    else
        return *new string("");
}

double LexGDL::CurrNum()
{
    if (currToken==LexGDL::NUM)
        return *(currVar.num);
    else
        return 0.0;
}

LexGDL::Token LexGDL::PrevToken()
{
    return prevToken;
}

bool LexGDL::PrevVar(string& str)
{
    if (prevToken==LexGDL::UNKNOWN||currToken==LexGDL::CHAR){
        str=*(prevVar.str);
        return true;
    }
    else {
        str="";
        return false;
    }
}

bool LexGDL::PrevVar(double& num)
{
    if (prevToken==LexGDL::NUM){
        num=*(prevVar.num);
        return true;
    }
    else {
        num=0.0;
        return false;
    }
}

bool LexGDL::PrevVar(long& num)
{
    if (prevToken==LexGDL::NUM){
        num=(long)(*(prevVar.num));
        return true;
    }
    else {
        num=0;
        return false;
    }
}
string& LexGDL::PrevStr()
{
    if (prevToken==LexGDL::UNKNOWN||prevToken==LexGDL::CHAR)
        return *(prevVar.str);
    else
        return *new string("");
}

double LexGDL::PrevNum()
{
    if (prevToken==LexGDL::NUM)
        return *(prevVar.num);
    else
        return 0.0;
}
```

```
void LexGDL::DelCurrVar()
{
    if (currToken==NUM) delete currVar.num;
    if (currToken==UNKNOWN || currToken==CHAR) delete currVar.str;
}

void LexGDL::DelPrevVar()
{
    if (prevToken==NUM) delete prevVar.num;
    if (prevToken==UNKNOWN || prevToken==CHAR) delete prevVar.str;


}

LexGDL::Token LexGDL::GetToken()
{
    if (currToken!=END && prevToken!=END) {
        DelPrevVar();
        prevVar = currVar;
        prevToken = currToken;
        char c = EOF;
        /* Ignore whitespace and newlines, get first nonwhite character.  */
        if (instream->good() && !instream->eof())  instream->get(c);
        while ((c == ' ' || c == '\t' || c == '\n' || c == '#' || c == '\r') && c != EOF
&& !instream->eof()) {
            if (c == '#') {
                while (c != '\n' && c != EOF && !instream->eof())
                    instream->get(c);
            }
            else {
                if (c=='\n')
                    lineCount++;
                instream->get(c);
            }
        }
        if (c==EOF || instream->bad() || instream->eof())
            return (currToken=END);

        /* Char starts a number => parse the number.  */
        if (c == '-' || c == '.' || isdigit (c)) {
            double value;
            instream->putback(c);
            (*instream) >> value;
            currVar.num = new double(value);
            return (currToken=NUM);
        }

        /* Char starts an identifier => read the name.  */
        string symbol;
        symbol = c;
        if (isalpha (c)) {
            instream->get(c);
            while (c != EOF  && !instream->eof() && isalnum (c)) {
                symbol += c;
                /* Get another character.  */
                instream->get(c);
            }
            instream->putback(c);

            /* check to see if the input is a keyword */
            if       (symbol=="length")          currToken = LENGTH;
            else if (symbol=="stator")          currToken = STATOR;
            else if (symbol=="rotor")            currToken = ROTOR;
            else if (symbol=="parts")            currToken = PART;
            else if (symbol=="bh")              currToken = BH;
            else if (symbol=="coils")            currToken = COIL;
            else if (symbol=="curves")          currToken = CURVE;
            else if (symbol=="line")            currToken = LINE;
            else if (symbol=="circ")            currToken = CIRC;
            else if (symbol=="linear")          currToken = LINEAR;
            else if (symbol=="exp")             currToken = EXP;
            else if (symbol=="nodes")           currToken = NODE;
            else if (symbol=="entry")           currToken = ENTRY;
            else if (symbol=="exit")             currToken = EXIT;
            else if (symbol=="interface")        currToken = INTERFACE;
            else if (symbol=="unitdistance")    currToken = UNIT;
            else if (symbol=="torquemodifier")  currToken = TORQUE;
            else if (symbol=="skewangle")        currToken = SKEW;
            else if (symbol=="d")                currToken = DEG;
            else if (symbol=="airgap")           currToken = AIRGAP;
            else                                currToken = UNKNOWN;
        }
        else {
            currToken = CHAR;
        }
        if (currToken==CHAR || currToken==UNKNOWN)
            currVar.str = new string(symbol);
    }
    return currToken;
}

// various constructs used by other routines
bool LexGDL::ParseBracket(long& v1, long& v2, long& v3)
{
    int state = 0;
    bool ok = true;
    do {
        switch (state) {
          case 0 : {
            ok &= (CurrStr()=="[");
            break;
          }
          case 1 : {
            ok &= CurrVar(v1);
            break;
          }
          case 2 : {
            ok &= CurrVar(v2);
            break;
          }
          case 3 : {
            ok &= CurrVar(v3);
            break;
          }
          case 4 : {
            ok &= (CurrStr()=="]");
            break;
          }
          default: ok = false;

        }
        state++;
    } while (GetToken()!=LexGDL::END && state!=5);
    return ok;
}

bool LexGDL::ParseBracket(long& v1, long& v2)
{
    int state = 0;
    bool ok = true;
    do {
        switch (state) {
          case 0 : {
            ok &= (CurrStr()=="[");
            break;
          }
          case 1 : {
            ok &= CurrVar(v1);
            break;
          }
          case 2 : {
            ok &= CurrVar(v2);
            break;
          }
          case 3 : {
            ok &= (CurrStr()=="]");
            break;
          }
          default: ok = false;
        }
        state++;
    } while (GetToken()!=LexGDL::END && state!=4);
    return ok;
}

bool LexGDL::ParseBracket(double& v1, double& v2)
{
    int state = 0;
    bool ok = true;
    do {
        switch (state) {
          case 0 : {
            ok &= (CurrStr()=="[");
            break;
          }
          case 1 : {
            ok &= CurrVar(v1);
            break;
          }
          case 2 : {
            ok &= CurrVar(v2);
            break;
          }
          case 3 : {
            ok &= (CurrStr()=="]");
            break;
          }
          default: ok = false;
        }
        state++;
    } while (GetToken()!=LexGDL::END && state!=4);
    return ok;
}
```

