Fall 2013

A Dynamic Magnetic Equivalent Circuit Model For Design And Control Of Wound Rotor Synchronous Machines

Xiaoqi Wang
Purdue University

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Xiaoqi Wang

Entitled

A Dynamic Magnetic Equivalent Circuit Model for Design and Control of Wound Rotor Synchronous Machines

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

STEVEN D. PEKAREK
Chair

ANAND RAGHUNATHAN

OLEG WASYNCZUK

SCOTT D. SUDHOFF

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): STEVEN D. PEKAREK

Approved by: M. R. Melloch 10-09-2013
Head of the Graduate Program Date

A DYNAMIC MAGNETIC EQUIVALENT CIRCUIT MODEL FOR DESIGN AND CONTROL OF WOUND ROTOR SYNCHRONOUS MACHINES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Xiaoqi Wang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2013

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

Foremost, I would like to gratefully and sincerely thank my advisor, Prof. Pekarek, for his support, guidance and patience. Although standing in front of his immense knowledge was like a sand looking at the sea, he gave me the courage and eagerness to make a jump and become part of the ocean. His mentorship and friendship are great treasures of my life. I would also like to thank Professors Sudhoff, Wasynczuk, and Raghunathan for the added support they have provided and for serving on my advisory committee. I acknowledge the financial support of Kohler Power Systems, the National Science Foundation under grant 1102303-ECCS, and the Office of Naval Research under grant N000-14-08-1-0080.

I truly appreciate for the endless love and unconditional support from my parents, even though they barely know what I was exactly doing on the other side of the earth. In addition, I owe many thanks to Michelle, Nir, Ross, Ahmed, Rob, Dan, Yimin, Grant, Anand, Rick, and Adam, for making this work possible. Finally, it is a blessing for me to meet the Szetos and the Cantonese groups, from whom I see light.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

ABSTRACT

Wang, Xiaoqi Ph.D., Purdue University, December 2013. A Dynamic Magnetic Equivalent Circuit Model for Design and Control of Wound Rotor Synchronous Machines. Major Professor: Steve Pekarek.

Recently, a new magnetic equivalent circuit (MEC) model was developed to support automated multi-objective design of wound-rotor synchronous machines (WRSMs). In this research, the MEC model and its application have been enhanced. Initial enhancement has focused on using the MEC model to explore machine design and control as a unified problem. Excitation strategies for optimal steady-state performance have been developed. The optimization is implemented in two phases. First, stator and field excitation at rated power is obtained as part of a WRSM design in which the objectives are to minimize machine mass and loss. Second, a map between current and torque is generated using a single-objective optimization in which core, resistive, and switch conduction loss are minimized. Optimal as well as sub-optimal and traditional controls are studied and compared. An interesting result is that a relatively straightforward field-oriented control is consistent with a desire for mass/loss reduction and control simplicity. The applicability of the excitation to systems in which prime mover angular velocity varies and is (un)controllable is considered, as well as its impact on machine design.

A second contribution has been the derivation of a mesh-based dynamic MEC model for WRSMs. As part of this effort, a reluctance network has been derived to model flux distribution around damper bar openings. The reluctance network is applicable to a user-defined damper bar pattern, which enables the study of optimal damper bar placement. In addition, Faraday's law is applied to establish a state model in which stator, field, and damper winding flux linkages are selected as state variables. The resulting coupled MEC/state model is solved to obtain transient machine dynamics, including

damper bar currents. In addition, skew of the rotor pole is incorporated using a multi-slices model. The proposed dynamic model opens new paths for exploration. Perhaps most significantly, it enables rigorous design of coupled synchronous machine/diode rectifier systems, which are used in numerous applications, but are often designed using rules of tradition created prior to the availability of efficient numerical simulation.

1. INTRODUCTION

Wound-rotor synchronous machine (WRSM) drive systems are widely used in utility, air, ship, and portable power generation. Numerous models including lumped parameter, Finite Element (FE), and magnetic equivalent circuits have been developed for electric machine design and performance analysis. A growing interest in the application of automated design optimization methods such as population-based design (PBD) motivates the need for an accurate and efficient machine model. Recently, a mesh-based steady state magnetic equivalent circuit (MEC) model has been proposed in [1] to address this need.

An initial focus of this research is to use the model proposed in [1] to explore excitation strategies that consider machine design and control as a united problem. A drive system that consists of a WRSM connected to a 3-phase active rectifier and a prime mover that holds the rotor speed constant is studied. In the optimization and excitation development process, several interesting results are observed. First, the *d*-axis current that is selected tends to be negative, which contrasts what would be expected from a traditional *qd* model, since the resulting salient torque opposes that of the torque produced by stator/field interaction. Second, it is shown that utilizing *qd* models with/without saturation incorporated along the *d*-axis leads to suboptimal excitation that is appreciably different than obtained from a MEC over much of the expected operating region. Third, it is observed that similar to the strategies considered in [2]-[4], a look-up table is the most convenient means to implement the optimal torque versus current map.

It is recognized that the traditional methods of excitation are often used for their relative simplicity, speed of response (i.e. very fast torque response), and in some cases the attractiveness of having closed form expressions that relate torque and current. Therefore, two alternative controls are considered using the MEC-based optimization strategy. In one, the field current is held constant (similar to a field-oriented control), and

the *q*- and *d*-axis current versus load is determined that minimizes overall system loss. In another, the *d*-axis current is held fixed at zero and the field and *q*-axis current versus load is selected to minimize system loss. Through evaluation of both of these sub-optimal strategies, it is found that a very simple field-oriented-type control (simplified control) approach can be established in which *q*-axis current maps linearly with torque, *d*-axis current is held at zero, and the field current is held constant. Since the resulting torque/*q*-axis current map is linear, the need to utilize a look-up table for control is eliminated. Moreover, there is a relatively minor impact on overall system loss.

Furthermore, although the machine was originally designed for fixed-speed operation, the applicability of the simplified control is considered for the case in which prime mover angular velocity varies but is not controlled by the electrical system. Envelopes that characterize the constant torque and constant output power region over a wide speed range are established to explore the impact that the 'optimal' and simplified controls have on the overall operating envelope of the machine/drive. Interestingly, under variable speed operation, it has been found that the power loss generated by the 'optimal' and simplified controls at different rotor speeds is relatively minor. However, it is also found that if one holds *d*-axis current at zero and only uses field current for field weakening, the range over which constant power can be achieved is reduced. Therefore, in an additional study, a new machine design study is performed in which Pareto-optimal fronts are established for a variable-speed drive in which one assumes an 'optimal' control and one in which the simplified control is applied. A comparison of the fronts and machines is used to assess the impact of the control on the design of a machine. Finally, the extension/applicability of the techniques to cases in which prime mover angular velocity varies and is controllable is discussed.

A second focus of this research is to develop a dynamic MEC model of a WRSM starting with the steady state MEC model in [1]. As part of this effort, a reluctance network has been derived to model flux tubes of stator tooth tips and damper bar openings. Damper bar leakage reluctance has been introduced to model the flux distribution around the damper bar openings for the case that the damper bar currents are active. The reluctance network is applicable to assign an arbitrary number of damper bars

placed at an arbitrary depth in the rotor pole tip. This enables a designer to explore alternative damper winding topologies as part of an optimization. In addition, Faraday's law is applied to establish a state model in which winding and damper bar flux linkages are selected as state variables and winding voltage is an input. The resulting coupled MEC/state model is solved to obtain transient machine dynamics, including damper bar currents. An important attribute of the model is that saturation is represented without the need for a relaxation factor, which enables its use as a practical tool in machine design. The proposed MEC model is validated by FEA or hardware results through various tests, including open circuit voltage, three-phase balanced load test, and stand still frequency response.

In order to model skewing effect, the dynamic MEC model is augmented to a multi-slices model with a uniformly shifted angle for each slice. The multi-slices model satisfies the constraint that each slice conveys the same stator, field, and damper currents. The convergence benefit and computational efficiency of the mesh-based MEC model ensure a relatively fast, well-converged solution for a large slice number.

Finally, the optimal design of coupled WRSM/rectifier systems has been explored. There is a desire within the community to understand the tradeoffs between using a machine/active rectifier and a machine/diode rectifier. Of particular interest is the expected difference in the size of the machines required under each topology. One may argue that a dynamic model is not required to assess the difference. However, the steady-state voltage versus current of the machine/diode rectifier is a function of subtransient inductances. In other words, damper bar currents are non-zero in a machine/diode rectifier system. As a result, a dynamic model that includes damper bars is required for rigorous optimization. Once the dynamic model validation was complete, GA-based optimization studies have been performed to compare the Pareto-optimal fronts of the machine/rectifier topologies.

1.1 Literature Review of WRSMs Control

A goal of exploring excitation strategies for synchronous machines is to consider its role when one attempts to minimize the active mass and power loss. Reduction of

active mass reduces component cost and also improves portability. Reducing power loss saves fuel, reduces emissions, and helps to reduce thermal signature. Presently, three common techniques for control of torque of a WRSM are field oriented control (FOC), maximum torque per ampere (MTPA) control, and direct torque control (DTC).

FOC algorithms in AC machines are intended to create torque versus current maps that are similar to DC machines. Specifically, in a DC machine, a field winding or magnet is used to establish a fixed magnetizing flux along a direct axis. The current in an armature (control) winding is then used to adjust torque. A convenience of an FOC is that (in theory) with the field winding flux held fixed, the torque versus armature current map is linear. A performance advantage is that FOC results in a relatively high torque bandwidth. Specifically electromagnetic torque can be changed nearly instantaneously. A disadvantage of FOC approaches in many AC machines is that at low values of torque, one pays a price of excess loss associated with maintaining a rated field flux. Details of FOC strategies are provided in [5]-[7].

In contrast to FOC, in maximum torque per ampere control (MTPA), a fixed *d*-axis constant field flux is abandoned in lieu of attempting to obtain the most torque from the moving charge. At low values of torque, this translates into lower loss than FOC. However, it does add complexity to the torque versus stator winding current map. It also reduces the torque bandwidth. In [8], a MTPA algorithm for an induction machine was presented and compared to the FOC. Instead of holding *d*-axis current as a constant value, both *qd*-axis currents are regulated to minimize the stator current amplitude for a given torque and speed. Decoupled analytical expressions for torque command in terms of *qd*-axis currents have been developed in [9], [10]. However, core loss and the inductance change due to saturation are not considered in the *qd* model based analysis. Since the power capability and the voltage limit constraints have a significant sensitivity on the machine parameters, especially the saliency ratio X_q/X_d[11], an online adaptive self-tuning parameters estimator and a feed-forward torque correction method are proposed in [12], [13] in order to analyze the effect of saturation and cross-magnetization.

The basic idea of DTC is to directly select a stator voltage vector according to the difference between a reference torque and stator flux linkage and their actual values. In

[14], a DTC scheme was applied to an interior permanent magnet synchronous machine (IPMSM). In [15] and [16], a similar DTC algorithm is used to control a surface PMSM with space vector modulation (SVM) so that a fixed switching frequency can be obtained. In [17] and [18], DTC is applied to a PM-assisted reluctance synchronous machine and an induction machine, respectively, in a starter alternator application. Although DTC is an inherently position sensorless control scheme, accurate stator flux and initial rotor position estimation is required.

However, the majority of the literature has focused on methods for permanent magnet and induction machines. Although well established, there remain interesting questions associated with these controls. Specifically, optimization of the excitation is rarely included as part of the machine design where geometry, turns, and core material are selected. Rather, the torque versus current map is derived subsequent to machine design using lumped parameter models that often assume linear magnetics and/or do not account for core loss or semiconductor loss. As a result, one can question whether these excitation approaches are consistent with design goals of minimization of mass or overall system loss. Indeed, when researchers in [2]-[4] used a finite element model and included core/semiconductor losses in calculating 'optimal' excitation of a wound-rotor synchronous machine, the control was implanted using a look-up table of currents versus speed and torque, rather than an analytical map. This of course raises a question as to whether a look-up table-based approach is required when one does include saturation, core loss, and semiconductor loss in the machine/excitation design.

In the 2 kW system considered in this research, the stator and field current at rated load are obtained as part of a multi-objective machine design optimization that includes 16 design variables. The optimization utilizes evolutionary strategies to establish the Pareto-optimal front between mass versus loss at rated load. Subsequently, the magnetic equivalent circuit is used to establish a map between torque and excitation that minimizes system loss (core, winding, conduction of the switches) at loads less than rated. Within this process, both optimal and sub-optimal control strategies have been developed and compared at rotor speed less than rated. Finally, the tradeoffs and limitations of the

proposed simplified control are explored when the desire is to optimize available torque at speeds beyond rated values.

1.2 Literature Review of Dynamic MEC Modeling

It has been decades since magnetic equivalent circuits (MEC) were introduced for machine analysis. After the basic properties and elements of the MEC were formally defined in 1941 [19], the duality between electric and magnetic circuits was proposed in [20], [21].This subsequently expanded the concept and use of MEC models. In [22], Ostović outlined the fundamental theory and structure of MEC for electric machines that forms the basis for most existing techniques. In his formulations, the MEC model solution is structured using Kirchoff's Current Law (KCL) utilizing nodal analysis.

A mesh-based alternative to the nodal MEC has been receiving more attention over the past decade. In such a formulation, Kirchhoff's Voltage Law (KVL) is applied to establish an algebraic system in which loop flux is an unknown and winding current (MMF) is the input. One of the challenges for the mesh-based MEC is that the flux tubes between stator and rotor appear and disappear as rotor position changes due to rotation. In a recent study [23], a relatively straightforward shape algorithm was proposed to automatically update the loop equations with rotor position. Within the shape algorithm, the airgap permeances are used to identify the number of meshes, and the permeance connections are used to determine the loop configuration so that the coefficients in the KVL equations can be updated. Utilizing the shape algorithm, a detailed steady state mesh-based MEC model for WRSMs is presented in [1].

The MEC model in [1] is used as a basis to derive a model that efficiently predicts the dynamic behavior of WRSMs. This is motivated by the fact that in many designs, dynamic performance is of interest. For example, to determine the voltage regulation characteristic of machine-diode rectifier systems requires one to model subtransient behavior. In addition, in some applications the subtransient inductances are constrained in an attempt to limit fault current. The impact that the constraints on inductance/fault current have on machine mass and efficiency has not been explored. The proposed model is intended to enable such exploration.

In contrast to steady state MEC formulations, dynamic MEC models have received relatively minor attention, particularly for WRSMs. In most cases, the transient responses of electric machines are obtained using equivalent electrical circuits [26]. A voltage behind reactance model with saturation in *d*-axis incorporated is proposed in [27]. Both stator and rotor dynamics are estimated using such model. In [28] and [29], an average-value model is introduced to analyze the transient response of the synchronous machine-rectifier system, in which the synchronous machines are modeled using a reduced order model and a full order model, respectively. In [30], a synchronous machine is modeled using a network formulation in *qd* variables. Magnetizing inductances in both axes are modified to portray saturation.

Of the research that has been placed on deriving dynamic MEC models for electrical machines, there are primarily five common approaches. One is to use a static MEC model to establish the lumped electric parameters of a dynamic machine model. For example, in [31], the winding inductances of an induction machine are determined using a static nodal-based MEC model within each simulation time step of a *q*-*d*-based model of the induction machine. In [32], a nodal-based MEC is applied to establish expressions for the stator winding back-emf and inductance of a non-salient-pole turbo-generator using within an electrical circuit simulation.

In a second approach, G. Slemon introduced what he referred to as a λ-i model in [33], [34], in which duality arguments are used to convert the steady-state MEC and damper bar current/flux linkage relationship into a dynamic electrical circuit consisting of inductors and capacitors. Although dualities can offer convenience, the proposed model structure relies on numerical differentiation to establish the coupling between the machine model and external circuits. This is not favorable for design studies requiring large numbers of evaluation owing to the ill conditioning of difference-based derivative approximations. In addition, the convergence behavior of the proposed model in saturation is unknown.

In a third approach, a differential gyrator model shown in Figure 1.1 is used to couple the electric and magnetic quantities so that the system can be solved as electric circuits with current-controlled voltage sources and voltage-controlled current sources.

One example of such approach is proposed in [35], where the magnetic circuit is represented by electric components using a permeance-capacitance analogy. A more extensive gyrator-based circuit of inverter-fed synchronous machines is presented in [36]. In [36], a gyrator circuit is used to couple the dynamic electric model of the stator and field windings to the MEC of the core so that the WRSM is represented using current-controlled voltage sources and capacitors. To structure the machine model in a gyrator form, winding flux and the rate of change of flux are taken to be analogous to electric charge and current, respectively. Although potentially convenient, a gyrator approach is generally limited to those who intend to use circuit solvers, such as SPICE or PLECS [37], to implement the model. In addition, in [36] the method to include saturation is to set the relative permeability of several iron elements to low values that are constant, rather than to determine values of permeability numerically within the simulation. This approach is more applicable for an analysis of a single machine in which flux levels are known apriori, rather than a design environment without such knowledge. The proposed gyrator model is applied to recent studies [38], [39] to couple the electric and magnetic domains for power electronic transformers, in which a HFMEC model that is considered as modular assembly of flux tubes is used to capture the eddy current dynamics.

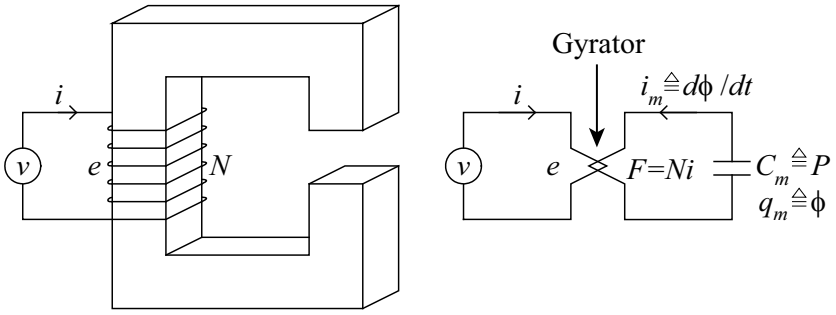


Figure 1.1: Depiction of an inductor and the equivalent electric circuit using a gyrator.

A fourth approach is that the MEC equations for a nodal-based model are differentiated with respect to time so that the node potentials and winding currents become state variables [40]. The inputs to resulting state model are the time changing rate of stator and rotor flux linkage. The outputs of the MEC state model are stator winding and rotor damper bar currents. The MEC state equations can be coupled to models of

external circuits that accept voltage as a model input, with current as a state. Instead of using a nonlinear solver, the permeability is calculated directly from the states, i.e. node potentials and currents, to avoid numerical issue. However, neither simulation nor hardware results are provided in [40]. Using a similar formulation, a nodal-based steady-state MEC model of a WRSM is proposed in [41]. A challenge with this approach is numerical convergence, which was cited in [42] and has been identified as an issue in nodal-based MEC formulations in general. Methods to address convergence using relaxation factors have been proposed, but add complexity and computational cost.

A fifth approach is one in which Faraday's law is used in tandem with the algebraic MEC relationships to establish a system of differential algebraic equations. Typically in such an approach, winding flux linkages are selected as the state variables. The winding flux linkages are established through numerical integration and used as an input to the algebraic MEC equations. The winding current is an output of the MEC model and is used along with winding voltage as an input for the winding flux linkage state equations. This type of formulation has been used to model induction machines under healthy [43], [44], [45] and faulted conditions [46], [47]. In [44], [45], the MEC network is expanded into 3-D so that local saturation, leakage and skewing can be represented. Although flux linkage is used as a state variable in [43]-[47], the formulations are all based upon a nodal-based MEC. It has been shown in [42] that mesh-based MECs have better convergence properties in components with nonlinear magnetic materials. In [48], mesh-based MEC techniques are applied to take place of a FE model with a MEC. The combined FE-MEC model is coupled to external electric circuit by augmenting the system equations. The augmented system is discretized in time and solved by numerical methods.

In this research, Faraday's law is used in tandem with the MEC expressions to establish a system of differential algebraic equations. This general approach has been applied in the dynamic models of machines using nodal-based MECs in [43]-[47], but has received limited attention in design owing to convergence issues. The judiciously restructured/scaled mesh-based model proposed herein has the strong convergence

properties necessary for population-based design. Indeed, the model is solved without the need for a relaxation factor to obtain convergence.

To model dynamic behavior, permeances are derived that represent the flux distribution of a damper winding structure that consists of an arbitrary number of bars of arbitrary radius with/without end connections between poles. This enables a designer to explore alternative damper winding topologies as part of an optimization. The model is readily coupled to models of external balanced or unbalanced electrical circuits, including passive or active rectifiers. The model is validated through comparison with hardware experiment as well as a finite element-based model.

2. BACKGROUND

This chapter describes the background information on the MEC modeling and GA optimization techniques applied within this research. The fundamental physics of MEC modeling are presented in the first section. The underlying theory of GA optimization is illustrated in the second section. The steady state mesh-based MEC model proposed in [1] is used as reference in this research and a brief review of it is given in the third section.

2.1 Magnetic Equivalent Circuit Basics

The basic element of the MEC is a flux tube defined by a volumetric space between two planes of equal magnetic scalar potential. Magnetic flux is assumed to enter the flux tube perpendicular to one equipotential plane to exit perpendicular to the other plane. The flux does not leave the boundaries of the volume except at the end surfaces. A diagram of a flux tube is shown in Figure 2.1. It is noted that u_1 and u_2 are the values of magnetic scalar potential at the two planes. The configurations and connections of each flux tube representing an electric machine depend on an analyst's knowledge and understanding of flux behavior.

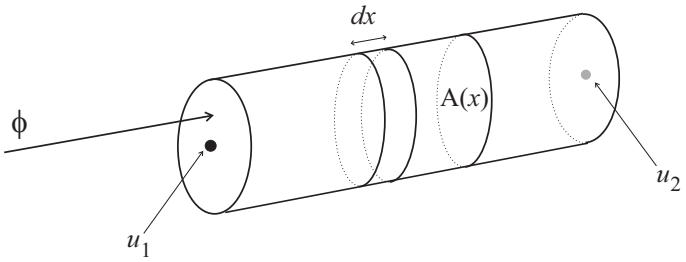


Figure 2.1: Uniform flux tube.

Conceptually, a flux tube is represented as a circuit element that is similar to the elements of an electric circuit. In particular, the equipotential planes are treated as equipotential nodes in an electric circuit, while the flux through the tube is analogous to the current in an electric conductor. Therefore, as a counterpart of resistance in electric circuit, the flux tube can be represented using a magnetic reluctance that is defined as,

$$R = \frac{u_1 - u_2}{\phi} \tag{2.1}$$

Similar to the calculation of electric resistance, the reluctance of a flux tube with uniform cross-sectional area and length can be calculated as

$$R = \frac{l}{\mu A} \tag{2.2}$$

where l is the length of the flux tube, A is the cross-sectional area of the flux tube, and μ is the permeability of the flux tube material. The inverse of reluctance is defined as the permeance (P), and can be expressed as the inverse of either (2.1) or (2.2).

In most cases, the flux tubes have non-uniform geometries where either the length or area changes along the flux path. In such applications, it is convenient to discretize the flux tube into differential sections and compute the overall reluctance or permeance using integration. Figure 2.2 shows two types of non-uniform flux tubes. For a flux tube with a varying area as shown in Figure 2.2(a), the reluctance can be derived as follows,

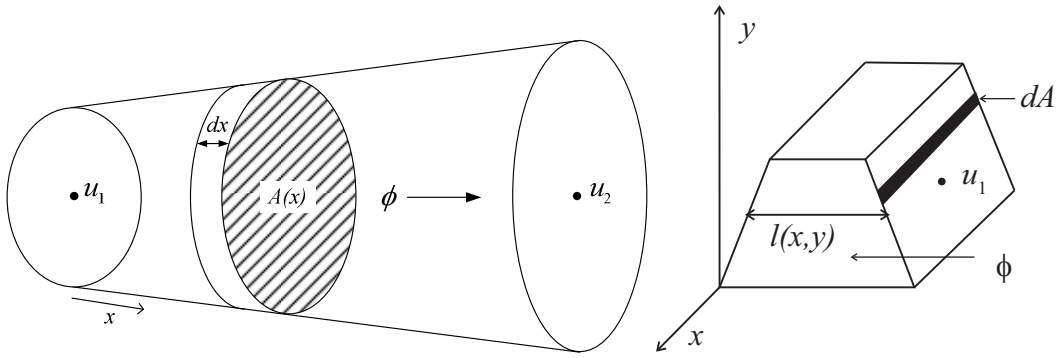
$$R = \int \frac{dx}{\mu A(x)} \tag{2.3}$$

where dx is the differential tube length and $A(x)$ is the position-dependent tube area. On the other hand, for a flux tube with a varying length as shown in Figure 2.2(b), the permeance can be calculated as follows,

$$P = \int_A \frac{\mu dA}{l(x,y)} \tag{2.4}$$

where dA is the differential flux tube area and $l(x, y)$ is the position dependent length. For the case that the length has an insignificant amount of variation, the mean path length can be approximated as the uniform length of the flux tube.

In general, an analytical expression of the flux tube area or length is required for calculating the reluctance or permeance respectively.



(a) Non-uniform area (b) Non-uniform length

Figure 2.2: Non-uniform flux tubes.

In an MEC, magnetomotive force (MMF) is used to represent the effect of electric current on the magnetic system. An MMF source is analogous to a voltage source in an electric circuit and its value can be determined using Ampere's law,

$$\oint \vec{\mathbf{H}} \bullet \mathbf{d}\vec{\mathbf{l}} = Ni \tag{2.5}$$

where $\vec{\mathbf{H}}$ is the magnetic field, and the integral is taken over a closed surface that encloses N turns of a current-carrying conductor. In (2.5), the MMF source F is defined as,

$$F \triangleq Ni \tag{2.6}$$

An example showing how an electromagnetic system can be related to an equivalent magnetic circuit is presented in Figure 2.3, where the magnetic behavior of the UI inductor on the left is modeled using the equivalent circuit on the right. The inductor winding is represented as the MMF source F_{ui}, the steel I component is represented by reluctance R_{i}, the steel U component is represented by reluctances R_{ub} and R_{us}, the flux tubes in the airgap are represented by the reluctance R_{ag}, and the leakage flux tube is represented by R_{l}.

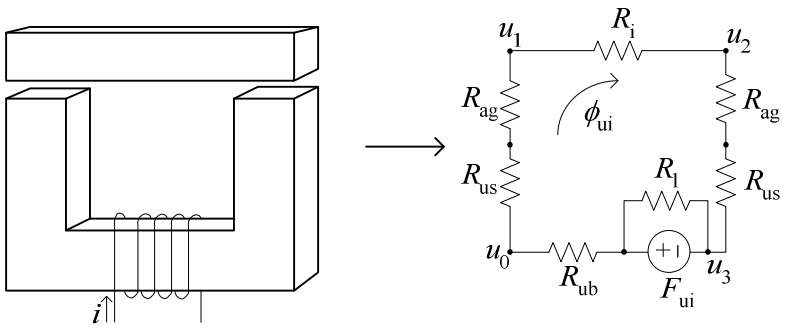


Figure 2.3: UI inductor and magnetic equivalent circuit.

Since the UI inductor in Figure 2.3 has a uniform geometry, the reluctance elements can be calculated using (2.2). The airgap reluctance and leakage reluctance have the permeability of free space, while the steel reluctances are calculated based upon the permeability of the steel material determined from its anhysteretic *B*-*H* curve. The steel reluctances are constant if the system is operating in the linear region. On the other hand, if saturation is considered, a nonlinear solver is needed to calculate to reluctances in the steel. Once the magnetic circuit network is created, the system can be described using a set of equations based upon common circuit analysis technique such as nodal or mesh analysis. Using appropriate solution algorithms, the flux (ϕ_{ui}) and/or node potentials (u_0-u_3) can be calculated.

2.2 Optimization Tools – Genetic Algorithm & Multi-Objective Optimization

A genetic algorithm (GA) based upon the theory of biological evolution is applied in this research to execute the single and multi-objective optimization. The essential steps of a GA are presented in [49] and shown in Figure 2.6. In the algorithm, each individual contains a set of genes. For the initial population, genes are selected arbitrarily within a user-defined range. Over subsequent generations, the population of individuals evolves based upon the evaluation of a user-defined fitness function.

The basic steps of evolution include selection, crossover, and mutation. During selection, an individual is considered as a parent to the next generation of designs and placed into a mating pool. During crossover, parts of the genetic information are exchanged between parents so that new individuals can be formed. Some parents will

stay without crossover to the next generation. At last, random gene mutation takes place in a small percentage of the population. Through repetition, the evolution process leads to a final population.

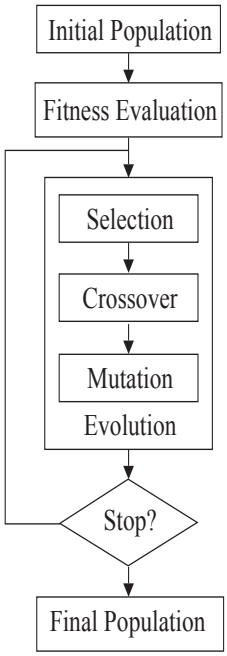


Figure 2.4: Basic steps of a genetic algorithm.

GA can be configured to solve single or multi-objective optimization problems. For single objective optimization, it is relatively straightforward that the best design is determined by determining the individual from the final population that has a maximum fitness. On the other hand, multi-objective optimization (MOO) employs a fitness value for each objective. The idea of dominance is thus introduced to evaluate how fit an individual is in general. An individual, x_1, is defined to dominate another individual, x_2, if x_1 performs as well as x_2 in all objectives and better than x_2 in at least one objective. For example, let mass and loss be the two objectives in the design. x_1 dominates x_2 if x_1 has the same loss and a better mass than x_2. However, neither x_1 nor x_2 are considered to dominate each other if x_1 has a better mass and a worse loss than x_2. If an individual is not dominated by any other members of the population, then it is considered as non-dominated. The Pareto-optimal set is defined as a collection of all of the non-dominated

solutions in the final population. Plotting this set in the objective space yields a boundary termed the Pareto-optimal front [50].

In this research, all GAs are executed using a Purdue-developed Genetic Optimization Systems Engineering Toolbox (GOSET) [51]. In this toolbox, more functionality, including elitism, migration, death, and diversity control, has been introduced than is shown in Figure 2.4. GOSET has been selected for ease of availability and its strong performance in addressing related machine optimization problems [52] and [53].

2.3 Reference MEC Model

2.3.1 Building MEC model

The steady-state MEC network upon which this research was initially based is designed to model the performance of a salient-pole WRSM with an arbitrary number of poles, integer number of slots/pole/phase, and symmetric winding configuration. Figure 2.5 shows an example cross section of a 4-pole WRSM. The flux tube geometries can be defined using the geometric variables indicated in Figure 2.5. The *q*-, *d*-, and *as*-axis of the machine are also listed. It is noted that mechanical rotor position θ_{rm} is defined by the position of *q*-axis with respect to the *as*-axis.

Figure 2.6 shows a representative network of the proposed MEC, wherein loop flux Φ is defined in the clockwise direction. The airgap reluctances correspond to the nonzero airgap permeances at the respective θ_{rm}. Within the network, each stator and field coil becomes a MMF source in the loop where the respective current locates.

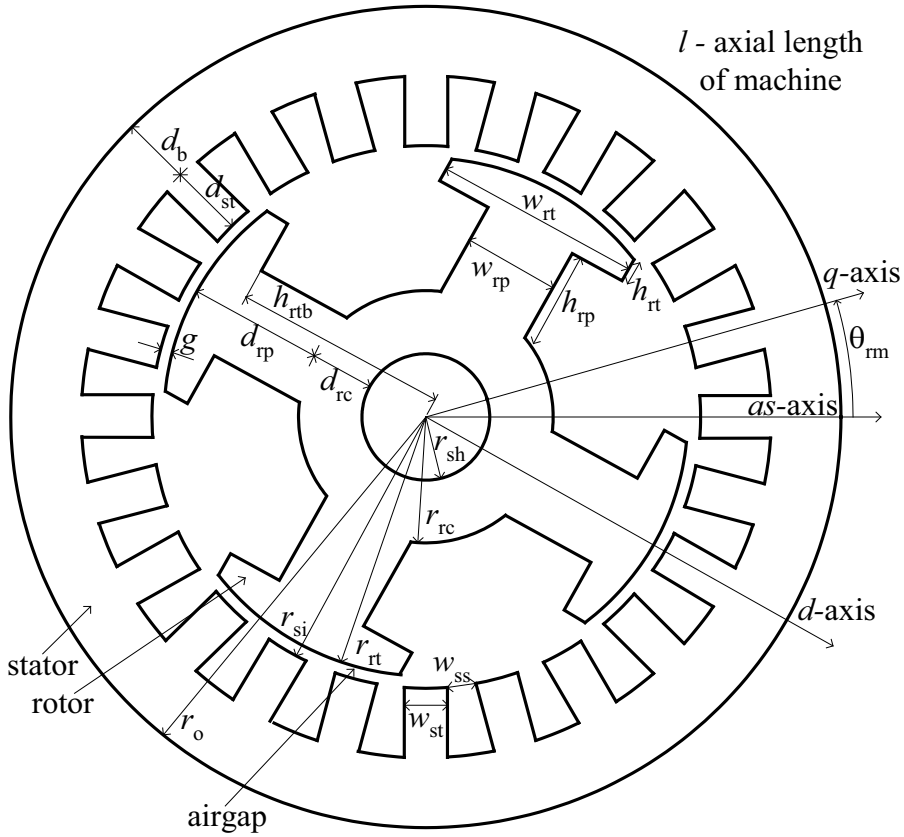


Figure 2.5: Representative cross-section of a WRSM.

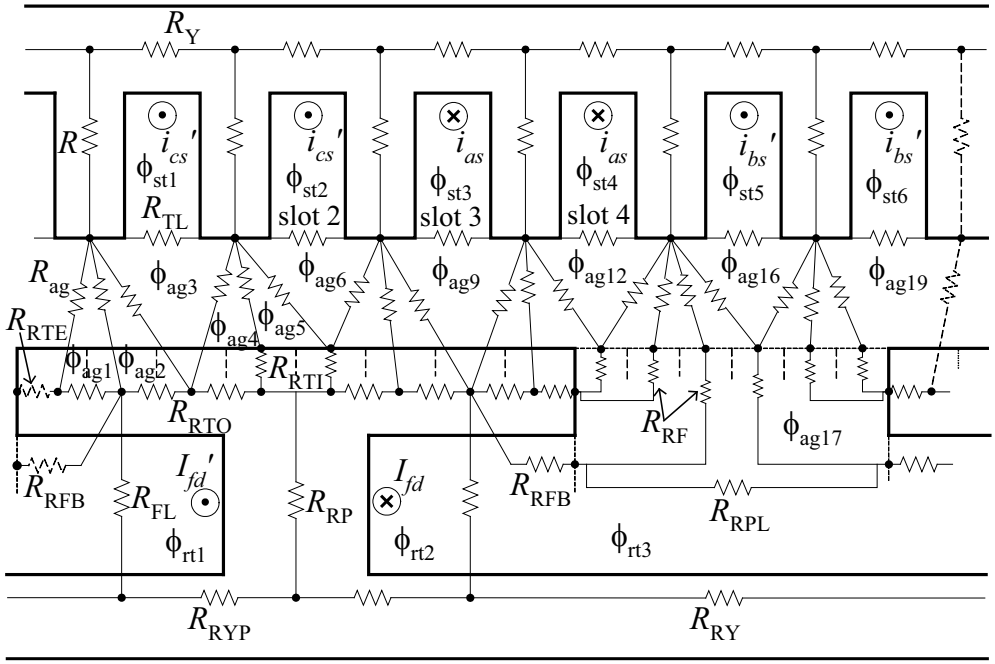


Figure 2.6: Representative WRSM MEC.

Details of the calculation of reluctance values used in the original model are presented in [1]. A few details related to this model are of note. First, the stator tooth flux tubes do not include stator tooth tips. Second, the airgap flux tubes are connection between the stator and rotor, which varies according to rotor position. In order to calculate the airgap permeances, the stator, rotor pole, and inter-polar region are discretized into subsections. Third, the rotor poles and rotor shank flux tubes are considered solid pieces, which have no damper bar slot. Fourth, the inter-polar region can be divided into four types of flux tubes, field winding leakage (R_{FL}), rotor pole leakage (R_{RPL}), rotor fringing (R_{RF}), and rotor fringing to the bottom of the pole tip (R_{RFB}).A challenge of implementing the MEC model shown in Figure 2.6 is that the reluctance network in the airgap changes with rotor position. Moreover, the values of the airgap permeances are dependent upon the dimensions of the stator teeth and rotor pole tip (genes of the GA). To enable a relatively large search space, the derivations of airgap permeance must account for many potential tooth width/pole body width combinations. In [23], the potential airgap permeance calculations was categorized into $5\times 8=40$ conditions, according to the relation of the width of stator tooth tip, stator tooth slot, and rotor pole tip section, as well as the relation of the position of stator tooth and rotor pole tip section.

As part of the initial research effort, tooth tips are added into the respective case conditions. Although at first glance one would consider that all the cases would need to be re-written, a straightforward alternative was developed. Specifically, the original stator tooth flux tube is reshaped as stator tooth tip flux tube in the updated model. An extra component called stator tooth shank is added in between with stator yoke and stator tooth tip, which shares the same flux loop with the stator tooth tip. By doing so, the interface between airgap and stator will not change, and what effectively happens is the automated program now sees a larger stator tooth width because the tips are included.

Once reluctance values in the network have been determined, a system of nonlinear algebraic equations related to each loop can be established based upon KVL,

$$\mathbf{A}_{\mathrm{R}}^{(nl\times nl)}\boldsymbol{\varphi}_{\mathrm{l}}^{(nl\times 1)}=\mathbf{F}_{\mathrm{l}}^{(nl\times 1)} \tag{2.7}$$

where $\mathbf{A}_R$ is a symmetric matrix composed of reluctances, $\boldsymbol{\varphi}_l$ is a vector of loop fluxes, $\mathbf{F}_l$ is a vector of MMF sources, and nl is the number of loops. The components of (2.7) can be expanded as

$$\boldsymbol{\varphi}_l = \left[\phi_{st1} \ \cdots \ \phi_{stns} \ \ \phi_{rt1} \ \cdots \ \phi_{rtnr} \ \ \phi_{ag1} \ \cdots \ \phi_{agna} \right]^T \tag{2.8}$$

where the subscripts "st", "rt", and "ag" denote loop fluxes in the stator, rotor, and airgap, respectively, and the subscripts "ns", "nr", and "na" denote the number of the stator slots per pole, the number of rotor loops per pole, and the number of airgap loops per pole, respectively. Using similar subscripts, $\mathbf{F}_l$ can be expressed as

$$\mathbf{F}_l = \left[\mathbf{F}_{st}^{(ns\times 1)^T} \ \ \mathbf{F}_{rt}^{(nr\times 1)^T} \ \ \mathbf{0}^{(na\times 1)^T} \right]^T \tag{2.9}$$

The mmf source in the stator loops is given by

$$\mathbf{F}_{st}^{(ns\times 1)} = \mathbf{N}_{abc}^{(ns\times 3)} \mathbf{i}_{abcs}^{(3\times 1)} \tag{2.10}$$

where $\mathbf{i}_{abcs}$ is a vector of balanced stator currents with rms value I_s and phase angle β, and the turns matrix $\mathbf{N}_{abc}$ is built using the *a*, *b*, and *c*-phase turn vectors. The mmf in the rotor loops is given by,

$$\mathbf{F}_{rt}^{(nr\times 1)} = \mathbf{N}_{rt}^{(nr\times 1)} I_{fd} = \left[-1 \ 1 \ 0 \right]^T N_{fd} I_{fd} \tag{2.11}$$

where I_{fd} is the field current and N_{fd} is the number of field turns. Due to the use of single-pole symmetry, the sign of the rotor mmf changes with respect to rotor position.

Within the model program, the matrix $\mathbf{A}_R$ is constructed using a building algorithm similar to that used in general circuit analysis programs (i.e., Spice). Details of the construction of matrix $\mathbf{A}_R$ are provided in [1].

2.3.2 Solving MEC model

The overall solution procedure for the static MEC model is shown in Figure 2.7. The inputs to the model are the machine geometry (including winding configuration), the material properties, and the stator and field currents. The outputs calculated in the post-processing include flux linkage, electromagnetic torque, power loss, and phase and field voltages.

Within the solution procedure shown in Figure 2.7, a Newton-Raphson (N-R) method is used to solve the nonlinear magnetic system in (2.7) at any given rotor position, and the solution procedure is described in Figure 2.8. The maximum possible relative permeability is used to calculate the initial guess of steel reluctance, which is further used to generate an initial guess of loop fluxes through (2.7). The permeability is updated in each iteration and ready for next step.

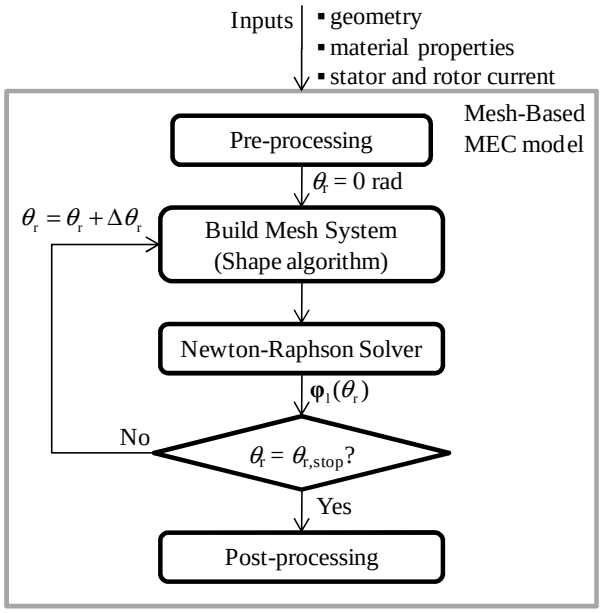


Figure 2.7: Block diagram of the overall solution procedure.