```
/***************************************************************************
                          main.cpp  -  description
                             -------------------
    begin                : Thu Apr 15 18:22:43 BST 1999

    copyright            : (C) 1999 by D. Downes
    email                : david.downes@ntu.ac.uk
 ***************************************************************************/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream>

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

#include "desc.h"


bool OpenFile(const char*, string, bool);
void ConvertSpice(const char*, string, string, long, bool, bool);

Description* description = NULL;

int main(int argc, char **argv)
{
    long sections=100;
    bool verbose=false;
    bool info=false;
    bool help=false;
    bool torq=true;
    int opt_char;
    string prefix("1:1:");
    while ((opt_char = getopt (argc, argv, "vihns:p:")) != -1) {
      switch(opt_char) {
        case 'v': verbose=true; break;
        case 'i': info=true; break;
        case 'h': help=true; break;
        case '?': help=true; break;
        case 'n': torq=false; break;
        case 's': sections=atol(optarg); break;
        case 'p': prefix=optarg; break;
        default: abort();
      }
    }
    if (argc-optind!=2 || help) {
        cout << "Usage: gdl2spice [-vihn] [-s sections] [-p prefix] in.gdl out.cct" <<
endl;
    } else {
        string name(argv[optind]);
        int x=name.find(".");
        if (x<string::npos) name.erase(x);
        if (OpenFile(argv[optind], name, verbose)) {
            if (info) {
              if (description) {
                description->Info();
              } else {
                cerr << "Error: description object pointer is null;" << endl;
              }
            } else {
              if (verbose) cout << "No. of sections to be used during integration: " <<
sections << endl;
              ConvertSpice(argv[optind+1],name,prefix,sections,verbose,torq);
            }
        } else {
            cerr << "Problem opening file: " << argv[optind] << endl;
        }
    }
//    cout << argc << " : " << argv[0] << endl;
}
```

```
bool OpenFile(const char *fileName, string name, bool verbose)
{
    ifstream* instream;
    bool ok=false;

    if ((instream = new ifstream(fileName))!=NULL) {
      if (instream->good()) {
//          delete description;
          description = NULL;
          if ((description = new Description(name.c_str()))!=NULL) {
              cout << "Parsing file ..." << endl;
              if (ok=description->Parse(instream,verbose))
                  cout << "Finished parsing file with no errors." << endl;
              else
                  cerr << "Error while parsing file!" << endl;
          }
          else {
              cerr << "Unable to create description class!" << endl;
          }
      }
      // Return an error message if we had a stream error and it wasn't the eof.
      if (instream->bad() && !instream->eof()) {
        cerr << "Stream error!" << endl;
      }
      delete instream;
    }
    else {
        cerr << "Unable to create description class!" << endl;
    }
    return ok;
}


void ConvertSpice(const char *cctFileName, string name, string prefix, long sections,
bool verbose, bool torq)
{
  if (description) {
    ofstream* cctstream = NULL;
    ofstream* lststream = NULL;
    string lstfile(name);
    lstfile+=".lst";
    name+=".dat";
    if ((cctstream = new ofstream(cctFileName)) && (lststream = new
ofstream(lstfile.c_str()))) {
      // write to file
      cout << "Converting model ..." << endl;
      if (description->Convert(*cctstream,*lststream,name.c_str(),prefix.c_str(),
sections,verbose,torq))
        cout << "Model successfully converted" << endl;
      else
        cerr << "Error occured while converting model" << endl;
      delete cctstream;
      delete lststream;
    } else {
      // Return an error message if we had a stream error.
      cerr << "Stream error!" << endl;
    }
  }  else {
    cout << "Must open a model before it can be converted" << endl;
  }
}
```

```
#if !defined(part_h)              // Sentry, use file only if it's not already included.
#define part_h

/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         part.h
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Class definitions for part.
*/

#include <string>
#include <cstdlib>

#include "unique.h"
#include "descobj.h"
#include "curve.h"
#include "coil.h"

class Description;
class Frame;
class Bh;

// Part
class Part : public DescriptionObject
{
private:
    Frame* frame;
    Curves curves;
    Bh* bh;
    Coils coils;
    Curves fluxEntry;
    Curves fluxExit;
    long fluxEntryCctNode;
    long fluxExitCctNode;
    double length;
    double minArea;
    double fluxEntryPot;
    double fluxExitPot;
    double diffPot;
    double lineIntegral;

public:
    Part();
    Part(Frame* f, const string& n);
    Part(Frame* f, const char* n = "unknown");
    ~Part();

    Bh* GetBh();
    Bh* SetBh(Bh*);
    bool IsMagPart();
    double GetLineIntegral();
    double SetLineIntegral(double);
    double GetMinArea();
    double SetMinArea(double);
    double SetLength(double);
    double GetLength();
    Curves& GetCurves();
    Coils& GetCoils();
    Curves& GetEntry();
    Curves& GetExit();
    Frame* GetFrame();
    Description* GetDescription();
    long GetEntryCctNode();
    long GetExitCctNode();
    long SetEntryCctNode(long);
    long SetExitCctNode(long);
    double GetEntryPot();
    double GetExitPot();
    double GetDiffPot();
    double SetEntryPot(double);
    double SetExitPot(double);
    double SetDiffPot(double);
    Curve* Find(Curve*);
    Coil* Find(Coil*);
    Curve* FindEntry(Curve*);
    Curve* FindExit(Curve*);
    bool Add(Curve*);
    bool Add(Coil*);
    bool AddEntry(Curve*);
    bool AddExit(Curve*);
    bool IsBoundary();
    bool IsEntryBoundary();
    bool IsExitBoundary();

    void ClearCctNodes();
    void WriteProperties(ofstream&);
    bool LinearMagCap(double&, long);
    static bool Parse(LexGDL&, Frame*, bool inform=true);
    bool LineIntegral(double&, int);
    bool Convert(ofstream&, long);
};

typedef Unique<Part *> Parts;
typedef Unique<Part *>::iterator PartsIterator;

void PartsWrite(ofstream&, Parts&);
#endif                                      // part_h sentry.
```