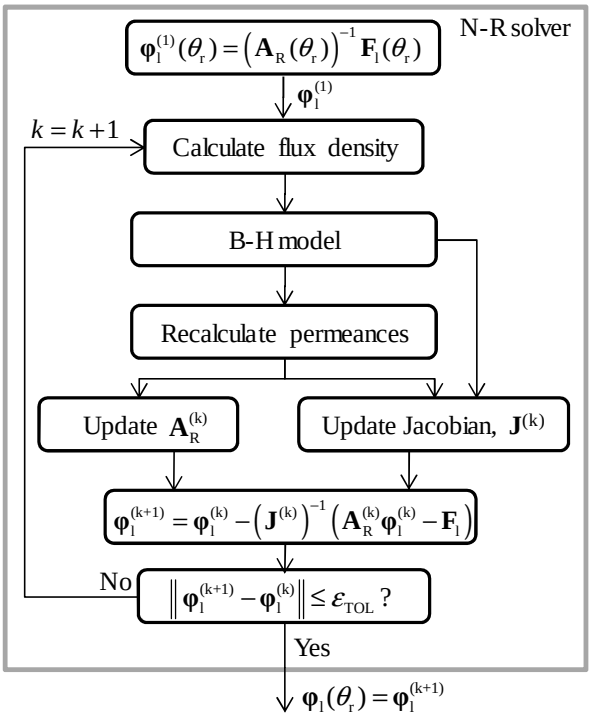


Figure 2.8: Block diagram of Newton-Raphson solution procedure.

The iterative solver starts by computing the branch flux density in the steel as,

$$\mathbf{B}_{\mathrm{br}} = \frac{\boldsymbol{\varphi}_{\mathrm{br}}}{\mathbf{A}_{\mathrm{br}}} \tag{2.12}$$

where $\mathbf{B}_{\mathrm{br}}$ is the branch flux density, $\boldsymbol{\varphi}_{\mathrm{br}}$ is the flux through the branch, and $\mathbf{A}_{\mathrm{br}}$ is the average cross-sectional area. Once the flux density is obtained, the relative permeability μ_{r} and the partial derivative of relative permeability $\partial\mu_{\mathrm{r}} / \partial B$ can be calculated in the B-H model using an exponential-based curve fit equation for $\mu_{\mathrm{r}}(B)$ presented in [54].

After the reluctance values are updated with the new permeabilities, the Jacobian matrix can be formed as follow,

$$\mathbf{J} = \frac{\partial\left(\mathbf{A}_{\mathrm{R}}\boldsymbol{\varphi}_{\mathrm{l}}\right)}{\partial\boldsymbol{\varphi}_{\mathrm{l}}} - \frac{\partial\mathbf{F}_{\mathrm{l}}}{\partial\boldsymbol{\varphi}_{\mathrm{l}}} \tag{2.13}$$

where the term $\partial\mathbf{F}_{\mathrm{l}}/\partial\boldsymbol{\varphi}_{\mathrm{l}}$ is zero since $\mathbf{F}_{\mathrm{l}}$ is not dependent on flux. Using the product rule, the Jacobian can be expanded in the form of

$$\mathbf{J} = \mathbf{A}_{\mathrm{R}} + \mathbf{D}_{\mathrm{R}} \tag{2.14}$$

where $\mathbf{D}_{\mathrm{R}}$ is a matrix containing all the partial derivative terms and is automatically generated from a branch connection matrix. If the branch connection matrix determined that R_i is only within one loop ϕ_v, then the following line of code is executed,

$$\mathbf{D}_{\mathrm{R}}(v,v) = \mathbf{D}_{\mathrm{R}}(v,v) + \frac{\partial R_i}{\partial \phi_v}(\phi_v) \tag{2.15}$$

If R_i is within two loops, x and y, then the following updates can be applied,

$$\begin{aligned}
\mathbf{D}_{\mathrm{R}}(x,x) &= \mathbf{D}_{\mathrm{R}}(x,x) + \partial R_i / \partial \phi_x \left(\phi_x - b\phi_y\right) \\
\mathbf{D}_{\mathrm{R}}(y,y) &= \mathbf{D}_{\mathrm{R}}(y,y) + \partial R_i / \partial \phi_y \left(\phi_y - b\phi_x\right) \\
\mathbf{D}_{\mathrm{R}}(x,y) &= \mathbf{D}_{\mathrm{R}}(x,y) + \partial R_i / \partial \phi_y \left(\phi_x - b\phi_y\right) \\
\mathbf{D}_{\mathrm{R}}(y,x) &= \mathbf{D}_{\mathrm{R}}(y,x) + \partial R_i / \partial \phi_x \left(\phi_y - b\phi_x\right)
\end{aligned} \tag{2.16}$$

where b is equal to +1 when R_i is a non-boundary reluctance and -1 when R_i does lie on the boundary of the pole. Calculation of the Jacobian and reluctance partial derivative terms is well established in [42].

2.3.3 Performance calculation

Electromagnetic Torque

An expression of field energy in terms of MEC quantities is presented in [22] as,

$$W_{\mathrm{mag}} = P\frac{1}{2}\sum_{\mathrm{j}=1}^{\mathrm{n}} \frac{\phi_{\mathrm{j}}^2}{P_{\mathrm{j}}} \tag{2.17}$$

where P_j is the j-th permeance and P is the number of poles. The torque equation based on (2.17) is developed in [55] as,

$$T_{\mathrm{e}}(\phi,\theta_{\mathrm{r}}) = \left(\frac{P}{2}\right)^2 \sum_{j=1}^{na} \left(\frac{\phi_{\mathrm{agj}}}{P_{\mathrm{agj}}}\right)^2 \frac{\partial P_{\mathrm{agj}}}{\partial \theta_{\mathrm{r}}} \tag{2.18}$$

where P_{agj} is the j-th airgap permeance and the number of airgap permeances changes with rotor position.

Stator Phase Voltage

The calculation of phase voltage is based on the phase voltage equations in the rotor reference frame [26],

$$v_{\mathrm{qs}}^{r} = r_{\mathrm{s}} i_{\mathrm{qs}}^{r} + \omega_{\mathrm{r}} \lambda_{\mathrm{ds}}^{r} + p\lambda_{\mathrm{qs}}^{r} \tag{2.19}$$

$$v_{\mathrm{ds}}^{r} = r_{\mathrm{s}} i_{\mathrm{ds}}^{r} - \omega_{\mathrm{r}} \lambda_{\mathrm{qs}}^{r} + p\lambda_{\mathrm{ds}}^{r} \tag{2.20}$$

where f_{qs}^{r} and f_{ds}^{r} are the *q*- and *d*-axis variables with *f* can be voltage (*v*), current (*i*), or flux linkage (λ), and p is the operator d/dt. From the machine geometry and conductor properties, the stator resistance (r_s) can be derived as,

$$r_{\mathrm{s}} = \frac{l_{\mathrm{c}}}{A_{\mathrm{c}}\sigma_{\mathrm{c}}} = \frac{\left(l_{\mathrm{slot}} + 2l_{\mathrm{end}}\right)}{A_{\mathrm{c}}\sigma_{\mathrm{c}}} \tag{2.21}$$

where A_c is the area of the conductor, σ_c is the conductivity (copper is used herein), and l_c is the length of the conductor including the length in both slots and end windings. The length of end windings is defined as the arc length between the centers of two adjacent stator tooth slots. Similarly, the field resistance and damper bar resistance can be calculated.

The phase winding flux linkages can be expressed in terms of MEC quantities as,

$$\boldsymbol{\lambda}_{\mathrm{abcs}} = P\,\mathbf{N}_{\mathrm{abc}}^{\mathrm{T}}\boldsymbol{\varphi}_{\mathrm{st}} \tag{2.22}$$

where$P$ is the number of poles, $\boldsymbol{\varphi}_{\mathrm{st}}$ is the vector of stator loop fluxes, and $\mathbf{N}_{\mathrm{abc}}$ is the turns matrix. $\lambda_{\mathrm{qs}}^{r}$ and $\lambda_{\mathrm{ds}}^{r}$ can be obtained by applying Park's transformation to the phase flux linkage $\boldsymbol{\lambda}_{\mathrm{abcs}}$. Considering slot harmonics and non-sinusoidally distributed windings, $p\lambda_{\mathrm{qs}}^{r}$ and $p\lambda_{\mathrm{ds}}^{r}$ are not zero. Application of a numerical differentiation can yield a voltage waveform. However, taking the average value of (2.19) and (2.20), the steady-state stator voltages can be expressed as:

$$\overline{v_{\mathrm{qs}}^{r}} = r_{\mathrm{s}}\overline{i_{\mathrm{qs}}^{r}} + \omega_{\mathrm{r}}\overline{\lambda_{\mathrm{ds}}^{r}} \tag{2.23}$$

$$\overline{v_{\mathrm{ds}}^{r}} = r_{\mathrm{s}}\overline{i_{\mathrm{ds}}^{r}} - \omega_{\mathrm{r}}\overline{\lambda_{\mathrm{qs}}^{r}} \tag{2.24}$$

where the superscript $^-$ represents average value. Once the stator voltages in the rotor reference frame are calculated, the values in machine variables can be determined by applying the inverse rotor reference frame transformation.

Power Loss

Within the static MEC model, the total machine/rectifier system loss is represented as,

$$P_{\text{loss}} = P_{\text{res}} + P_{\text{core}} + P_{cond} \tag{2.25}$$

where P_{res} is the total resistive loss in the machine, P_{core} is the core loss in the stator, and P_{cond} is the semiconductor conduction losses. Notice that core loss in the rotor, losses associated with switching (turning on and off semiconductor devices), and friction and windage losses are neglected within the model. The resistive loss is calculated as,

$$P_{res} = r_{fd}\, I_{fd}^2 + \frac{3}{2\pi}\int_0^{2\pi} r_s\, i_{as}^2\left(\theta_r\right) d\theta_r \tag{2.26}$$

where the phase currents are balanced and the field current is a constant dc value.

In the core loss calculation, a volumetric power loss density (W/m^3), P_{ld} ,is approximated based on the Modified Steinmetz Equation (MSE) [56],

$$P_{ld}\left(B\right) = \underbrace{k_h\, f_{eq}^{\alpha-1}\left(\frac{B_{\max}}{B_b}\right)^{\beta} f}_{\textit{Hysteresis Loss}} + \underbrace{\frac{k_e\, f}{B_b^2}\int_0^T\left(\frac{dB}{dt}\right)^2 dt}_{\textit{Eddy Current Loss}} \tag{2.27}$$

where f and T are the fundamental frequency and period of the current; B_b is the base flux density ($B_b = 1\text{T}$); $B_{\max}$ is the maximum value of the flux density waveform; α , β , k_h and k_e are parameters of the MSE that are defined in [54] and their values are listed in Table 2.1. The equivalent frequency is given by,

$$f_{eq} = \frac{2}{\left(B_{\max} - B_{\min}\right)^2 \pi^2}\int_0^T\left(\frac{dB}{dt}\right)^2 dt \tag{2.28}$$

In (2.27) and (2.28), the derivative and integral terms are calculated using a forward Euler formula and the composite trapezoidal rule, respectively. Thus, the final value of core loss in the stator can be developed as,

$$P_{core} = P_{ld,T} V_{ST} + P_{ld,Y} V_{SY} \tag{2.29}$$

where $P_{ld,T}$ and $P_{ld,Y}$ are the volumetric power loss density in the stator teeth and stator yoke, respectively; and V_{ST} and V_{SY} are the volume of the stator teeth and stator yoke, respectively.

Table 2.1
Parameters for core loss estimation using MSE for M19.

α	1.338	β	1.817
k_e	5.044e-5	k_h	0.09294

By assuming the forward voltage drop of a transistor and a diode are the same, the conduction losses is given by,

$$P_{cond} = 3V_{drop} \frac{1}{2\pi} \int_0^{2\pi} \left| i_{as}(\theta_r) \right| d\theta_r \tag{2.30}$$

where V_{drop} is the forward switch and diode voltage drop and

$$\sqrt{2} I_{rms} = \sqrt{I_{qs}^2 + I_{ds}^2} \tag{2.31}$$

Switching loss is not represented in the model. Its potential influence is the subject of ongoing research. In the studies conducted herein it was assumed that $V_{drop} = 2$ V for all devices.

3. CONTROL OPTIMIZATION OF WRSMS

3.1 Motivation

Prior to derivations, it is convenient to view the block diagram of a representative WRSM drive shown in Figure 3.1 to place the questions addressed in this research in context. In Figure 3.1 it can be seen that the WRSM is connected mechanically to a prime mover. The stator windings are connected to an active rectifier, which is used to control the stator phase currents and convert ac to dc. The field winding is connected to a dc source, which herein is assumed to regulate the field current. Typically, the dc bus is capacitive, as shown. Although the prime mover could be categorized by type (i.e. diesel engine, gas turbine, wind turbine), herein it is classified by whether one does or does not have the capability to adjust commanded prime mover angular velocity (speed). An example where one does not have the capability to control speed is aircraft power generation systems, where the turbine or engine speed is not specified by the electrical power system and indeed varies considerably. A similar situation is encountered in traditional automotive charging systems. A third example is ship and portable power applications where the commanded speeds of turbine or engine sets are often fixed by the manufacturer.

A representative control for systems without access to commanded prime mover velocity is shown in Figure 3.2. As shown, the difference between commanded and measured dc voltage is input to a voltage regulator (often a proportional plus integral control). The output of the voltage regulator is the commanded electromagnetic torque that is desired from the WRSM. The electric drive controller is responsible for translating the commanded torque to stator and field current commands that are used to adjust the switching devices in the active rectifier and field winding circuits. Through this process,

the commanded torque effectively sets the dc current out of the electric drive that in steady-state will match the load current at the commanded voltage.

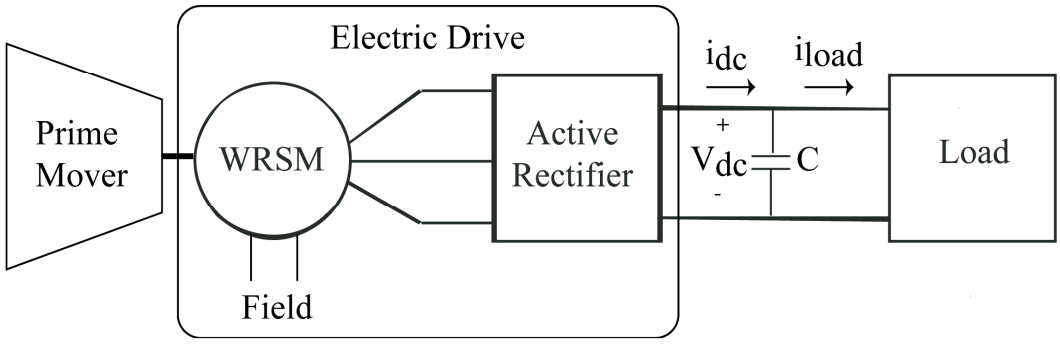


Figure 3.1: Block diagram of a representative WRSM drive.

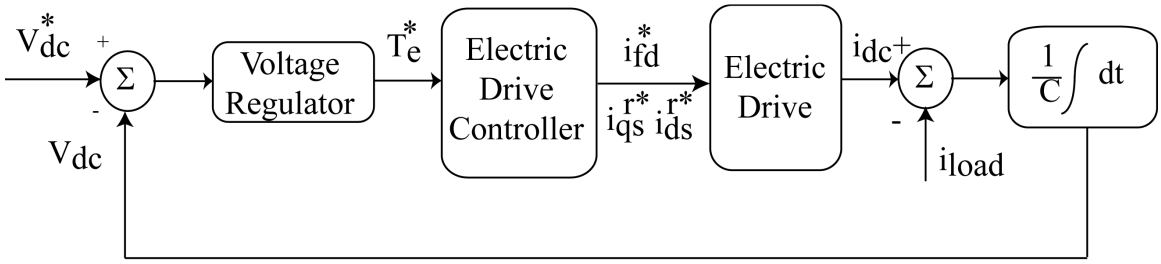


Figure 3.2: Representative electric drive control without access to commanded prime mover velocity. (Starred quantities represent commanded values.)

The overall question addressed herein is how to establish the map between commanded torque and commanded winding currents. This question yields additional questions as to what model should be applied to create the map, whether the proposed map is consistent with the goals of mass/loss reduction, the simplicity of the control, and what is the influence on the machine design? These questions are addressed for the case in which one does not have the capability to adjust commanded prime mover angular velocity in Sections 3.3-3.7. It is noted that without this capability, the prime mover has no role in the design of the electric drive controls, other than to provide the lower/upper limits on angular velocity.

The case in which prime mover commanded angular velocity is adjustable does change the overall picture and enables one to consider the coupled prime mover/electric

drive together when establishing controls. A description of how this can be approached is provided in Section 3.8. Finally, although the questions raised are herein considered for a generator application, the answers presented are directly applicable for a WRSM drive operating as a motor (i.e. as an engine starter).

3.2 Background

Over the past several years, a multi-objective (i.e. minimize mass, minimize loss) evolutionary-based design toolbox [51] has been created for WRSMs. The variables listed in Table 3.1 are used as genes. Genes 1-7 are geometric variables that define the depth/length of all the major machine sections. These are shown in Figure 2.5. Genes 8-11 are scaling factors between 0 and 1 that are used to establish the geometry of the stator teeth/slots and the rotor poles. Genes 12-13 are used to define the stator and field windings. Genes 14-16 are used to define the field and stator winding excitation.

Table 3.1
Genes Used in the WRSM Design Program.

#	Gene	Gene Description
1	r_{sh}	Shaft radius (m)
2	d_{rc}	Rotor core depth (m)
3	d_{rp}	Rotor pole depth (m)
4	g	Airgap length (m)
5	d_{st}	Stator tooth depth (m)
6	d_{b}	Stator yoke depth (m)
7	l	Stack length (m)
8	fw_{ss}	Fraction to find w_{ss}
9	fh_{rt}	Fraction to find h_{rt}
10	fw_{rt}	Fraction to find w_{rt}
11	fw_{rp}	Fraction to find w_{rp}
12	N_{s}	Turns per slot
13	N_{fd}	Number of field turns
14	I_{s}	Stator current, rms (A)
15	β	Stator phase angle (rad)
16	I_{fd}	Field current (A)

Within the toolbox, the constraints and fitness function are evaluated using the steady-state MEC model described in Section 2.3. The MEC model has been structured for rapid evaluation of candidate designs by modeling only a single pole and using a mesh-based solution of the circuit. Within optimization studies, a single machine is evaluated at 91 discrete positions over half of an electrical cycle. This requires on the order of 0.6-.0.8 son a single-core desktop PC. The variance in the time is due to the convergence of the Newton Raphson algorithm, which has been found to require less than 5 iterations, regardless of saturation level.

The toolbox is configured for the electromagnetic design of machines of arbitrary power level. To date, thermal effects are considered in a simplified way by setting a current density limit on the stator and rotor windings. Initial testing and toolbox implementation has focused upon an air-cooled drive system with constraints of a dc-link voltage < 200 V, output power > 2 kW, and winding current densities < 7.6 A/mm^2 at a rotor speed of 3600 rpm. An initial optimization was performed using a population of 600 individuals over 800 generations. The Pareto-optimal front from which a design to be constructed was selected is shown in Figure 3.3. Details of the design process and hardware validation are provided in [1].

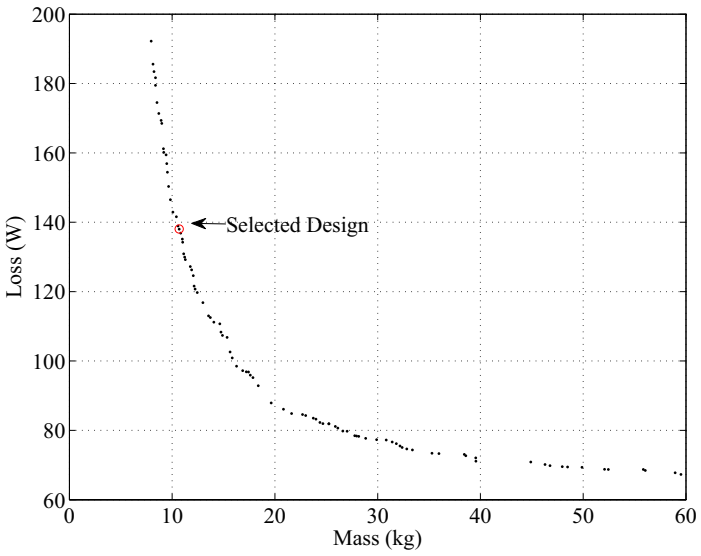


Figure 3.3: Pareto-optimal front for 2 kW machine design.

It is noted that in the initial validation, focus was placed upon the machine. Losses of the rectifier were not included in establishing the Pareto-optimal front. Subsequently, conduction loss of the rectifier has been included. In using the tool to study machine designs with/without conduction loss, it has been found that the machines are similar in terms of geometry and field and stator winding ampere-turns [57]. The notable difference is that in the machines with rectifier conduction loss included, ampere-turns are achieved by higher turns and lower current compared to machines without rectifier conduction loss.

Among the lessons learned in the design and validation is that there can be relatively wide variability in the anhysteretic *BH* curves of M19 steel. Specifically, toroidal samples of the core material obtained pre- and post-machine construction were obtained and were found to have differences. This is not unexpected, since material classification is based upon a loss characterization and not an anhysteretic *BH* characterization [58]. The *qd*-axis flux linkage calculated with *BH*1 (pre-construction) and *BH*2 (post-construction) are shown in Figure 3.4. *BH2* was shown to have a more accurate material characterization in [59] and thus is used in developing the excitation strategies in the following sections.

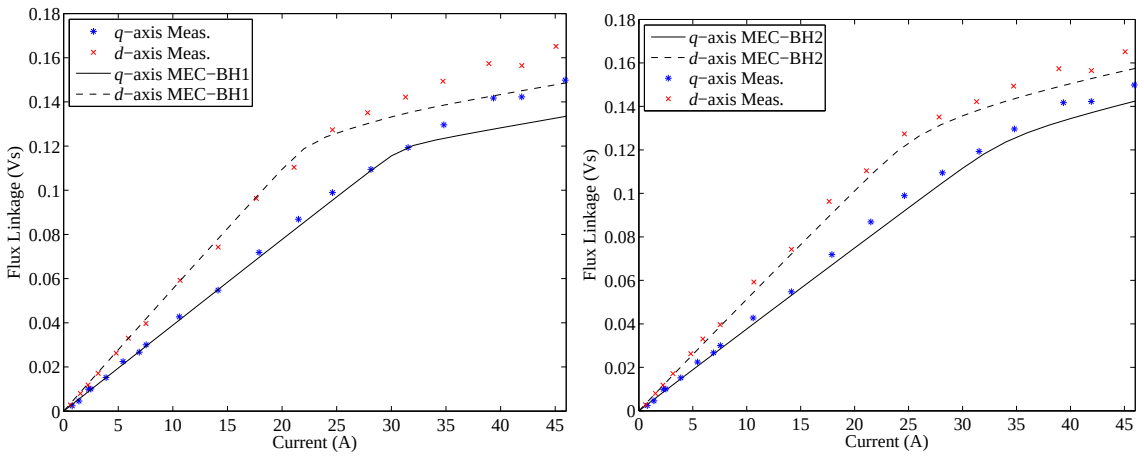


Figure 3.4: *q*- and *d*-axis flux linkage versus current.

3.3 Model Details

From Table 3.1, it is observed that the genes of the machine design optimization include stator and field excitation for rated output power. One may suggest that the currents over a range of operating points could be obtained within the machine design optimization. However, the computational effort required to do so is significant, since each operating point would require the solution of the MEC. In addition, as one increases the number of genes (which would need to include currents at each operating point), the time required to obtain convergence increases.

As an alternative, for power less than rated, a second optimization is performed to establish a map between commanded torque and stator/field excitation for any machine upon the Pareto-optimal front. Herein, the map is obtained for the machine that was constructed using three approaches. In the first, a standard *qd*-model is utilized. In the second, a *qd*-model in which saturation is included along the *d*-axis is utilized. In the final approach, the MEC model is applied.

qd model – saturation neglected

Since the machines are connected to an active rectifier, damper windings are not utilized in the rotor of the machine studied. The voltage and flux linkage equations of a traditional *qd* model that are used for optimization are expressed as,

$$V_{qs}^{r} = r_s I_{qs}^{r} + \omega_r \lambda_{ds}^{r} \tag{3.1}$$

$$V_{ds}^{r} = r_s I_{ds}^{r} - \omega_r \lambda_{qs}^{r} \tag{3.2}$$

$$V_{fd}^{'} = r_{fd}^{'} I_{fd}^{'} \tag{3.3}$$

$$\lambda_{qs}^{r} = L_q i_{qs}^{r} \tag{3.4}$$

$$\lambda_{ds}^{r} = L_d i_{ds}^{r} + L_{md} i_{fd}^{'} \tag{3.5}$$

The electromagnetic torque is expressed as,

$$T_e = \frac{3}{2}\frac{P}{2}(\lambda_{ds}^{r} i_{qs}^{r} - \lambda_{qs}^{r} i_{ds}^{r}) \tag{3.6}$$

In (3.1)-(3.6), L_{md} and L_{mq} are the *d*-axis and *q*-axis magnetizing inductances, respectively, r_s is the stator winding resistance, and r'_{fd} is the referred field resistance. In (3.3) and (3.5), the primes are used to denote that the field quantities are referred to the stator winding.

The values of L_d, L_q are obtained from the MEC model by taking the ratio of the respective *q*- and *d*-axis flux linkage to *q*- and *d*-axis test currents. L_{md} and L_{mq} are obtained by subtracting the stator leakage inductance L_{ls} from L_d and L_q. The stator leakage inductance is approximated as the zero-sequence inductance L_0 which is the ratio of zero-sequence flux linkage to zero-sequence current. The ratio between actual and referred rotor windings was obtained using a developed diagram of the MMF of the rotor and stator windings [26]. The equivalent turns of a sinusoidally distributed winding were computed and used to establish $I'_{fd} = \frac{2}{3}\frac{N_{fd}}{N_s} I_{fd}$ and $V'_{fd} = \frac{N_s}{N_{fd}} V_{fd}$. The stator and field winding resistances are calculated within the machine design program using (2.21) to calculate dc winding resistance. All parameters of the steady-state *qd* model are shown in Table 3.2.

qd model – saturation along d-axis

Often, in the analysis of salient-pole synchronous machines, saturation is represented along the *d*-axis. With knowledge that the machine selected has flux densities that are beyond the knee of the *BH* curve, it was of interest to observe the influence that modeling *d*-axis saturation has on the optimized winding currents. To model saturation, (3.5) is represented in a form

$$\lambda^r_{ds} = L_{ls} i^r_{ds} + \lambda^r_{md} \tag{3.7}$$

where

$$\lambda^r_{md} = f(i_{md}) \tag{3.8}$$

$$I_{md} = I^r_{ds} + I'_{fd} \tag{3.9}$$

To determine the relationship between magnetizing current and flux linkage, the MEC model was utilized. The rotor was positioned at $\theta_r = 90°$ and the stator winding currents were set to zero. The field current was increased and the respective *d*-axis flux linkage determined. The relationship between magnetizing current and flux linkage can be expressed mathematically using the map proposed in [27] as,

$$i_{md} = \frac{2M_d}{\pi}\left\{(\lambda_{md} - \lambda_T)a\tan[\tau_T(\lambda_{md} - \lambda_T)] + \lambda_T a\tan(-\tau_T\lambda_T)\right\} + \frac{M_d}{\pi\tau_T}\left\{\ln(1+\tau_T^2\lambda_T^2) - \ln[1+\tau_T^2(\lambda_{md} - \lambda_T)^2]\right\} + M_a\lambda_{md} \quad (3.10)$$

where M_d and M_a are related to the initial and final slopes, τ_T and λ_T define the tightness of the transition from initial slope to final slope and the point of transition, respectively. The values are shown in Table 3.2.

Table 3.2
Parameters used in *qd* Models.

M_{d}	451.42	M_{a}	612.83
τ_{T}	173.09	λ_{T}	0.127
$r_{\mathrm{s}}(\Omega)$	0.16	$r_{\mathrm{fd}}(\Omega)$	2.55
L_{q} (mH)	3.76	L_{d} (mH)	5.15
L_0 (mH)	0.82	N_s	19.67
N_{fd}	215.26		

MEC model

The electromagnetic torque and power loss calculation are shown in Section 2.3.3. The same equations can be used to calculate the resistive and conduction loss for the *qd* models, however, core loss can only be calculated in the MEC model.

3.4 Optimal Excitation

Consistent with the desire to minimize loss, an optimization was established to minimize loss subject to the constraint of meeting the specified electromagnetic torque command. Additional constraints include not exceeding the current limit and the phase voltage limit. Mathematically, the optimization is expressed as,

$$\text{Minimize} \qquad P_{loss}(i_{qs}^{r}, i_{ds}^{r}, i_{fd}) \tag{3.11}$$

Subject to:

$$T_e = T_e^* \tag{3.12}$$

$$J_{stator} \le J_{s\max} \tag{3.13}$$

$$J_{rotor} \le J_{r\max} \tag{3.14}$$

$$\sqrt{3}\sqrt{V_{qs}^{r2} + V_{ds}^{r2}} \le V_{dc} \tag{3.15}$$

In (3.12), and throughout this chapter, a * is used to denote a commanded value. The maximum stator and rotor current densities were assumed $J_{s\max} = J_{r\max} = 7.6\,\text{A/mm}^2$ The dc bus voltage limit was $V_{dc} = 200\,\text{V}$. The optimization was performed using the evolutionary approach used for the machine design.

Initially, the optimization was performed using the MEC model upon which the design was based. The optimization was then repeated using the traditional *qd* model (no saturation) and the *qd* model incorporated with *d*-axis saturation. Within the *qd* models, only resistive loss and switch conduction loss is represented (no core loss). The resulting currents obtained from optimization of (3.11)-(3.15) using the three models are shown in Figure 3.5. The comparison of the total power loss obtained from the optimization using the three models is shown in Figure 3.6. From the plots in Figure 3.6, one can see that the power loss is significantly under estimated when using the *qd* models due to the absence of core loss within these models. This would lead to an overestimate of the output power from the *qd* models. In addition, if one applies the currents obtained from the *qd* models into the MEC model, one finds that at higher power levels the torque is significantly less than the commanded torque.

From Figure 3.5, there are several interesting observations. First, at lighter loads both *qd* models yield nearly the same optimal stator current commands. This is expected since under the relatively small currents, saturation is unlikely to play a dominant role. As load increases, the currents obtained by the three models tend to have more significant differences.

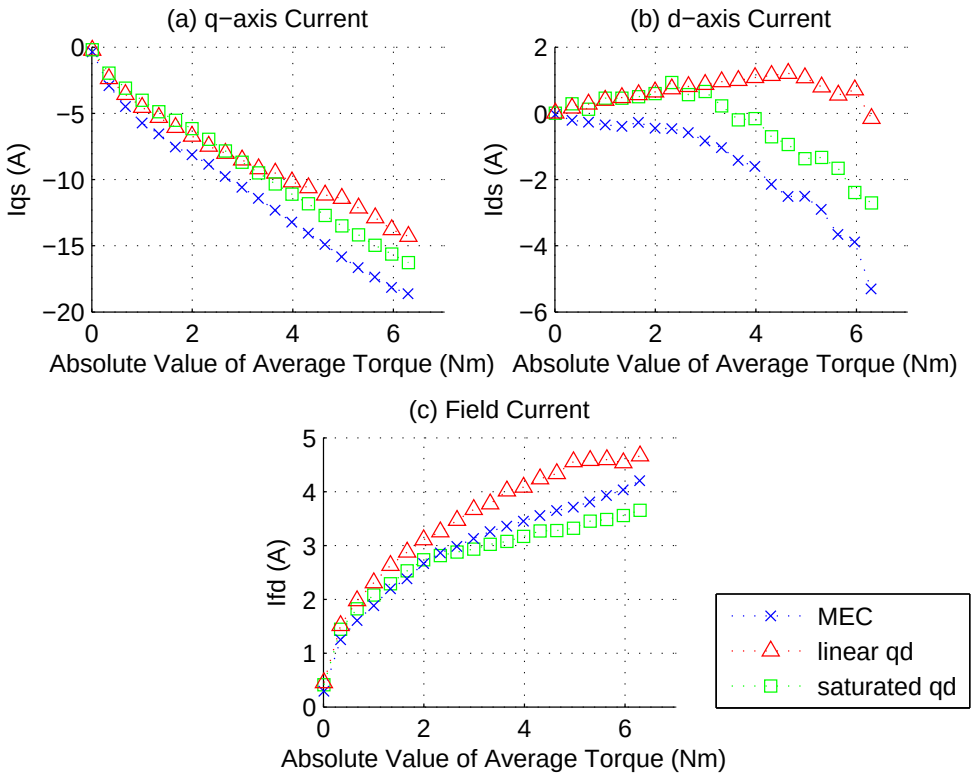


Figure 3.5: Current control schemes for optimal control based on MEC model, linear *qd* model, and nonlinear *qd* model respectively. a) *q*-axis current, b) *d*-axis current, c) field current.

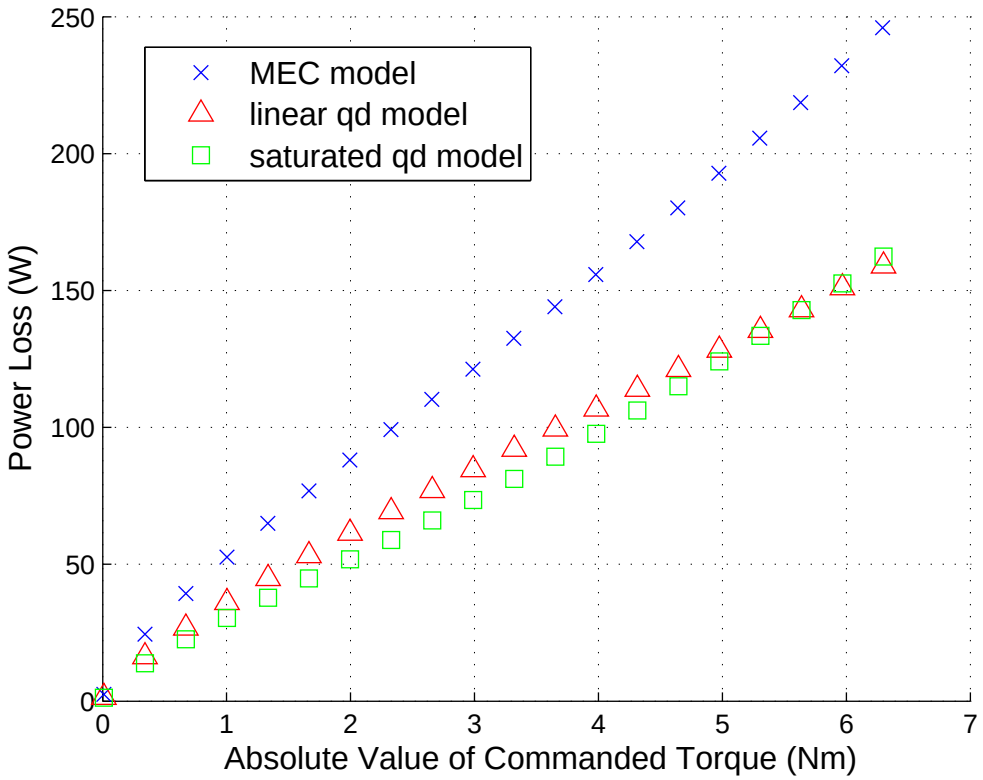


Figure 3.6: Comparison of power loss for the MEC model, linear *qd* model, and nonlinear *qd* model.

One of the more interesting trends is in the *d*-axis current. From Figure 3.5(b), it can be seen that the *d*-axis currents obtained from the unsaturated and saturated *qd* models transition from positive to negative values as load increases, while those obtained from the MEC model are always negative. Without considering saturation, core loss, or a dc link voltage constraint, one would expect that the *d*-axis current would be positive in order to provide additional torque resulting from saliency (note that torque is defined negative for generator operation). Therefore, the *d*-axis current in the unsaturated *qd* model is positive in most of the load region until a voltage constraint is met at higher load and becomes negative to weaken the field. It is interesting that the field current is not used to weaken the field. As for the saturated *qd* model, saturation effects the selection of *qd*-axis currents so that the *d*-axis current transition to negative occurs earlier than the unsaturated model.

Within the MEC model, saturation and core loss are included. At light load, one can argue that the only possible reason for a negative *d*-axis is to minimize core loss. A careful inspection at the *q*-axis and field currents under light load shows that they are larger in magnitude than those obtained by the *qd* models. This is to counteract the reduction in torque created by the negative *d*-axis current.

To help explain the prevalence of negative *d*-axis current at higher loads, an additional study was performed. Specifically, taking the phase current amplitude and field current at rated load, the current phase angle was varied and the impact on the machine performance was investigated using the MEC model. Variation of the phase angle directly impacts the amount of *q*-axis and *d*-axis current. This variation has no impact on the resistive/conduction loss since rms stator current remains the same. The main variables of interest for this study were core loss and torque, and these variables along with *q*- and *d*-axis current are plotted in Figure 3.7 as a function of phase angle. These results illustrate that a negative *d*-axis current provides a benefit in terms of core loss, although the amount of the reduction in core loss is perhaps relatively small. In addition, if one looks at the impact on torque, it can be seen that for a set of field current, the maximum torque point is achieved by using a negative *d*-axis current. Referring back to (3.6) with (3.4) and (3.5) substituted for the flux linkages, this seems counterintuitive, but it is reasonable considering that the MEC model accounts for saturation whereas the lumped parameter equation does not. Indeed, this also explains why the *d*-axis current obtained by the saturated *qd* model becomes negative as load increases.

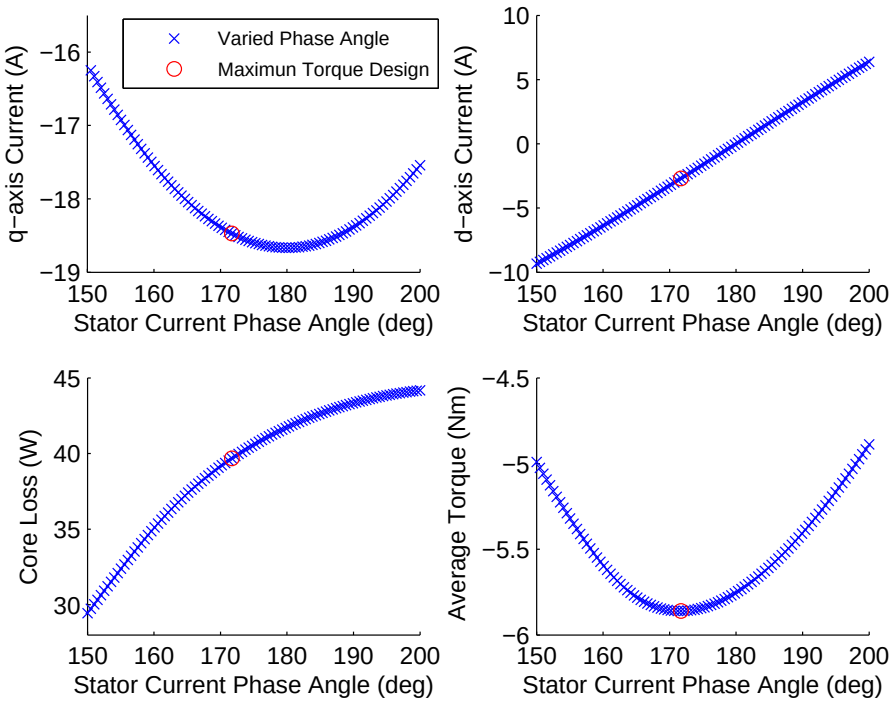


Figure 3.7: *q*- and *d*-axis stator current, torque and core loss versus stator current phase angle for constructed machine.

3.5 Sub-optimal Excitation

Simplicity of control is often of interest. In addition, to provide a rapid dynamic response there is often a desire to establish a field-oriented approach similar to that of a DC machine in which the rotor field is constant and the torque command is mapped directly to armature excitation.

To address these potential interests, three alternative excitation schemes are considered. In the first, the field current is held fixed at the optimized 2 kW level (3.8 A), and the *q*- and *d*-axis currents are optimized to minimize system loss at each value of commanded torque. A second control is considered in which the torque attributed to saliency is eliminated by setting *d*-axis current to zero. Therein, the field and *q*-axis current are solved to minimize system loss. The currents obtained for these two schemes are shown along with the optimal control currents in Figure 3.8. The total system loss resulting from these controls are shown in Figure 3.9.

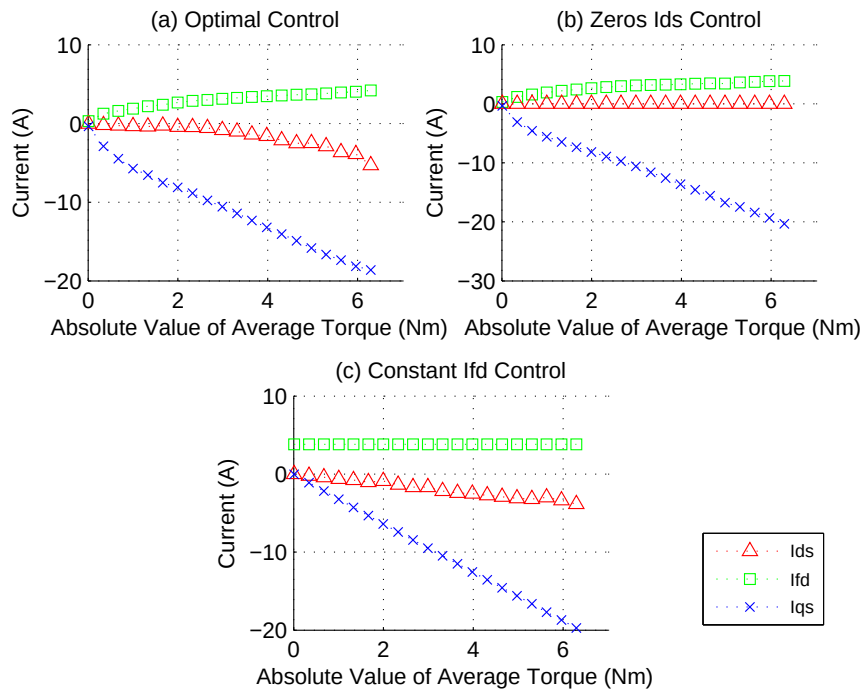


Figure 3.8: Current control schemes for a) optimal control, b) zero *d*-axis, and c) constant field controls.