```
/*  Project gdl
    TNTU
    Copyright © 1996

    SUBSYSTEM:    gdl.exe Application
    FILE:         part.cpp
    AUTHOR:       D. Downes


    OVERVIEW
    ========
    Source file for implementation of part.
*/

#include <cmath>
#include <strstream>

#include "desc.h"
#include "part.h"
#include "frame.h"
#include "bh.h"

void PartsWrite(ofstream& out, Parts& parts)
{
    out << "parts {\n";
    PartsIterator iter=parts.begin();
    while (iter != parts.end()) {

        (*iter)->WriteProperties(out);
        out << "\n";
        iter++;
    }
    out << "}";
}

void Part::WriteProperties(ofstream& out)
{
    CurvesIterator curvesiter=curves.begin();
    CoilsIterator coilsiter=coils.begin();
    CurvesIterator entryiter=fluxEntry.begin();
    CurvesIterator exititer=fluxExit.begin();
    out << GetName() << " { length " << length << "\n";
    if (bh) out << "bh " << bh->GetName() << "\n";
    if (curvesiter != curves.end()) {
        out << "curves { ";
        do {
            out << (*(curvesiter++))->GetName() << " ";
        } while (curvesiter != curves.end());
        out << "}\n";
    }
    if (coilsiter != coils.end()) {
        out << "coils { ";
        do {
            out << (*(coilsiter++))->GetName() << " ";
        } while (coilsiter != coils.end());
        out << "}\n";
    }
    if (entryiter != fluxEntry.end()) {
        out << "entry { ";
        do {
            out << (*(entryiter++))->GetName() << " ";
        } while (entryiter != fluxEntry.end());
        out << "}\n";
    }
    if (exititer != fluxExit.end()) {
        out << "exit { ";
        do {
            out << (*(exititer++))->GetName() << " ";
        } while (exititer != fluxExit.end());
        out << "}\n";
    }
    out << "}";
}

bool Part::LinearMagCap(double& cap, long sections)
```

```
{
    bool ok = false;
    if (bh!=NULL) {
        if ((bh->GetType())==Bh::LINEAR) {
            double integral;
            if (LineIntegral(integral,sections)) {
                ok = bh->ConvertLinear(cap, integral);
            }
        }
    }
    return ok;
}

bool Part::Convert(ofstream& cct, long sections)
{
    bool ok = true;
    double integral = 0;
    double volume = 0;
    if (IsMagPart()) {  // check that cct nodes and bh exist
        // ok=LineIntegral(integral,volume,sections);    // check line integral is valid
        ok=LineIntegral(integral,sections);    // check line integral is valid
        if (ok && integral!=0) {  // bh curve exists and integral isn't zero
            frame->AddCctNode(fluxEntryCctNode);  // add nodes to the list of cct nodes
held by frame
            frame->AddCctNode(fluxExitCctNode);
            if (bh->GetType()==Bh::LINEAR) {
              strstream num1; // used to convert numeric to string
              num1 << fluxEntryCctNode;
              frame->AddDiffNode(num1.str());
              strstream num2;
              num2 << fluxExitCctNode;
              frame->AddDiffNode(num2.str());
            }
            else {
              frame->AddDiffNode(bh->GetCapNode(*this));
            }
            long cctNode;
            // generate circuit elements
            if (!coils.empty())
                cctNode=GetDescription()->GetCctNode(); // at least one coil exists
            else
                cctNode=fluxExitCctNode; // no coils therfore use the existing exit node
            cct  << "* part '" << GetName() << "' length: " << length << " integral: "
<< integral
                << " entry cct node: " << fluxEntryCctNode << " exit cct node: " <<
fluxExitCctNode << "\n";
            // generate magnetic capacitor
            bh->Convert(cct, *this, fluxEntryCctNode, cctNode, integral, volume);

            cct << "* coils:\n";
            // generate gyrators, if any exist for the part
            CoilsIterator coilsiter=coils.begin();
            if (coilsiter != coils.end()) {
                long midCctNode;
                midCctNode=GetDescription()->GetCctNode(); // mid point connection of
src pair
                cct << "Vsens" << GetName() << " " << cctNode << " " << midCctNode  << "
0\n";
                Coil* coilptr;
                do {
 //                     frame->AddCctNode(midCctNode); // provide ground point for
mid node
                    coilptr = *(coilsiter++); // set coil pointer ahead so as to find
out if its the last coil in the list
                    if (coilsiter == coils.end()) // last coil
                        cctNode=fluxExitCctNode; // last coil so use the exit node
                    else
                        cctNode=GetDescription()->GetCctNode(); // needs a new cct node
                    cct << "H" << GetName() << coilptr->GetName() << " " << cctNode << "
" << midCctNode
                        << " Vsens" << coilptr->GetName() << " " << (-1*(coilptr-
>GetTurns())) << "\n";
                    midCctNode=cctNode;
                } while (coilsiter != coils.end());
            }
        }
    }
    return ok;
}

Part::Part(Frame* f, const string& n):DescriptionObject(n)
{
    frame=f;
    bh=NULL;
    fluxEntryCctNode=-1;
    fluxExitCctNode=-1;
    fluxEntryPot=0;
    fluxExitPot=0;
    diffPot=0;
    length=f->GetLength();
    minArea=-1;
    lineIntegral=-1;
}

Part::Part(Frame* f, const char* n):DescriptionObject(n)
{
    frame=f;
    bh=NULL;
    fluxEntryCctNode=-1;
    fluxExitCctNode=-1;
    fluxEntryPot=0;
    fluxExitPot=0;
    diffPot=0;
    length=f->GetLength();
    minArea=-1;
    lineIntegral=-1;
}

Part::~Part()
{
}

bool Part::IsMagPart() // check that cct nodes and bh exist

{
    return (fluxEntryCctNode>=0 && fluxExitCctNode>=0 && bh);
}

double Part::GetLineIntegral()
{
    return lineIntegral;
}

double Part::SetLineIntegral(double i)
{
    return (lineIntegral=i);
}

double Part::GetMinArea()
{
    return minArea;
}

double Part::SetMinArea(double a)
{
    return (minArea=a);
}


double Part::SetLength(double l)
{
    return (length=l);
}

double Part::GetLength()
{
    return length;
}

Frame* Part::GetFrame()
```

```
{
    return frame;
}

Description* Part::GetDescription()
{
    return (frame?frame->GetDescription():NULL);
}

Bh* Part::GetBh()
{
    return bh;
}

double Part::GetEntryPot()
{
    return fluxEntryPot;
}

double Part::GetExitPot()
{
    return fluxExitPot;
}

double Part::SetEntryPot(double value)
{
    return (fluxEntryPot=value);
}

double Part::SetExitPot(double value)
{
    return (fluxExitPot=value);
}

double Part::GetDiffPot()
{
    return diffPot;
}

double Part::SetDiffPot(double value)
{
    return (diffPot=value);
}

long Part::GetEntryCctNode()
{
    return fluxEntryCctNode;
}

long Part::GetExitCctNode()
{
    return fluxExitCctNode;
}

long Part::SetEntryCctNode(long node)
{
    return (fluxEntryCctNode=node);
}

long Part::SetExitCctNode(long node)
{
    return (fluxExitCctNode=node);
}

void Part::ClearCctNodes()
{
    fluxEntryCctNode=-1;
    fluxExitCctNode=-1;
}

Bh* Part::SetBh(Bh* b)
{
    return (bh = b);
}
```

```
Curves& Part::GetCurves()
{
    return curves;
}

Curve* Part::Find(Curve* c)
{
    return curves.Find(c);
}

bool Part::Add(Curve* c)
{
    return curves.Add(c);
}

Coils& Part::GetCoils()
{
    return coils;
}

Coil* Part::Find(Coil* c)
{
    return coils.Find(c);
}

bool Part::Add(Coil* c)
{
    return coils.Add(c);
}

Curves& Part::GetEntry()
{
    return fluxEntry;
}

Curve* Part::FindEntry(Curve* c)
{
    return fluxEntry.Find(c);
}

bool Part::AddEntry(Curve* c)
{
    return fluxEntry.Add(c);
}

Curves& Part::GetExit()
{
    return fluxExit;
}


Curve* Part::FindExit(Curve* c)
{
    return fluxExit.Find(c);
}

bool Part::AddExit(Curve* c)
{
    return fluxExit.Add(c);
}

bool Part::IsBoundary()
{
    bool b = false;
    CurvesIterator iter=curves.begin();
    while (iter!=curves.end() && !b) {
        b |= ((*(iter++))->IsBoundary());
    }
    return (b);
}

bool Part::IsEntryBoundary()
{
    bool b = false;
    CurvesIterator iter=fluxEntry.begin();
    while (iter!=fluxEntry.end() && !b) {
        b |= ((*(iter++))->IsBoundary());
    }
    return (b);
}

bool Part::IsExitBoundary()
{
    bool b = false;
    CurvesIterator iter=fluxExit.begin();
    while (iter!=fluxExit.end() && !b) {
        b |= ((*(iter++))->IsBoundary());
    }
    return (b);
}

bool Part::Parse(LexGDL& lex, Frame* frame, bool inform) // expects to find new token
already current
{
    Part* part;
    bool ok = true;
    string nameId;
    bool currok = true;
    if (lex.CurrStr()=="{") {
        lex.GetToken(); // get new token
        do {
            if (lex.CurrToken()==LexGDL::UNKNOWN) {
                nameId = lex.CurrStr();
                lex.GetToken(); // get new token
                if (lex.CurrStr()=="{") {
                    lex.GetToken(); // get new token
                    part = new Part(frame, nameId);
                    if (frame->Add(part)) {
                        if (inform) {
                            cout << "Part: " << part->GetName() << " #" << frame-
>GetParts().size() << " added" << endl;
                        }
                        currok = true;
                        do {
                            switch (lex.CurrToken()) {
                                case LexGDL::LENGTH : {
                                    lex.GetToken(); // get new token
                                    double l;
                                    if (lex.CurrVar(l)) {
                                        part->SetLength(l*frame->GetDescription()-
>GetUnitDistance());

                                        lex.GetToken();
                                    }
                                    else {
                                        cerr << "ERROR (line: " << lex.LineCount() << ")
numeric argument for length expected" << endl;
                                        currok = false;
                                    }
                                }
                                break;
                                case LexGDL::BH : {
                                    lex.GetToken(); // get new token
                                    if (lex.CurrToken()==LexGDL::UNKNOWN && frame-
>GetDescription()) {

                                        Bh* b = frame->GetDescription()->Find(&Bh(frame-
>GetDescription(),lex.CurrStr()));

                                        if (b) {
                                            part->SetBh(b);
                                        }
                                        else {
                                            cerr << "ERROR (line: " << lex.LineCount()
<< " part: " << nameId << ") bh: " << lex.CurrStr() << " doesn't exist" << endl;
                                            currok = false;
                                        }
                                        lex.GetToken();
                                    }
                                    else {
                                        cerr << "ERROR (line: " << lex.LineCount() << ")
bh name expected" << endl;
```