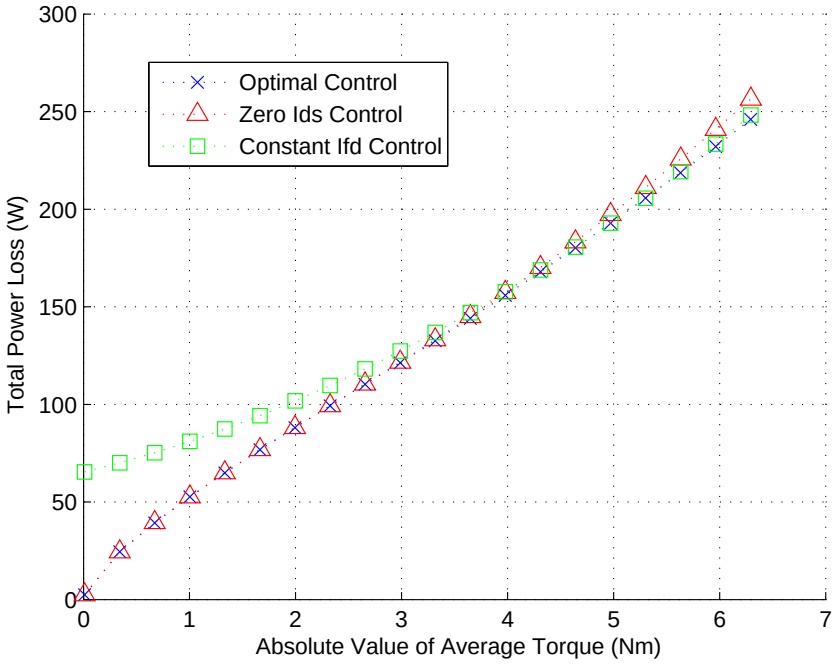


Figure 3.9: Total power loss for optimal control, zero *d*-axis current control, and constant field controls.

From Figure 3.9, it can be seen that with the exception of powers below 30% rated, a constant field control provides minimal difference with those of the original optimized current. It can also be observed that setting *d*-axis current to zero does lead to an increase in loss at higher power levels. However, this loss increase is relatively minor.

The results of the first two alternative controls sets the stage for a third control in which the *d*-axis current is set to zero, the field current is held constant, and a torque versus *q*-axis current map is utilized over the entire power range. Figure 3.10 shows this simplified current control. The field current in this scheme is obtained so that the overall power loss in creating electromagnetic torque from 0 to rated (6.3 Nm) is minimized. Its value is 3.28 A in this case. A comparison of total system loss resulting from the simplified control and the optimal control is also shown in Figure 3.10. From the results, it can be seen that the simplified control is nearly as efficient as the optimal control over much of the power range.

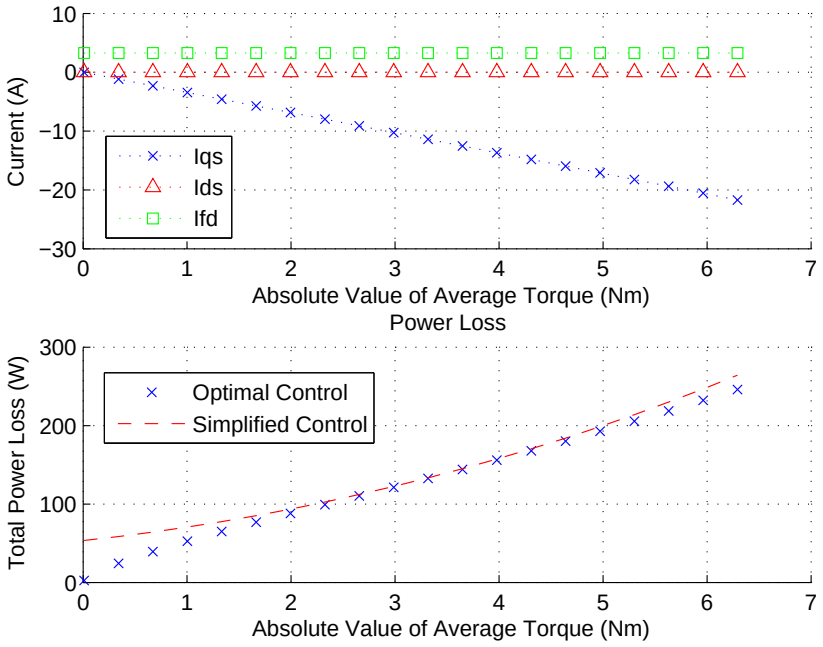


Figure 3.10: Current control schemes and total power loss for simplified control.

In many applications, operation at low and high powers occurs over a relatively low percent of time, which would tend to minimize the overall energy loss if such a control were implemented. In addition, one notes that since the field current is held fixed,

such a control is relatively straightforward for both brushed and brushless exciters. Specifically, for a brushless excitation system, a single point map is needed between field current and excitation field voltage. Another point to consider is that it is interesting that the torque versus *q*-axis current is indeed linear in this simplified control, despite the machine operating in saturation. This is a result that saturation is primarily set by the field current. The *q*-axis current from zero through rated value appears to have relatively minor influence on the magnetic operating point. In addition, since the *d*-axis current is held fixed at zero and the *q*-axis does not have an appreciate influence on the magnetic operating point, changes in torque would not translate to transients in the field current. Theoretically, this would ensure a fast transient response.

Finally, there was interest in establishing performance for speeds less than rated. Within this region, studies were performed to establish the power loss between the simplified and optimal controls at various speeds and torque levels. The results are shown in Figure 3.11. From these curves one can see there is relatively minor difference between the loss obtained from the two controls, as one might expect.

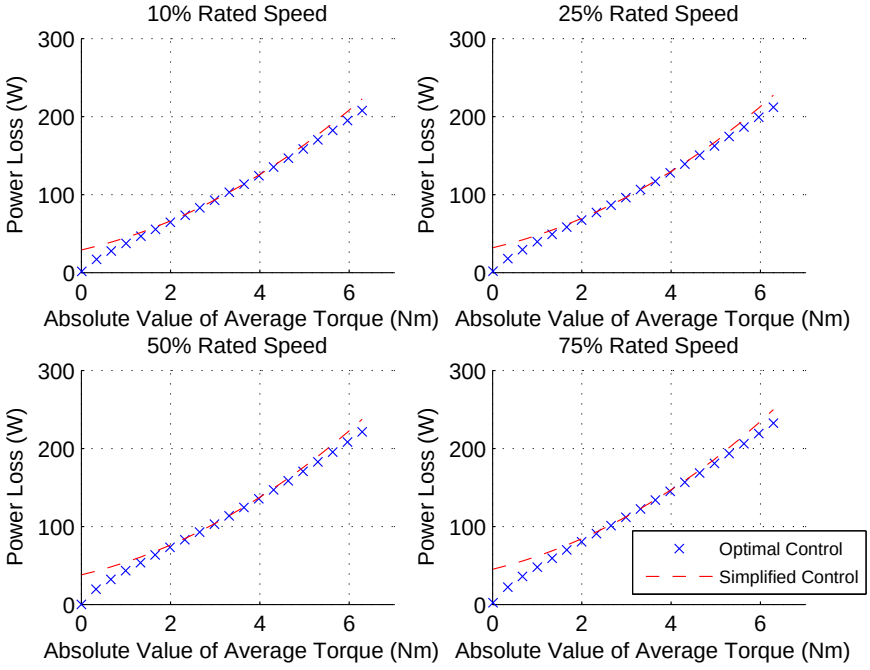


Figure 3.11: Comparison of power loss between optimal and simplified control at variable speed.

3.6 Hardware Validation

Hardware-based performance of the controls for speeds up to 3600 rpm was evaluated using the bench set up shown in Figure 3.12. A dynamometer working as a prime-mover was connected with the WRSM through a torque transducer. The WRSM is driven electrically by an active rectifier that used a ST microelectronics STG3P3M25N60 3-phase inverter bridge with an International Rectifier half-bridge gate driver (IR2183) to perform delta-hysteresis current regulation. The delta interval and hysteresis band were set to 50 μs and 0.5 A, respectively. At the output of the inverter is a 6.6 mF capacitor in parallel with a 20 Ω resistor. An encoder is used to obtain rotor position and a power supply operating as a current source is used to provide field excitation. The current control vectors generated by the optimal control and the simplified control were tested at 3600 rpm and 1800 rpm.

Figure 3.12: Hardware test bench.

Prior to the experiments, the dynamometer was used to spin the de-energized machine to 3600 rpm and 1800 rpm. An in-line torque transducer was used to establish an estimate of 100 W and 50 W loss due to friction/windage at rated and half-rated speed, respectively. Stator and field windings resistances were measured as 0.2 Ω and 2.81 Ω, respectively. The measured values of resistance are used for loss calculation in the MEC

model in this section. The machine was then run under load and the respective mechanical input power and dc output power were measured. 'Measured' electromagnetic torque was estimated by taking the measurement of the torque transducer, and subtracting the torque associated with friction/windage. Total loss was estimated as

$$P_{loss} = \left|T_e \omega_{rm}\right| + V_{fd} I_{fd} - \overline{V_{dc} I_{dc}} \tag{3.16}$$

where T_e is the estimated electromagnetic torque, ω_{rm} is the rotor angular velocity, $V_{fd} I_{fd}$ is the input power to the exciter, and $\overline{V_{dc} I_{dc}}$ is the average of the product of measured dc-link current and voltage.

Table 3.3 and Table 3.4 contain the MEC and hardware performance for the optimal control and simplified control at rated speed, respectively.

Table 3.5 and Table 3.6 contain the MEC and hardware performance for the optimal control and simplified control at half-rated speed, respectively. From the tables, one can see that the simulation and experimental results match well. As expected, the simplified control produces slightly more machine loss than the optimized control at higher torque levels.

For both current commands one can see error between the MEC model predicted loss and the measured loss. A difference is certainly expected, since in the MEC model, switching loss is not represented. In addition, only core loss of the stator (not the rotor) is considered. Within the delta-hysteresis control a synchronous current regulator was not applied. Thus, there is likely some minor error between commanded current and actual current that could lead to a difference in expected/measured torque that was perhaps favorable in some instances and unfavorable in others. However, it does not appear that these differences were appreciable. In general, one can conclude that the difference between the loss of the optimal and simplified controls is minor, and that a relatively simple control can be achieved that is consistent with goals of minimizing mass and loss of this machine/drive system.

Table 3.3
Comparison of MEC and hardware for optimal control currents at 3600 rpm.

Current (A)				Torque (Nm)	DC Output Power (W)	Power Loss (W)
Iq	Id	Ifd				
-10.6	-0.8	3.13	MEC	2.99	1022.3	131.3
			Hardware	3.16	1067.3	151.6
			Error	5.7%	4.4%	15.5%
-14.0	-2.1	3.56	MEC	4.31	1476.3	184.6
			Hardware	4.60	1565.8	204.0
			Error	6.7%	6.1%	10.5%
-15.8	-2.5	3.71	MEC	4.97	1698.9	213.5
			Hardware	5.37	1826.0	237.2
			Error	8.0%	7.5%	11.1%
-17.4	-3.7	3.93	MEC	5.63	1923.9	243.5
			Hardware	6.06	2055.3	272.7
			Error	7.6%	6.8%	12.0%
-18.6	-5.3	4.20	MEC	6.29	2146.3	275.4
			Hardware	6.65	2236.6	320.0
			Error	5.7%	4.2%	16.2%

Table 3.4
Comparison of MEC and hardware for simplified control currents at 3600 rpm.

Current (A)				Torque (Nm)	DC Output Power (W)	Power Loss (W)
Iq	Id	Ifd				
-10.3	0	3.28	MEC	2.99	1024.4	132.2
			Hardware	3.17	1071.9	153.3
			Error	6.0%	4.6%	16.0%
-14.8	0	3.28	MEC	4.31	1467.7	187.7
			Hardware	4.55	1538.6	207.0
			Error	5.6%	4.8%	10.3%
-17.1	0	3.28	MEC	4.97	1683.8	220.8
			Hardware	5.25	1752.0	257.4
			Error	5.6%	4.1%	16.7%
-19.4	0	3.28	MEC	5.63	1896.6	257.7
			Hardware	5.93	1986.6	279.1
			Error	5.3%	4.7%	8.3%
-21.7	0	3.28	MEC	6.29	2105.1	298.5
			Hardware	6.55	2165.5	334.0
			Error	4.1%	2.9%	11.9%

Table 3.5
Comparison of MEC and hardware for optimal control currents at 1800 rpm.

Current (A)				Torque (Nm)	DC Output Power (W)	Power Loss (W)
Iq	Id	Ifd				
-10.4	-0.6	3.20	MEC	2.99	477.3	112.9
			Hardware	3.12	484.6	132.2
			Error	4.3%	1.5%	17.1%
-13.9	-2.1	3.61	MEC	4.31	684.9	163.5
			Hardware	4.54	704.9	187.4
			Error	5.3%	2.9%	14.6%
-15.7	-2.8	3.78	MEC	4.97	785.5	191.1
			Hardware	5.30	827.9	211.2
			Error	6.6%	5.4%	10.5%
-17.3	-3.7	3.97	MEC	5.63	885.5	220.2
			Hardware	6.01	932.7	244.4
			Error	6.7%	5.3%	11.0%
-18.8	-4.6	4.16	MEC	6.29	984.1	250.7
			Hardware	6.57	1017.4	269.6
			Error	4.5%	3.4%	7.5%

Table 3.6
Comparison of MEC and hardware for simplified control currents at 1800 rpm.

Current (A)				Torque (Nm)	DC Output Power (W)	Power Loss (W)
Iq	Id	Ifd				
-10.2	0	3.28	MEC	2.99	449.6	113.4
			Hardware	3.13	484.2	136.0
			Error	4.7%	7.7%	19.9%
-14.8	0	3.28	MEC	4.31	646.0	166.3
			Hardware	4.45	672.7	196.3
			Error	3.2%	4.1%	18.0%
-17.1	0	3.28	MEC	4.97	739.2	197.8
			Hardware	5.13	776.2	221.0
			Error	3.2%	5.0%	11.7%
-19.4	0	3.28	MEC	5.63	828.9	232.8
			Hardware	5.85	878.1	254.8
			Error	3.9%	5.9%	9.5%
-21.7	0	3.28	MEC	6.29	914.9	271.5
			Hardware	6.48	955.5	296.2
			Error	3.0%	4.4%	9.1%

3.7 Variable Speed Operation

Although the given machine was not originally designed for variable speed application, it is interesting to consider the impact of these alternative excitations strategies as speed increases beyond rated value. To do so, the envelopes that establish the maximum possible torque at each speed were created following an optimization:

$$\text{Maximize} \qquad T_e(i_{qs}^r, i_{ds}^r, i_{fd}, \omega_r) \tag{3.17}$$

Subject to:

$$T_e \le T_{e\_rated} \tag{3.18}$$

$$J_{stator} \le J_{s\max} \tag{3.19}$$

$$J_{rotor} \le J_{r\max} \tag{3.20}$$

$$\sqrt{3}\sqrt{V_{qs}^{r2} + V_{ds}^{r2}} \le V_{dc} \tag{3.21}$$

where the rated torque is 6.3 Nm. To create the envelopes for the simplified control, i_{ds}^r in (3.17) is set to zero for all speeds, and the field current is held constant at 3.28 A. The maximum torque versus speed under each of the controls is shown in Figure 3.13.

Comparing the envelopes of performance, one notes that the torque achievability from the simplified control is a subset of that of the optimal control. Of course, for speeds up to rated there is no difference in the torque availability and the performance was considered in Sections 3.5 and 3.6. However, if one extends beyond rated speed, the peak torque that can be obtained is much different between the two controls. Considering Figure 3.13, several details catch the eye. First, although the machine was not designed for variable speed operation, when using the optimal control, one can achieve rated torque for speeds exceeding roughly twice rated speed. Moreover, once the available torque decreases, the decrease is proportional to rotor speed and therefore a constant power region extends to at least four times rated speed. Of course, design constraints and mechanical loss for high speeds were not considered and so this result is useful in that it allows comparison to the performance from the simplified control.

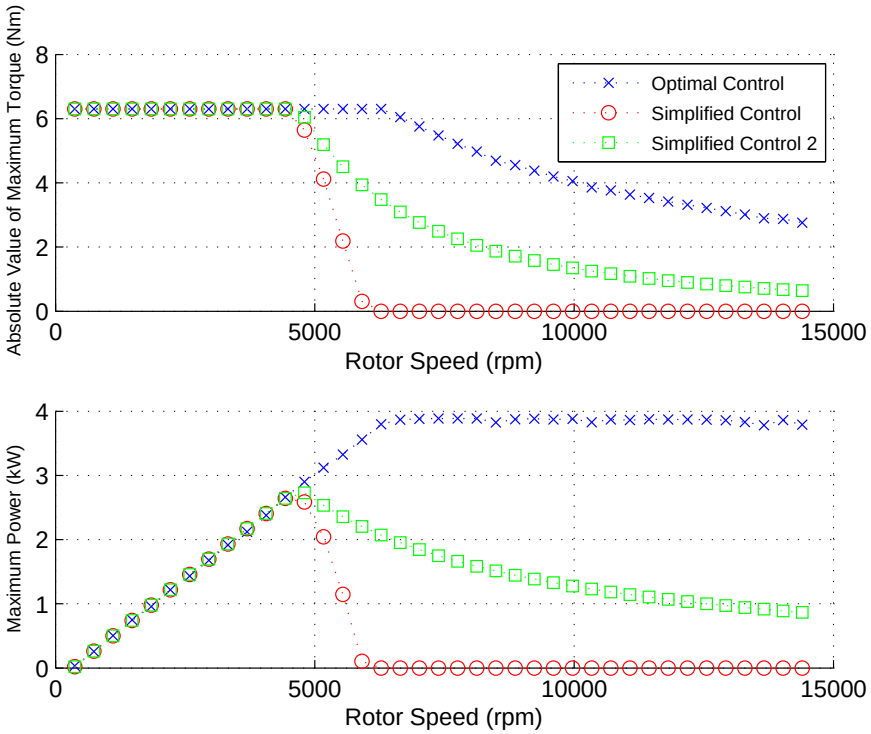


Figure 3.13: Torque and output power envelopes of optimal and simplified controls.

As one would expect, the torque that can be achieved using the simplified control is much less as speeds extend beyond rated speed. This is due to the fact that if the field current is held fixed and the *d*-axis current fixed at zero, no field weakening occurs. Thus the *q*-axis current achievable is diminished. Of course, the simplified control can be modified so that the field current is reduced in proportion to rotor speed. To consider such a method, a study was performed in which the field current is adjusted according to

$$\begin{cases} i_{fd} = i_{fd,sc}, & \omega_{rm} \le \omega_{rm,rated} \\ i_{fd} = i_{fd,rated}\left(\dfrac{\omega_{rm,rated}}{\omega_{rm}}\right), & \omega_{rm} > \omega_{rm,rated} \end{cases} \tag{3.22}$$

and the *q*-axis current adjusted to solve (3.17). In (3.22), $i_{fd,sc}$ is the field current for simplified control at less than rated speed (3.28 A), $i_{fd,rated}$ is the field current at the optimized 2 kW level (3.8 A) and $\omega_{rm,rated}$ is the rated speed of the machine (3600 rpm). The resulting torque envelope is shown in Figure 3.13 as Simplified Control 2. One can

observe from the curve that the field weakening of the field does enable an increase in available torque. However, it remains much less than that of the optimal control.

There are many questions that arise for optimization of a machine intended to operate over a wide speed range. Addressing them is outside the scope of this paper. However one question that was of interest is whether a machine can be designed with a wide speed range and yet with *d*-axis current fixed at zero. To consider this question, an optimization study was performed. Within the study, the stator phase angle, which is gene 15 in Table 3.1, is set to 180° so that *d*-axis current is zero. In addition, an extra constraint is added so that the rated output power (2 kW) is obtained at four times rated speed with one fourth of the field current used at rated speed.

To obtain a perspective on the potential mass penalty that results from setting *d*-axis current to zero, a repeat of the original 2 kW design was performed with the updated *BH* properties included. For this case the *d*-axis current is allowed to be nonzero. With the three currents to manipulate, all the machines can achieve rated torque at 2 kW at 3600 rpm and constant power at four times rated speed. Conduction loss of the rectifier is included within the loss calculation in both cases.

The resulting Pareto fronts of power loss at rated speed versus mass is shown in Figure 3.14. As shown in Figure 3.14, at rated speed, the machines that are designed assuming the use of Simplified Control 2 with the constant power constraint have more mass for a given loss than the machines designed for the machines with the optimal control. Comparing the fronts provides some measure of the cost (increase in mass) of keeping with a simplified control under variable speed operation. For systems with higher loss, the mass difference is relatively small. However, as loss decreases, the difference in mass becomes more appreciable.

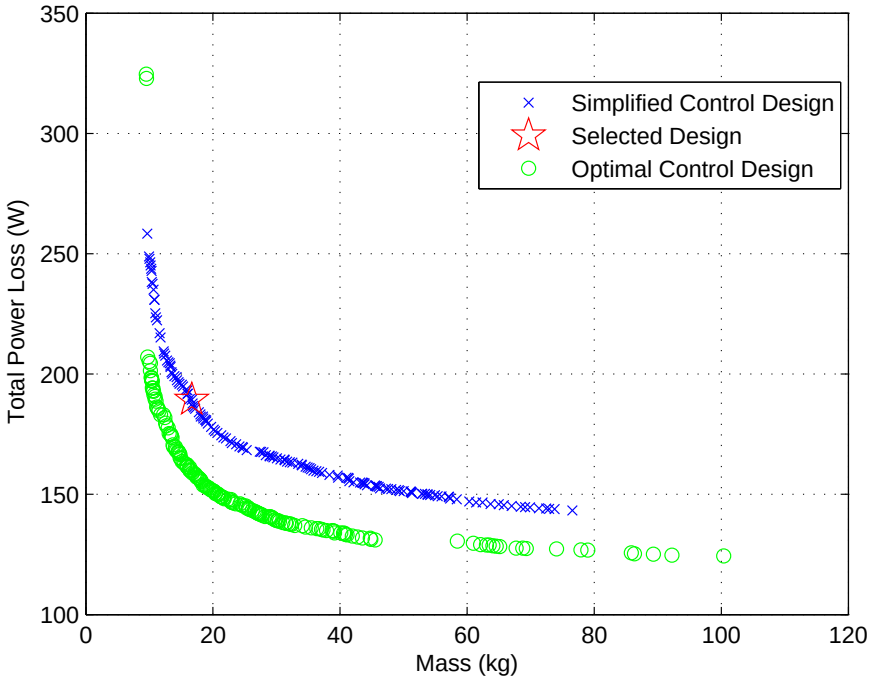


Figure 3.14: Comparison of Pareto fronts.

To investigate the difference in the machines from the two fronts, the conduction loss, core loss, and resistive loss have been compared for rated speed/rated torque conditions in Figure 3.15. A comparison of different design variables in the two design optimizations is shown in Figure 3.16 to help understand how the designs are different. As shown in Figure 3.16, although the field currents at rated speed for both designs are very close, the designs assuming Simplified Control 2 tend to have a much smaller field current at high speed (one fourth of the rated value at four times rated speed). Therefore, a larger stator current is required to compensate for the torque reduced by the smaller field current. This increases the rectifier conduction loss. Since the modified designs have larger stator current, the optimization process tends to use less stator turns in order to reduce the stator resistance, thus the resistive loss of the machines designed to use Simplified Control 2 and optimal control are very close. Moreover, since negative *d*-axis current helps to reduce core loss as discussed previously, setting them to zero one expects to have more core loss. It is also interesting that in general, the size of machines and turns of the field winding created by both designs are very close. The key difference is the

stator winding turns and current. If one summarizes these trends, one can surmise that for systems in which the conduction loss is a small percentage of overall loss, the two fronts would approach each other.

As a final study, a machine (shown as a star in Figure 3.14) was selected for evaluation of the excitation optimization. Following (3.17)-(3.21), the same control optimization process was applied to generate the torque and output power envelopes of simplified control 2 for this machine. The envelopes are shown in Figure 3.17. Comparing the torque and power envelopes of the Simplified Control 2 with that observed for Simplified Control 2 of the original machine shown in Figure 3.13, one can see that the speeds over which constant torque is achieved is expanded significantly. In addition, once rated torque cannot be achieved, the field weakening leads to a torque envelop that yields in excess of 2 kW power at speeds up to four times rated. Thus, one observes that it is possible to have a simplified field-oriented type control with *d*-axis current set to zero and yet have a wide constant power range, provided the simplified control is included in the design stage.

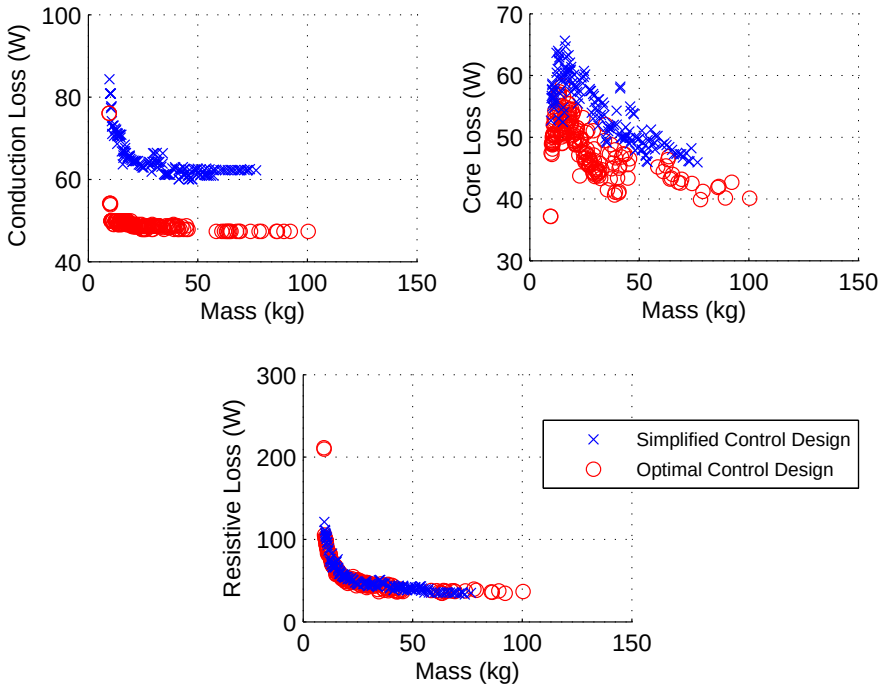


Figure 3.15: Comparison of conduction loss, core loss, and resistive loss.

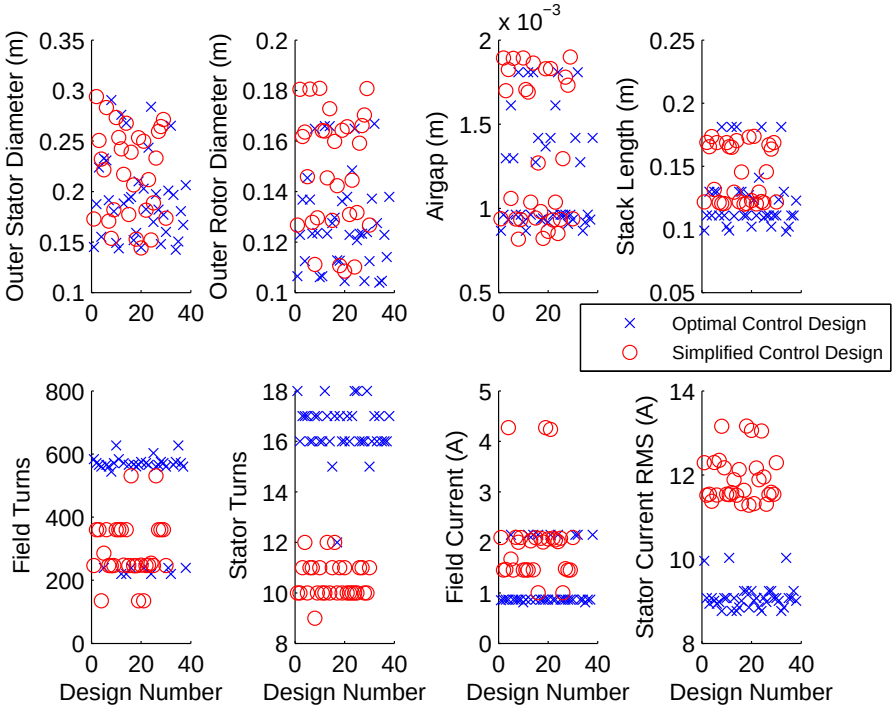


Figure 3.16: Comparison of design variables in variable speed design and rated speed design.

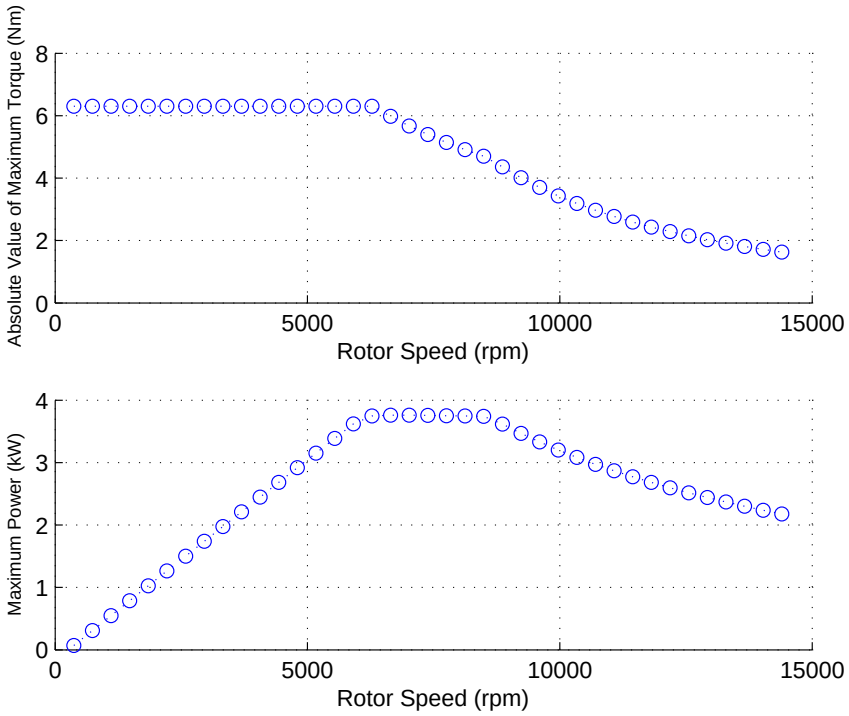


Figure 3.17: Torque and output power envelopes of simplified control 2 using a simplified control design.

3.8 Discussion

In Section 3.1 several questions were raised and it is helpful to consider them in light of the results presented. From the control perspective, it has been found that when establishing the torque versus current map there are significant differences between the currents that are obtained from the MEC and those one would obtain using traditional *qd* models. The differences come from the impact of saturation as well as the influence of core loss. At first glance this is discouraging since the resulting 'optimal' torque/current map from the MEC model is difficult to express analytically. However, through analysis of the optimized currents, an alternative simplified control is obtained that is straightforward to implement. Its main property – a linear map between torque and *q*-axis current- is precisely what drive control designers seek. The caveat of the simplified control is that one must be willing to accept an increase in loss over an 'optimized' current.

This leads to a question of whether a control designer needs to communicate their desire to use a simplified control to the machine designer? For variable speed applications the results in Section 3.7 show the answer is yes. Without this communication, the torque versus speed capabilities of the drive is greatly diminished under the simplified control. Moreover, the machine designer will be able to inform the control engineer of the added cost of the simplified control since, as shown in Figure 3.14, the mass of the machine may increase.

Finally, one may ask how this research applies to applications in which one can adjust the commanded prime mover angular velocity. Going back to Figure 3.2, in this case, the output of the voltage regulator is now a power command. Due to the capability to adjust speed, an optimization can be performed to obtain the torque/speed combination:

$$\text{Minimize} \qquad P_{loss}(T_e, \omega_r) \tag{3.23}$$

$$\text{Subject to} \qquad P(T_e, \omega_r) = P^* \tag{3.24}$$

where the loss includes those of the WRSM, active rectifier, prime mover, and rotation. The output of the optimization is a torque command provided to the electric drive and a speed command provided to the prime mover. Again, a torque command to current command map is required. The results of Section 3.7 are readily applied, with the caveat that for a wide speed range, any desire to use a simplified control requires one to include the control as part of the machine design process.

4. DYNAMIC MAGNETIC EQUIVALENT CIRCUIT MODEL

4.1 Enhanced MEC Network

The dynamic MEC-based model is designed to predict the performance of a salient-pole WRSM with an arbitrary number of poles, integer number of slots/pole/phase, and damper bars. An example cross section of a 4-pole WRSM with 3 damper bars is shown in Figure 4.1. The *q*-, *d*-, and *as*-axis of the machine are also listed. It is noted that mechanical rotor position θ_{rm} is defined by the position of the *q*-axis with respect to the *as*-axis.

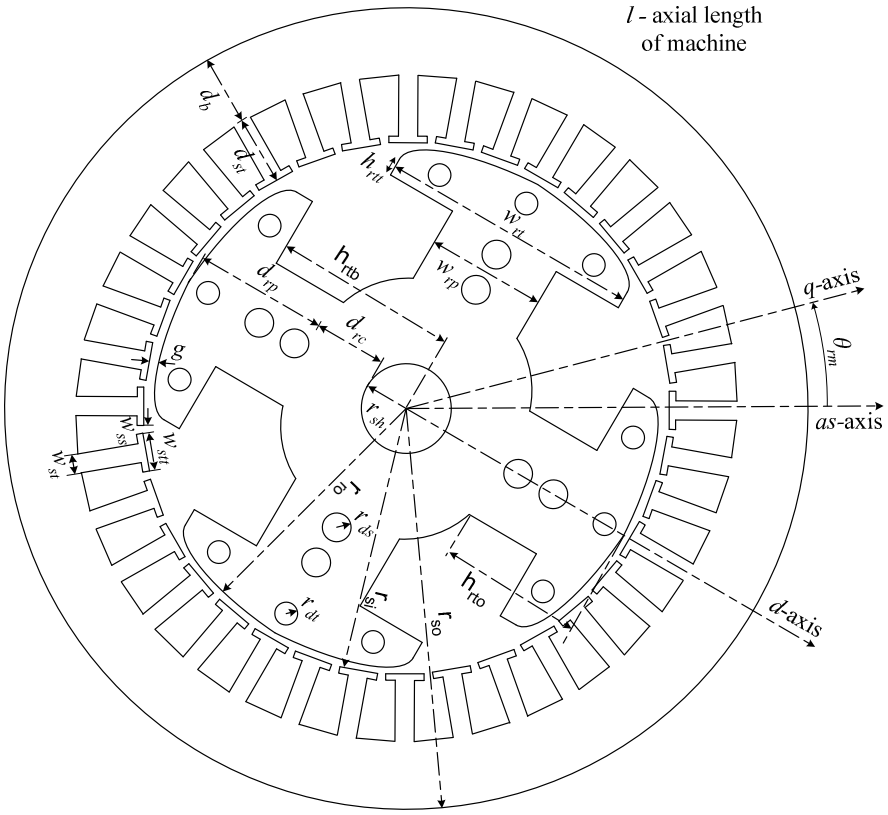


Figure 4.1: Example WRSM geometry/configuration.

An initial network of the proposed MEC is shown in Figure 4.2, wherein loop flux Φ is defined in the clockwise direction. Within the network, each stator and field coil becomes a MMF source in the loop where the respective current is located. Single-pole symmetry is applied to reduce the number of unknowns [23]. Therefore, the MEC network shown includes a single pole. Regarding the network, the reluctances of the stator leakage (R_{TL}), stator yoke (R_Y), rotor interpolar region (R_{RY}), rotor shank (R_{RSH}), rotor yoke (R_{RYP}), and the nonzero airgap reluctances (R_{ag}) at the respective θ_{rm} are identical to those developed for the steady state model [1].

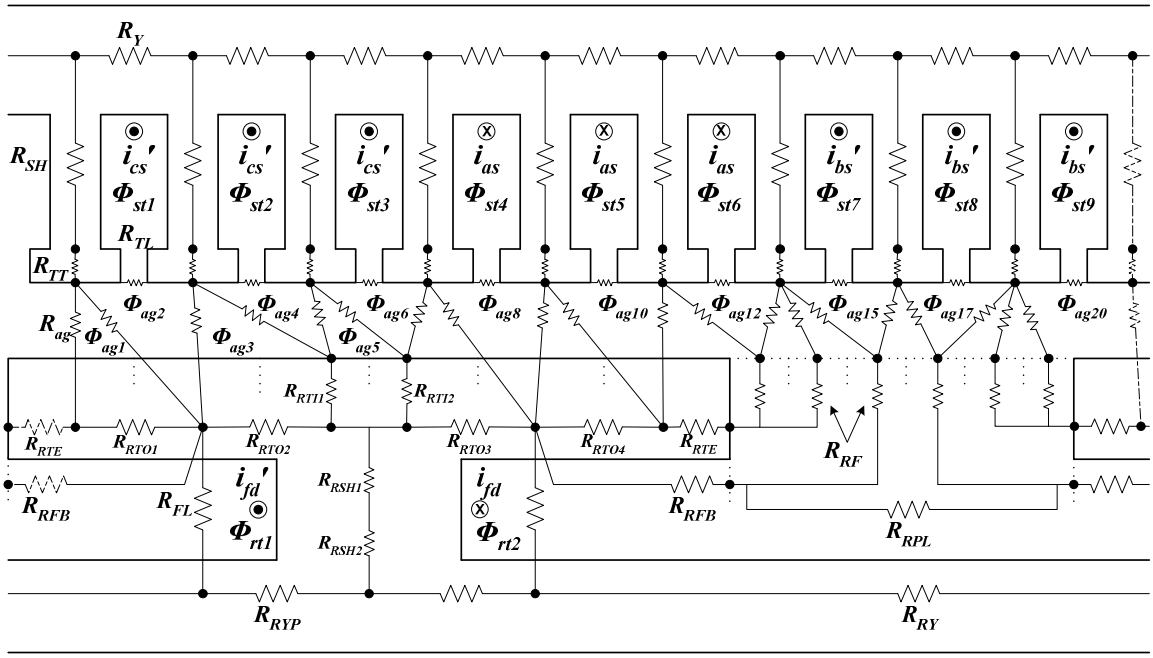


Figure 4.2: Representative WRSM MEC with damper bars inactive.

The first enhancement that the dynamic MEC network provides is that the stator tooth is divided into two components, that is stator tooth shank (R_{SH}) and stator tooth tip (R_{TT}). The challenge of this effort is to determine the airgap permeance based on the updated network. Since the airgap flux tubes are formed between the stator and rotor, they are dependent on the rotor position. For purposes of calculating the airgap permeances, the stator, rotor pole, and rotor slot are all discretized into subsections. The stator is discretized by the number of stator teeth. In general, the number of rotor pole and

slot sections can be user-defined variables. The airgap permeance between the i-th stator tooth (STi) and the j-th rotor section (RSj) is calculated as a parallel combination of flux tubes that represent flux paths directly from a stator tooth to a rotor section and fringing from the side of a stator tooth to a rotor section. In an automated design program considering arbitrary geometries, the calculation of the airgap permeance is dependent on several factors. Specifically, one must know how the angular span of the rotor section compares to the angular spans of the stator tooth and half the stator slot. In [1], logic that was used to determine overlap angles for arbitrary geometries is provided, assuming the stator teeth do not have tooth tips. Within the enhanced MEC model, the same logic is used to determine the reluctance between stator teeth and rotor sections is applied using the geometry of the stator tooth tip to establish angular overlap.

The uniqueness of the MEC network for the dynamic model is centered on the reluctance network of the rotor pole tips. A goal is to develop a general model that can be applied for arbitrary number of damper bars and also, at their arbitrary positioning (with some limitation), both horizontally and vertically. An issue that is often confronted by manufacturers is that a single lamination is used across a large product range. Thus, damper bar holes are often included in rotor laminations, but in some products left unfilled. Within the model, provisions are included to represent damper bar holes that are inactive and those that are active.

For the case in which the damper bar currents are inactive, the MEC network is shown in Figure 4.2. Therein it is shown the flux tubes that represent the rotor pole tip include the “inner” pole tip (R_{RTIi}), the “outer” pole tip (R_{RTOi}), and the “outer end” of the pole tip (R_{RTEi}). Within the model, it is assumed that to the left and right of the pole body flux mainly flows tangentially, and directly above the rotor pole body, flows radially. If an outer section includes a damper hole, the value of R_{RTOi} is derived assuming the tube geometry is a rectangular section of steel with a cylindrical damper hole at the center. This has been found to provide a reasonable estimate of the tangential flux flow in the outer sections.

For the case in which the damper bar currents are active, the MEC network in the rotor changes appreciably as shown in Figure 4.3. Specifically, it is observed from 2D

FEA that a leakage path exists around a damper hole and the leakage flux varies appreciably according to the depth of damper hole. Therefore, if an outer section R_{RTOi} includes an active damper bar, then the section is represented using a parallel combination of two reluctances R_{RTOi}^{*} and R_{RLOi}. The reluctance R_{RTOi}^{*} is used to represent a main path in which flux flows in the same direction of R_{RTOi}. The reluctance R_{RLOi} is used to represent a leakage path around a damper bar.