```
                                        currok = false;
                                    }
                                }
                                break;
                                case LexGDL::CURVE : {
                                    lex.GetToken();
                                    if (lex.CurrStr()=="{") {
                                        lex.GetToken(); // get new token
                                        do {
                                            if (lex.CurrToken()==LexGDL::UNKNOWN) {
                                                string name;
                                                lex.CurrVar(name);
                                                lex.GetToken();
                                                if (lex.CurrStr()=="[") { // is it a
pattern?
                                                    long start, rep, inc;
                                                    currok &= lex.ParseBracket(start,
rep, inc); // parses current and leaves new token as current
                                                    if (currok) {
                                                        long curr, c;
                                                        for (curr=start,c=0; c<rep;
curr+=inc,c++) {
                                                            strstream number;
                                                            number << curr; // change
integer to char string
                                                            string full = name +
number.str();
                                                            Curve* curve = frame-
>Find(&Curve(frame,full));
                                                            if (curve) {
                                                                part->Add(curve);
                                                            }
                                                            else {
                                                                cerr << "ERROR (line: "
<< lex.LineCount() << " part: " << nameId << ") curve: " << full << " doesn't exist" <<
endl;
                                                                currok = false;
                                                            }
                                                        }
                                                    }
                                                }
                                                else {
                                                    Curve* curve = frame-
>Find(&Curve(frame,name));
                                                    if (curve) {
                                                        part->Add(curve);
                                    }
                                                    else {
                                                        cerr << "ERROR (line: " <<
lex.LineCount() << " part: " << nameId << ") curve: " << lex.CurrStr() <<
exist" << endl;
                                                        currok = false;
                                                    }
                                                }

                                            }
                                            else {
                                                cerr << "ERROR (line: " <<
lex.LineCount() << ") curve name expected" << endl;
                                                currok = false;
                                                lex.GetToken();
                                            }
                                        } while (lex.CurrToken()!=LexGDL::END &&
lex.CurrStr()!="}");
                                        if (lex.CurrStr()=="}") {
                                            lex.GetToken();
                                        }
                                        else {
                                            cerr << "ERROR (line: " << lex.LineCount()
<< " part: " << nameId << ") '}' expected" << endl;
                                            currok = false;
                                        }
                                    }
                                    else {
                                        cerr << "ERROR (line: " << lex.LineCount() << "
part: " << nameId << ") '{' expected" << endl;
```

```
                                        currok = false;
  }
                                }
                                break;
                                case LexGDL::COIL : {
                                    lex.GetToken(); // get new token
                                    if (lex.CurrStr()=="{") {
                                        lex.GetToken(); // get new token
                                        do {
                                            string name;
                                            if (lex.CurrVar(name)) {
                                                lex.GetToken(); // get new token
                                                Coil* coil = frame->Find(&Coil(frame,
name));

                                                if (coil) {
                                                    part->Add(coil);
                                                }
                                                else {
                                                    cerr << "ERROR (line: " <<
lex.LineCount() << " part: " << nameId << ") coil: " << name << " doesn't exist" << endl;
                                                    currok = false;
                                                }
                                            }
                                            else {
                                                cerr << "ERROR (line: " <<
lex.LineCount() << " part: " << nameId << ") coil_name expected" << endl;
                                                currok = false;
                                                lex.GetToken();
                                            }
                                        } while (lex.CurrToken()!=LexGDL::END &&
lex.CurrStr()!="}");
                                        if (lex.CurrStr()=="}") {
                                            lex.GetToken();
                                        }
                                        else {
                                            cerr << "ERROR (line: " << lex.LineCount()
<< " part: " << nameId << ") '}' expected" << endl;
                                            currok = false;
                                        }
                                    }
                                    else {
                                        cerr << "ERROR (line: " << lex.LineCount() << "
part: " << nameId << ") '{' expected" << endl;
                                        currok = false;
                                    }
                                }
                                break;
                                case LexGDL::ENTRY : {
                                    lex.GetToken(); // get new token
                                    if (lex.CurrStr()=="{") {
                                        lex.GetToken(); // get new token
                                        do {
                                            // entry
                                            if (lex.CurrToken()==LexGDL::UNKNOWN) {
                                                string name;
                                                lex.CurrVar(name);
                                                lex.GetToken();
                                                if (lex.CurrStr()=="[") { // is it a
pattern?
                                                    long start, rep, inc;
                                                    currok &= lex.ParseBracket(start,
rep, inc); // parses current and leaves new token as current
                                                    if (currok) {
                                                        long curr, c;
                                                        for (curr=start,c=0; c<rep;
curr+=inc,c++) {

                                                            strstream number;
                                                            number << curr; // change
integer to char string

                                                            string full = name +
number.str();

                                                            Curve* curve = part-
>Find(&Curve(frame,full));

                                                            if (curve) {
                                                                part->AddEntry(curve);
```

```
                                                              }
                                                              else {
                                                                  cerr << "ERROR (line: "
<< lex.LineCount() << " part: " << nameId << ") curve: " << full <<
in this part" << endl;
                                                                  currok = false;
                                                              }
                                                          }
                                                      }
                                                  }
                                                  else {
                                                      Curve* curve = part-
>Find(&Curve(frame,name));
                                                      if (curve) {
                                                          part->AddEntry(curve);
                                                      }
                                                      else {
                                                          cerr << "ERROR (line: " <<
lex.LineCount() << " part: " << nameId << ") curve: " << lex.CurrStr() <<
defined as in this part" << endl;
                                                          currok = false;
                                                      }
                                                  }
                                              }
                                              else {
                                                  cerr << "ERROR (line: " <<
lex.LineCount() << ") curve_name expected" << endl;
                                                  currok = false;
                                                  lex.GetToken();
                                              }
                                          } while (lex.CurrToken()!=LexGDL::END &&
lex.CurrStr()!="}");
                                          if (lex.CurrStr()=="}") {
                                              lex.GetToken();
                                          }
                                          else {
                                              cerr << "ERROR (line: " << lex.LineCount()
<< " part: " << nameId << ") '}' expected" << endl;
                                              currok = false;
                                          }
                                      }
                                      else {
                                          cerr << "ERROR (line: " << lex.LineCount() << "
part: " << nameId << ") '{' expected" << endl;
                                          currok = false;
                                      }
                                  }
                                  break;
                                  case LexGDL::EXIT : {
                                      lex.GetToken(); // get new token
                                      if (lex.CurrStr()=="{") {
                                          lex.GetToken(); // get new token
                                          do {
                                              // exit
                                              if (lex.CurrToken()==LexGDL::UNKNOWN) {