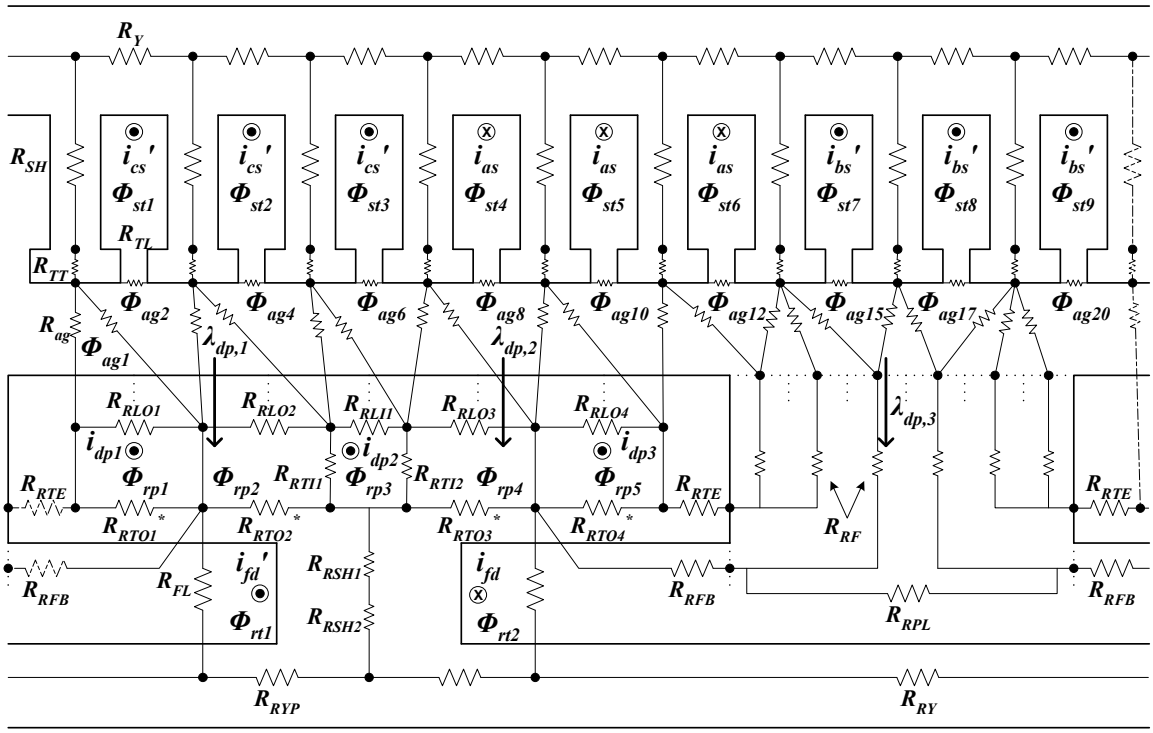


Figure 4.3: Representative WRSM MEC with damper bars active.

4.1.1 Stator flux tubes

As can be seen from Figure 4.2 and Figure 4.3, the stator is composed of 4 types of flux tubes, the stator tooth tip (R_{TT}), stator tooth shank (R_{SH}), stator yoke (R_Y), and stator tooth leakage (R_{TL}). A close-up of configuration of stator flux tubes is shown in Figure 4.4.

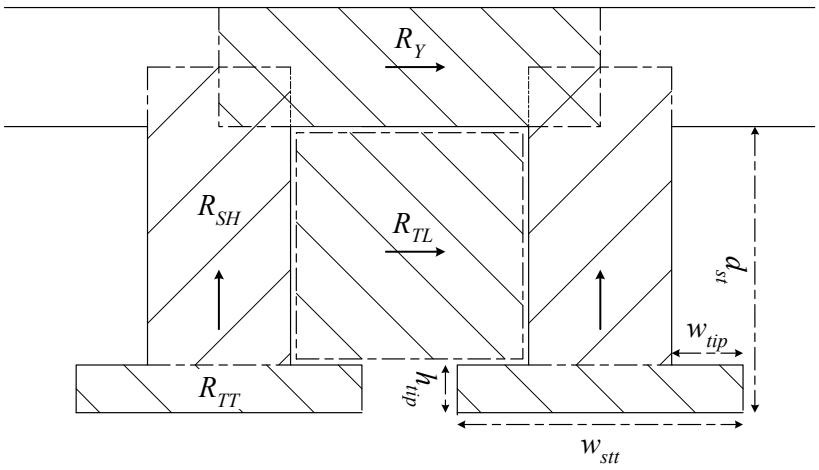


Figure 4.4: Illustration of stator flux tubes.

Within the model, w_{stt} and h_{tip} represent the width and length of stator tooth tip respectively, while $\left(w_{stt} - 2w_{tip}\right)$ and $\left(d_{st} - h_{tip}\right)$ are the width and length of stator tooth shank respectively. As for the stator yoke, the width and length are given as d_b and $2\pi / N_{st}\left(r_o - d_b / 2\right)$ respectively, where N_{st} is the number of stator teeth. The lengths of R_{SH} and R_Y are selected as the mean path length and the equipotential planes intersect to form a node in the MEC. The reluctances for the stator tooth shank, the tooth tip, and the yoke are calculated as,

$$R_{SH} = \frac{d_{st} - h_{tip}}{\mu l(w_{stt} - 2w_{tip})} \tag{4.1}$$

$$R_{TT} = \frac{h_{tip}}{\mu l w_{stt}} \tag{4.2}$$

$$R_Y = \frac{2\pi / N_{st}\left(r_{so} - d_b / 2\right)}{\mu l d_b} \tag{4.3}$$

where μ is the magnetic permeability. The calculation of stator tooth leakage reluctance (R_{TL}) is provided in [60].

4.1.2 Flux tubes in the rotor pole with damper holes

In [1], the reluctance network of the rotor did not account for support or damper bar holes. In the enhanced model, such holes are included. To establish the difference between the models, it is convenient to first consider the model without damper holes which is taken directly from [1]. Without damper holes, the general configuration of the various rotor tooth tip flux tubes is illustrated in Figure 4.5. The basic idea behind the configuration is that to the left and right of the shank, flux mainly flows tangentially in the tooth tip; and directly above the rotor shank, flux flows radially.

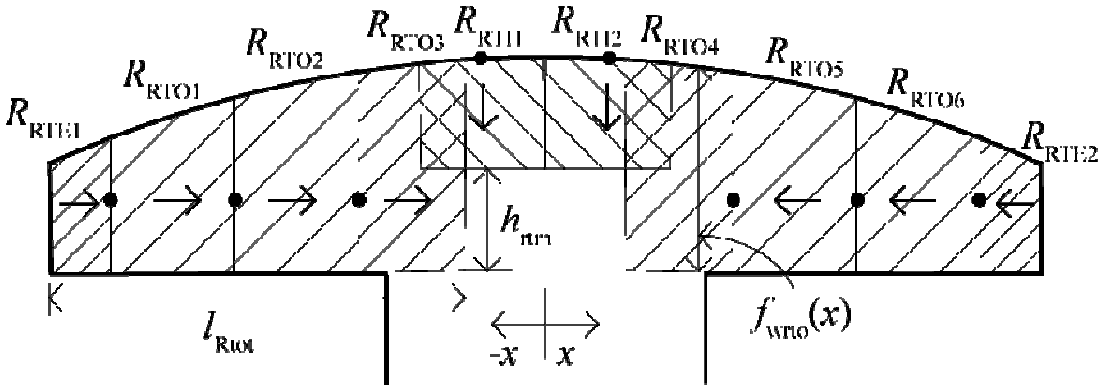


Figure 4.5: Description of rotor tooth tip flux tubes.

The flux tube at the outer edge of the rotor tooth tip is represented by reluctance R_{RTEi}, and it provides a path for fringing through the side of the rotor tooth. The length of the flux tube is half of a rotor tooth section. The width can be estimated by the function $f_{wrto}(x)$ shown in Figure 4.5, which is established from simple geometry,

$$f_{wrto}(x) = \sqrt{r_{ro}^2 - (x)^2} - h_{rtb} \tag{4.4}$$

where x is the distance from the center of the rotor shank to the middle of the respective flux tube. The reluctance expression is given by

$$R_{RTEi} = \frac{w_{rts}/2}{\mu_{RTEi} f_{wrto}\left(w_{rt}/2 - w_{rts}/4\right) l} \tag{4.5}$$

where w_{rts} is the width of a rotor tooth section (w_{rt}/N_{rts}), and N_{rts} is the number of user-defined rotor tooth sections.

The remaining tangential flux tubes are represented by the outer rotor tooth tip reluctances (R_{RTOi}). The number of outer reluctances is dependent on the number of rotor tooth sections (N_{rts}) and on the total length of tangential reluctances, l_{Rtot}, which is defined herein as,

$$l_{Rtot} = \left(w_{rt} - w_{rp}\right)/2 + \min\left(w_{rp}/4, h_{rtm}\right) \tag{4.6}$$

where $h_{rtm} = f_{wrto}(w_{rp}/2)/2$. The length of the individual flux tubes is equal to w_{rts} except for the inner-most flux tube which has a length in Figure 4.5 of $l_{RTO3} = l_{Rtot} - 2.5w_{rts}$. The approximate width of each flux tube is again determined using $f_{wrto}(x)$ from (4.4).

As for the inner rotor pole tip section (R_{RTI}), the width is equal to w_{rts}, and the length is calculated as,

$$l_{RTIi} = f_{wrto}(x) - h_{rtm} \tag{4.7}$$

Similarly, the rotor pole shank reluctance (R_{RSH}) has a width of w_{rp} and a length of $\left(d_{rc}/2 + h_{rp} + h_{rtm}\right)$.

Next, if damper bar opening are included and the damper currents are inactive, the flux tubes of the rotor sections, except for the outer edge of the pole tip (R_{RTE}), become non-uniform flux tubes. This is shown using a representative pole with hole openings in Figure 4.6. In general, the ideas of having tangential flux tubes to the left and right of the shank and radial tubes above the rotor shank is continued. However, the tubes are modified to incorporate the effects of the holes.

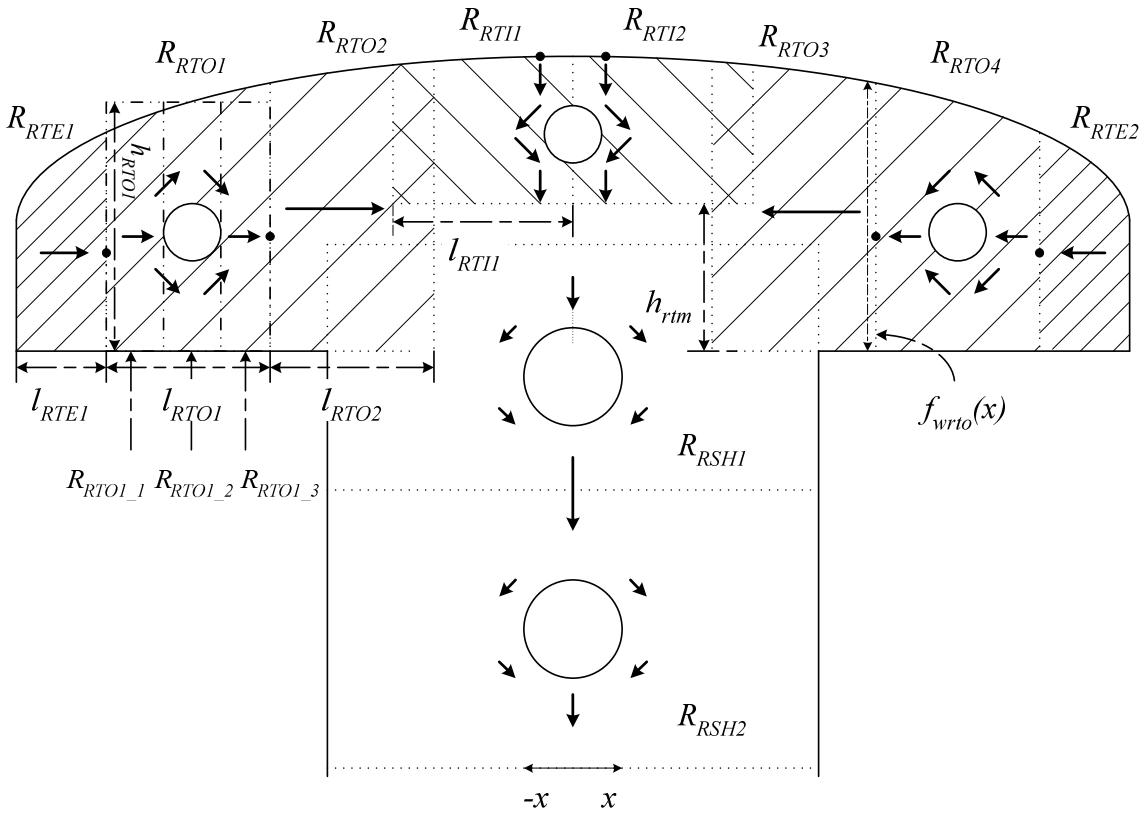


Figure 4.6: Illustration of rotor pole shank and rotor pole tip flux tubes.

Herein the highlighted section R_{RTO1} is used as an example to illustrate the derivation of the reluctance for a flux tube with a damper bar opening. A close-up of the highlighted section R_{RTO1} is shown in Figure 4.7. In order to derive the reluctance of the flux tube, an assumption is made that the damper holes are placed at the center of a respective rotor pole section.

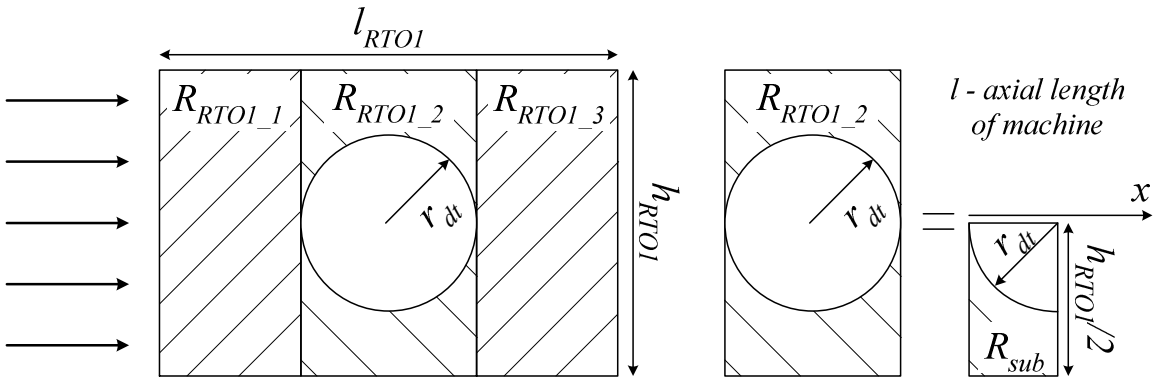


Figure 4.7: Configuration of rotor pole tip flux tube with damper bar.

In Figure 4.7, R_{RTO1} is divided into three subsections, R_{RTO1_1}, R_{RTO1_2}, and R_{RTO1_3}. R_{RTO1_1}, and R_{RTO1_3} are subsections of R_{RTO1} before and after the damper hole. R_{RTO1_2} is the subsection of R_{RTO1} that contains the damper hole. Since all three are serially connected and R_{RTO1_1} and R_{RTO1_3} are assumed to have the same cross-sectional area, R_{RTO1_1} and R_{RTO1_3} can be combined as a single reluctance $R_{RTO1\_1,3}$. It is determined using

$$R_{RTO1\_1,3} = \frac{l_{RTO1} - 2r_{dt}}{\mu l h_{RTO1}} \tag{4.8}$$

where h_{RTO1} is the width of the section obtained using (4.4). Using symmetry and considering the appropriate series and parallel combinations, one can obtain the reluctance of the subsection with the damper hole (R_{RTO1_2}) through consideration of the reluctance of only a quarter of the subsection region as shown in Figure 4.7. Specifically, it can be shown that R_{RTO1_2} and R_{sub} are equal. To calculate their value,

$$R_{RTO1\_2} = \int_0^{r_{dt}} \frac{dx}{\mu l\left(\frac{h_{RTO1}}{2} - \sqrt{r_{dt}^2 - x^2}\right)} = -\frac{\pi}{2\mu l} + \frac{h_{RTO1}}{\mu l\sqrt{h_{RTO1}^2 - 4r_{dt}^2}}\left[\frac{\pi}{2} + \tan^{-1}\left(\frac{2r_{dt}}{\sqrt{h_{RTO1}^2 - 4r_{dt}^2}}\right)\right] \tag{4.9}$$

The cross-sectional area of the component R_{RTO1_2} used to evaluate the permeability value is the mean value of the section. Finally, the component $R_{RTO1\_1,3}$ and R_{RTO1_2} are combined and represented as R_{RTO1} in the MEC network in Figure 4.2.

A similar approach has been applied to calculate R_{RTIi} when a damper hole is included within the inner pole region.

4.1.3 Flux tubes of rotor pole tip leakage

From observations of flux line distribution using finite elements, leakage path exists around a damper hole when damper current is active. Therefore, leakage reluctance in rotor pole tip is incorporated to the MEC network as shown in Figure 4.3. To derive R_{RLO1}, the section R_{RTO1} is highlighted in Figure 4.6 and enlarged in Figure 4.8 to illustrate the configuration of rotor pole tip leakage flux tube.

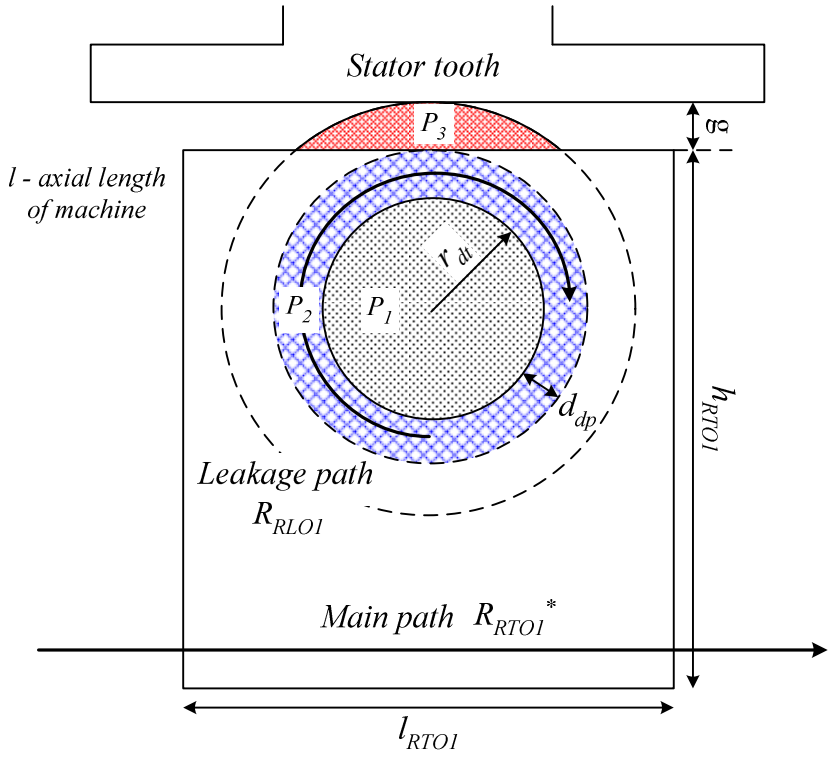


Figure 4.8: Configuration of rotor pole tip leakage flux tubes.

An assumption has been made that the leakage path is circling around the damper slot. Thus the damper slot leakage flux tube (R_{RLO1}) is structured as a parallel combination of three leakage permeances, that is P_1, P_2, and P_3, as shown in the shaded area of Figure 4.8. P_1 represents the leakage path in the copper (or air) inside the damper slot, with a radius of r_{dt}. P_2 represents the leakage path in the steel in the rotor section, which is modeled as a ring with a width of d_{dp} that is equal to the depth of the damper hole. Herein, a scaling factor α_{dp} is introduced to describe the vertical position of the damper holes with respect to the section height. Specifically $\alpha_{dp} = 0$ or $\alpha_{dp} = 1$ then the damper holes locate at the top or the bottom of the rotor pole tip, respectively. Thus, the depth of damper hole d_{dp} is equal to $\alpha_{dp}(h_{RTO1} - 2r_{dt})$. P_3 represents the leakage path in the air gap, in which the MMF drop in the steel is neglected. Therefore, the reluctance of damper slot leakage R_{RLO1} for the cylindrical tube can be expressed as,

$$\frac{1}{R_{RLO1}} = \underbrace{\frac{\mu_0 l}{8\pi}}_{P_1} + \underbrace{\frac{\mu l}{2\pi}\ln\left(\frac{d_{dp}+r_{dt}}{r_{dt}}\right)}_{P_2} + \underbrace{\frac{\mu_0 l}{2}\ln\left(\frac{\sqrt{2g(d_{dp}+r_{dt})+g^2}+g+d_{dp}+r_{dt}}{d_{dp}+r_{dt}}\right)}_{P_3} \tag{4.10}$$

where μ is the magnetic permeability in the steel, and μ_0 is the magnetic permeability in the air. The value of R_{RTO1}^{*} is then calculated in a way to keep the parallel combination of it and R_{RLO1} to be the same as R_{RTO1}. Doing so, the reluctance of R_{RTO1}^{*} can be expressed as,

$$R_{RTO1}^{*} = \left\{ \left\{ \frac{h_{RTO1}}{\mu l\sqrt{h_{RTO1}^{2}-4r_{dt}^{2}}} \left[\frac{\pi}{2} + \tan^{-1}\left(\frac{2r_{dt}}{\sqrt{h_{RTO1}^{2}-4r_{dt}^{2}}}\right) \right] - \frac{\pi}{2\mu l} + \frac{l_{RTO1}-2r_{dt}}{\mu l h_{RTO1}} \right\}^{-1} - \frac{1}{R_{RLO1}} \right\}^{-1} \tag{4.11}$$

One can observe from Figure 4.3 that in the outer pole sections the two reluctances are placed in parallel by assuming that the reluctance in the vertical direction is negligible. As for those rotor sections without damper bars (e.g. R_{RTO2} and R_{RTO3}), the total reluctance of the section is decomposed into two equivalent reluctances placed in parallel in the rotor pole network. For instance, the rotor section R_{RTO2} is decomposed into two branches, that is R_{RLO2} and R_{RTO2}^{*}, in the reluctance network, with values that $R_{RLO2} = R_{RTO2}^{*} = 2R_{RTO2}$.

For an inner section with a damper bar, a leakage reluctance R_{RLIi} calculated in the same fashion of R_{RLOi} is added in between the adjacent two inner sections R_{RTIi}.

In practice, the topology of the network in Figure 4.3 can be applied to machines without active damper bars by simply removing all of the rotor pole tip leakage reluctances and the MMF sources of damper currents. Therefore, the initial MEC network in Figure 4.2 can be replaced by the enhanced MEC network in Figure 4.3.

4.1.4 Damper bar placement

In general, the rotor pole tip can be discretized into a user-defined arbitrary number of sections. The number of damper bars is also a user-defined arbitrary number. If the number of rotor pole tip damper bars is an odd number, then one of the bars is located in the center of the most inner two R_{RTI} sections. Otherwise, with an even number, there is no hole in the center of the most inner two R_{RTIi} sections, but they are symmetrically distributed on the two sides of the rest of the rotor pole sections. Within the design program, the horizontal distribution of the damper bars is described using a damper winding vector as

$$\mathbf{damper\_rtip} = [... \quad r_{dt3} \quad r_{dt2} \quad r_{dt1} \quad r_{dt2} \quad r_{dt3} \quad ...] \tag{4.12}$$

where r_{dti} is the radius of the one in the middle of the rotor pole and the other values are the radii of damper bars at two sides. By manipulating the value of r_{dti} in (4.12), the horizontal distribution and the shape of the damper bars is readily modified. For example, if the number of damper bars on each rotor pole tip is three, a damper winding vector $[0 \quad r_{dt2} \quad 0 \quad r_{dt1} \quad 0 \quad r_{dt2} \quad 0]$ gives a more scattered damper bars distribution compare to a damper winding vector $[0 \quad 0 \quad r_{dt2} \quad r_{dt1} \quad r_{dt2} \quad 0 \quad 0]$.

In addition, the vertical depth of the damper bars can be assigned by adjusting the scaling factor α_{dp} . Therefore, the proposed MEC model provides the ability to investigate both horizontal and vertical placement of the damper bar in the rotor pole tips.

Practically, damper current is not present in the rotor shank. The slot openings in the rotor shank are used to bind the rotor laminations and confine the field windings. Therefore, in practice they are likely not located in the center of the rotor shank but at the edges of the rotor shank. However, the reluctance of the rotor shank component does not change when the holes are placed at different locations along the radial direction.

4.2 Meshed-Based MEC Model Formulation

4.2.1 Single-pole symmetry

Single-pole symmetry has already been studied in [23] in which it was shown that only a single pole is required for analysis of an integer slot/pole/phase machine. Therefore, Figure 4.9 shows an example MEC network with a single pole span. One can imagine that if the MEC network was continued for the pole to the right, the MEC network over a full pole pair can be formed.

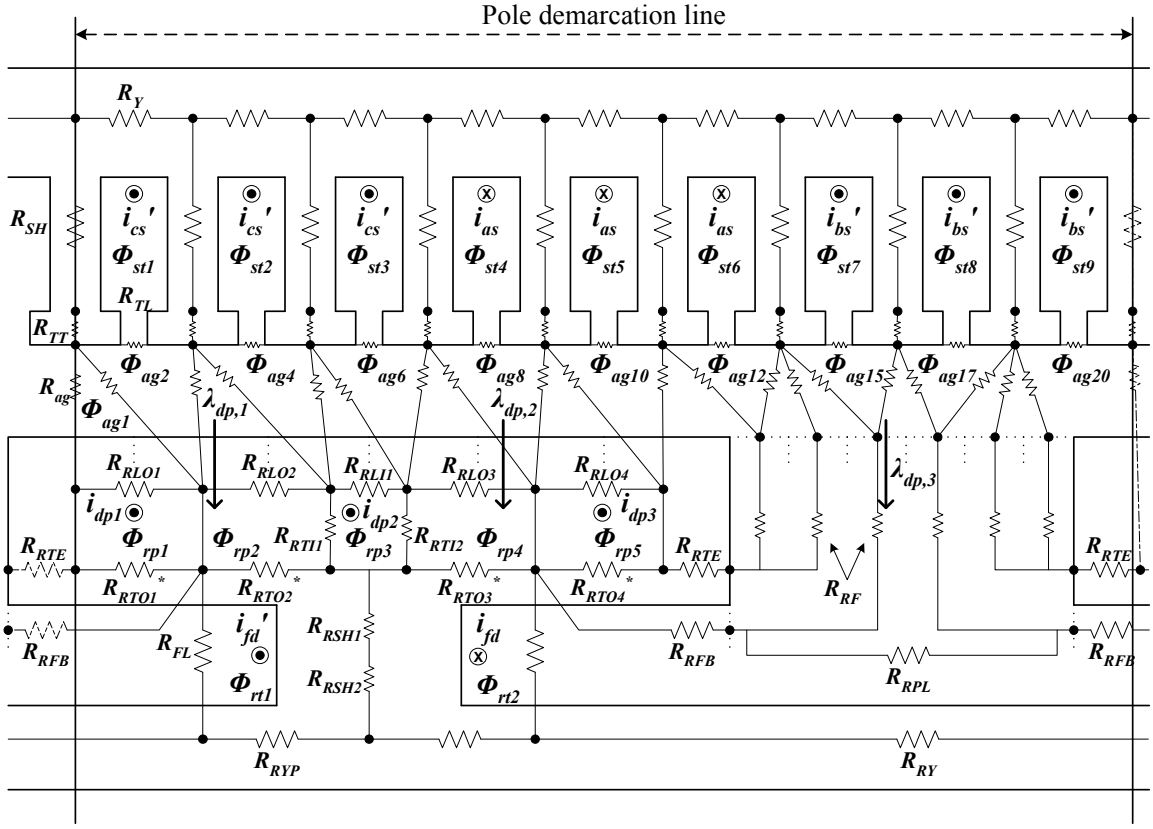


Figure 4.9: Single pole representative of the MEC network.

Considering the symmetry of the magnetic circuit topology in each pole, the reluctance networks are identical on both sides of the pole demarcation line. In addition, the MMF sources have the same amplitude but opposite polarity to the left and right of

the line. As a result, it is apparent that the loop fluxes in the left of the line are equal to the negative of the corresponding loop fluxes in the right.

If damper windings are activated, care must be given to incorporate the single-pole symmetry. Specifically, whenever a rotor tooth tip section crosses the pole demarcation line, the direction of damper winding current must be reversed so that the MMF source of the damper winding has the same amplitude but opposite direction. Moreover, the direction of the flux linkage crossing each of two damper windings has to be reversed since the positive direction changes from one pole to the other.

4.2.2 KVL MEC model

Often, MEC models are structured to explore steady-state behavior in which case the model is structured to accept stator and rotor currents as inputs. Once reluctance values in the network have been determined, a system of nonlinear algebraic equations related to each loop is then established based upon KVL as,

$$\mathbf{A}_{\mathrm{R}}^{(nl\times nl)}\boldsymbol{\varphi}_{\mathrm{l}}^{(nl\times 1)}=\mathbf{F}_{\mathrm{l}}^{(nl\times 1)} \tag{4.13}$$

where $\mathbf{A}_{\mathrm{R}}$ is a symmetric matrix composed of reluctances, $\boldsymbol{\varphi}_{\mathrm{l}}$ is a vector of loop fluxes, $\mathbf{F}_{\mathrm{l}}$ is a vector of MMF sources, and nl is the number of loops. The loop flux vector $\boldsymbol{\varphi}_{\mathrm{l}}$ can be expanded as,

$$\boldsymbol{\varphi}_{\mathrm{l}}=\left[\phi_{\mathrm{st1}}\ \cdots\ \phi_{\mathrm{st}ns}\ \ \phi_{\mathrm{rt1}}\ \cdots\ \phi_{\mathrm{rt}nr}\ \ \phi_{\mathrm{ag1}}\ \cdots\ \phi_{\mathrm{ag}na}\ \ \phi_{\mathrm{rp1}}\ \cdots\ \phi_{\mathrm{rp}np}\right]^{T} \tag{4.14}$$

where the subscripts ‘st’, ‘rt’, ‘ag’, and ‘rp’ indicate loop fluxes in the stator, rotor, airgap, and rotor pole tip leakage respectively, and the subscripts ‘ns’, ‘nr’, ‘na’, and ‘np’ denote the number (per pole) of the stator slots, rotor loops, airgap loops, and rotor pole tip leakage loops respectively. The source vector $\mathbf{F}_{\mathrm{l}}$ can be expressed as,

$$\mathbf{F}_{\mathrm{l}}=\left[\mathbf{F}_{\mathrm{st}}^{(ns\times 1)^{\mathrm{T}}}\ \ \mathbf{F}_{\mathrm{rt}}^{(nr\times 1)^{\mathrm{T}}}\ \ \mathbf{0}^{(na\times 1)^{\mathrm{T}}}\ \ \mathbf{F}_{\mathrm{rp}}^{(np\times 1)^{\mathrm{T}}}\right]^{\mathrm{T}} \tag{4.15}$$

The mmf source in the stator loops is given by,

$$\mathbf{F}_{\mathrm{st}}^{(ns\times 1)}=\mathbf{N}_{\mathrm{abc}}^{(ns\times 3)}\mathbf{i}_{\mathrm{abcs}}^{(3\times 1)} \tag{4.16}$$

where $\mathbf{i}_{\mathrm{abcs}}$ is a vector of balanced stator currents and the turns matrix $\mathbf{N}_{\mathrm{abc}}$ is built using the *a*, *b*, and *c*-phase turn vectors. The mmf in the rotor loops is given by,

$$\mathbf{F}_{\mathrm{rt}}^{(nr\times 1)} = \mathbf{N}_{\mathrm{rt}}^{(nr\times 1)} I_{\mathrm{fd}} = \left[-1\ 1\ 0\right]^{\mathrm{T}} N_{\mathrm{fd}} I_{\mathrm{fd}} \tag{4.17}$$

where I_{fd} is the field current and N_{fd} is the number of field turns. Due to the use of single-pole symmetry, the sign of the rotor MMF changes with rotor position.

The last element in $\mathbf{F}_{\mathrm{l}}$, i.e. $\mathbf{F}_{\mathrm{rp}}$, represents the damper winding mmf source within the meshes of the rotor pole tip leakage. It can be expressed as,

$$\mathbf{F}_{\mathrm{rp}}^{(np\times 1)}(j) = \mathbf{N}_{\mathrm{dp}}^{(np\times nd)}(j,k)\,\mathbf{i}_{\mathrm{dp}}^{(nd\times 1)}(k) \tag{4.18}$$

where the subscript 'nd' denotes the number of damper bars on each rotor pole tip. $\mathbf{F}_{\mathrm{rp}}^{(np\times 1)}(j)$ is the j^{th} rotor pole tip leakage loop MMF, $\mathbf{i}_{\mathrm{dp}}^{(nd\times 1)}(k)$ is the k^{th} damper winding current. $\mathbf{N}_{\mathrm{dp}}^{(np\times nd)}(j,k)$ indicates the number of damper winding turns, which has a value of 1 if the k^{th} damper winding current is in the j^{th} rotor pole tip leakage loop and, otherwise, has a value of 0. For example, for the geometry shown in Figure 4.3,

$$\mathbf{F}_{\mathrm{rp}}^{(5\times 1)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{i}_{\mathrm{dp}}^{(3\times 1)} \tag{4.19}$$

The derivation of dynamic system equations in the remainder of this section is based upon a configuration in which there is a pole to pole connection between the damper windings. However, the proposed model is readily modified to the case in which damper winding connections are only made on a single pole by using the fact that the damper winding currents satisfy the relationship,

$$\mathbf{i}_{\mathrm{dp}}(nd) = -\sum_{k=1}^{nd-1} \mathbf{i}_{\mathrm{dp}}(k) \tag{4.20}$$

Using (4.20) one can see that, a number of (*nd*-1) damper winding current is needed to be solved and all of the entries of the m^{th} row of the matrix $\mathbf{N}_{\mathrm{dp}}$ are -1, where *m* is the rotor pole tip leakage loop index that $\mathbf{i}_{\mathrm{dp}}(nd)$ is present.

4.3 Dynamic System Equations

Prior to deriving the dynamic model, it is convenient to view the intended dynamic model structure in the block diagram form shown in Figure 4.10. Therein it can be seen that a dynamic model is obtained by first restructuring the KVL MEC system of equations so that stator and damper winding flux linkage is used as an input to the MEC model, and stator and damper winding current is an output of the MEC model. State equations are then established to obtain stator and damper winding flux linkage based upon winding voltage and current, which is obtained from the coupling to external circuits and the MEC respectively. From Figure 4.10, unlike the stator and damper winding currents, the field winding currents remain an input to the MEC derived herein. This is used to consider machines in which the field winding is coupled to a power electronic circuit that acts as a current source. For the case in which the field winding is connected to a power electronic circuit that appears as a voltage source (i.e. a rotating rectifier exciter), the field winding dynamics are readily included using a similar approach that is applied to the stator and damper windings.

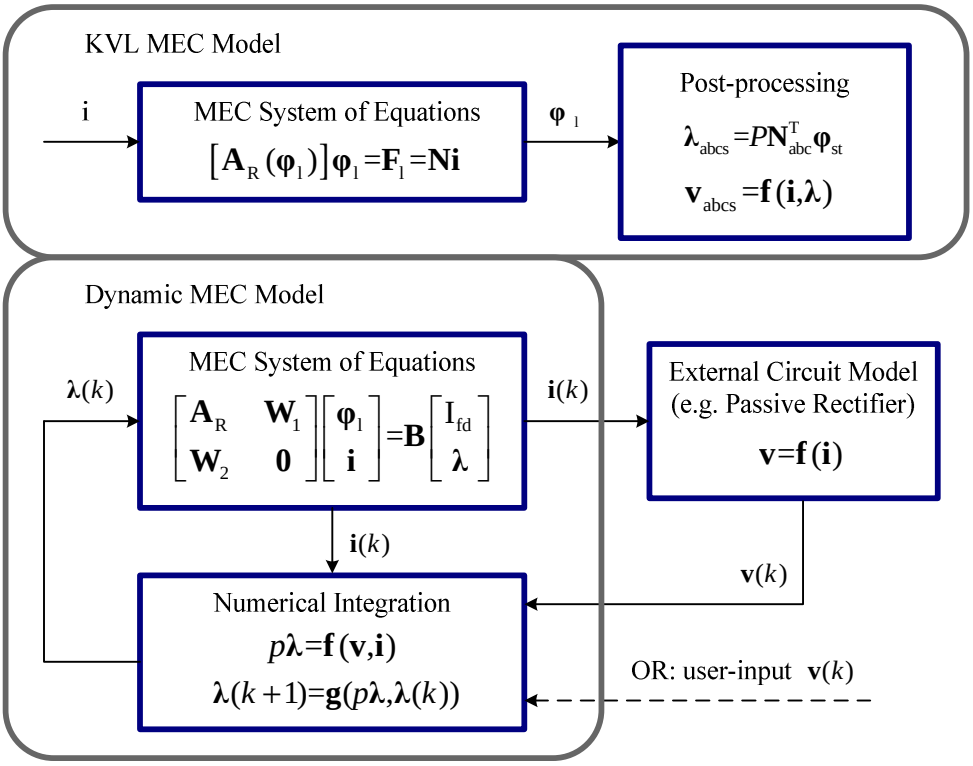


Figure 4.10: Basic structure of the dynamic model shown in contrast with the KVL model.

As a first step in restructuring the MEC model, (4.13) is expanded as,

$$\mathbf{A}_R\boldsymbol{\varphi}_l-\mathbf{N}_{l,abc}\mathbf{i}_{abcs}-\mathbf{N}_{l,dp}\mathbf{i}_{dp}=\mathbf{N}_{l,fd}\mathrm{I}_{fd} \tag{4.21}$$

where the turns matrices are defined as,

$$\mathbf{N}_{l,abc}^{(nl\times 3)}=\begin{bmatrix}\mathbf{N}_{abc}^{(ns\times 3)}\\ \mathbf{0}^{((nr+na+np)\times 3)}\end{bmatrix} \tag{4.22}$$

$$\mathbf{N}_{l,fd}^{(nl\times 1)}=\begin{bmatrix}\mathbf{0}^{(ns\times 1)}\\ \mathbf{N}_{rt}^{(nr\times 1)}\\ \mathbf{0}^{((na+np)\times 1)}\end{bmatrix} \tag{4.23}$$

$$\mathbf{N}_{l,dp}^{(nl\times nd)}=\begin{bmatrix}\mathbf{0}^{((ns+nr+na)\times nd)}\\ \mathbf{N}_{dp}^{(np\times nd)}\end{bmatrix} \tag{4.24}$$

Next, the system matrix $\mathbf{A}_R$ in (4.21) is augmented so that the loop flux is not only related to the MMF sources, but also to the flux linkage. To do so, the stator flux linkage is first expressed as,

$$\boldsymbol{\lambda}_{\text{abcs}} = P\mathbf{N}_{\text{l,abc}}\boldsymbol{\varphi}_{\text{st}} \tag{4.25}$$

where P is the number of poles. A matrix $\mathbf{M}_{\text{l,dp}}$ is used to relate the damper bar flux linkage (which is identical to flux since there is only a single turn) to the loop fluxes $\boldsymbol{\varphi}_{\text{l}}$. Specifically,

$$\boldsymbol{\lambda}_{\text{dp}}^{(nd\times 1)} = \mathbf{M}_{\text{l,dp}}^{(nd\times nl)}\boldsymbol{\varphi}_{\text{l}}^{(nl\times 1)} = \left[\mathbf{0}^{(nd\times(nl-nd))} \;\; \mathbf{M}_{\text{l,dp\_sub}}^{(nd\times nd)}\right]\boldsymbol{\varphi}_{\text{l}}^{(nl\times 1)} \tag{4.26}$$

where $\boldsymbol{\lambda}_{\text{dp}}$ is the net flux linkage (flux) between two adjacent damper bars. The net flux crossing damper bars is readily established through inspection of the circuit. In general, it can be shown that the only contributions to the net flux are from the loop flux that circulates around the two corresponding damper bars. For example, from Figure 4.3, the net flux between bars 1 and 2 can be expressed as $\lambda_{dp,1} = \phi_{rp1} - \phi_{rp3}$. The net flux between bars 2 and 3 can be expressed as $\lambda_{dp,2} = \phi_{rp3} - \phi_{rp5}$. The net flux between bar 3 and the first bar in the next pole is obtained using symmetry. Specifically, the loop flux of the first damper bar in the next pole is the opposite of ϕ_{rp1}. Therefore, $\lambda_{dp,3} = \phi_{rp5} + \phi_{rp1}$. Thus, for the circuit shown in Figure 4.3, the relationship between loop fluxes and damper flux linkages can be expressed as,

$$\begin{bmatrix}\lambda_{dp,1}\\ \lambda_{dp,2}\\ \lambda_{dp,3}\end{bmatrix} = \underbrace{\begin{bmatrix}1 & -1 & 0\\ 0 & 1 & -1\\ 1 & 0 & 1\end{bmatrix}}_{\mathbf{M}_{\text{l,dp\_sub}}}\begin{bmatrix}\phi_{rp1}\\ \phi_{rp3}\\ \phi_{rp5}\end{bmatrix} \tag{4.27}$$

Straightforward logic is used to generate the matrix for an arbitrary damper structure.