                                                  string name;
                                                  lex.CurrVar(name);
                                                  lex.GetToken();
                                                  if (lex.CurrStr()=="[") { // is it a
pattern?
                                                      long start, rep, inc;
                                                      currok &= lex.ParseBracket(start,
rep, inc); // parses current and leaves new token as current
                                                      if (currok) {
                                                          long curr, c;
                                                          for (curr=start,c=0; c<rep;
curr+=inc,c++) {

                                                              strstream number;
                                                              number << curr; // change
integer to char string
                                                              string full = name +
number.str();
                                                              Curve* curve = part-
>Find(&Curve(frame,full));
```

```
                                                             if (curve) {
                                                                 part->AddExit(curve);
                                                             }
                                                             else {
                                                                 cerr << "ERROR (line: "
<< lex.LineCount() << " part: " << nameId << ") curve: " << full <<
in this part" << endl;
                                                                 currok = false;
                                                             }
                                                         }
                                                     }
                                                 }
                                                 else {
                                                     Curve* curve = part-
>Find(&Curve(frame,name));
                                                     if (curve) {
                                                         part->AddExit(curve);
                                                     }
                                                     else {
                                                         cerr << "ERROR (line: " <<
lex.LineCount() << " part: " << nameId << ") curve: " << lex.CurrStr() <<
defined as in this part" << endl;
                                                         currok = false;
                                                     }
                                                 }
                                             }
                                             else {
                                                 cerr << "ERROR (line: " <<
lex.LineCount() << ") curve name expected" << endl;
                                                 currok = false;
                                                 lex.GetToken();
                                             }
                                         } while (lex.CurrToken()!=LexGDL::END &&
lex.CurrStr()!="}");
                                         if (lex.CurrStr()=="}") {
                                             lex.GetToken();
                                         }
                                         else {
                                             cerr << "ERROR (line: " << lex.LineCount()
<< " part: " << nameId << ") '}' expected" << endl;
                                             currok = false;
                                         }
                                     }
                                     else {
                                         cerr << "ERROR (line: " << lex.LineCount() << "
part: " << nameId << ") '{' expected" << endl;
                                         currok = false;
                                     }
                                 }
                                 break;
                                 default : {
                                     cerr << "ERROR (line: " << lex.LineCount() << ") '"
<< lex.CurrStr() << "' not valid" << endl;
                                     currok = false;
                                     lex.GetToken();
                                 }
                             }
                         } while (lex.CurrToken()!=LexGDL::END && lex.CurrStr()!="}");
                         if (lex.CurrStr()!="}") {
                             cerr << "ERROR (line: " << lex.LineCount() << " part: " <<
nameId << ") '}' expected" << endl;
                             currok = false;
                         }
                         ok &= currok;
                     }
                     else {
                         cerr << "ERROR (part: " << nameId << ") duplicate" << endl;
                         delete part;
                         ok = false;
                     }
                 }
                 else {
                     cerr << "ERROR (line: " << lex.LineCount() << " part: " << nameId <<
") '{' expected" << endl;
                     ok = false;
```

```
                }
            }
            else {
                cerr << "ERROR (line: " << lex.LineCount() << ") part name expected" <<
endl;
                ok = false;
            }
            lex.GetToken(); // get new token
        } while (lex.CurrToken()!=LexGDL::END && lex.CurrStr()!="}");
        lex.GetToken();
        if (lex.PrevStr()!="}") {
            cerr << "ERROR (line: " << lex.LineCount() << ") '}' expected" << endl;
            ok = false;
        }
    }
    else {
        cerr << "ERROR (line: " << lex.LineCount() << ") '{' expected" << endl;
        ok = false;
    }
    return ok;
}

bool Part::LineIntegral(double& result, int sections)
{
    bool ok=(lineIntegral>=0);
    if (!ok) {
      result = 0;
      if (!fluxEntry.empty() && !fluxExit.empty()) { // needs entry and exit curves
          Curves newlist;
          Curves checklist;
          CurvesIterator entryiter=fluxEntry.begin(); // point to first curve
          Curve* curve=(*entryiter);
          Coord* coord=(curve->GetEnd()); // get the first curves end node
          newlist.push_back(curve); // add to new list
          checklist.Add(curve); // do same to check list
          CurvesIterator curvesiter=curves.begin();
          Coord* other;
          do { // build list
              other = NULL; // other coord must be null at start of search
              curvesiter=curves.begin(); // start search from begining
              do { // find curve that has this current coord
                  if (!checklist.Find(*curvesiter)) // ignore any curves already added
to linked list
                      other=((*curvesiter)->OtherIfValid(coord)); // get other coord if
valid coord
                  if (!other) curvesiter++; // go to next curve
              } while (curvesiter!=curves.end() && other==NULL); // while not end of
curves in part
              if (other && curvesiter!=curves.end()) { // valid other coord
                  curve=*curvesiter; // next curve is current in part curves
                  newlist.push_back(curve); // add to linked list
                  checklist.Add(curve); // do same to check list
                  coord=other; // prepare for next pass
              }
          } while (other); // still getting valid curves
          // if all curves have been transfered then must be a closed topology
          ok = (newlist.size()==curves.size());
          if (ok) { // carry on with performing the line integral
              bool upadd = true;
              double lengthU = 0;
              double lengthD = 0;
              CurvesIterator listiter=newlist.begin();
              do { // step up through curve list changing total variable as exit curves
are passed
                  if (!fluxEntry.Find(*listiter)) { // check not an entry curve
                      if (!fluxExit.Find(*listiter)) {
                          if (upadd)
                              lengthU += (*listiter)->Length();
                          else
                              lengthD += (*listiter)->Length();
                      }
                      else { // move to down section of curve list
                          upadd = false;
                      }
                  }
                listiter++;
            } while(listiter!=newlist.end());
            bool validU=(lengthU!=0.0);
            bool validD=(lengthD!=0.0);
            if (validU || validD) { // at least one length must not be zero
                double incU=lengthU/sections; // one of the increments may be zero so
                double incD=lengthD/sections;  // miss that section from the iteration
                CurvesIterator iterU=newlist.begin();
                Curve* prevU = *iterU;
                while (iterU!=newlist.end()?fluxEntry.Find(*iterU):false) { // move up
past the entry curve
                    prevU = *(iterU++);
                }
                Curve* curveU = *iterU;
                CurvesReverseIterator iterD=newlist.rbegin(); // reset to end of
section for down list iteration
                Curve* prevD = *iterD; // set previous to the  known entry curve
                while (iterD!=newlist.rend()?fluxEntry.Find(*iterD):false) { // move
down past any entry curves
                    prevD = *(iterD++);
                }
                Curve* curveD = *iterD;
                // start stepping through the curves with calculated increment
                double distU=0;
                double distD=0;
                if (validU) {
                    distU += incU;
                    lengthU -= incU;
                }
                if (validD) {
                    distD += incD;
                    lengthD -= incD;
                }
                Coord* nodeU=SharedCoord(prevU,curveU);
                Coord* nodeD=SharedCoord(prevD,curveD);
                if (nodeU!=NULL && nodeD!=NULL) {
                    Vector2d currposU=nodeU->GetPos();
                    Vector2d prevposU=currposU;
                    Vector2d currposD=nodeD->GetPos();
                    Vector2d prevposD=currposD;
                    do {
                        // check for curves long enough for the increment
                        if (validU) {
                            while (iterU!=newlist.end() && distU>curveU->Length()) {
// go to next curve

                                distU-=curveU->Length(); // subtract smaller curve

length from distance

                                prevU=curveU;
                                curveU=*(++iterU); // go to next curve and make current
                            }
                            // find new position
                            prevposU=currposU;
                            currposU=curveU->FractionPos(SharedCoord(prevU,curveU),
distU/curveU->Length());
                        }
                        if (validD) {
                            while (iterD!=newlist.rend() && distD>curveD->Length()) {
// go to next curve

                                distD-=curveD->Length(); // subtract smaller curve

length from distance

                                prevD=curveD;
                                curveD=(*(++iterD)); // go to next curve and make

current

                            }
                            // find new position
                            prevposD=currposD;
                            currposD=curveD->FractionPos(SharedCoord(prevD,curveD),
distD/curveD->Length());
                        }
                        // calc of area
                        double area = fabs(length/cos(frame-
>GetSkew()))*(fabs(Mag(currposU-currposD))+fabs(Mag(prevposU-prevposD)))/2;
                        if (minArea==0 || area<minArea) minArea=area;
                        double dist = (fabs(Mag(currposU-prevposU))+fabs(Mag(currposD-
prevposD)))/2;
```

```
                                result += dist/area;
                                // go to next position
                                if (validU) {
                                    distU += incU;
                                    lengthU -= incU;
                                }
                                if (validD) {
                                    distD += incD;
                                    lengthD -= incD;
                                }
//                                cout << "Name: " << GetName() << " distU: " << distU << "
distD: " << distD << endl;
                                // make sure list of curves isn't exhausted and both positions
are not on exit curves
                            } while (iterU!=newlist.end() && iterD!=newlist.rend() &&
(validU?!fluxExit.Find(curveU):true) && (validD?!fluxExit.Find(curveD):true) &&
lengthU>=0 && lengthD>=0);
                        }
                    }
                }
            }
//          cout << "Name: " << GetName() << " Integral: " << result << endl;
        if (ok) lineIntegral=result;
    } else {
        result=lineIntegral;
    }
    return ok;
}
```

```
// 2d vector utuility class
// (c) D. Downes 96
// Impliments basic operations for a 2d vector class.
// Internal storage of the coordintes is cartesian

#ifndef VECTOR2D_CLASS_DECLARED
#define VECTOR2D_CLASS_DECLARED

#include <cmath>
#include <iostream>

class Vector2d
{
protected:
  double x,y;
public:
  enum Type {CART, POLAR};

  Vector2d()  { x=0; y=0; }                                // no argument constructor
  Vector2d(const Vector2d& v)  { x=v.x; y=v.y; }           // 1 argument constructor
  Vector2d(double sx, double sy)  { x=sx; y=sy; }          // 2 argument constructor
  Vector2d(Type system, double set1, double set2) {
    x=(system==CART)?set1:(set1*cos(set2));
    y=(system==CART)?set2:(set1*sin(set2)); }              // 3 argument constructor

  double X()  { return x; }                                // x value of vector
  double Y()  {  return y; }                               // y value of vector

  int operator == (const Vector2d& v)  {
    return (x==v.x && y==v.y); }                           // comparison
  int operator != (const Vector2d& v)  {
    return (x!=v.x && y!=v.y); }