From the first two steps, the MEC system of equations for the dynamic model is expanded as,

$$\begin{bmatrix}\mathbf{A}_{\text{R}} & -\mathbf{N}_{\text{l,abc}} & -\mathbf{N}_{\text{l,dp}}\\ \mathbf{N}_{\text{l,abc}}^{\text{T}} & 0 & \\ \mathbf{M}_{\text{l,dp}} & & \end{bmatrix}\begin{bmatrix}\boldsymbol{\varphi}_{\text{l}}\\ \mathbf{i}_{\text{abcs}}\\ \mathbf{i}_{\text{dp}}\end{bmatrix} = \begin{bmatrix}\mathbf{N}_{\text{l,fd}} & 0 & 0\\ 0 & \mathbf{I}/P & 0\\ 0 & 0 & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathrm{I}_{\text{fd}}\\ \boldsymbol{\lambda}_{\text{abcs}}\\ \boldsymbol{\lambda}_{\text{dp}}\end{bmatrix} \tag{4.28}$$

where $\mathbf{I}$ is an identity matrix. To simplify further the stator windings can be transformed into an arbitrary reference frame using the following transformation,

$$\mathbf{K}_s = \frac{2}{3}\begin{bmatrix} \cos(\theta) & \cos(\theta - 2\pi/3) & \cos(\theta + 2\pi/3) \\ \sin(\theta) & \sin(\theta - 2\pi/3) & \sin(\theta + 2\pi/3) \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \tag{4.29}$$

$$\mathbf{f}_{qd0s} = \mathbf{K}_s \mathbf{f}_{abcs}$$

where θ is the reference frame position, and f can be voltage (v), current (i), or flux linkage (λ).

Applying the arbitrary reference frame transformation to the MEC system of equations (4.28), the following dynamic MEC system can be obtained,

$$\underbrace{\begin{bmatrix} \mathbf{A}_R & -f_{scale}\mathbf{N}_{l,abc}\left(\mathbf{K}_s\right)^{-1} & -f_{scale}\mathbf{N}_{l,dp} \\ f_{scale}\mathbf{K}_s\mathbf{N}_{l,abc}^T & 0 & \\ f_{scale}\mathbf{M}_{l,dp} & & \end{bmatrix}}_{\mathbf{A}_{dyn}} \begin{bmatrix} \boldsymbol{\varphi}_l \\ \mathbf{i}_{qd0s,scl} \\ \mathbf{i}_{dp,scl} \end{bmatrix} = \begin{bmatrix} \mathbf{N}_{l,fd} & 0 & 0 \\ 0 & f_{scale}\mathbf{I}/P & 0 \\ 0 & 0 & f_{scale}\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathrm{I}_{fd} \\ \boldsymbol{\lambda}_{qd0s} \\ \boldsymbol{\lambda}_{dp} \end{bmatrix} \tag{4.30}$$

where

$$\mathbf{i}_{qd0s} = f_{scale}\mathbf{i}_{qd0s,scl} \tag{4.31}$$

$$\mathbf{i}_{dp} = f_{scale}\mathbf{i}_{dp,scl} \tag{4.32}$$

and f_{scale} is a user-defined scaling factor that is used to increase the magnitude of the smallest terms to avoid an ill-conditioned system matrix. In practice, with $f_{scale} = 10^3$, it has been observed that potential ill-conditioning is eliminated.

In comparing (4.30) to the block diagram in Figure 4.10, relations among the notations are,

$$\mathbf{W}_1 = \left[-f_{\text{scale}}\mathbf{N}_{\text{l,abc}} \left(\mathbf{K}_{\text{s}}\right)^{-1} \; -f_{scale}\mathbf{N}_{\text{l,dp}} \right] \tag{4.33}$$

$$\mathbf{W}_2 = \begin{bmatrix} f_{scale}\mathbf{K}_{\text{s}}\mathbf{N}_{\text{l,abc}}^{\text{T}} \\ f_{scale}\mathbf{M}_{\text{l,dp}} \end{bmatrix} \tag{4.34}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{N}_{\text{l,fd}} & 0 & 0 \\ 0 & f_{scale}\mathbf{I}/P & 0 \\ 0 & 0 & f_{scale}\mathbf{I} \end{bmatrix} \tag{4.35}$$

A Newton-Raphson method is used to solve the dynamic model for the loop fluxes and currents. The Jacobian matrix of the dynamic model in (4.30) is expressed,

$$\begin{aligned} \mathbf{J}_{\text{dyn}} &= \begin{bmatrix} \dfrac{\partial \mathbf{A}_{\text{R}}\boldsymbol{\varphi}_{\text{l}}}{\partial \boldsymbol{\varphi}_{\text{l}}} & -\dfrac{\partial f_{scale}\mathbf{N}_{\text{l,abc}}\left(\mathbf{K}_{\text{s}}\right)^{-1}\mathbf{i}_{\text{qd0s,scl}}}{\partial \mathbf{i}_{\text{qd0s,scl}}} & -\dfrac{\partial f_{scale}\mathbf{N}_{\text{l,dp}}\mathbf{i}_{\text{dp,scl}}}{\partial \mathbf{i}_{\text{dp,scl}}} \\ \dfrac{\partial f_{scale}\mathbf{K}_{\text{s}}\mathbf{N}_{\text{l,abc}}^{\text{T}}\boldsymbol{\varphi}_{\text{l}}}{\partial \boldsymbol{\varphi}_{\text{l}}} & & \\ \dfrac{\partial f_{scale}\mathbf{M}_{\text{l,dp}}\boldsymbol{\varphi}_{\text{l}}}{\partial \boldsymbol{\varphi}_{\text{l}}} & 0 & \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{A}_{\text{R}} + \mathbf{D}_{\text{R}} & -f_{scale}\mathbf{N}_{\text{l,abc}}\left(\mathbf{K}_{\text{s}}\right)^{-1} & -f_{scale}\mathbf{N}_{\text{l,dp}} \\ f_{scale}\mathbf{K}_{\text{s}}\mathbf{N}_{\text{l,abc}}^{\text{T}} & 0 & \\ f_{scale}\mathbf{M}_{\text{l,dp}} & & \end{bmatrix} \\ &= \mathbf{A}_{\text{dyn}} + \begin{bmatrix} \mathbf{D}_{\text{R}} & 0 \\ 0 & 0 \end{bmatrix} \end{aligned} \tag{4.36}$$

where $\mathbf{A}_{\text{dyn}}$ is the augmented system matrix in (4.30), and $\mathbf{D}_{\text{R}}$ is a matrix that contains the partial derivation of the network reluctances with respect to the loop fluxes. Derivation of $\mathbf{D}_{\text{R}}$ is provided in [1].

The next step in the dynamic model development is to establish the state equations of the system that enable calculation of the flux linkages that are inputs in (4.30). The derivation of the state equations is divided into two parts, one for the rotor electrical system, and the second for the stator electrical system.

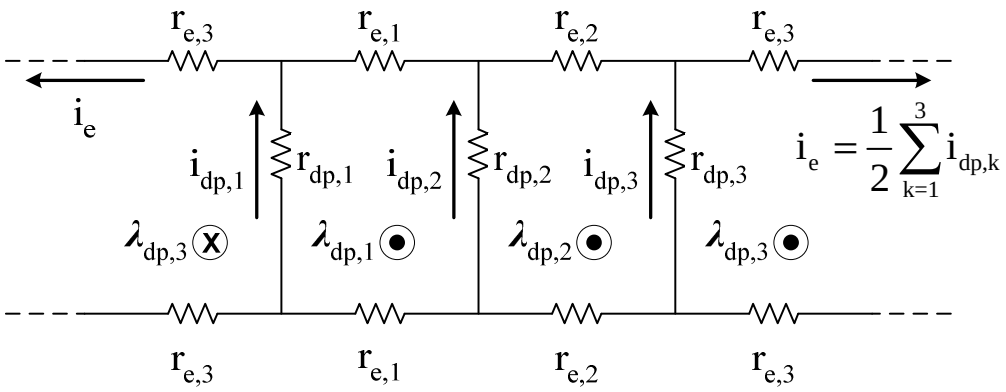


Figure 4.11: Damper winding circuit.

To help describe the state equations for the rotor, the electrical connection circuit shown in Figure 4.11 is used. Within the circuit, $r_{dp,k}$ is the resistance of each damper bar, and $r_{e,k}$ is the resistance of the connection end between bars. From Ohm's and Faraday's laws, the damper winding currents are related to the flux linkage crossing each of two bars as follow,

$$p\boldsymbol{\lambda}_{\text{dp}} = \mathbf{T}_{\text{dp}}\mathbf{i}_{\text{dp}} \tag{4.37}$$

where $p = d/dt$ is the Heaviside operator for differentiation, and $\mathbf{T}_{\text{dp}}$ can be express as,

$$\mathbf{T}_{\text{dp}} = \begin{bmatrix} r_{dp,1} + r_{e,1} & -r_{dp,2} - r_{e,1} & -r_{e,1} \\ r_{e,2} & r_{dp,2} + r_{e,2} & -r_{dp,3} - r_{e,2} \\ r_{dp,1} + r_{e,3} & r_{e,3} & r_{dp,3} + r_{e,3} \end{bmatrix} \tag{4.38}$$

where $r_{dp,k}$ is the resistance of each damper bar, and $r_{e,k}$ is the resistance of the connection end between bars.

As for the stator electrical system, the stator winding voltage equations can be expressed in the arbitrary reference frame as,

$$p\boldsymbol{\lambda}_{\text{qd0s}} = \mathbf{v}_{\text{qd0s}} - r_s\mathbf{i}_{\text{qd0s}} - \omega\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\boldsymbol{\lambda}_{\text{qd0s}} \tag{4.39}$$

where the stator voltage $\mathbf{v}_{\text{qd0s}}$ can be either a user-defined input or calculated by an external circuit model Numerical integration is used to solve (4.37) and (4.39) for the

damper and stator winding flux linkages, given the stator winding voltages, stator winding currents, and damper bar currents.

Based upon the calculations of electromagnetic torque and power losses presented in Section 2.3.3, the calculation of resistive loss is updated to incorporate the damper current loss with an expression as,

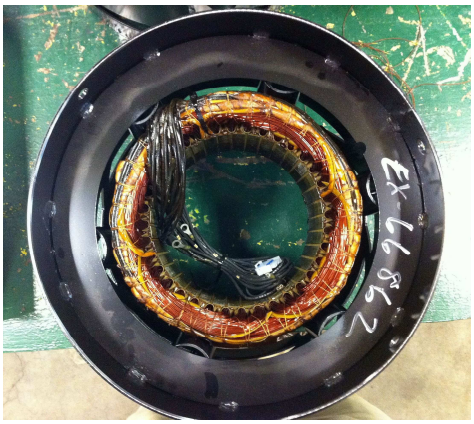
$$P_{res} = \underbrace{\frac{3}{2\pi}\int_0^{2\pi} r_s\, i_{as}^2(\theta_r)\, d\theta_r + r_{fd}\, i_{fd}^2}_{stator+field} + \underbrace{\frac{P}{2\pi}\sum_{k=1}^{nd}\int_0^{2\pi}\left[r_{dp,k}\, i_{dp,k}^2(\theta_r) + r_{e,k}\, i_{e,k}^2(\theta_r)\right] d\theta_r}_{damper} \tag{4.40}$$

where $r_{dp,k}$ and $i_{dp,k}$ are the resistance and current of each damper bar, and $r_{e,k}$ and $i_{e,k}$ are the resistance and current of the connection end between bars.

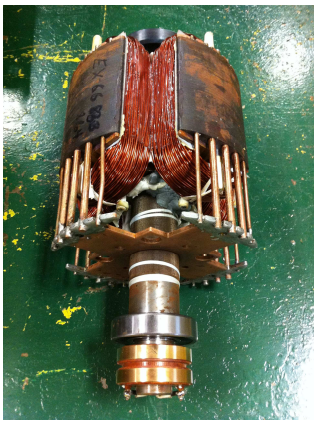
4.4 Validation of Dynamic MEC Model

4.4.1 Hardware environment

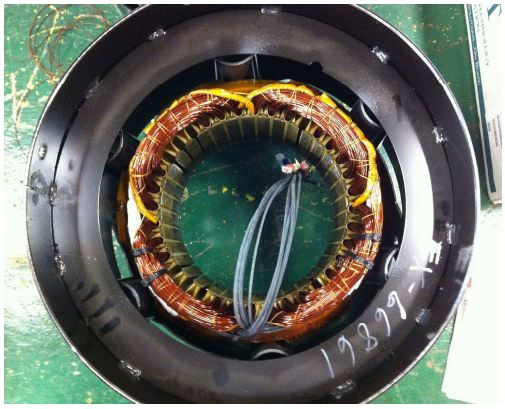
Two stator and rotor geometries were created by Kohler Power System Co. for the validation of the dynamic MEC model. The two stators are identical with the exception that one is wound for single-phase and three-phase generation, respectively. The two rotors are identical with the exception that one has rotor poles (and damper bars) that are straight as one proceeds from the front to the back of the machine. The other has rotor poles and damper bars that are skewed. Skewing is a common method to reduce harmonics introduced by non-ideal magnetic fields. Using the two stators and rotors, four WRSMs can be assembled for test. Figure 4.12 shows the three-phase and single-phase stator, and the straight and skewed rotor. In the following sections, a 3-phase 10 kW WRSM with a straight rotor that is designed to operate at 1800 rpm was built for hardware validation.



(a) Three-phase stator



(b) Straight rotor



(c) Single-phase stator

(d) Skewed rotor

A view of the cross-section of the stator and rotor laminations of the MEC model and hardware is shown in Figure 4.13. The geometry of the stator and rotor laminations, as well as the measured values of stator and field resistances are listed in Table 4.1. In the rotor geometry, there are 5 damper slots with unequal radii filled with copper. Dimensions and resistances of the damper bars and end connections are shown in Table 4.2. It is noted that the temperature of the stator and rotor are measured by wireless temperature sensor so that the resistance values can be calculated at loaded condition.

The *BH* curve of the steel material used in laminations is characterized using the fit equations developed in [61] and expressed as,

$$\mu_B(B) = \mu_0 \frac{f(B)}{f(B)-1} \tag{4.41}$$

$$f(B) = \frac{B}{M} = \frac{\mu_r}{\mu_r - 1} + \sum_{k=1}^{K} \left[\alpha_k |B| + \delta_k \ln(\varepsilon_k + \zeta_k e^{-\beta_k |B|}) \right] \tag{4.42}$$

$$\delta_k = \alpha_k / \beta_k \quad , \quad \varepsilon_k = \zeta_k e^{-\beta_k \gamma_k} = \frac{e^{-\beta_k \gamma_k}}{1 + e^{-\beta_k \gamma_k}} \tag{4.43}$$

where μ_r, α_k, β_k, and γ_k are the parameters with values listed in Table 4.3, and M is the magnetization. The parameters for core loss estimation using MSE is shown in Table 4.4.

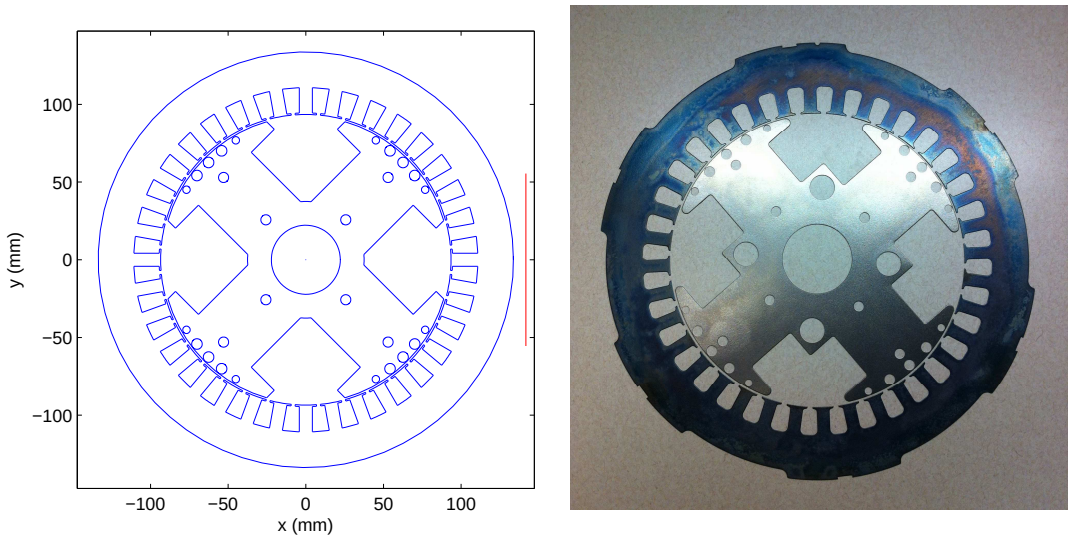


Figure 4.13:Comparison of design cross-section to the stator and rotor laminations.

Table 4.1
Wound-rotor synchronous machine parameters.

r_{sh}: 44.4 mm	d_{rc}: 15.3 mm	g: 1.21 mm	l: 11.1 cm
d_{stt}: 1.02 mm	d_{st}: 17.3 mm	w_{stt}:13.8 mm	w_{st}: 8.4 mm
w_{ss}: 2.5 mm	d_b: 22.97 mm	h_{rto}: 52.1 mm	h_{rtt}: 6.9 mm
w_{rp}: 4.81cm	w_{rt}: 8.99 cm	d_{rp}: 54.76 mm	r_{ro}:18.45 cm
Nph: 3	Pp: 2	α_{dp}: 0.08	
Number of stator teeth: 36		Field winding turns per pole: 214	
Stator turns (N_s): 14		Stator winding connection: series	
a-phase winding distribution: $[N_s \quad N_s \quad N_s \quad N_s \quad N_s \quad N_s \quad 0 \quad 0 \quad 0]$			
Stator resistance r_s: 0.748 ohm (25 °C) / 0.852 ohm (58.5 °C)			
Field resistance r_{fd}: 3.046 ohm (25 °C) / 3.627 ohm (72.2 °C)			

Table 4.2
Damper bar dimension and resistance.

Number of rotor tip dampers: 5	Number of rotor shank dampers: 2
Radius of damper bars on rotor tip (r_{dt}): r_{dt1} - 3.4mm , r_{dt2} - 2.4mm	
Radius of damper bars on rotor shank (r_{ds}): 3.3mm	
Damper winding vector: $[0 \quad 0 \quad r_{dt2} \quad 0 \quad r_{dt1} \quad r_{dt1} \quad r_{dt1} \quad 0 \quad r_{dt2} \quad 0 \quad 0]$	
Damper bar body resistance (r_{dp}): $[0.184 \quad 0.091 \quad 0.091 \quad 0.091 \quad 0.184]$ mohm (25 °C) / $[0.219 \quad 0.108 \quad 0.108 \quad 0.108 \quad 0.219]$ mohm (72.2 °C)	
Damper bar end connection resistance (r_e): $[0.133 \quad 0.100 \quad 0.100 \quad 0.133 \quad 0.871]$ mohm (25 °C) / $[0.158 \quad 0.119 \quad 0.119 \quad 0.158 \quad 1.037]$ mohm (72.2 °C)	

Table 4.3
Parameters for calculating permeability for 50WW800.

μ_r = 5349.922 (initial relative permeability), K = 4
α = [0.12542 0.00019835 0.00019835 0.00019835] 1/T
β = [13.14573 0.1971988 129.4606 8.358885] 1/T
γ = [1.6445 0.01 1.4157 0.58577] T

Table 4.4
Parameters for core loss estimation using MSE for 50WW800.

α	1.0529	β	1.5969
k_e	8.2813e-5	k_h	0.3314

4.4.2 Open circuit voltage

For validation a series of experiments was performed. In the first, the machine was operated under open-circuit conditions at rated speed. The instantaneous and RMS value of the line-line voltage was then obtained for a range of field currents from 0 to 10.2 A. The RMS values of line-to-line voltage are compared in Figure 4.14. The largest difference between the predicted and measured values is approximately 5.0%. Plots of the line-line voltage for three of the field currents are shown in Figure 4.15. Therein it can be seen that there are significant slot harmonics in both measured and MEC waveforms. This is due to the fact that neither the stator slots nor the rotor poles are skewed.

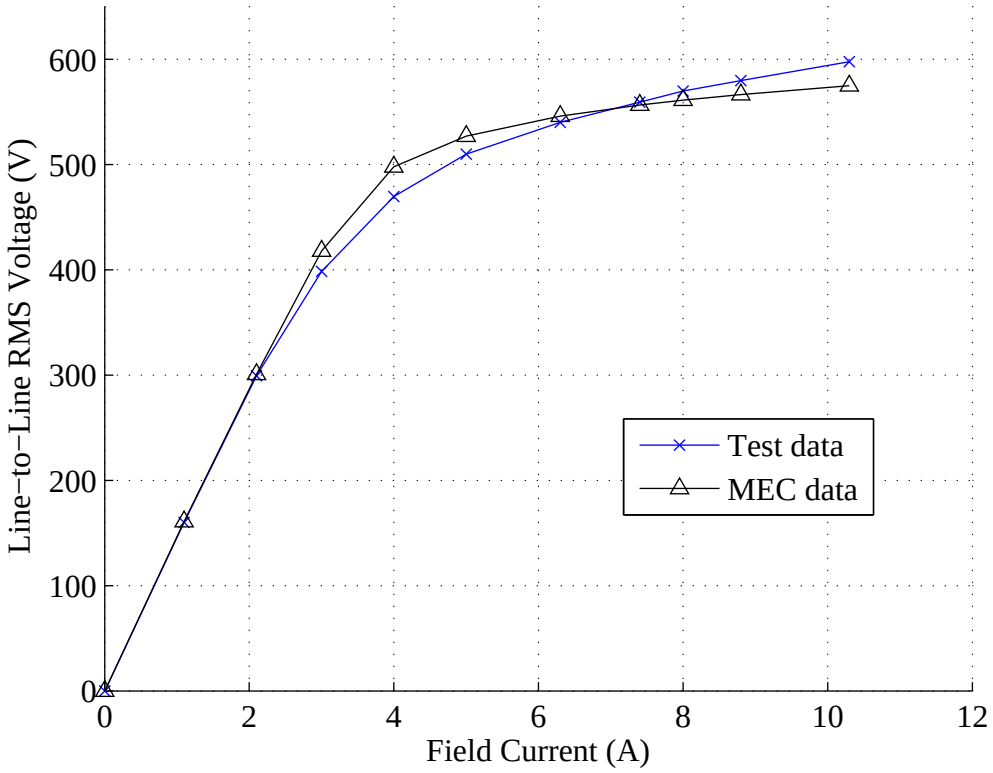


Figure 4.14: Comparison of RMS values of open circuit line-to-line voltage.

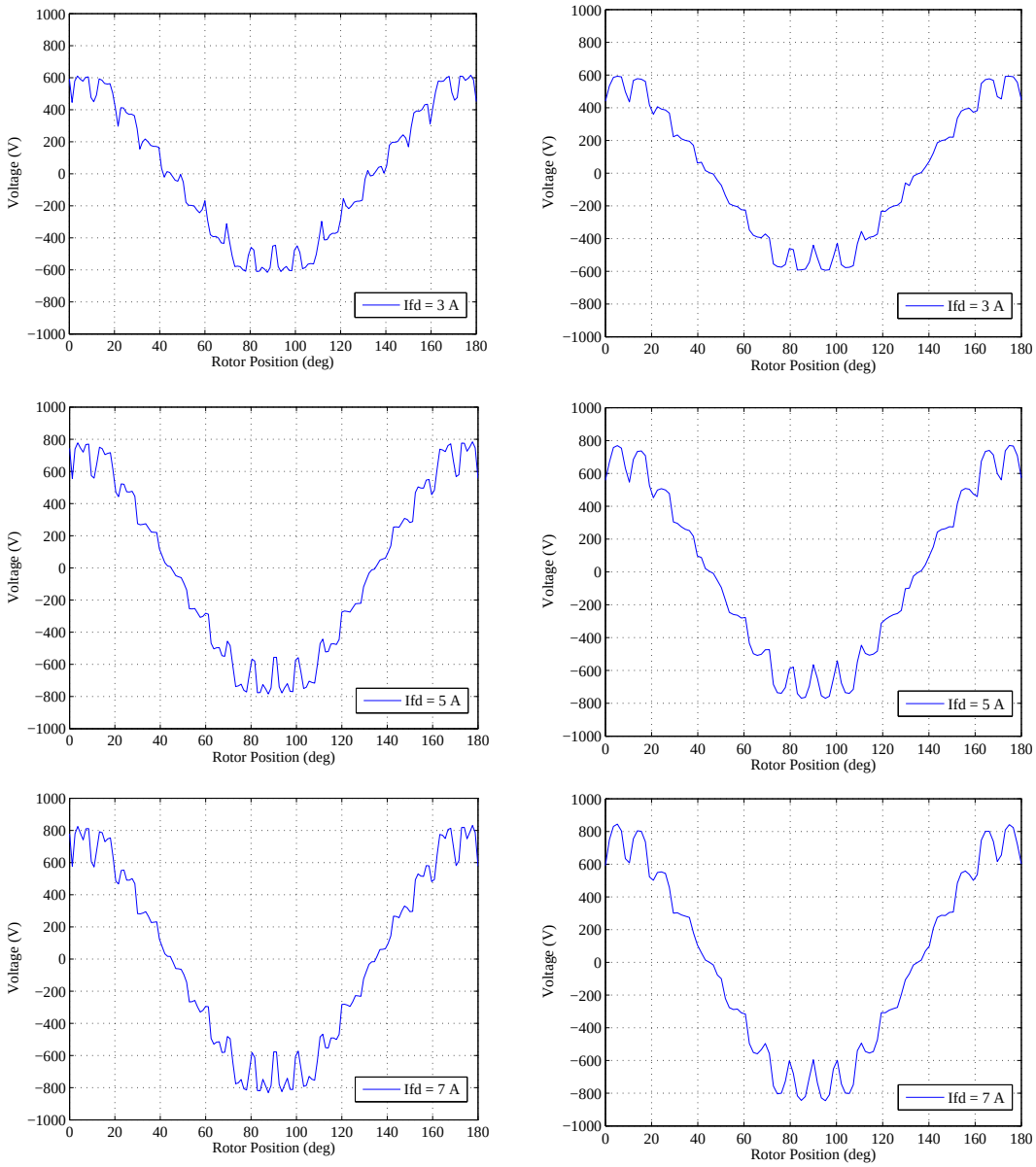


Figure 4.15: Comparison of MEC (left) and hardware (right) open circuit line-to-line voltage waveforms.

4.4.3 Excitation scheme generation

In practice, commercial alternators typically operate a at fixed power factor and line-to-line voltage. Field excitation is adjusted to change power level. The proposed MEC model provides the ability to determine the stator and field excitations for a given output power, power factor, and line-to-line voltage. Values for the MEC under

alternative loading were obtained as part of an optimization in which the objective is expressed as,

$$\text{Minimize} \qquad (\frac{V_{\_error}}{V^*})^2 + (\frac{P_{\_error}}{P^*})^2 + (\frac{pf_{\_error}}{pf^*})^2 \tag{4.44}$$

where V^*, P^*, and pf^* are commanded values for line-to-line voltage, output power, and power factor, respectively. $V_{\_error}$, $P_{\_error}$, and $pf_{\_error}$ are the difference between calculated and commanded values. The output power, reactive power and power factor in the MEC model are calculated using,

$$P_{out} = \frac{3}{2}(v_{qs}^r i_{qs}^r + v_{ds}^r i_{ds}^r) \tag{4.45}$$

$$Q_{out} = \frac{3}{2}(v_{qs}^r i_{ds}^r - v_{ds}^r i_{qs}^r) \tag{4.46}$$

$$pf = sign(Q_{out}) \frac{P_{out}}{\sqrt{P_{out}^{\ 2} + Q_{out}^{\ 2}}} \tag{4.47}$$

where $sign(Q_{out})$ is the sign of reactive power. A negative value represents leading power factor and vice versa. The MEC model is run under steady-state mode, with RMS value of stator current, stator current phase angle and field current set as genes. Using a population of 100 and a generation of 25, the optimization for each operation point takes about 3 minutes. The rotor speed was set 1800 rpm, the power factor to -0.8, and the RMS value of line-to-line voltage to 480 V. The results obtained by MEC model is compared with those from measurement in Table 4.5.

From Table 4.5, the RMS value of stator phase current has a very strong relation between the MEC model and measured values. The error in the field current increases as the load increases, although is remains at a reasonable level. One reason to explain this could be the material is not precisely characterized, particularly in saturation as shown in the open circuit test in Figure 4.14. Another reason could be that the damper winding currents are deactivated in the steady state analysis, which in fact changes the rotor equivalent circuit.

Table 4.5
Stator and field excitation estimations.

Output Power (kW)	Field current			RMS values of stator phase current		
	Measured (A)	MEC (A)	Error (%)	Measured (A)	MEC (A)	Error (%)
10.103	11.6	10.69	7.84	15.2	15.19	0.07
8.6446	10.4	9.67	7.02	13.0	13.0	0.00
6.5613	8.7	8.28	4.83	9.8	9.87	0.71
4.7491	7.5	7.13	4.93	7.1	7.14	0.56
2.2018	5.7	5.63	1.23	3.3	3.31	0.30

4.4.4 Balanced three-phase load test

In a third validation, the WRSM was operated at rated speed and stator windings connected to Y-connected balanced three-phase parallel RL (resistance and inductance) loads that have 0.8 lagging power factor. For each load, the RMS value of the line-to-line voltage was regulated to 480 V by adjusting the field current applied. The measured values of RMS stator currents, average input torque, and output power were measured and are compared to those predicted by the MEC Model in Table 4.6-Table 4.8, respectively.

It is important to note that within the MEC model the electromagnetic torque is calculated, not the input torque. In addition, the core loss is not within the dynamic model, but rather it is obtained as part of post-processing calculations. Thus, input torque from the MEC model was estimated using,

$$T_{in\_avg} = \frac{T_e \omega_{rm} + P_{mech} + P_{core}}{\omega_{rm}} \tag{4.48}$$

where the electromagnetic torque T_e is defined as positive in generation mode here, P_{core} is the core loss, ω_{rm} is the mechanical rotor speed, P_{mech} = 303W is the rotational loss that was measured experimentally at no load conditions. The output power is calculated using,

$$P_{out} = T_e \omega_{rm} - P_{res} \tag{4.49}$$

where the calculation of P_{res} is shown in (4.40). In practice, the brushes attached to the rotor slip ring increase the field resistance by 1 Ω.

Table 4.6
Comparison of RMS values of phase current.

Field Current (A)	Load	RMS values of phase current		
		Measured (A)	MEC (A)	Error (%)
6.2	77.16 (Ω) 0.2729 (H)	4.5	4.8	6.25
7.6	45.44 (Ω) 0.1607 (H)	7.6	8.0	5.00
9.6	30.35 (Ω) 0.1073 (H)	11.4	12.0	5.00
11.6	22.81 (Ω) 0.0807 (H)	15.2	15.9	4.40

Table 4.7
Comparison of average input torque.

Field Current (A)	Load	Average input torque		
		Measured (Nm)	MEC (Nm)	Error (%)
6.2	77.16 (Ω) 0.2729 (H)	19.98	21.01	4.90
7.6	45.44 (Ω) 0.1607 (H)	32.26	33.81	4.58
9.6	30.35 (Ω) 0.1073 (H)	47.78	49.95	4.34
11.6	22.81 (Ω) 0.0807 (H)	64.16	66.11	2.95

Table 4.8
Comparison of output power.

Field Current (A)	Load	Output power		
		Measured (kW)	MEC (kW)	Error (%)
6.2	77.16 (Ω) 0.2729 (H)	3.1775	2.9858	6.03
7.6	45.44 (Ω) 0.1607 (H)	5.3641	5.0707	5.47
9.6	30.35 (Ω) 0.1073 (H)	7.9783	7.5915	4.85
11.6	22.81 (Ω) 0.0807 (H)	10.4445	10.1030	3.27

Table 4.9
Comparison of power loss.

Load	MEC				Test	
	P_{s+f} (W)	P_{core} (W)	P_{dp} (W)	$P_{core+dp}$ (W)	P_{s+f} (W)	$P_{core+dp}$ (W)
1	235.5	232.3	12.5	244.7	229.5	247.9
2	431.2	241.8	32.9	274.7	414.5	292.8
3	793.2	254.2	86.4	340.6	757.6	354.4
4	1267.4	265.3	181.9	447.2	1211.2	477.0

From the results in Table 4.6-Table 4.8, there is a strong correlation between the model and hardware results. The error is approximately 6% at low load, and 3% at full load. A study of the power loss components is shown in Table 4.9, in which P_{s+f} is the resistive loss in the stator and field windings, P_{core} is the core loss, and P_{dp} is the damper loss. The difference between the measured and predicted P_{s+f} values causes by the difference of stator currents. The measured $P_{core+dp}$ is calculated by subtracting P_{s+f} and P_{mech} from the total power loss. One might see that the predicted $P_{core+dp}$ values are slightly lower than the measured values. This is due to the fact that in practice the field winding is sourced by the stator winding through an exciter, which is not modeled in the MEC.

In addition, The line-to-line voltage waveforms at rated output power (10 kW) are compared between MEC and hardware in Figure 4.16. The error of RMS values of phase current and voltage is approximately 5%.

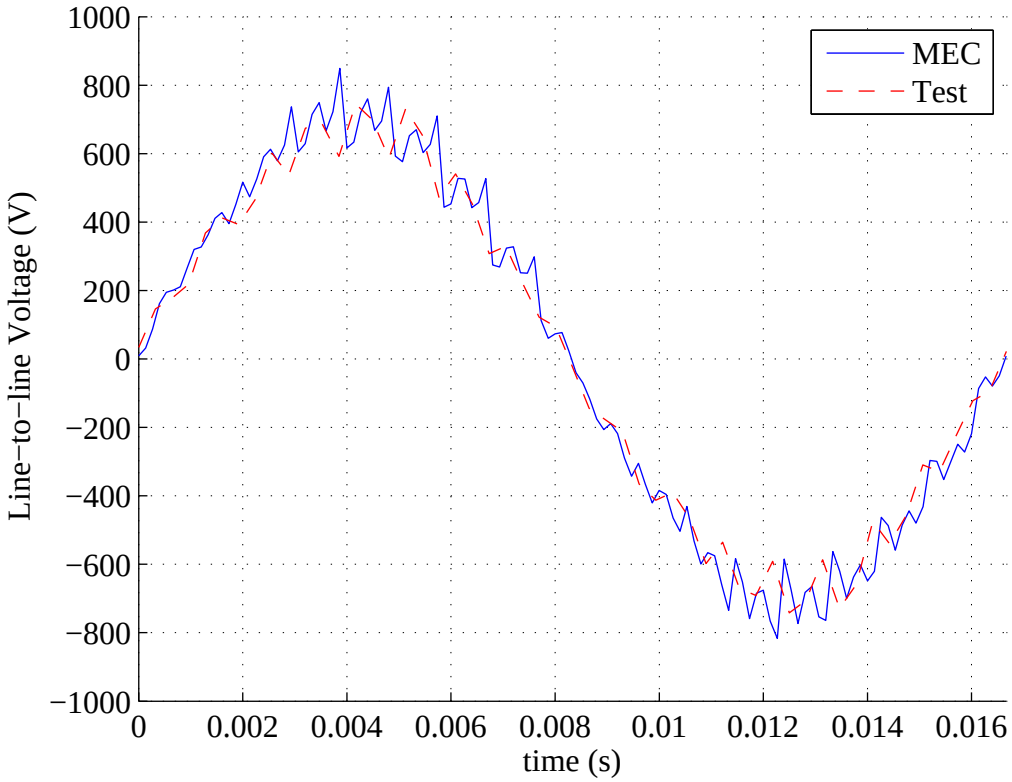


Figure 4.16: Comparison of line-to-line voltage waveforms at rated power (10 kW).

4.4.5 Stand still frequency response

As a final experiment, a standstill frequency response test [62] was applied to obtain *qd*-axis operational impedances. This test was motivated by the fact that in many cases, the subtransient inductances are used in design specifications. In addition, the switching behavior of the diodes in machine-rectifier systems is a function of the subtransient inductances [24], [25]. The SSFR circuit configuration and test procedure have been described in details in IEEE Std. 115. An SSFR similar to the standard has been executed to date. The circuit configuration used for the test is shown in Figure 4.17, where *b* and *c* phase stator windings are parallel-connected, and the field winding is short circuited. A function generator was connected to a power amplifier which was used to provide ac voltage in a range of frequencies from 0.1 to 1 k Hz.

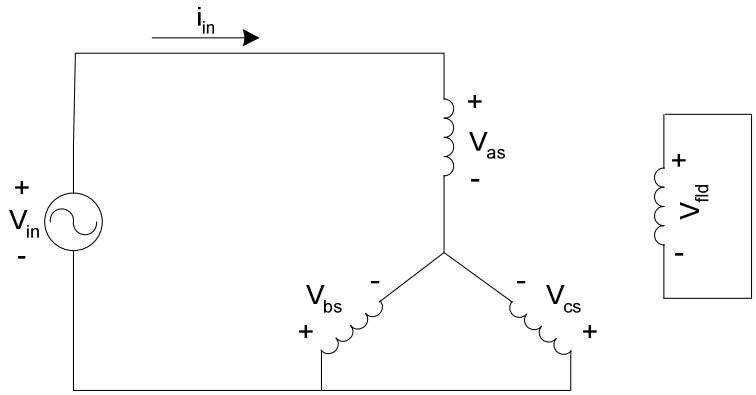


Figure 4.17: Measurement of *q*- and *d*- axis operational impedance.

The test procedure for *d*-axis operational reactance measurement can be divided into two steps. First, the rotor is positioned at $\theta_r = 90°$. The rotor angle was determined by setting the source voltage to a frequency of 100Hz, and rotating the rotor until the induced field voltage becomes a maximum value. At this point, the magnetic axis of field winding is aligned and ready to be used for *d*-axis test. Second, after applying a variable-frequency source voltage, v_{in} and i_{in} signals are measured so that *d*-axis impedance can be calculated as

$$Z_d(s) = \frac{2}{3}\frac{v_{in}(s)}{i_{in}(s)} \tag{4.50}$$

And the *d*-axis operational reactance can be calculated using

$$X_d(s) = \frac{\omega_b(Z_d(s) - r_s)}{s} \tag{4.51}$$

where ω_b is the base radian frequency, and $s = j\omega$. Finally, the rotor is tuned at a position such that the induced field voltage achieves its null and a *q*-axis impedance measured.

Prior to describing the results, it is noted that in the physical construction, connections between damper end bars is made through copper plates that are connected to each end of the rotor. In constructing the machine with these plates, an additional conductive path is created through the rotor shaft, which was not modeled.

The magnitude of the operational impedance between hardware and the MEC is shown in Figure 4.18. The high-frequency asymptote of the operational impedances

corresponds to subtransient impedances. The low-frequency asymptotes correspond to magnetizing impedances. Comparing results from the MEC model with measurement, the *d*-axis data matches very well. However, a discrepancy does exist in the *q*-axis. At low frequencies (below 0.4 Hz), the measured and predicted values match closely.

At mid frequencies (between 0.4 Hz and 20 Hz), the experimental data begins to deviate. This is attributed to the additional conduction path that exists between the copper plates and the rotor shaft. These components provide a path for *q*-axis current which is not modeled in the MEC. To confirm this conjecture, a 2D FEA model was created and used to obtain *q*-axis (and *d*-axis) operational impedances. Within the FEA model, eddy currents were not represented, which is consistent with the MEC model. Comparing the FEA and MEC curves, the match is very strong through the mid frequency range.

At higher frequencies (above 100 Hz), a slight difference exists between the FEA and MEC impedances. This is likely caused by some error in modeling the rotor pole tip leakage flux paths, since the operational impedance is dominated by leakage impedance at high frequency. It is also noted that at the measured *q*-axis impedance drops more significantly than both the FEA and MEC-based curves. This is mainly attributed to the eddy currents in the shaft/copper plates. Other factors such as the variation of resistance due to skin effect could also lead to some difference among the three traces.

As shown in the Section 4.1.3, the reluctance of the rotor pole tip leakage is a function of the depth of the damper bars. Therefore, in order to study the influence that damper bar placement has on the operational impedances, two additional machines were modeled in MEC and FEA. In these two machines, the geometries of the hardware-based machine were used. However, the depth of bars was adjusted by modifying the scaling factor α_{dp}. In the first case, the bars were positioned relatively deep into the rotor tips by setting α_{dp}=0.5, which provides for a leakage flux paths with relatively small reluctance. In the second case, the bars were placed at the top of the rotor tips very close to the airgap by setting α_{dp}=0.0001, which nearly eliminates the leakage flux path around the damper bars. The frequency responses obtained are provided in Figure 4.19 and Figure 4.20. In Figure 4.19, one observes that the MEC and FEA models match very well. One can note that under this design, the q-axis impedance is nearly constant. This is attributed to the

fact that the damper leakage inductances are relatively large. In Figure 4.20, there is a small difference in the high frequency asymptotes in the *q*-axis curves. This is attributed to the fact that the damper slot leakage between the poles is not represented within the MEC model, and thus the leakage inductance is under estimated. The study in Figure 4.20 is then repeated with damper bar connections are only made on a single pole in order to eliminate the leakage path between poles. The result is shown in Fig. 15 and indicates a strong match between the two models.

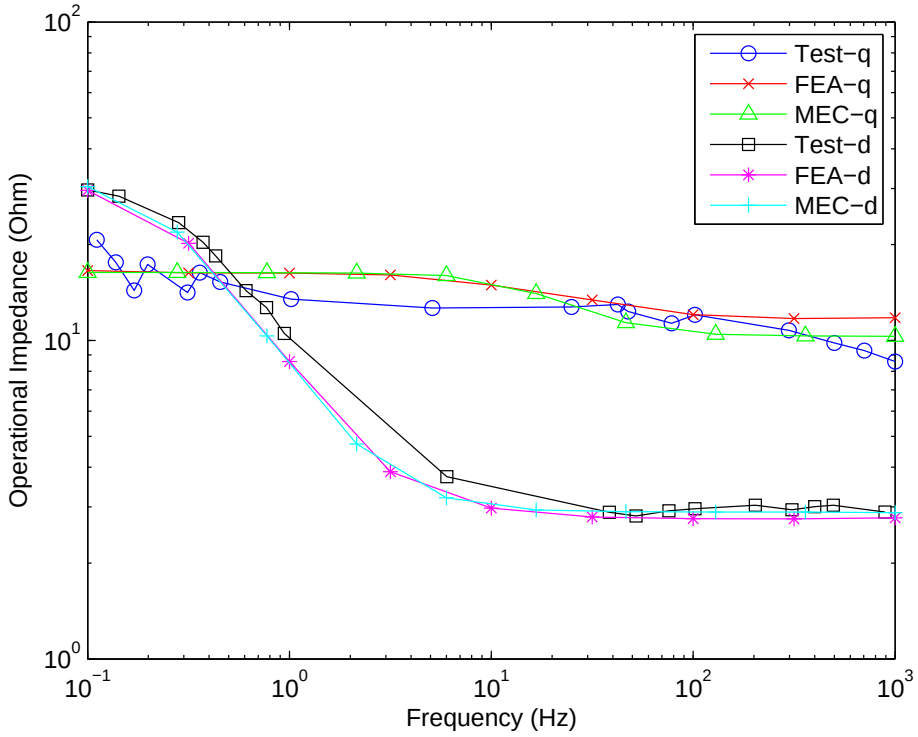


Figure 4.18: Standstill frequency response test with α_{dp}=0.08.

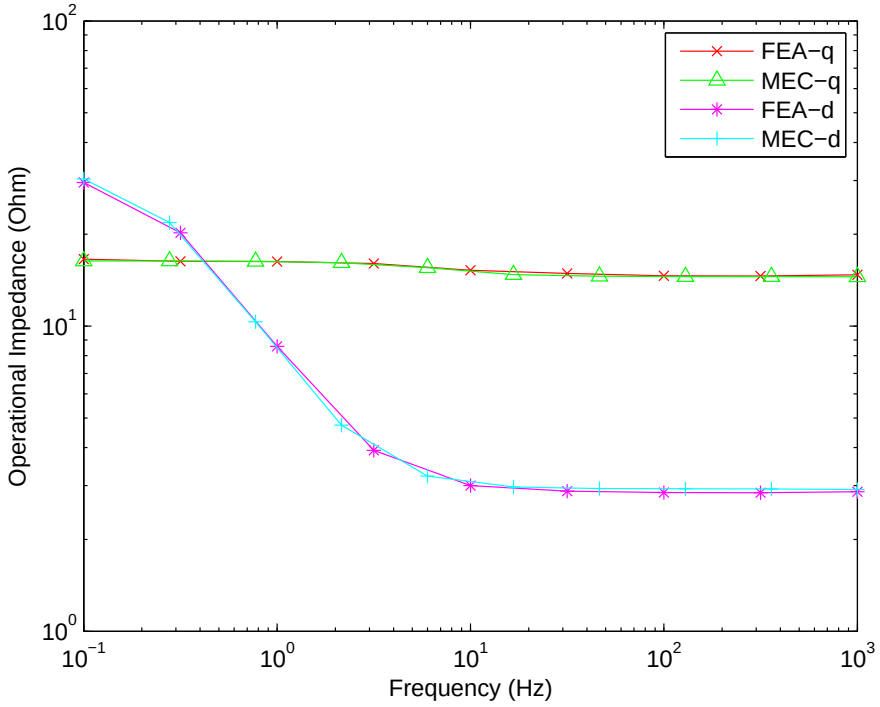


Figure 4.19: Standstill frequency response test with α_{dp}=0.5.

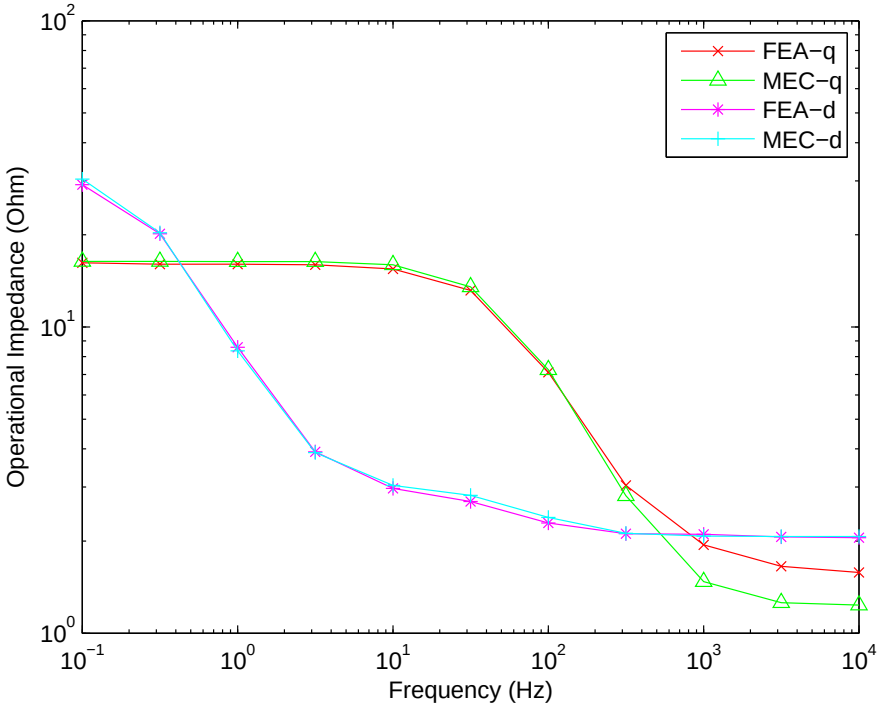


Figure 4.20: Standstill frequency response test with α_{dp}=0.0001.

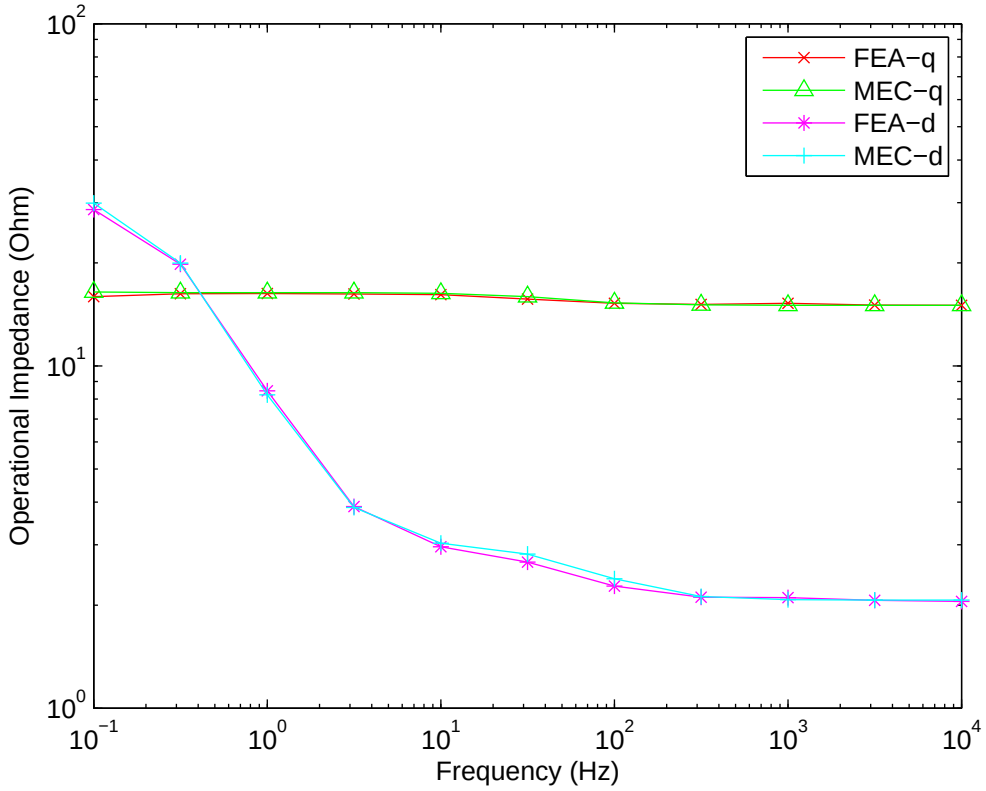


Figure 4.21: Standstill frequency response test with α_{dp}=0.0001, with damper bar connections are only made on a single pole.

It is also important to note that the precision in capturing leakage flux behavior and calculating the flux crossing the damper bar paths for the MEC and FEA is different. Specifically, within the FEA model, the flux densities are vector quantities and thus the normal component of the flux crossing the damper path is readily modeled. However, flux densities in the MEC model are represented as scalars. Therefore, a difference between FEA and MEC results is certainly expected.

5. IMPLEMENTATION OF SKEWING

5.1 Literature Review of Skewing

The electromagnetic torque in synchronous machines include three main components: 1) torque produced by the interaction of poles resulting from the stator currents and the rotor field; 2) reluctance torque, which is generated by the interaction of the poles produced by the stator winding attempting to align with a minimum reluctance path; and 3) cogging torque, which is created by the interaction of poles produced by the rotor field and the attempting to align with stator teeth. Often, to minimize acoustic noise and vibration there is a need to minimize cogging torque.

Stacking the stator teeth or rotor poles with a slight offset down the axial length, which is often referred to as skewing, tends to reduce cogging torque and also eliminate stator-slot-induced harmonics in current and voltage waveforms [63]-[65]. Approaches to model skewing in electric machine models generally fall into one of five categories. Within lumped-parameter models, a conventional approach is to apply analytically-derived skew factors to represent its impact on airgap flux density harmonics, which is then used to calculate skew-based induced voltages and machine parameters [60], [66]. In a second path, an analytical model that describes flux density and airgap permeance with axial variation was proposed in [67], [68]. Within the model, input data from a finite number of magnetostatic FEA solutions is used to predict the flux density that includes slot harmonics and saturation. In a third method which is focused on MEC models, V. Ostovic introduced a ‘3D’ calculation of airgap permeances that is based upon the overlap of a stator and rotor tooth sections with axial variation included [69]. In general, this requires sophisticated logic, and hence is impractical in generalized machine design problems. A fourth approach is to create separate 2D models with appropriate shifts of the rotor relative to the stator teeth [70]. The energy values obtained from each model are averaged and used to calculate electromagnetic torque. Similarly, flux linkage values are

averaged and used to calculate open-circuit voltage. Although straightforward to implement, it has been shown in [71] that using this process leads to inaccuracies in machines that have short-circuited rotor cages. This inaccuracy results from the neglecting of the coupling of the flux linkage and induced cage currents.

The fifth approach, which has been used in FEA [71]-[74], is referred to as a multi-slice method. Within a multi-slice model, the machine is separated into a finite number of cross-sections along axial direction. Within each of the finite sections, a shift is introduced between the stator teeth and rotor poles. Each of the slices is then modeled in two-dimensions with a constraint that the axial currents are the same. Therefore, within the model, flux and currents of the respective slices are not averaged, but are all solved within a unified system matrix. Herein, this approach is extended to both the steady-state and dynamic MEC models.

5.2 Multi-slices MEC Model

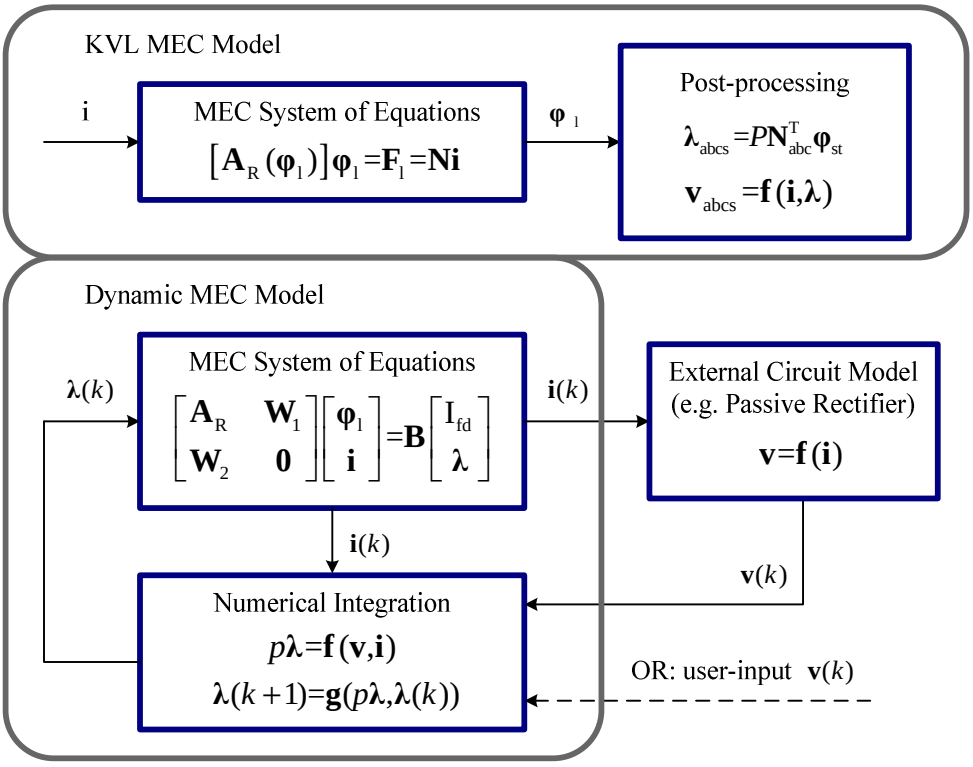


Figure 5.1: Basic structure of the dynamic model shown in contrast with the KVL model.

To establish the multi-slice method, the block diagram of the steady state and dynamic MEC models shown in Figure 4.10 is useful and is included here as Figure 5.1 for convenience. Applying the multi-slice technique to the steady state MEC model is relatively straightforward due to the fact that currents are set as inputs to the model so that the constraint that each of the slices shares the same axial currents is automatically satisfied. The steady state MEC system equations are shown in Figure 5.1 as,

$$\mathbf{A}_{\mathrm{R}}\boldsymbol{\varphi}_{\mathrm{l}}=\mathbf{N}\mathbf{i} \tag{5.1}$$

Here we consider n slices of equal axial length. The step angle, which is used to provide the offset between the stator teeth and pole sections within each slice, can be expressed as,

$$\alpha=\frac{\theta_{skew}}{(n-1)} \tag{5.2}$$

where θ_{skew} is the complete skew angle down the axial length of the machine. If the rotor position for the first slice is θ_1, then the *k*th slice has a rotor position that is expressed as,

$$\theta_k=\theta_1+\alpha(k-1),\quad k=1,...,n \tag{5.3}$$

Applying θ_k to the algorithm to determine the reluctance and turns matrices, the system equations (5.1) for the *k*th slice can be written as,

$$\mathbf{A}_{\mathrm{R,k}}\boldsymbol{\varphi}_{\mathrm{l,k}}=\mathbf{N}_{\mathrm{k}}\mathbf{i} \tag{5.4}$$

By combining and manipulating all of the slice models, the overall multi-slice system equations can be expressed in matrix form as,

$$\begin{bmatrix}\mathbf{A}_{\mathrm{R},1}^{(nl\times nl)} & & \mathbf{0}\\ & \ddots & \\ \mathbf{0} & & \mathbf{A}_{\mathrm{R,n}}^{(nl\times nl)}\end{bmatrix}\begin{bmatrix}\boldsymbol{\varphi}_{\mathrm{l},1}^{(nl\times 1)}\\ \vdots\\ \boldsymbol{\varphi}_{\mathrm{l,n}}^{(nl\times 1)}\end{bmatrix}=\begin{bmatrix}\mathbf{N}_1^{(nl\times 4)}\\ \vdots\\ \mathbf{N}_{\mathrm{n}}^{(nl\times 4)}\end{bmatrix}\mathbf{i}^{(4\times 1)} \tag{5.5}$$

As can be seen from (5.5), the inputs for the multi-slice system equations are stator and field currents (i), and the unknowns are the loop fluxes ($\boldsymbol{\varphi}_{\mathrm{l}}$) for each individual slice.

A Newton-Raphson method is used to solve the multi-slice system of equations. The derivation of the Jacobian matrix for the single slice dynamic system has been presented in [1]. The same technique is applied to the system equation in (5.4), and thus the Jacobian matrix for the *k*th slices is expressed as,

$$\mathbf{J}_{\mathrm{k}} = \mathbf{A}_{\mathrm{R,k}} + \mathbf{D}_{\mathrm{R,k}} \tag{5.6}$$

By combining and manipulating all of the slice Jacobian matrices, the overall multi-slice Jacobian matrix for (5.5) can be expressed as,

$$\mathbf{J}_{\mathrm{tot}} = \mathbf{A}_{\mathrm{R,tot}} + \begin{bmatrix} \mathbf{D}_{\mathrm{R,1}} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{D}_{\mathrm{R,n}} \end{bmatrix} \tag{5.7}$$

where $\mathbf{A}_{\mathrm{R,tot}}$ is the system matrix in (5.5). The calculation of electromagnetic torque can be expressed as,

$$T_{\mathrm{e}}(\phi, \theta_{\mathrm{r}}) = \sum_{k=1}^{n} \left\{ \left(\frac{P}{2} \right)^2 \sum_{j=1}^{na} \left(\frac{\phi_{\mathrm{agj,k}}}{P_{\mathrm{agj,k}}} \right)^2 \frac{\partial P_{\mathrm{agj,k}}}{\partial \theta_{\mathrm{r}}} \right\} \tag{5.8}$$

In the steady-state MEC model, stator phase voltage is calculated as a post-process of the flux linkage, which is calculated as the product of phase winding function and flux. When the multi-slice model is applied, the phase flux in the calculation is substituted by the sum of each separate slice.

As for the dynamic MEC model, the currents are no longer the inputs. Therefore, one of the challenges to implement the multi-slice technique to the dynamic MEC model is that the same currents should be solved for each separate slice. The dynamic MEC system equations are shown in Figure 5.1 as,

$$\begin{bmatrix} \mathbf{A}_{\mathrm{R}} & \mathbf{W}_1 \\ \mathbf{W}_2 & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\varphi}_{\mathrm{l}} \\ \mathbf{i} \end{bmatrix} = \begin{bmatrix} \mathbf{N}_{\mathrm{l,fd}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}/P \end{bmatrix} \begin{bmatrix} \mathbf{I}_{\mathrm{fd}} \\ \boldsymbol{\lambda} \end{bmatrix} \tag{5.9}$$

Similar to the steady-state MEC model, applying θ_k to the algorithm to determine the reluctance and turns matrices, the system equations (5.9) for the *k*th slice can be written as,

$$\begin{bmatrix} \mathbf{A}_{\mathrm{R,k}} & \mathbf{W}_{\mathrm{1,k}} \\ \mathbf{W}_{\mathrm{2,k}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\varphi}_{\mathrm{l,k}} \\ \mathbf{i} \end{bmatrix} = \begin{bmatrix} \mathbf{N}_{\mathrm{l,fd}} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}/P \end{bmatrix} \begin{bmatrix} \mathbf{I}_{\mathrm{fd}} \\ \boldsymbol{\lambda}_{\mathrm{k}} \end{bmatrix} \tag{5.10}$$

Since the total flux linkage ($\boldsymbol{\lambda}$) is the sum of the flux linkage for each separate slices ($\boldsymbol{\lambda}_{\mathrm{k}}$), therefore

$$\sum_{k=1}^{n} \mathbf{W}_{\mathrm{2,k}} \boldsymbol{\varphi}_{\mathrm{l,k}} = \mathbf{I}\boldsymbol{\lambda}/P \tag{5.11}$$

By combining and manipulating (5.10) and (5.11), the overall multi-slice system of equations can be expressed in matrix form as,

$$\begin{bmatrix} \mathbf{A}_{\mathrm{R},1}^{(nl\times nl)} & & \mathbf{0} & \mathbf{W}_{1,1}^{(nl\times(2+nd))} \\ & \ddots & & \vdots \\ \mathbf{0} & & \mathbf{A}_{\mathrm{R},1}^{(nl\times nl)} & \mathbf{W}_{1,n}^{(nl\times(2+nd))} \\ \mathbf{W}_{2,1}^{((2+nd)\times nl)} & \cdots & \mathbf{W}_{2,n}^{((2+nd)\times nl)} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\varphi}_{\mathrm{l},1}^{(nl\times 1)} \\ \vdots \\ \boldsymbol{\varphi}_{\mathrm{l},n}^{(nl\times 1)} \\ \mathbf{i}^{((2+nd)\times 1)} \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{N}_{\mathrm{l,fd}}^{(nl\times 1)} & & & \\ & \ddots & & \mathbf{0} \\ & & \mathbf{N}_{\mathrm{l,fd}}^{(nl\times 1)} & \\ & \mathbf{0} & & \mathbf{I}^{((2+nd)\times(2+nd))}/P \end{bmatrix} \begin{bmatrix} \mathbf{I}_{\mathrm{fd}} \\ \vdots \\ \mathbf{I}_{\mathrm{fd}} \\ \boldsymbol{\lambda}^{((2+nd)\times 1)} \end{bmatrix} \tag{5.12}$$

As can be seen from (5.12), the inputs for the multi-slice system of equations are field current ($\mathbf{I}_{\mathrm{fd}}$) for each separate slices and the flux linkage ($\boldsymbol{\lambda}$) of the phases and the damper bars, which can be calculated by the same state equation and numerical integration as shown in Figure 5.1. The unknowns are the loop fluxes ($\boldsymbol{\varphi}_{\mathrm{l,k}}$) for each separate slice and the currents ($\mathbf{i}$) of the phases and the damper bars. The currents ($\mathbf{i}$) in the unknown vector satisfy the constraint that all slices share the same axial currents.

A Newton-Raphson method is used to solve the multi-slice system of equations for the loop fluxes and currents. The derivation of the Jacobian matrix for the single slice dynamic system has been presented in Chapter 4. The same technique can be applied to the system equation in (5.12), and thus the Jacobian matrix for the *k*th slices is expressed as,

$$\mathbf{J}_{\mathrm{dyn,k}} = \mathbf{A}_{\mathrm{dyn,k}} + \begin{bmatrix} \mathbf{D}_{\mathrm{R,k}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{5.13}$$

By combining and manipulating all of the slice Jacobian matrices, the overall multi-slice Jacobian matrix for (5.12) can be expressed as,

$$\mathbf{J}_{\mathrm{dyn,tot}} = \mathbf{A}_{\mathrm{dyn,tot}} + \begin{bmatrix} \mathbf{D}_{\mathrm{R},1} & & \mathbf{0} & \\ & \ddots & & \mathbf{0} \\ \mathbf{0} & & \mathbf{D}_{\mathrm{R},n} & \\ & \mathbf{0} & & \mathbf{0} \end{bmatrix} \tag{5.14}$$

where $\mathbf{A}_{\mathrm{dyn,tot}}$ is the system matrix in (5.12).

5.3 Simulation Results

5.3.1 Open circuit voltage

For validation, the multi-slice MEC model is configured with a slice number of five and a skew degree of one stator slot. The skewed and non-skewed open circuit voltage waveforms are obtained for the machine described in Chapter 4, at a field excitation Ifd = 7 A, as shown in Figure 5.2. The open circuit voltage waveforms for each of the slices in the multi-slices model are shown in Figure 5.3. One can see in Figure 5.3 that the waveforms are shifted evenly by $\frac{\theta_{skew}}{4}$. The harmonics spectrum of the open circuit voltage waveforms in Figure 5.2 is shown in Figure 5.4. One can see that the skewing reduces the $(6k+1)^{th}$ and $(6k-1)^{th}$ harmonics, where k = 1, 2, … . The results match analytical prediction [60] that the skew influence on the *h*th harmonic of open-circuit voltage can be modeled as,

$$k_{skew} = \frac{\sin(\frac{h\theta_{skew}}{2})}{\frac{h\theta_{skew}}{2}} \tag{5.15}$$

The comparison of skew factors calculated based upon (5.15) and Figure 5.3 is shown in Table 5.1. From Table 5.1 one can see that the harmonic components are significantly reduced. The differences in the higher slot-induced harmonics is attributed to saturation, numerical error, and approximations of flux behavior around slots used in both analytical and MEC derivations.

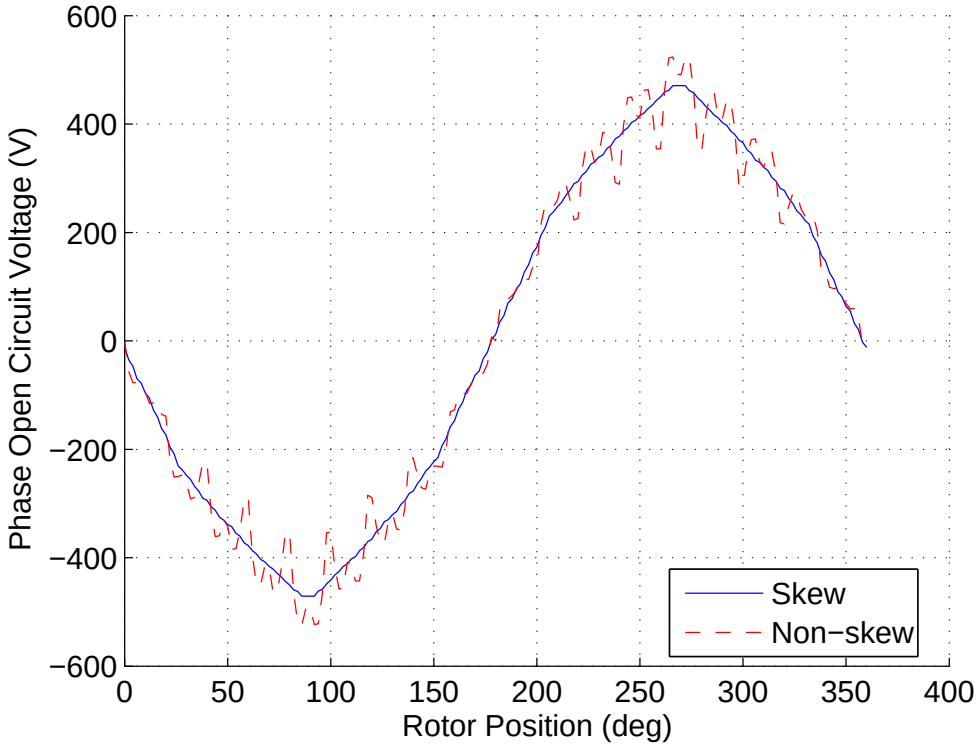


Figure 5.2: Comparison of the skewed and non-skewed open circuit voltage waveforms.

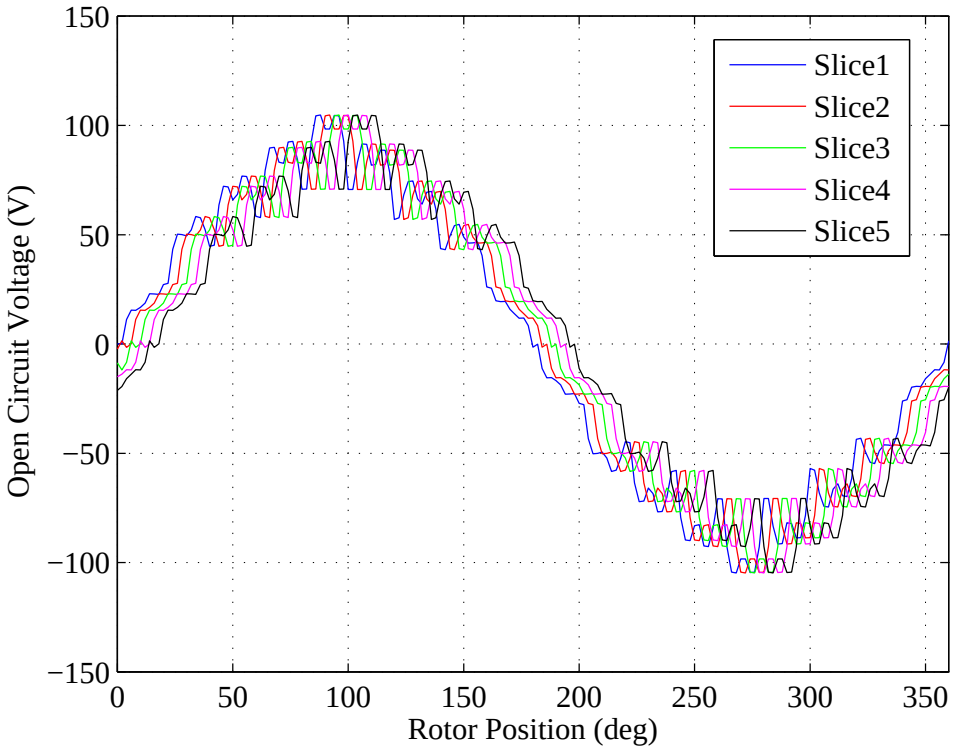


Figure 5.3: Open circuit voltage for each slice.

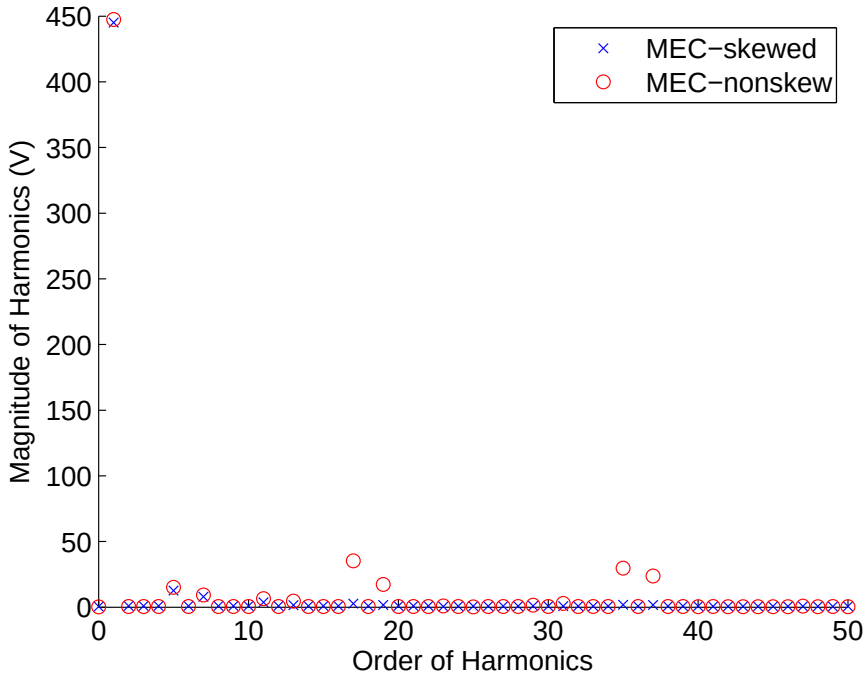


Figure 5.4: Harmonics spectrum of the open circuit voltage waveforms.

Table 5.1
Comparison of skew factors calculated by analytical and MEC models.

Order of Harmonics	Analytical	MEC	Error
1	0.9949	0.9946	0.1%
5	0.8778	0.8393	5.4%
7	0.7691	0.8557	11.3%
11	0.4895	0.5812	18.7%
13	0.3376	0.3481	3.1%
17	0.0585	0.0783	33.8%
19	0.0524	0.0994	89.7%

5.3.2 Balanced three-phase load test

As a second validation, it is assumed that the WRSM is connected to 3-phase balanced resistive load, providing output power of 7 kW. The load resistance is 40 Ω, and the field excitation is set to 7 A. Comparisons of the skewed and non-skewed waveforms, including phase current, phase voltage, and electromagnetic torque, are shown in Figure 5.5-Figure 5.7. As expected, the waveforms predicted by the multi-slice model have much lower harmonic content.

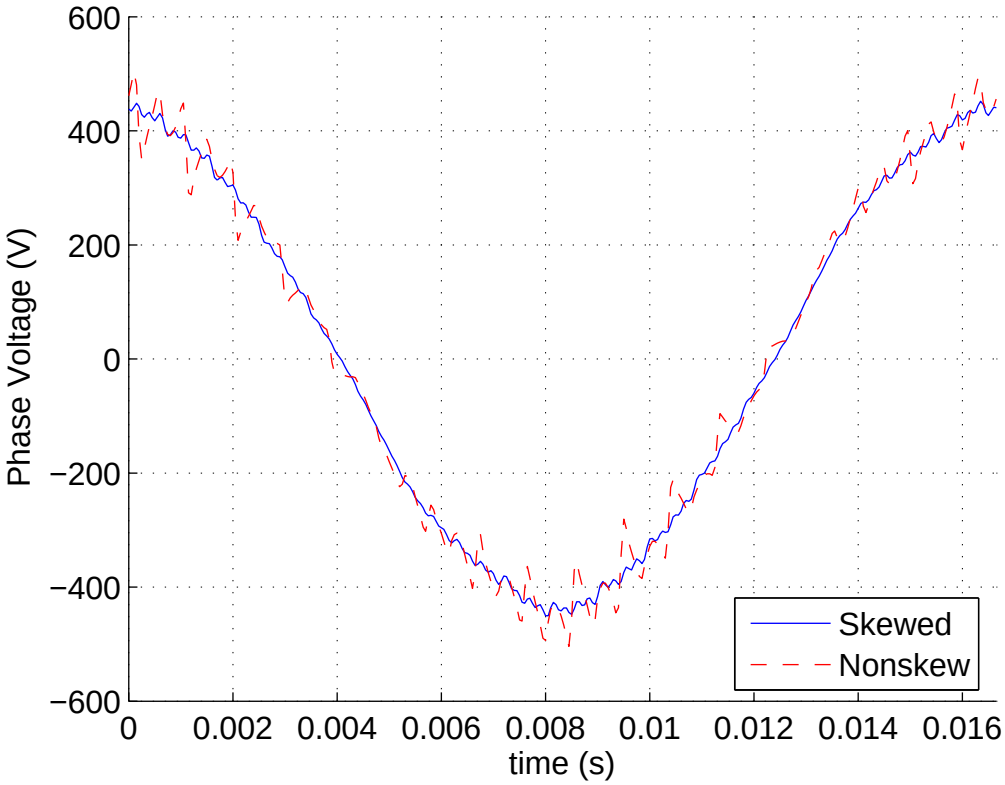


Figure 5.5: Comparison of skewed and non-skewed phase voltage.

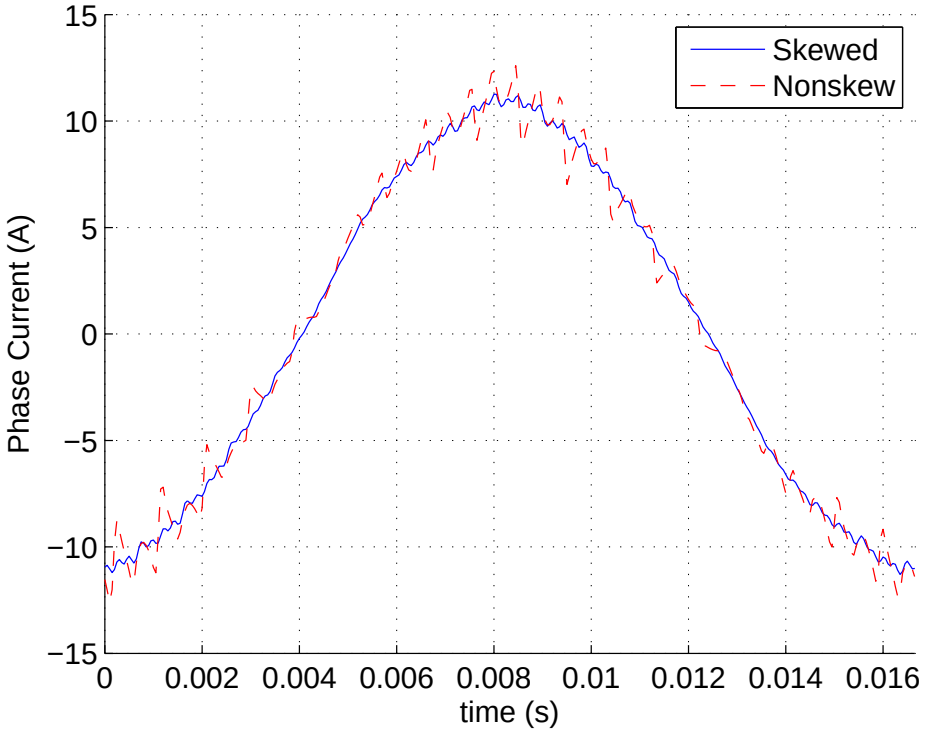


Figure 5.6: Comparison of skewed and non-skewed phase current.

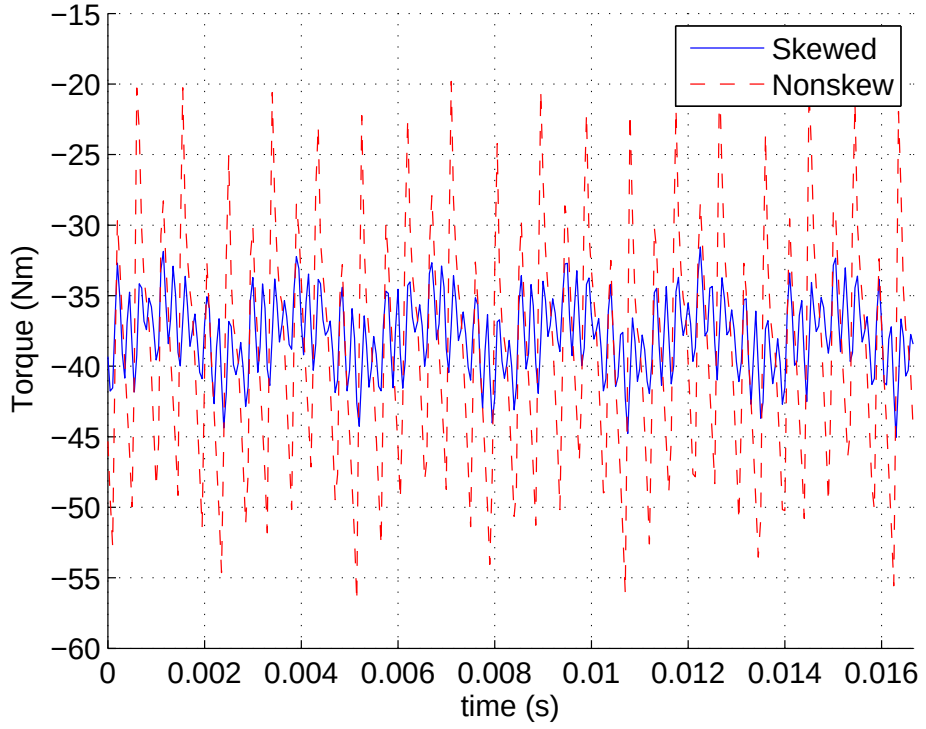


Figure 5.7: Comparison of skewed and non-skewed electromagnetic torque.

6. OPTIMAL DESIGN OF WRSM/RECTIFIER SYSTEMS

6.1 Design Overview

In this chapter, a goal was to apply the model developed in Chapter 4 to demonstrate its use in machine design. Toward this goal, the design of electric machines for a 25 MW, 3600 rpm dc power generation system is considered. As shown in Figure 6.1, the generation system consists of a prime mover, e.g. the turbine of the vessel. The output shaft of the turbine is connected to an electric machine that sources power electronic converters used to supply dc power. Designs were explored for connection of the WRSM to power electronic converters that enable the control of winding current. Such converters are herein referred to as active rectifiers. In addition, designs were explored for connection of the WRSM to diode-based converters, which are herein referred to as passive rectifiers. The passive rectifier designs are also applicable to architectures in which thyristors are used in place of diodes for fault protection.

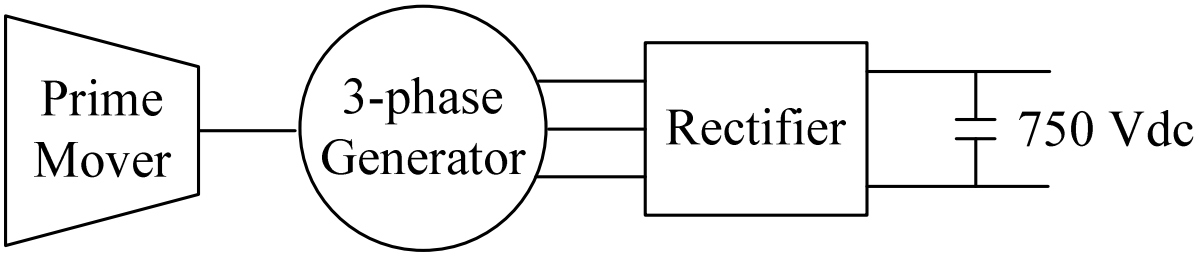


Figure 6.1: 25 MW generation system.

A question of particular interest in formulating the design studies was whether passive rectifiers can be applied in such high power applications. Passive rectifiers have the desirable property that they do not require a rotor position sensor to establish the converter switching and do not require gate-drive circuitry. Thus, they are simpler to control/maintain and have a higher reliability. However, it is generally believed that the

harmonics associated with diode rectifiers lead to electric machines that are too large for practical consideration, particularly at high power levels. To consider whether this is indeed the case, multi-objective optimizations were performed. Within each optimization, the performance metrics were machine mass and machine/rectifier loss. A Pareto optimal front, which represents the tradeoff between mass and loss (including resistive loss, core loss in the stator, and conduction loss), was obtained for machine/active rectifier and machine/passive rectifier systems.

For the design of all machines considered herein, it was assumed that the dc bus voltage is 5 kV, the output power required is 25 MW, the prime mover operates at a fixed speed of 3600 rpm, and all winding current densities are less than 10 A/mm^2. Although a thermal analysis was not performed, the current density limit is within reason, provided that the machines are liquid cooled. The maximum packing factor of the stator slots was assumed to be 0.5 and that of the rotor 0.6. The on-state voltage drop of IGBTs was assumed to be 6 V. The drop of the diodes (thyristors) was taken to be 4 V. These were based upon values obtained from datasheets of high power switching devices. The multi-objective optimization of each topology was carried out using GOSET 2.4 [51].