  Vector2d& operator = (const Vector2d& v) {
    x=v.x;
    y=v.y;
    return *this; }                                        // assignment
  Vector2d& operator () (double set1, double set2)  {
    x=set1;
    y=set2;
    return *this; }                                        // cartesian values
  Vector2d& operator () (Type system, double set1, double set2)  {
    x=(system==CART)?set1:(set1*cos(set2));
    y=(system==CART)?set2:(set1*sin(set2));
    return *this; }                                        // polar or cartesian values

  Vector2d& operator += (const Vector2d& v)  {
    x+=v.x;
    y+=v.y;
    return *this; }                                        // addition with equal
  Vector2d& operator -= (const Vector2d& v)  {
    x-=v.x;
    y-=v.y;
    return *this; }                                        // subtraction with equal
  Vector2d& operator *= (double s)  {
    x*=s;
    y*=s;
    return *this;  }                                       // scaler multilication with equal

  friend Vector2d operator + (const Vector2d& a, const Vector2d& b)  {
    return Vector2d(a.x+b.x,a.y+b.y);  }                   // addition
  friend Vector2d operator - (const Vector2d& a, const Vector2d& b)  {
    return Vector2d(a.x-b.x,a.y-b.y);  }                   // subtraction
  friend Vector2d operator * (double a, const Vector2d& b)  {
    return Vector2d(b.x*a,b.y*a);  }                       // scaler multiplication
  friend Vector2d operator * (const Vector2d& b, double a)  {
    return Vector2d(b.x*a,b.y*a);  }
  friend Vector2d operator / (const Vector2d& b, double a)  {
    return Vector2d(b.x/a,b.y/a);  }                        // scaler division

  friend ostream& operator << (ostream& out, Vector2d& v)  {
    out << v.x << ", " << v.y;
    return out;  }                                         // output
  friend istream& operator >> (istream& in, Vector2d& v) {
    in >> v.x >> v.y;
    return in;  }                                    // input

  friend double Mag(const Vector2d& v)  {
    return sqrt(v.x*v.x+v.y*v.y);  }                  // magnutude of vector
  friend double Angle(const Vector2d& v)  {
    double mag = sqrt(v.x*v.x+v.y*v.y);
    double angle = (mag?acos(v.x/mag):0);
    return ((v.y<0)?(2*M_PI-angle):angle);  }        // angle of vector wrt x axis
  friend Vector2d Norm(const Vector2d& v)  {
    return Vector2d(-1.0*v.y,v.x);  }                // normal of vector
  friend Vector2d UnitNorm(const Vector2d& v)  {
      double mag = sqrt(v.x*v.x+v.y*v.y);
      if (mag==0)
        return Vector2d(0,0);
      else
          return Vector2d(-1.0*v.y/mag,v.x/mag);  }    // unit normal of vector
  friend double Dot(const Vector2d& a, const Vector2d& b)  {
    return (a.x*b.x+a.y*b.y);  }                      // dot product
  friend double Cross(const Vector2d& a, const Vector2d& b)  {
    return (a.x*b.y-a.y*b.x);  }                      // cross product
};

#endif
```

Appendices

I

'coincident area' program listing

```
// Program to calculate the coincident area under the rotor and stator teeth
// Uses brute force numerical integration of area under rotor and stator

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream.h>
#include <stdlib.h>
#include <math.h>

float length=0.1; // mutual length of rotor and stator
float rotorskew=3.5*2*M_PI/360;  // skew of rotor
float statorskew=0;  // skew of stator
float rotorangle=5.64*2*M_PI/360; // angle in radians of rotor tooth
float statorangle=6.88*2*M_PI/360; // angle in radians of stator tooth
float radius=0.045; // average radius in meters of the stator and rotor
int sections=100; // number of sections to use in the numerical aproximation

// given a value pos, which is a fraction of the total tooth pithch, calculate the high
boundary value
float calcHi(float skew, float length, float radius, float angle, float pos)
{
  float x=pos*(radius*angle+fabs(tan(skew)*length));
  float y=fabs(x/tan(skew));
  return (y>length?length:y);
}

// given a value pos, which is a fraction of the total tooth pithch, calculate the low
boundary value
float calcLo(float skew, float length, float radius, float angle, float pos)
{
  float x=(1-pos)*(radius*angle+fabs(tan(skew)*length));
  float y=length-fabs(x/tan(skew));
  return (y<0?0:y);
}

int main(int argc, char *argv[])
{
  float lr=radius*rotorangle+fabs(tan(rotorskew)*length); // total length of rotor tooth
  float ls=radius*statorangle+fabs(tan(statorskew)*length);  // total length of stator
tooth
  float incx=(lr+ls)/sections; // increment of rotor past stator
  for (float x=-ls; x<=lr; x+=incx) {  // loop for the full motion of the rotor
    float xs=x>0?x:0; // find coincident start of rotor and stator
    float xe=x+ls<lr?x+ls:lr;  // find coincident end of rotor and stator
    float area=0;
    float incz=(xe-xs)/sections;  // setup increment for area
    if (incz>0) {  // only perform calculation if valid coincident area
      for (float z=xs; z<=xe; z+=incz) { // integrate over the coincident area
         float pr=z/lr; // current point on rotor as ratio of length
         float ps=xs>0?z/ls:(z-x)/ls; // current point on stator as ratio of length
         float rhi=calcHi(rotorskew,length,radius,rotorangle,pr); // calculate the
current values
         float rlo=calcLo(rotorskew,length,radius,rotorangle,pr); // for the position of
the rotor teeth
         float shi=calcHi(statorskew,length,radius,statorangle,ps);
         float slo=calcLo(statorskew,length,radius,statorangle,ps);
         if (rhi>slo && shi>rlo) {  // check teeth are coincident
            area+=((rhi<shi?rhi:shi)-(rlo>slo?rlo:slo))*incz;  // approximate coincident
area
         }
      }
    }
    cout << (x+(ls-lr)/2)/radius << ", " << area << endl; // output position and area
data
  }
  return EXIT_SUCCESS;
}
```