The core of the WRSM/rectifier system design study is the dynamic MEC model for WRSMs proposed in Chapter 4. The example cross-sectional WRSM geometry and representative MEC network are shown in Figure 6.2 and Figure 6.3, respectively. In Figure 6.2, the rotor of the machine consists of the shaft with radius r_{sh}, the rotor core which conducts flux circumferentially around the machine with depth d_{rc}, and the rotor teeth with depth d_{rp} and outer radius r_{ro}. The rotor teeth consist of a tooth shank connected to the rotor core with width w_{rp} and a tooth tip with width w_{rt}. The rotor damper bar has a radius r_{ds} on the rotor shank and a radius r_{dt} on the rotor tip. The number of rotor teeth is equivalent to the number of poles. The airgap has a uniform depth between stator and rotor teeth of g. The stator of the machine consists of the stator teeth with depth d_{st} and inner radius r_{si}, the stator slots with a width of w_{ss} at the airgap, and the stator back iron of depth d_b and outer radius r_{so}. The stator can have any integer number of slots per pole per phase. The length of the active part of the machine is denoted l.

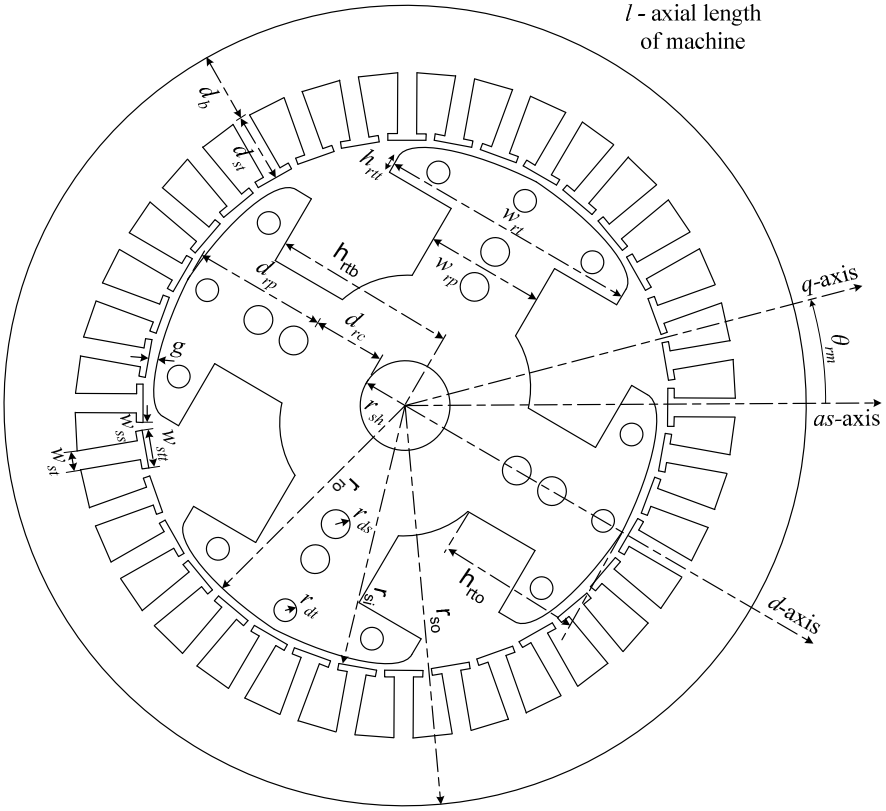


Figure 6.2: Example WRSM geometry/configuration.

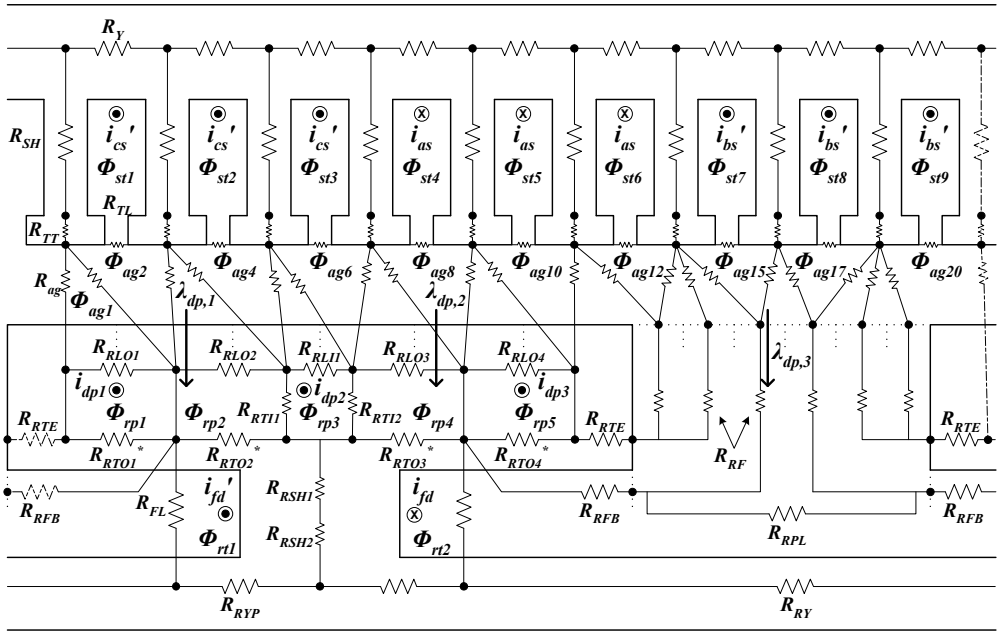


Figure 6.3: Representative WRSM MEC.

6.2 Design of WRSM/Active Rectifier Systems

The first system was structured as a wound-rotor synchronous machine connected to an active rectifier as shown in Figure 6.4. In this system, it was assumed that the phase currents i_{as}, i_{bs}, and i_{cs} are regulated to be sinusoidal waveforms, and the field current I_{fd} is also regulated to be an ideal current source.

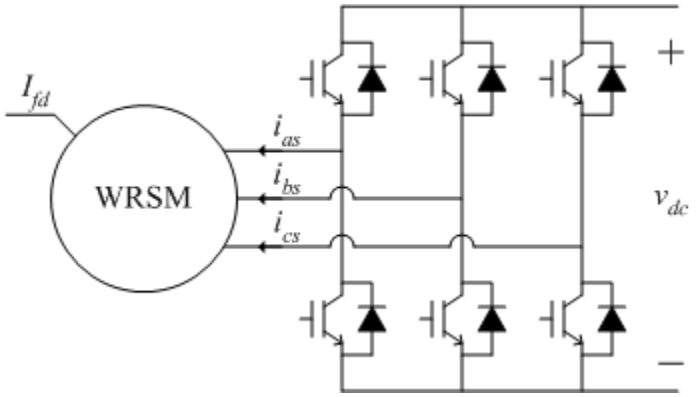


Figure 6.4: WRSM/active rectifier system.

The MEC model is structured as a current input voltage output, steady-state model, in which damper bars are not included. The design space variables for the studies relating to the WRSM/active rectifier are given by,

$$\begin{aligned}\boldsymbol{\theta} = \big[& d_{rc} \quad l \quad d_{rt} \quad g \quad d_{st} \quad d_{bs} \quad fw_{ss} \quad fh_{rt} \quad fw_{rt} \quad \cdots \\ & fw_{rp} \quad N_s \quad I_s \quad \beta \quad N_{fd} \quad I_{fd} \quad P_p \quad ftipw \quad ftiph \big]^T\end{aligned} \tag{6.1}$$

Within (6.1), the first six variables relate to the machine geometry, and they were defined earlier when discussing Figure 6.2. Genes number 7, 8, 9, 10, 17, and 18 are scaling factors between 0 and 1 that are used to establish machine geometry based upon calculations. For instance, with the stator tooth height known, *ftiph* defines the height of the stator tooth tip as a fraction of the total height of the stator tooth. The gene N_s represents the number of turns in the phase windings function. It is noted that the stator winding has a slots/pole/phase number of 2 and the *a*-phase winding function is expressed as $\begin{bmatrix} 0 & 0 & N_s & N_s & 0 & 0 \end{bmatrix}$. With appropriate phase shift, the b- and c-phase

winding function can be achieved. The variable I_s defines the rms phase current, and β defines the phase angle of the stator currents. Thus the phase currents are expressed as,

$$\begin{bmatrix} i_{as} \\ i_{bs} \\ i_{cs} \end{bmatrix} = \sqrt{2} I_s \begin{bmatrix} \cos(\theta_r + \beta) \\ \cos(\theta_r + \beta - 2\pi/3) \\ \cos(\theta_r + \beta + 2\pi/3) \end{bmatrix} \tag{6.2}$$

The variable P_p denotes the number of pole pairs. The choice of material can be readily included in the design space; however, M19 was selected as the stator and rotor lamination and copper was used for the stator and field windings in this study. The properties of these materials are defined in [61].

6.3 Design of WRSM/Passive Rectifier Systems

The second system considered was a wound-rotor synchronous machine interfaced to the dc bus with a passive rectifier as shown in Figure 6.5. The influence of damper bars is of interest in passive rectifier systems because the system behavior is based upon subtransient dynamics of the machine [24], [25]. More specifically, the regulation characteristic of the output voltage is dependent on the subtransient reactances. Arguably, a lower value of these reactance could yield an increase in current (power) for a given dc voltage. However, adjusting subtransient reactance requires one to introduce damper bars. The current in the damper bars produces additional resistive losses, which likely impacts the loss for a given generator size. Therefore, the sizing, number, and true benefit of the bars were largely unknown prior to this study.

The MEC model is structured as a voltage input current output, dynamic model, in which damper current dynamics are included. By coupling with the passive rectifier model, the phase voltage can be calculated using the phase currents through the following steps. First, the phase currents ($\mathbf{i}_{abcs}$ or $\mathbf{i}_{qd0s}$) can be transformed to the rectifier line currents ($\mathbf{i}_{abcl}$). For a machine with wye-connection, the rectifier line currents are equal to the negative of the phase currents based on the assumption that phase currents flow outside to the machine in generator mode. For a machine with delta-connection, the line currents are calculated as

$$\mathbf{i}_{abcl} = \begin{bmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \left(\mathbf{K}_s\right)^{-1} \mathbf{i}_{qd0s} \tag{6.3}$$

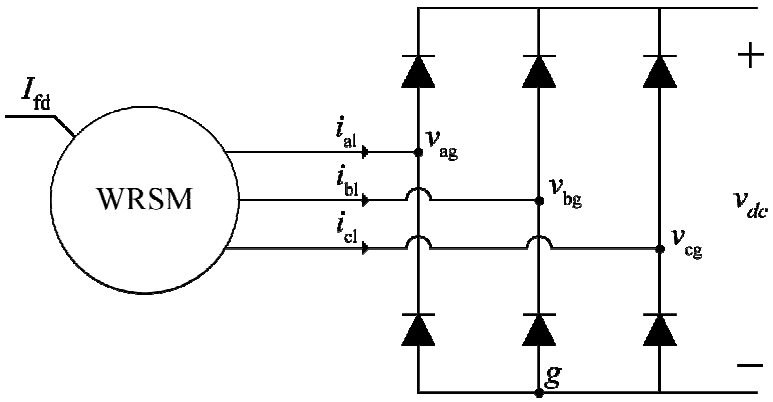


Figure 6.5: WRSM/passive rectifier system.

Next, the rectifier voltage ($\mathbf{v}_{abcg}$) can be calculated by the rectifier line current ($\mathbf{i}_{abcl}$), using the relationship shown in Figure 6.6. When the magnitude of the rectifier line current is above ε, the rectifier voltage logarithmically approaches either $v_{dc} + V_{drop}$ or $-V_{drop}$, according to the current direction. Otherwise, a linear relationship is used when the current magnitude is below ε. It is noted that ε has a value of 0.005 and V_{drop} is the diode voltage drop.

Using the above logic, the rectifier voltages can be determined and finally be used to calculate the phase voltage in the arbitrary reference frame as,

$$\mathbf{v}_{qds} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{K}_s \mathbf{v}_{abcg} \tag{6.4}$$

for a machine with wye-connection and,

$$\mathbf{v}_{qd0s} = \frac{3}{2} \begin{bmatrix} 1 & \sqrt{3}/3 & 0 \\ -\sqrt{3}/3 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{K}_s \mathbf{v}_{abcg} \tag{6.5}$$

for a machine with delta-connection.

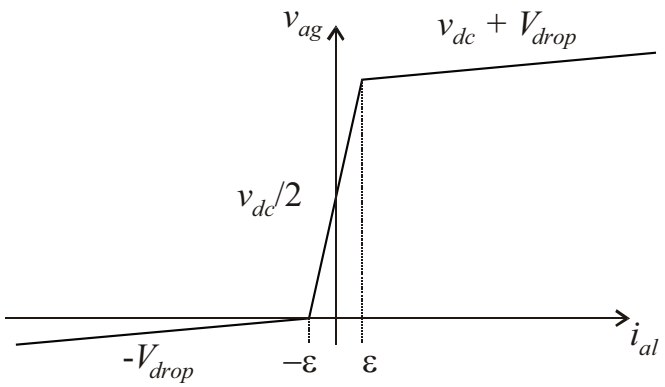


Figure 6.6: Relationship between the rectifier voltage and rectifier line current.

The design space variables for the studies relating to the WRSM/active rectifier are given by,

$$\boldsymbol{\theta} = \begin{bmatrix} d_{rc} & l & d_{rt} & g & d_{st} & d_{bs} & fw_{ss} & fh_{rt} & fw_{rt} & \cdots \\ fw_{rp} & N_s & N_{fd} & I_{fd} & P_p & ftipw & ftiph & r_{dt} & d_{num} & d_{con} \end{bmatrix}^T \tag{6.6}$$

where the variables related to the phase currents have been removed and the radius r_{dt}, number $dnum$, and connection type (pole or pole-pole) $dcon$, of the damper bars have been included. All design assumptions and constraints of active rectifier design also apply with one exception. The constraint on the calculated dc bus voltage is no longer needed since the output bus voltage is pre-defined.

6.4 Results and Discussion

Design optimizations for both of the system topologies have been studied using the GOSET tool box, with a population number of 800 and a generation number of 600. An estimation of the elapsed time for the optimizations process of the WRSM/active rectifier system is approximately 10 hours, while it takes about 250 hours for the WRSM/passive rectifier system. The design optimization was performed several times to ensure convergence and repeatability of the design process. The final Pareto front obtained for the passive rectifier design is shown in Figure 6.7.

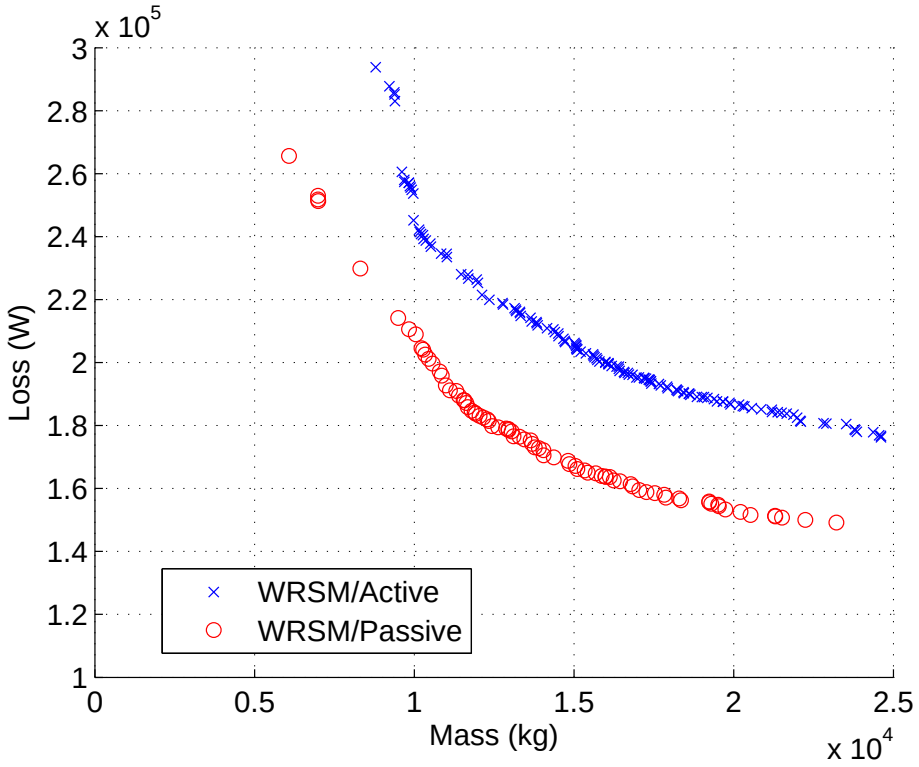


Figure 6.7: Pareto fronts of alternative WRSM/rectifier topologies.

Comparing the pareto fronts of alternative WRSM/rectifier topologies in Figure 6.7, a surprising result is that for a given system loss, the mass of a passive rectifier machine is less than that of an active rectifier machine. This is partly due to the fact that the on-state voltage drop of the power diodes are less than those of IGBTs. In addition, through the evolutionary optimization process, the core and winding geometry of the passive rectifier machines are different than those of the active rectifier machines. This difference effectively compensates for the difference in harmonic content of the stator current that results from diode rectification. In order to observe differences in geometry/configuration of the alternative WRSM/rectifier systems, the comparison of genes in the design studies are shown in Figure 6.8 and Figure 6.9.

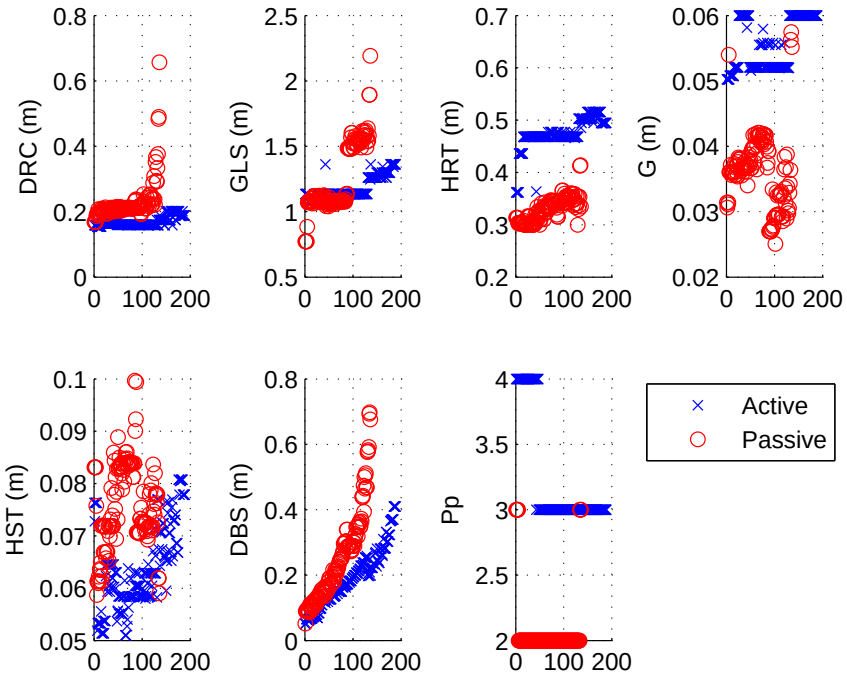


Figure 6.8: Comparison of genes of alternative WRSM/rectifier systems (a).

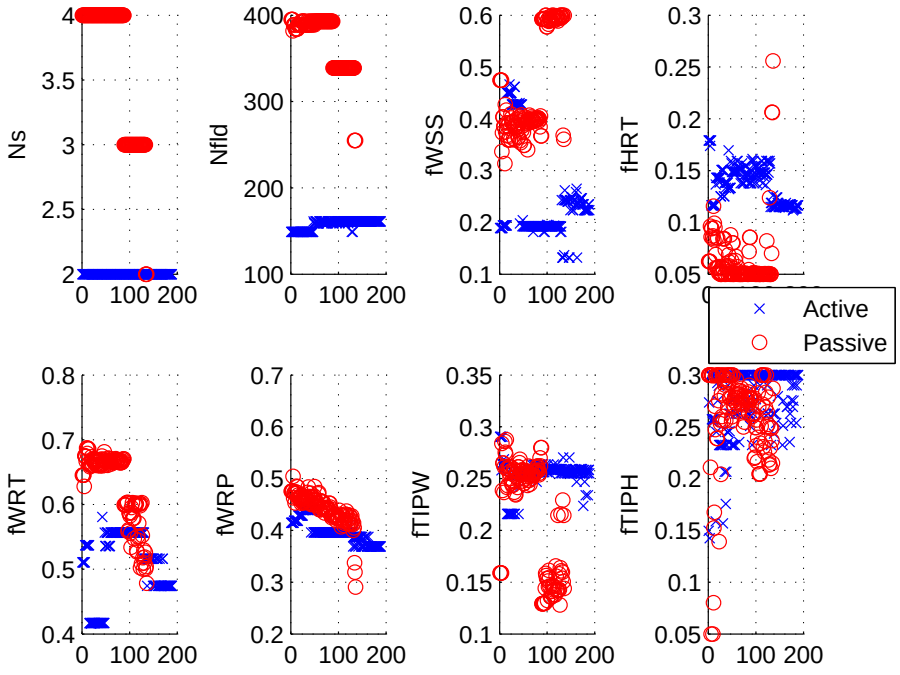


Figure 6.9: Comparison of genes of alternative WRSM/rectifier systems (b).

From Figure 6.8 and Figure 6.9 one can see that the WRSM/active rectifier design has larger height of rotor teeth (HRT), airgap length (G), and pole pair (Pp). On the other hand, the WRSM/passive rectifier design has larger stack length (GLS), stator turns (Ns) and field turns (Nfld). Of note is that all machines on the front have zero damper bars. Thus, a conclusion is that there appears to be no advantage, in either mass or loss, to utilize damper bars in the system topologies considered. In addition, since the pole pair (Pp) number is 3 or 4 for the WRSM/active rectifier design, and is 2 or 3 for the WRSM/passive rectifier design, thus four example machine designs with different pole pair number are shown in Figure 6.10 - Figure 6.13 for comparison.

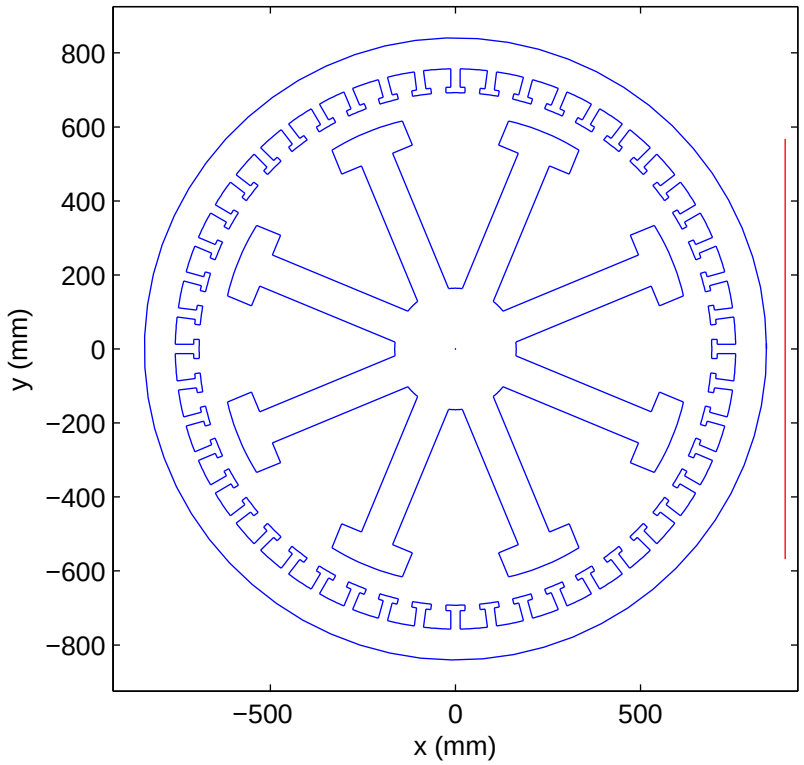


Figure 6.10: Example design of an 8-pole WRSM connected to active rectifier.

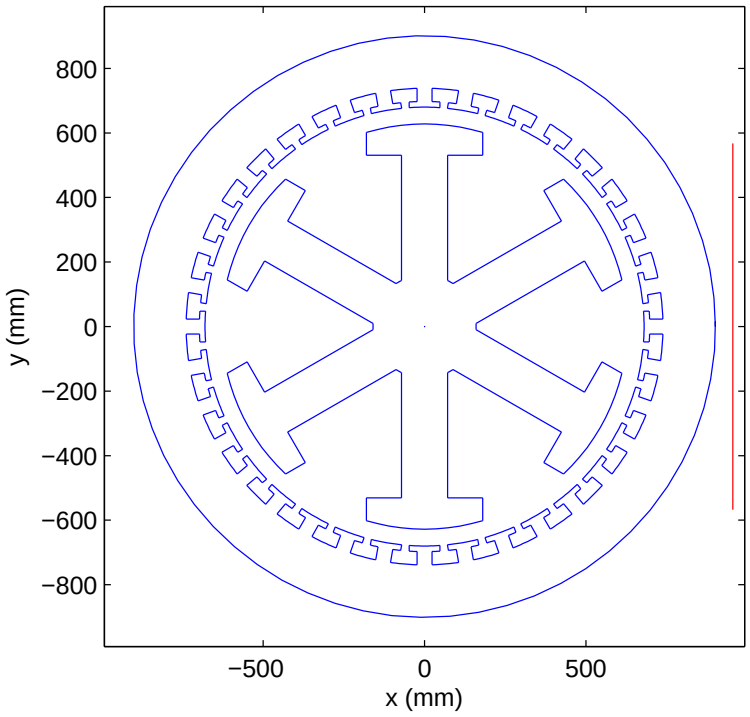


Figure 6.11: Example design of a 6-pole WRSM connected to active rectifier.

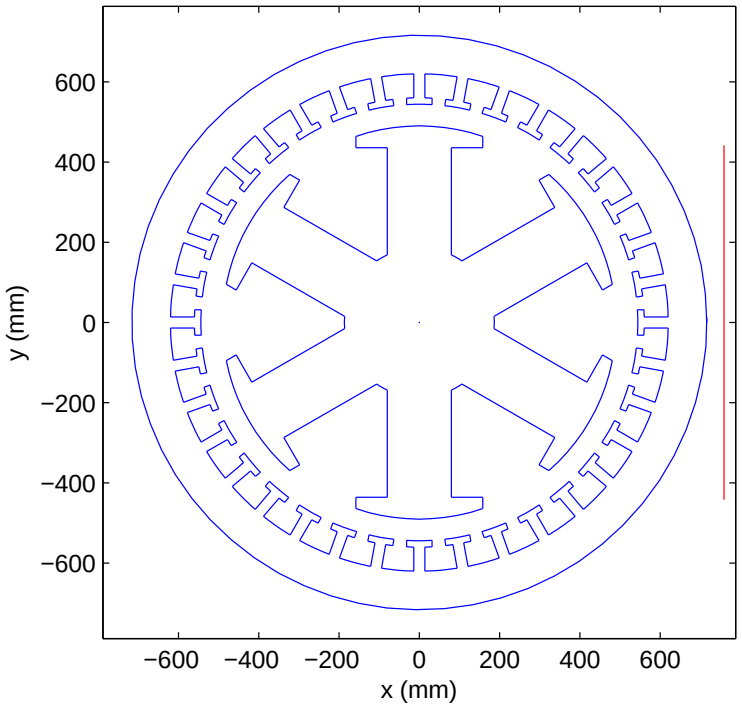


Figure 6.12: Example design of a 6-pole WRSM connected to passive rectifier.

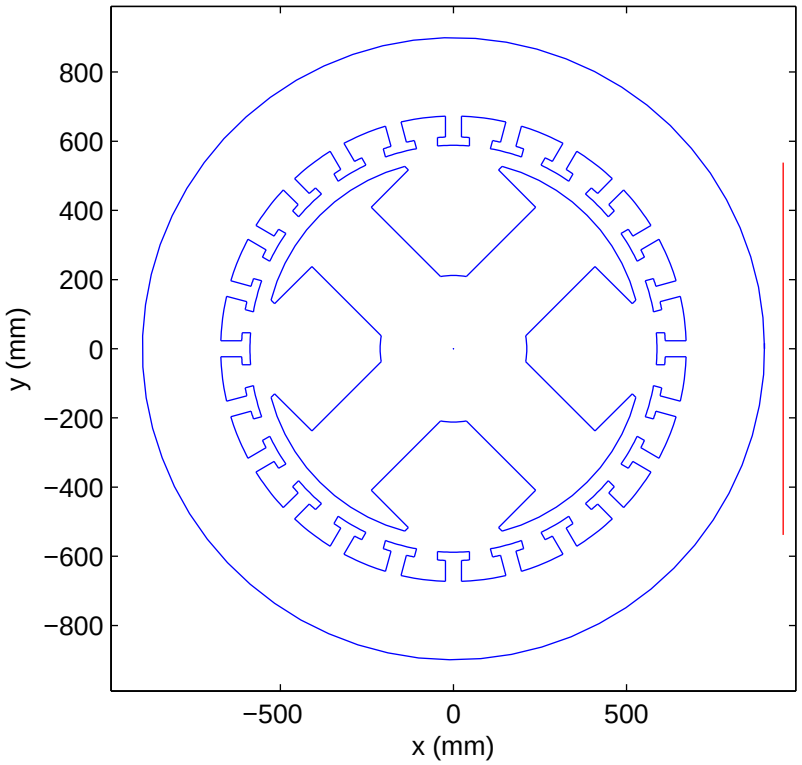


Figure 6.13: Example design of a 4-pole WRSM connected to passive rectifier.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

The key contributions of this research are twofold. First has been the result of the study of optimized excitation strategies that are consistent with goals of minimizing mass and loss for a WRSM drive system. It is shown that utilizing *qd* models with/without saturation incorporated along the *d*-axis leads to suboptimal excitation that is different than obtained from a MEC over much of the expected operating region. However, based upon analysis of several alternative strategies, a simplified control is derived in which *d*-axis current is zero, field current is held fixed, and *q*-axis current is varied linearly with torque. This control results in system-level efficiencies nearly the same as a control designed to maximize efficiency. Finally, the tradeoffs and limitations of the simplified control are explored when the desire is to optimize available torque over variable speeds that may or may not be controllable.

Second, an enhanced dynamic MEC model for WRSMs has been developed. The model enables one to include the dynamics of an arbitrary number of damper bars with and without connection between poles. The dynamic model is structured to accept terminal winding voltage as input, which leads to relatively straightforward coupling with external circuits. As part of the dynamic model development, new geometry features, including stator tooth tips and rotor damper bars have been added, which greatly increases the dimension of potential machine topologies that can be analyzed and design. In addition, a multi-slices approach has been implemented to the steady state and dynamic MEC to model the skewing effect. Finally, alternative WRSM/rectifier systems are compared based on the steady-state and dynamic MEC models.

A 10 kW and a 2 kW WRSM have been used to validate the proposed dynamic MEC model and control approaches, respectively. Several test cases have been run and have shown relatively strong correlation among MEC, FEA, and hardware results.

7.2 Future Work

Further validation of the dynamic model will be performed. The required parts to assemble and set up the 10 kW WRSM have been machined. Once the WRSM has been mounted on the test bench and ready to operate, more time-domain waveforms of phase current, phase voltage, and torque will be measured and analyze. Different load circuits, including resistive-inductive (RL) load, active rectifier, and passive rectifier are of interest. A particular focus will be to compare transients in the time-domain and the operational impedances at various frequencies. To compare hardware and simulated transient performance, it is desirable to measure the damper winding currents. This is challenging when the rotor is moving at 1800 rpm. Thus, a goal will be to develop a technique to measure the damper winding currents in-situ. A Rowgowski coil connected with a wireless voltage sensor will be evaluated for this purpose.

Simulation results of the skew model has been presented and compared to analytical model. In order to achieve more thorough validation, a 10 kW WRSM with a skewed rotor will be constructed so that different time-domain waveforms can be measured. 3D FEA analysis is also preferred if more computational power is accessible.

In addition, although the dynamic MEC model is designed for three-phase machines, it sets up the baseline to explore the applications of single-phase or multi-phase machines design. Compared to three-phase machines, the single-phase machines operate at lower power level and usually constant frequency. The single-phase machines can be connected to the ac grid directly without any power electronics, but with an auxiliary winding, which draws an industrial desire for its simplicity. On the other hand, the multi-phase machines provide lower harmonics content at the price of extra phases of windings. Due to the extra number of phases, the rating of the semiconductor switches for each inverter leg can be reduced accordingly, and the fault tolerance for phase failure may be improved as well.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Bash, M.L., Pekarek, S.D., "Modeling of Salient-Pole Wound-Rotor Synchronous Machines for Population-Based Design," *IEEE Transactions on Energy Conversion*, vol.26, no.2, pp.381-392, June 2011.

[2] Friedrich, G., "Experimental comparison between Wound Rotor and permanent magnet synchronous machine for Integrated Starter Generator applications," *2010 IEEE Energy Conversion Congress and Exposition (ECCE)*, 12-16 Sept. 2010.

[3] G Friedrich, A Girardin, "Integrated starter generator," *IEEE Industry Applications Magazine*, 2009.

[4] Janiaud N., Vallet F.-X., Petit M., Sandou G., "Electric Vehicle Powertrain Architecture and Control Global Optimization," *EVS24 International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium, Stavanger, Norway*, May 13-16, 2009.

[5] Sudhoff, S.D., Corzine, K.A., Glover, S.F., Hegner, H.J., Robey, H.N., Jr. "DC link stabilized field oriented control of electric propulsion systems,"*IEEE Transactions on Energy Conversion*, vol.12, no.1, pp.27-33, 1998.

[6] Nordin, Kamarudin B., Novotny, Donald W., Zinger, Donald S., "The Influence of Motor Parameter Deviations in Feedforward Field Orientation Drive Systems," *IEEE Transactions on Industry Applications*, vol.IA-21, no.4, pp.1009-1015, 1985.

[7] Corley, M.J., Lorenz, R.D., "Rotor position and velocity estimation for a salient-pole permanent magnet synchronous machine at standstill and high speeds," *IEEE Transactions on Industry Applications*, vol.34, no.4, pp.784-789, 1998.

[8] Wasynczuk, O., Sudhoff, S.D., Corzine, K.A., Tichenor, J.L., "A maximum torque per ampere control strategy for induction motor drives," *IEEE Transactions on Energy Conversion*, vol.13, no.2, pp.163-169, 1998.

[9] Ching-Tsai Pan, Sue, S.-M., "A linear maximum torque per ampere control for IPMSM drives over full-speed range," *IEEE Transactions on Energy Conversion*, June 2005.

[10] Morimoto, S., Sanada, M., Takeda, Y., "Wide-speed operation of interior permanent magnet synchronous motors with high-performance current regulator," *IEEE Transactions on Industry Applications*, Jul/Aug 1994.

[11] Uddin, M.N., Radwan, T.S., Rahman, M.A., "Performance of interior permanent magnet motor drive over wide speed range," *IEEE Transactions on Energy Conversion*, Mar 2002.

[12] Mohamed, Y.A.-R.I., Lee, T.K., "Adaptive self-tuning MTPA vector controller for IPMSM drive system," *IEEE Transactions on Energy Conversion*, Sept. 2006.

[13] Bing Cheng, Tesch, T.R., "Torque Feedforward Control Technique for Permanent-Magnet Synchronous Motors," *IEEE Transactions on Industrial Electronics*, March 2010.

[14] M.F. Rahman, L. Zhong, and K. W. Lim, "A direct torque controlled interior permanent magnet synchronous motor drive incorporating field weakening," *Industry Application, IEEE Transactions on*, vol. 34, iss. 6, pp. 1246-1253, Nov. 1998.

[15] Lixin Tang, LiminZhong, M. F. Rahman, "A novel direct torque controlled interior permanent magnet synchronous machine drive with low ripple in flux and torque and fixed switching frequency," *Power Electronics, IEEE Transactions on*, vol. 19, iss. 2, pp. 346-354, 2004.

[16] Sanda V. Paturca, MirceaCovrig, Leonard Melcescu, "Direct torque control of permanent magnet synchronous motor (PMSM) – an approach by using space vector modulation (SVM)," *Proceedings of the 6th WSEAS/IASME Int. Conf. on Electric Power Systems, High Voltage, Electric Machines*, Tenerife, Spain. Dec. 2006.

[17] Ion Boldea, "DTFC-SVM motion-sensorless control of a PM-assisted reluctance synchronous machine as starter-alternator for hybrid electric vehicles," *Power Electronics, IEEE Transactions on*, vol. 21, iss. 3, pp. 711-719, 2006.

[18] Jun Zhang, M. F. Rahman, "A direct-flux-vector-controlled induction generator with space-vector modulation for integrated starter-alternator," *Industrial Electronics, IEEE Transactions on*, vol. 54, iss. 5, pp. 2512-2520, 2007.

[19] H.C. Roters, *Electromagnetic Devices*, John Wiley & Sons, Inc., New York, 1941.

[20] E.C. Cherry, "The duality between interlinked electric and magnetic circuits and the formation of transformer equivalent circuits," *Proc. Physical Soc. of London*, pp. 101-111, 1949.

[21] E.R. Laithwaite, "Magnetic equivalent circuits for electrical machines," *Proc. IEE*, vol.114, pp. 1805-1809, Nov. 1967.

[22] V. Ostović, *Dynamics of Saturated Electric Machines*, Springer-Verlag, New York, 1989.

[23] M. Bash, J. Williams, and S. Pekarek, "Incorporating motion in mesh-based magnetic equivalent circuits," *IEEE Transactions on Energy Conversion*, vol. 25, iss. 2, pp. 329 – 338, 2010.

[24] Warner, T. H., Kassakian, J.G., "Transient Characteristics and Modeling of Large Turboalternator Driver Rectifier/Inverter Systerms Based on Field Test Data," *IEEE Transactions on Power Apparatus and Systems*, vol. 104, No. 7, pp. 1804-1811, Jul. 1985.

[25] Sudhoff, Scott D., Wasynczuk, Oleg, "Analysis and average-value modeling of line-commutated converter-synchronous machine systems," *IEEE Trans. Energy Conv.*, vol. 8, No. 1, pp. 92-99, Mar. 1993.

[26] P. Krause, O. Wasynczuk, and S. Sudhoff, *Analysis of Electric Machinery and Drive Systems*, 2nd ed. Piscataway, NJ: IEEE Press, 2002.

[27] Pekarek, S.D., Walters, E.A., Kuhn, B.T., "An efficient and accurate method of representing magnetic saturation in physical-variable models of synchronous machines," *IEEE Transactions on Energy Conversion*, vol.14, no.1, pp.72-79, Mar 1999.

[28] Sudhoff, S.D., Corzine, K.A., Hegner, H.J., Delisle, D.E., "Transient and dynamic average-value modeling of synchronous machine fed load-commutated converters," *IEEE Transactions on Energy Conversion*, vol.11, no.3, pp.508-514, Sept 1996.

[29] Jatskevich, J., Pekarek, S.D., Davoudi, A., "Parametric average-value model of synchronous machine-rectifier systems," *IEEE Transactions on Energy Conversion*, vol.21, no.1, pp.9-18, Mar 2006.

[30] Aliprantis, D.C., Sudhoff, S.D., Kuhn, B.T., "A Synchronous Machine Model With Saturation and Arbitrary Rotor Network Representation," *IEEE Transactions on Energy Conversion*, vol.20, no.3, pp.584-594, Sept 2005.

[31] K.J. Bradley and A. Tami, "Reluctance mesh modeling of induction motors with healthy and faulty rotors," *Proc. IEEE Ind. Applications Conf.*, vol. 1, pp. 625-632, Oct. 1996.

[32] W. Shuting, L. Heming, L. Yonggang, and X. Zhaofeng, “Reluctance network model of turbo-generator and its application – Part I: Model,” *Proc. 8th Int. Electrical Machines and Systems*, vol. 3, pp. 1988-1993, Sept. 2005.

[33] G.R. Slemon, “An equivalent circuit approach to analysis of synchronous machines with saliency and saturation,” *IEEE Trans. Energy Conv.*, vol.5, No. 3, Sept. 1990.

[34] Xiao, Y., Slemon, G.R., Iravani, M.R., “Implementation of an equivalent circuit approach to the analysis of synchronous machines,”*IEEE Trans. Energy Conv.*, vol.9, No. 4, Dec. 1994.

[35] Hamill, David C.," Lumped equivalent circuits of magnetic components: the gyrator-capacitor approach," *IEEE Transactions on Power Electronics*, vol. 8, no. 2, pp. 97-103, Apr. 1993.

[36] Carpenter, M.J., Macdonald, D.C., “Circuit representation of inverter-fed synchronous motors,” *IEEE Trans. Energy Conv.*, vol. 4, No. 3, pp.531-537, Sep. 1989.

[37] *Piecewise Linear Electrical Circuit Simulation (PLECS) User Manual*, Version 1.5, Plexim GmbH, 2006. Available at: http://www.plexim.com (accessed on January 21, 2013).

[38] Davoudi, A., Chapman, P.L., Jatskevich, J., Khaligh, A., “Reduced-Order Modeling of High-Fidelity Magnetic Equivalent Circuits,” *IEEE Trans. Power Electronics*, vol.24, No. 12, pp. 2847 - 2855, Dec. 2009.

[39] Davoudi, A., Chapman, P.L., Jatskevich, J., Behjati, H. , “Reduced-Order Dynamic Modeling of Multiple-Winding Power Electronic Magnetic Components,” *IEEE Trans. Power Electronics*, vol.27, No. 5, pp. 2220 - 2226, May 2012.

[40] V. Ostović, “A novel method for evaluation of transient states in saturated electric machines,” *IEEE Trans. Ind. Appl.*, vol. 25, pp. 96-100, Jan.-Feb. 1989.

[41] Li Yao, “Magnetic Field Modelling of Machine and Multiple Machine Systems Using Dynamic Reluctance Mesh Modelling,” PhD dissertation, Department of Electrical and Computer Engineering, University of Nottingham, Nottinghan, England, May 2006.

[42] H.W. Derbas, J.M. Williams, A.C. Koenig, and S.D. Pekarek, “A comparison of nodal- and mesh-based magnetic equivalent circuit models,” *IEEE Trans. Energy Conv.*, vol. 24, iss. 2, pp. 388-396, June 2009.

[43] S.D. Sudhoff, B.T. Kuhn, K.A. Corzine, and B.T. Branecky, "Magnetic equivalent circuit modeling of induction motors," *IEEE Trans. Energy Conv.*, vol. 22, no. 2, pp. 259-270, June 2007.

[44] M. Amrhein and P.T. Krein, "Magnetic equivalent circuit modeling of induction machines design-oriented approach with extension to 3-D," *Proc. IEEE Electric Mach & Drives Conf.*, vol. 2, pp. 1557-1563, May 2007.

[45] M. Amrhein, "Induction machine performance improvements – design-oriented approaches," PhD dissertation, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, Apr. 2007.

[46] E. Sarani, K, Abbaszadeh, and M. Ardebili, "Modeling and simulation of turn fault and unbalanced magnetic pull in induction motor based on magnetic equivalent circuit method," *Proc. 8th Int. Electrical Machines and Systems*, vol. 1, pp. 52-56, Sept. 2005.

[47] G.Y. Sizov, A. Sayed-Ahmed, Y. Chia-Cho, and N.A. Demerdash, "Analysis and Diagnostics of Adjacent and Nonadjacent Broken-Rotor-Bar Faults in Squirrel-Cage Induction Machines"; *Industrial Electronics, IEEE Transactions on*, vol. 56, iss. 11, pp. 4627 – 4641, 2009.

[48] J. Gyselinck, L. Vandevelde, and J. Melkebeek, "Coupling finite elements and magnetic and electrical networks in magnetodynamics," *International conference on electrical machines*, vol. 2, Sept. 1998, pp. 1431-1436.

[49] E.K.P. Chong and S.H. Żak, *An Introduction to Optimization*, 2nd ed. John Wiley & Sons, Inc., 2001, pp. 237-250.

[50] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, Inc., 2001.

[51] *Genetic Optimization System Engineering Tool (GOSET) For Use with MATLAB*, Manual Version 2.4, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, with United States Naval Academy, Annapolis, MD, 2005. Avaliable: https://engineering.purdue.edu/ECE/Research/Areas/PEDS/go_system_engineering_toolbox.

[52] B.N. Cassimere and S.D. Sudhoff, "Population-based design of surface-mounted permanent-magnet synchronous machines," *IEEE Trans on Energy Conversion*, vol. 24, no. 2, June 2009, pp. 338-346.

[53] J. Cale and S.D. Sudhoff, "EI core inductor designs using population-based design algorithms," *22nd IEEE Applied Power Electronics Conference, APEC 2007*, pp. 1062-1069, Feb. 2007.

[54] Sudhoff, S.D. and G. Shane, PMM Toolbox: Power Magnetic Material Toolbox (Version 1.0) [MATLAB Toolbox], School of Electrical and Computer Engr., Purdue Univ., https://engineering.purdue.edu/ECE/Research/Areas/PEDS/power_magnetic_material_toolbox, 2010.

[55] W. Zhu, S. Pekarek, and B. Fahimi, "Derivation of a magnetic equivalent circuit model for analysis and design of synchronous generators," University of Missouri-Rolla, Tech. Report, Dec. 2003.

[56] J. Reinert, A. Brockmeyer, and R. W. A. A. De Doncker, "Calculation of losses in ferro- and ferrimagnetic materials based on the modified Steinmetz equation," *IEEE Trans. Ind. Appl.*, vol. 37, no. 4, pp. 1055–1061, Jul. 2001.

[57] Bash, M.L., Pekarek, S.D., "Trends on the Pareto-optimal fronts of a portable generator drive system," *2012 Twenty-Seventh Annual IEEEApplied Power Electronics Conference and Exposition (APEC)*, vol., no., pp.931-937, 5-9 Feb. 2012.

[58] Standard Specification for Nonoriented Electrical Steel Fully Processed Types, ASTM Standard A677-07, American Society for testing and Materials International, West Conshohocken, PA, 2003.

[59] Bash, M.L., Pekarek, S.D., "Analysis and validation of a population-based design of a wound-rotor synchronous machine," *IEEE Transactions on Energy Conversion*, vol.27, no.3, pp.603-614, Sept 2012.

[60] T.A. Lipo, *Introduction to AC Machine Design*, 2nd ed. Madison,WI: Wisconsin Power Electronics Research Center, 2004.

[61] Shane, Grant M., Sudhoff, Scott D., "Refinements in Anhysteretic Characterization and Permeability Modeling," *IEEE Transactions on Magnetic*, vol. 46, No. 11, pp. 3834-3843, Nov. 2010.

[62] "IEEE Standard Procedures for Obtaining Synchronous Machine Parameters by Standstill Frequency Response Testing (Supplement to ANSI/IEEE Std 115-1983, IEEE Guide: Test Procedures for Synchronous Machines)," *IEEE Std 115A-1987*, 1987.

[63] Jahns, T.M. , Soong, W.L. , "Pulsating torque minimization techniques for permanent magnet AC motor drives-a review," *IEEE Transactions on Industrial Electronics*, vol.43, No. 2, pp. 321-330, Apr 1996.

[64] Bianchi, N. , Bolognani, S. , “Design techniques for reducing the cogging torque in surface-mounted PM motors,” *Conference Record of the 2000 IEEE Industry Applications Conference*, vol.1, pp. 179-185, Oct 2000.

[65] Islam, R. , Husain, I. , Fardoun, A. , McLaughlin, K. , “Permanent-Magnet Synchronous Motor Magnet Designs With Skewing for Torque Ripple and Cogging Torque Reduction,” *IEEE Transactions on Industrial Applications*, vol.45, No. 1, pp. 152-160, Jan-Feb 2009.

[66] Alhamadi, M.A. , Demerdash, N.A. , “Modeling of effects of skewing of rotor mounted permanent magnets on the performance of brushless DC motors,” *IEEE Transactions on Energy Conversion*, vol.6, No. 4, pp. 721-729, Dec 1991.

[67] Knight, A.M. , Troitskaia, S. ; Stranges, N. ; Merkhouf, A. , “Analysis of large synchronous machines with axial skew, part 1: flux density and open-circuit voltage harmonics,” *IET Electric Power Applications*, vol.3, No. 5, pp. 389-397, Sept 2009.

[68] Knight, A.M. , Troitskaia, S. ; Stranges, N. ; Merkhouf, A. , “Analysis of large synchronous machines with axial skew, part 2: inter-bar resistance, skew and losses,” *IET Electric Power Applications*, vol.3, No. 5, pp. 398-406, Sept 2009.

[69] V. Ostovic, *Dynamics of saturated electric machines*, New York: Springer Verlag Press, 1989.

[70] Sizov, G.Y. , Zhang, P. , Ionel, D.M. , Demerdash, N.A.O. , Brown, I.P. , Smith, A.O. , Solveson, M.G. , “Modeling and analysis of effects of skew on torque ripple and stator tooth forces in permanent magnet AC machines,” *IEEE Energy Conversion Congress and Exposition (ECCE)*, pp. 3055-3061, Sept 2012.

[71] Williamson, Stephen , Flack, T.J. ; Volschenk, A.F. , “Representation of skew in time-stepped two-dimensional finite-element models of electrical machines,” *IEEE Transactions on Industrial Applications*, vol.31, No. 5, pp. 1009-1015, Sept-Oct 1995.

[72] Gyselinck, Johan, Melkebeek, Jan, “Modelling of electrical machines with skewed slots using the two-dimensional finite element method,” *Proceedings of the 13th International Conference on Electrical Machines*, pp. 125-130, Sept 1996.

[73] Piriou, F. , Razek, A. , “A model for coupled magnetic-electric circuits in electric machines with skewed slots,” *IEEE Transactions on Magnetics*, vol.26, No. 2, pp. 1096-1100, Mar 1990.

[74] Gyselinck, J.J.C. , Vandevelde, L. ; Melkebeek, J.A.A. , “Multi-slice FE modeling of electrical machines with skewed slots-the skew discretization error,” *IEEE Transactions on Magnetics*, vol.37, No. 5, pp. 3233-3237, Sept 2001.

APPENDICES

A. MATLAB CODE

The code for the enhanced steady-state and dynamic MEC models are provided herein. A list of the filenames with the corresponding description is shown in Table A.1.

Table A.1
Filenames and description.

File	Description	Page
wrsm_design.m	Run WRSM design study.	127
wrsmfit.m	Evaluates a particular machine design (set of design variables) based on the constraints and objectives. Assigns each design a fitness value.	129
wrsm_model.m	Intializes MEC simulation variables, solves the MEC system of equations and plots results.	135
design_param.m	Creates a vector of machine/simulation parameters for a given machine using design variables.	140
wrsmdynamics.m	Solves the Dynamics of the MEC network.	149
get_reluctances.m	Calculates all terms in the reluctance equation except for the relative permeability. This is done for all iron permeances in the stator and rotor. Calculates cross-sectional area. Calculates all reluctances residing in air.	164
get_Pag.m	Determines the airgap permeance between	171

	each rotor tooth/slot section and stator tooth.	
get_J.m	Determines the Jacobian.	180
get_meshmatrices.m	Builds the matrices A and d used to solve for flux. Outputs Cr for use by get_J.m	182
shape_alg.m	Determines the mesh connections for each reluctance and mmf source for a given rotor position. The first column of the connection matrics is left as zero and is later updated with the specific reluctance/source value.	184
get_mur_exp.m	Calculate mur and pmur from exponential curve fit formulation in PMMT.	197
get_mass.m	Calculates the weight of the machine.	198
coreloss.m	Calculates the core loss of the iron sections for any given material.	201
calc_dploss.m	Calculates damper loss.	202
wrsmpostprocess.m	Calculates postprocessing values (voltage, flux linkage, etc.) after modeling a machine.	203
plotwrsm.m	Depicts the machine topology in a plot.	204
rect.m	Calculates the rectifier voltages based on the rectifier currents.	209
tools.m	Finds the average value, rms value, and/or ripple of a given signal.	211
wrsmdynamics_ multislice.m	Similar to wrsmdynamics.m, but with skew model incorporated.	213
wrsmdynamics_ss_multislice.m	Similar to wrsmdynamics.m, but with skew model incorporated to analyze steady-state model.	232

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% Apr 1, 2013
%-------------------------------------------------------------------------
% M-FILE: wrsm_design.m
%
% Run WRSM design study
%-------------------------------------------------------------------------
close all
clear
clc
addpath([pwd,'\goset_2.5'])

% units
mm  = 1e-3;
cm  = 1e-2;

% set up parameters for machine design
param.SD = 0*mm;
param.damper_rshank = 0*mm;
param.damper_nshank = 0;
param.damper_dtip = 0.5;
param.vrms = 0;
param.vph = 0;
param.vfreq = 60;
param.NCYC = 2;
param.NPTS = 1e3;

% -------------------------------------------------------------------------
% Multi-objective optimization
GAP = gapdefault(2,0,500,500);
GAP.op_list = [1 2];
GAP.pp_list = [1 2];
% GAP.rp_lvl  = 0;
GAP.mc_alg  = 6.0;
GAP.ev_pp = true;               % parallel process   [Set to true]
GAP.ev_npg = 2;                 % number of evaluation groups for non-
block [Set to number of cores allocated by matlabpool]
% Set up genes
% 1-min, 2-max, 3-type, 4-chromosome
GAP.gd = [  10*cm     80*cm    3   1; % DRC-1
            0.5       3        3   1; % GLS-2
            30*cm     80*cm    3   1; % HRT-3
            20*mm     60*mm    3   1; % G1-4
            1*cm      40*cm    3   1; % HST-5
            5*cm      80*cm    3   1; % DBS-6
            0.1       0.6      3   1; % fB0-7
            0.05      0.3      3   1; % fHRTT-8
```

```
0.3       0.9       3   1; % RPIT-9
0.1       0.7       3   1; % fWRTSH-10
1         5         1   1; % Ns-11
1         1000      1   1; % Nfld-12
50        150       2   1; % ifld-13
1         7         1   1; % Pp-14
0         0.3       2   1; % tipw-15
0.05      0.3       2   1; % tiph-16
0         20*mm     3   1; % damper_rtip_1-17
0         20*mm     3   1; % damper_rtip_2-18
0         3         1   1; % damper_ntip-19
0         2         1   1]; % bar connection type-20


% START GENETIC ALGORITHM OPTIMIZATION
[fp,GAS,final_designs] = gaoptimize(@wrsmfit,GAP,param);
% ------------------------------------------------------------------------

save results final_designs fp GAS GAP param
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% Apr 1, 2013
%-------------------------------------------------------------------------
% fitness = wrsmfit(design)
%
% Evaluates a particular machine design (set of design variables) based
on
% the constraints and objectives. Assigns each design a fitness value.
% OUTPUTS: fitness  - fitness of a machine design
%
% INPUTS:  design   - design variables
%-------------------------------------------------------------------------
function fitness = wrsmfit(GAP,param,varnum)

design(1) = param.SD;
design(2) = GAP(1);
design(3) = GAP(2);
design(4) = GAP(3);
design(5) = GAP(4);
design(6) = GAP(5);
design(7) = GAP(6);
design(8) = GAP(7);
design(9) = GAP(8);
design(10) = GAP(9);
design(11) = GAP(10);
design(12) = GAP(11);
design(13) = GAP(12);
design(14) = param.vrms;
design(15) = param.vph;
design(16) = GAP(13);
design(17) = GAP(14);
design(18) = GAP(15);
design(19) = GAP(16);
design(20) = GAP(17);
design(21) = GAP(18);
design(22) = param.damper_rshank;
design(23) = GAP(19);
design(24) = param.damper_nshank;
design(25) = param.damper_dtip;
design(26) = GAP(20);
design(27) = param.vfreq;
design(28) = param.NCYC;
design(29) = param.NPTS;

% GET GEOMETRY, WINDING, AND SIMULATION PARAMETERS
[parx,pars,turns,damperdata,mudata] = design_params(design);
%-------------------------------------------------------------------------
```

```
% CONSTRAINTS:
%-------------------------------------------------------------------------
nc = 8; % Number of constraints
constraints = zeros(1,nc);
%-------------------------------------------------------------------------
% Constraint 1: Realizable and realistic geometry.
% Rotor tooth shank does not overlap at core
WRTSHchord  = pars(56);
DC          = pars(25);
RP          = pars(28);
chordmax = (DC)*sin(pi/RP);
c1a = lessthan(WRTSHchord,chordmax,0.1*chordmax);
% rotor pole tips do not overlap in the slot
WAIRT       = pars(35);
HRTT        = pars(44);
ROD         = pars(24);
maxHRT = sin(pi/2-
pi/RP)/sin(2*pi/RP)*ROD*sin(WAIRT/ROD)+2*HRTT*(RP==2);
c1b = lessthan(HRTT,maxHRT,0.01*maxHRT);
% Length constraint is met - no pancake machines
% c1c = lessthan(pars(1)/pars(3),1.82,0.182);
c1c = 1;
% HRTT is real and positive
if abs(HRTT) ~= HRTT
    HRTT = -1;
end
c1d = greaterthan(HRTT,0,0.01);
% Radius of damper bars has to be less than width of rotor sections
if c1d == 1
    SPT     = parx(2);
    WRT     = pars(34);
    ROD     = pars(24); % Rotor outer diameter, m
    WRTang  = 2*WRT/ROD;
    xout    = sin(WRTang/2)*ROD/2;  % (xout = WRTchord/2)
    WRTS2   = xout*2/SPT; % Horizontal width (not arc width) of the
rotor tooth sections
    damper_rtip = damperdata.damper_rtip; % Radius of damper windings
    if max(damper_rtip) > WRTS2/2
        Rxm = -1;
    else
        mu0     = pi*4e-7;      % Permeability of free space
        [Rxm,areas,Rair,NPRTS,NPRTB] =
get_reluctances(mu0,parx,pars,damperdata);
    end
    c1e     = greaterthan(min(Rxm),0,0.01);
else
    c1e = 0;
end
% Outer diameter constraint
c1f = lessthan(pars(1),2.5,0.25);
constraints(1) = (c1a+c1b+c1c+c1d+c1e+c1f)/6;
% constraints(1) = 1;
if constraints(1) == 1      % Machine is realizable
    %---------------------------------------------------------------------
    % Evaluate the MEC model over one stator tooth and slot
```

```
    % PARAMETERS
    NCYC    = parx(6);              % Number of cycles
    DT      = parx(12);             % Time step in s
    wrm     = parx(4)*2*pi/60;      % Mechanical rotor speed in rad/s
    wr      = (pars(28)/2)*wrm;     % Electrical rotor speed in rad/s
    rs      = pars(23);                 % Phase resistance in ohm
    rfld    = pars(43);                 % Field resistance in ohms
    ifld    = pars(47);                 % Field current (A)
    Pmin    = parx(24);                 % Minimun output power (w)
    synfreq = (pars(28)/2)*parx(4)/60;  % Frequency of vas,vbs,vcs -
(assumed to be synchronized with rotor speed)
    damper_ntip = damperdata.damper_ntip; % Number of damper windings
on rotor tip
    Rd = damperdata.Rd; % Resistance of damper windings on rotor tip
    % DYNAMICS DESCRIPTION
[t,vabc,lamabcpp,lamdamper,iabc,idamper,idc,vdc,vc,torque,qrm,phit,BY,B
T,BTT,nrconverge,saturate,BIRON] =
wrsmdynamics(parx,pars,turns,damperdata,mudata,0);
    SL = parx(3);
    SPPPP = SL/RP/3;
    SPT     = parx(2);
    NRrtrt  = parx(27);
    damper_nshank = damperdata.damper_nshank; % Number of damper
windings on rotor shank
    BRY = abs(BIRON(SPPPP*9+[1 3+damper_nshank 4+damper_nshank],:));
    BRTSH = abs(BIRON(SPPPP*9+2,:));
    BRT = abs(BIRON(SPPPP*9+4+damper_nshank+[1:(SPT - 2*NRrtrt) (2*SPT
- 4*NRrtrt)+1:(2*SPT - 4*NRrtrt)+2*NRrtrt],:));
    %-----------------------------------------------------
    % Constraint 2: Newton-Raphson Nonlinear Solver Converges &
Operation meets flux density constraint
    constraints(2) = nrconverge & min(saturate);
    %     constraints(2) = 1;
    if constraints(2) == 1
        %-------------------------------------------------------------
        % Constraint 3: Avarage torque be negative.
        Te_avg = tool_avg(torque,1,synfreq,DT); % Compute average
torque
        constraints(3) = lessthan(Te_avg,-(0.7*Pmin/wrm),0.1*Pmin/wrm);
        if constraints(3) == 1;
            %---------------------------------------------------------
            % Constraint 4: Voltage is above minimum allowed value, vdc
is actually Vas_rated.
            vdcmax = parx(25);
            % Calculaion of current, voltage rms, avg
            irms = tools('tool_rms',iabc(1,:),1,synfreq,DT);
            vrms = tools('tool_rms',vabc(1,:),1,synfreq,DT);
            constraints(4) = lessthan(vrms,vdcmax/sqrt(6),0.01*vdcmax);
%             V_error = abs(vrms-vdcmax)/vdcmax;
%             constraints(4) =
lessthan(V_error,0.01*vdcmax,0.001*vdcmax);
%             constraints(4) = 1;
            %---------------------------------------------------------
            % Constraint 5: Minimum power output met.
            % WEIGHT CALCULATION
```

```
            [wstt,wst,wsy,wrt,wrsh,wry,wsw,wrw,weight] =
get_mass(pars,parx,turns,damperdata);
            % LOSS CALCULATION
            DENS    = pars(37);
            GLS     = pars(3);
            clBTT   =
coreloss(BTT(1,:),synfreq,DT,mudata.s)*wstt/DENS*1000;
            clBT    =
coreloss(BT(1,:),synfreq,DT,mudata.s)*wst/DENS*1000;
            clBY    =
coreloss(BY(1,:),synfreq,DT,mudata.s)*wsy/DENS*1000;
            clWRT   =
coreloss(sum(BRT,1)/SPT,synfreq,DT,mudata.s)*wrt/DENS*1000;
            clWRSH  =
coreloss(BRTSH,synfreq,DT,mudata.s)*wrsh/DENS*1000;
            clWRY   =
coreloss(BRY(1,:),synfreq,DT,mudata.s)*wry/DENS*1000;
            core_losses = clBTT+clBT+clBY+clWRT+clWRSH+clWRY;
            resistive_losses    = parx(1)*rs*irms^2 +
rfld*mean(ifld)*mean(ifld);
            conduction_losses   = parx(20)*(irms*sqrt(2)*2/pi)*parx(1);
            damper_losses       = calc_dploss(idamper, damperdata,
pars, parx);
            total_losses  = resistive_losses + core_losses +
damper_losses + conduction_losses;
            Pelec = abs(Te_avg*wrm) - total_losses;
            constraints(5) = greaterthan(Pelec,Pmin,0.1*Pmin);
            %         P_error = abs(Pelec-Pmin)/Pmin;
            %         constraints(5) =
lessthan(P_error,0.01*Pmin,0.001*Pmin);
            %         constraints(5) = 1;
            %-------------------------------------------------------------
            % Constraint 6: Stator Current Density less than max.
            B1          = pars(10);
            BS          = pars(12);
            Ncond       = max(turns);
            H3          = pars(8);
            slotarea    = (0.5*(B1+BS))*H3;
            pfs         = pars(48);
            Js = irms*sqrt(2)*Ncond/(slotarea*pfs);
            Jmax = parx(26);
            constraints(6) = lessthan(Js,Jmax,0.1*Jmax);
            %         constraints(6) = 1;
            %-------------------------------------------------------------
            % Constraint 7: Rotor Current Density less than max.
            HRTSH       = pars(45);
            WCOIL       = pars(51);
            Nfld        = pars(41);
            slotareaf   = WCOIL*HRTSH;
            pfr         = pars(52);
            ifld = pars(47);
            Jr = ifld*Nfld/(slotareaf*pfr);
            constraints(7) = lessthan(Jr,Jmax,0.1*Jmax);
            %         constraints(7) = 1;
            %-------------------------------------------------------------
```

```
            % Constraint 8: power factor above 0.8.
            pf = sign(Qelec)*Pmin/sqrt(Pmin^2+Qelec^2);
            pf_error = abs(pf-0.8);
            constraints(8) = lessthan(pf_error,0.01*0.8,0.001*0.8);
            %         constraints(8) = lessthan(pf,0.8,0.1*0.8);
                    constraints(8) = 1;
        end
    end
end
%-------------------------------------------------------------------------
% FITNESS EVALUATION:
%-------------------------------------------------------------------------
cmin = min(constraints); % Minimum value of the constraint variables.
Value of 1 indicates that the constraint is met.
if cmin < 1
    fitness = (sum(constraints) - 1e12*nc)*[1;1];
else
    fitness = [-total_losses;-weight];
end
%-------------------------------------------------------------------------
if nargin>2
    disp('Geometric Parameters')
    disp(['Shaft Diameter (SD): ',num2str(1e3*param.SD),'mm'])
    disp(['Depth of Rotor Core (DRC): ',num2str(1e3*GAP(1)),'mm'])
    disp(['Core length (GLS): ',num2str(1e2*GAP(2)),'cm'])
    disp(['Height of Rotor Tooth (HRT): ',num2str(1e3*GAP(3)),'mm'])
    disp(['Airgap Length (G1): ',num2str(1e3*GAP(4)),'mm'])
    disp(['Height of Stator Tooth (HST): ',num2str(1e3*GAP(5)),'mm'])
    disp(['Depth of Stator Yoke (DBS): ',num2str(1e3*GAP(6)),'mm'])
    disp(['Width of Stator Tooth Shank (STW):
',num2str(1e3*pars(20)),'mm'])
    disp(['Height of Rotor Tooth Tip Side (HRTT):
',num2str(1e3*pars(44)),'mm'])
    disp(['Chord Length of Rotor Tooth Tip (WRTchord):
',num2str(1e3*pars(55)),'mm'])
    disp(['Chord Width of Rotor Tooth Shank (WRTSHchord):
',num2str(1e3*pars(56)),'mm'])
    disp(['Stator Turns (Ns): ',num2str(GAP(11))])
    disp(['Field Turns (Nfld): ',num2str(GAP(12))])
    disp(['Pole Pairs (Pp): ',num2str(GAP(14))])
    disp(['Width of Stator Tooth Tip (STTW):
',num2str(1e3*pars(21)),'mm'])
    disp(['Height of Stator Tooth Tip (STTW):
',num2str(1e3*pars(58)),'mm'])
    disp(['Number of Damper bars on Rotor Tip (damper_ntip):
',num2str(damperdata.damper_ntip)])
    fprintf('Radius of Damper bars on Rotor Tip (damper_rtip): %f
mm.\n',1e3*damperdata.damper_rtip);
    disp(['Number of Damper bars on Rotor Shank (damper_nshank):
',num2str(damperdata.damper_nshank)])
    disp(['Radius of Damper bars on Rotor Shank (damper_rshank):
',num2str(1e3*damperdata.damper_rshank),'mm'])
    disp('Electric Parameters')
    disp(['Phase Current RMS: ',num2str(irms),'A'])
```

```
    disp(['Phase Current Angle: ',num2str(atan2(-
mean(idsr),mean(iqsr))*180/pi),'deg'])
    disp(['Phase Voltage RMS: ',num2str(vrms),'V'])
    disp(['Phase Voltage Angle: ',num2str(atan2(-
mean(vdsr),mean(vqsr))*180/pi),'deg'])
    disp(['Field Current: ',num2str(ifld),'A'])
    disp(['Stator Current Density: ',num2str(Js),'A/m^2'])
    disp(['Rotor Current Density: ',num2str(Jr),'A/m^2'])
    disp(['Output Power: ',num2str(Pelec/1000),'kW'])
    disp(['Reactive Power: ',num2str(Qelec/1000),'kVA'])
    disp(['Electromagnetic Torque: ',num2str(Te_avg),'Nm'])
    disp(['Total Loss: ',num2str(total_losses),'W'])
    disp(['Efficiency: ',num2str(Pelec/abs(Te_avg*wrm))])
    disp('Losses')
    disp(['Resistive Loss: ',num2str(resistive_losses),'W'])
    disp(['Core Loss: ',num2str(core_losses),'W'])
    disp(['Conduction Loss: ',num2str(conduction_losses),'W'])
    disp(['Damper Loss: ',num2str(damper_losses),'W'])
    disp('Mass')
    disp(['Stator Mass: ',num2str(wsy+wst+wstt),'kg'])
    disp(['Rotor Mass: ',num2str(wry+wrt+wrsh),'kg'])
    disp(['Copper Mass: ',num2str(wsw+wrw),'kg'])
    disp(['Total Machine Mass: ',num2str(weight),'kg'])

    plotwrsm(pars,parx,damperdata,0,varnum);
end
% greaterthan and lessthan functions used to compute constraint values.
function c = greaterthan(x,xmin,deltax)
if x > xmin
    c = 1;
else
    c = 1/(1+abs((xmin-x)/deltax));
end

    function c = lessthan(x,xmax,deltax)
        if x < xmax
            c = 1;
        else
            c = 1/(1+abs((x-xmax)/deltax));
        end
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% April 1, 2012
%-------------------------------------------------------------------------
% M-FILE: wrsm_model.m
%
% Intializes MEC simulation variables, solves the MEC
% system of equations and plots results.
%-------------------------------------------------------------------------
clear all
close all
clc
% ---------------------------------------------------------
% % EVALUATE A MACHINE FROM MULTI-OBJECTIVE DESIGN RESULTS
% fdi = input('Which design would you like to evaluate: ');
% filename = input('Filename of the saved data: ');
% % Load design results and process genes
load('init_test.mat')
[parx,pars,turns,damperdata,mudata] = design_params(final_design(:,1));
% ---------------------------------------------------------
fprintf('********* Dynamic Mesh Based MEC Model **********\n')
% SIMULATION TIME AND PARAMETERS
NCYC    = parx(6);                  % Number of cycles
DT      = parx(12);                 % Time step in s
iter    = parx(30);                 % Number of iterations
wrm     = parx(4)*2*pi/60;          % Mechanical rotor speed in rad/s
wr      = (pars(28)/2)*wrm;         % Electrical rotor speed in rad/s
rs      = pars(23);                 % Phase resistance in ohm
rfld    = pars(43);                 % Field resistance in ohms
ifld    = pars(47);                 % Field current (A)
synfreq   = (pars(28)/2)*parx(4)/60;  % Frequency of vas,vbs,vcs -
(assumed to be synchronized with rotor speed)
damper_ntip = damperdata.damper_ntip; % Number of damper windings on
rotor tip
damper_nshank = damperdata.damper_nshank; % Number of damper windings
on rotor shank
Rd = damperdata.Rd; % Resistance of damper windings on rotor tip
Re = damperdata.Re; % Resistance of damper windings connection

% DYNAMICS DESCRIPTION
qr_init = 0;
[t,vabc,lamabcpp,lamdamper,iabc,idamper,idc,vdc,vc,torque,qrm,phit,BY,B
T,BTT,nrconverge,saturate,BIRON] =
wrsmdynamics(parx,pars,turns,damperdata,mudata,qr_init);

% POST-PROCESSING
qrmdeg = qrm*180/pi;
RP = pars(28);
```

```
SL = parx(3);
SPPPP = SL/RP/3;
SPT     = parx(2);
NRrtrt  = parx(27);
BRY = abs(BIRON(SPPPP*9+[1 3+damper_nshank 4+damper_nshank],1:iter));
BRTSH = abs(BIRON(SPPPP*9+2,1:iter));
BRT = abs(BIRON(SPPPP*9+4+damper_nshank+[1:(SPT - 2*NRrtrt) (2*SPT -
4*NRrtrt)+1:(2*SPT - 4*NRrtrt)+2*NRrtrt],1:iter));
if wrm > 0
    ias = iabc(1,:);
    ibs = iabc(2,:);
    ics = iabc(3,:);
    wrsmpostprocess;
    % CALCULATING AVERAGE AND RIPPLE TORQUE
    [Te_rms,Te_avg,Te_rip] = tools('tool_all',torque,1,synfreq,DT);
    % Calculaion of current, voltage rms
    irms = tools('tool_rms',iabc(1,:),1,synfreq,DT);
    vrms = tools('tool_rms',vabc(1,:),1,synfreq,DT);
    % Calculate current density in a slot
    B1          = pars(10);
    BS          = pars(12);
    Ncond       = max(turns);
    H3          = pars(8);
    slotarea    = (0.5*(B1+BS))*H3;
    pfs         = pars(48);
    Ac          = slotarea*pfs/Ncond;
    Js          = irms*sqrt(2)/Ac;
    fprintf('The current density in a stator slot is %f
A/mm^2.\n',Js*1e-6);
    HRTSH       = pars(45);
    WCOIL       = pars(51);
    Nfld        = pars(41);
    slotareaf   = WCOIL*HRTSH;
    pfr         = pars(52);
    Acfld       = slotareaf*pfr/Nfld;
    Jr          = ifld/Acfld;
    fprintf('The current density in a field slot is %f
A/mm^2.\n',Jr*1e-6);
    % WEIGHT CALCULATION
    [wstt,wst,wsy,wrt,wrsh,wry,wsw,wrw,weight] =
get_mass(pars,parx,turns,damperdata);
    msg = sprintf('Stator Mass = %f kg',wsy+wst+wstt); disp(msg);
    msg = sprintf('Rotor Mass = %f kg',wry+wrt+wrsh); disp(msg);
    msg = sprintf('Copper Mass = %f kg',wsw+wrw); disp(msg);
    msg = sprintf('Total Machine Mass = %f kg',weight); disp(msg);
    % LOSS CALCULATION
    DENS    = pars(37);
    GLS     = pars(3);
    clBTT   = coreloss(BTT(1,:),synfreq,DT,mudata.s)*wstt/DENS*1000;
    clBT    = coreloss(BT(1,:),synfreq,DT,mudata.s)*wst/DENS*1000;
    clBY    = coreloss(BY(1,:),synfreq,DT,mudata.s)*wsy/DENS*1000;
    clWRT   =
coreloss(sum(BRT,1)/SPT,synfreq,DT,mudata.s)*wrt/DENS*1000;
    clWRSH  = coreloss(BRTSH,synfreq,DT,mudata.s)*wrsh/DENS*1000;
    clWRY   = coreloss(BRY(1,:),synfreq,DT,mudata.s)*wry/DENS*1000;
```

```
    core_losses = clBTT+clBT+clBY+clWRT+clWRSH+clWRY;
    resistive_losses    = parx(1)*rs*irms^2 +
rfld*mean(ifld)*mean(ifld);
    conduction_losses   = parx(20)*(irms*sqrt(2)*2/pi)*parx(1);
    damper_losses       = calc_dploss(idamper, damperdata, pars, parx);
    total_losses  = resistive_losses + core_losses + damper_losses +
303;
    % Input mechanical torque calculation
    Te_mech = sign(Te_avg)*(abs(Te_avg*wrm)+core_losses+303)/wrm;
    % OUTPUT INFO TO COMMAND WINDOW
    fprintf('Current: %f A\n', irms);
    fprintf('Voltage: %f V\n', vrms*sqrt(3));
    fprintf('Output power: %f kW\n', (abs(Te_avg*wrm)-resistive_losses-
damper_losses)/1000);
    fprintf('Mechanical torque: %f Nm\n', Te_mech);
    fprintf('Electrical torque: %f Nm\n', Te_avg);
    fprintf('Torque ripple: %f Nm\n\n', Te_rip);
    fprintf('The resistive loss is %f W\n',resistive_losses);
    fprintf('Core loss in the teeth: %f W\n', clBT+clBTT);
    fprintf('Core loss in the yoke: %f W\n', clBY);
    fprintf('The core loss is %f W\n',core_losses);
    fprintf('The damper loss is %f W\n',damper_losses);
    fprintf('The conduction loss is %f W\n',conduction_losses);
    fprintf('The total loss is %f W\n',total_losses);
    fprintf('The machine efficiency is %f\n\n',(abs(Te_mech*wrm)-
total_losses)/abs(Te_mech*wrm));
    fprintf('Max stator yoke flux density: %f T\n', max(max(BY)));
    fprintf('Max stator tooth flux density: %f T\n', max(max(BT)));
    fprintf('Max rotor yoke flux density: %f T\n', max(max(BRY)));
    fprintf('Max rotor shank flux density: %f T\n', max(max(BRTSH)));
    fprintf('Max flux density: %f T\n', max(max(abs(BIRON))));
end

% PLOT RESULTS
if wrm > 0
    xax = qrmdeg; % xaxis value
else
    xax = linspace(0,DT*(iter-1),iter);
end
figure(1)
box on
hold on
plot(t,iabc(1,:),'b');
plot(t,iabc(2,:),'r');
plot(t,iabc(3,:),'g');
plot(t,ifld,'c');
set(gca,'FontName','Times New Roman')
set(gca,'FontSize',12)
title('Phase Currents');
xlabel('Time (s)')
ylabel('Current (A)')
figure(2)
box on
hold on
plot(t,torque(1:iter),'b');
```

```
set(gca,'XLim',[t(1) t(iter)])
set(gca,'FontName','Times New Roman')
set(gca,'FontSize',12)
title('Torque');
xlabel('Time (s)')
ylabel('Torque (Nm)')
figure(3)
hold on
box on
plot(t,-(BY(1,1:iter)),'b');
plot(t,-(BY(2,1:iter)),'r');
set(gca,'FontName','Times New Roman')
set(gca,'FontSize',12)
xlabel('Time (s)')
title('Stator Yoke Section Flux Density');
ylabel('Flux Density (T)')
figure(4)
hold on
box on
plot(t,(BT(1,1:iter)),'b');
plot(t,(BT(2,1:iter)),'b');
plot(t,(BT(3,1:iter)),'r');
set(gca,'FontName','Times New Roman')
set(gca,'FontSize',12)
title('Stator Tooth Flux Density');
xlabel('Time (s)')
ylabel('Flux Density (T)')
figure(5)
hold on
box on
plot(t,(BRY(1,1:iter)),'r');
plot(t,(BRY(2,1:iter)),'b');
plot(t,(BRY(3,1:iter)),'b');
set(gca,'FontName','Times New Roman')
set(gca,'FontSize',12)
title('Rotor Yoke Flux Density');
xlabel('Time (s)')
ylabel('Flux Density (T)')
figure(6)
hold on
box on
plot(t,BRTSH(1,1:iter),'b');
set(gca,'FontName','Times New Roman')
set(gca,'FontSize',12)
title('Rotor Tooth Shank Flux Density');
xlabel('Time (s)')
ylabel('Flux Density (T)')
figure(7)
box on
hold on
plot(t,vabc(1,:),'b');
plot(t,vabc(2,:),'r');
plot(t,vabc(3,:),'g');
set(gca,'XLim',[t(1) t(iter)])
set(gca,'FontName','Times New Roman')
```

```
set(gca,'FontSize',12)
title('Phase and field voltage');
xlabel('Time (s)')
ylabel('Voltage (V)')
figure(8)
box on
hold on
plot(t,lamabcpp(1,:),'b');
plot(t,lamabcpp(2,:),'r');
plot(t,lamabcpp(3,:),'g');
set(gca,'XLim',[t(1) t(iter)])
set(gca,'FontName','Times New Roman')
set(gca,'FontSize',12)
title('Phase and field flux linkage');
xlabel('Time (s)')
ylabel('Flux linkage (Vs)')
plotwrsm(pars,parx,damperdata,0,9);
figure(10)
hold on
% for i = 1:damper_ntip
%     plot(t,idamper(i,:))
% end
plot(t,idamper(1,:),'b');
plot(t,idamper(2,:),'r');
plot(t,idamper(3,:),'g');
plot(t,idamper(4,:),'c');
plot(t,idamper(5,:),'m');
set(gca,'FontName','Times New Roman')
set(gca,'FontSize',12)
title('Damper Winding Currents');
xlabel('Time (s)')
ylabel('Current (A)')
legend('1','2','3','4','5')
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% April 1, 2012
%-------------------------------------------------------------------------
% [parx,pars,turns,damperdata,matdata] = design_params(design)
%
% Creates a vector of machine/simulation parameters for a given machine
% using design variables.
%
% OUTPUTS: pars       - geometric parameters
%          parx       - simulation parameters
%          turns      - conductor turns
%          damperdata - damper properties
%          matdata    - magnetic material properties
%
% INPUTS:  design     - vector of genes from machine optimization
%-------------------------------------------------------------------------
function [parx,pars,turns,damperdata,matdata] = design_params(design)
% USER DEFINED MACHINE PARAMETERS --------->
%-------------------------------------------------------------------------
% MEC Simulation Data
%-------------------------------------------------------------------------
NPH     = 3;                % NUMBER OF PHASES
damper_rtip_1 = design(20); % Radius of damper windings on rotor tip
damper_rtip_2 = design(21); % Radius of damper windings on rotor tip
damper_rshank = design(22); % Radius of damper windings on rotor shank
damper_ntip = design(23);   % Number of damper windings on rotor tip
damper_nshank = design(24); % Number of damper windings on rotor shank
damper_dtip = design(25); % Ratio of the depth of dampers on rotor tip
bartype = design(26); % Bartype: 0-no connection, 1-connent within
poles, 2-connect between poles
%-------------------------------------------------------------------------
% Rotor section division & Damper windings distribution
% And this is a "mirror half" vector, for example
%
%  (rotor sections)
%   ---------------------------------------------------
%  |      |      |      |      |      |      |      |
%   ---------------------------------------------------
%                          [ rdp1      rdp2   rdp3   rdp4 ...]
%
switch damper_ntip
    case 0
        damper_rtip = zeros(4,1);
    case 1
        damper_rtip = [damper_rtip_1 0 0 0]';
    case 2
        damper_rtip = [0 damper_rtip_1 0 0]';
```

```
    case 3
        damper_rtip = [damper_rtip_1 0 damper_rtip_1 0]';
    case 4
        damper_rtip = [0 damper_rtip_1 0 damper_rtip_2 0]';
    case 5
        damper_rtip = [damper_rtip_1 damper_rtip_1 0 damper_rtip_2 0
0]';
    otherwise
        damper_rtip = [damper_rtip_1*mod(damper_ntip,2) damper_rtip_1
damper_rtip_2 damper_rtip_2*ones(1,floor((damper_ntip+2)/2)-3)]';
end
SPT  = 2*length(damper_rtip);
%-------------------------------------------------------------------------

SPAIR   = 3;                % SECTIONS PER HALF THE ROTOR "SLOT"
SLL     = 3*design(17)*3*2; % NUMBER OF STATOR SLOTS (change to
correspond with poles)
RP      = design(17)*2;     % NUMBER OF POLES
vfreq   = design(27);       % Input voltage frequency
WRMRPM  = 1800;             % MECHANICAL ROTOR SPEED, RPM
% If WRMRPM==0, then the system is in SSFR mode
if WRMRPM == 0
    ONECYC  = 1/vfreq;      % ONE PERIOD, s
else
    ONECYC  = 1./(WRMRPM/60*RP/2); % ONE PERIOD, s
end
NCYC    = design(28);       % NUMBER OF ELECTRICAL CYCLES TO SIMULATE
NPTS    = design(29);       % NUMBER OF DATA POINTS PER CYCLE
TSTART  = 0;                % INITIAL TIME, s
TSTOP   = NCYC*ONECYC;      % FINAL TIME, s
ITER    = NCYC*NPTS+1;      % NUMBER OF ITERATIONS
DT      = ONECYC/NPTS;      % TIME STEP, s
ALPHAX  = 1;                % CONVERGENCE FACTOR FOR NEWTON-RAPHSON
MAXIT   = 50;               % MAXIMUM NUMBER OF ITERATIONS
VDROP   = 2;                % FORWARD SWITCH AND DIODE DROP, V
scl1    = 1e3;              % scaling factor for stator windings
scl2    = 1e1;              % scaling factor for field windings
DALPHA  = 0.442307;         % Rectifier Parameters
DBETA   = 2.352236;
%-------------------------------------------------------------------------
% Stator Input Data
%-------------------------------------------------------------------------
SLTINS  = 0;                % SLOT INSULATION WIDTH, m
ESC     = 2.5e-2;           % ARMATURE WINDING EXTENSION BEYOND STACK,
M
%-------------------------------------------------------------------------
% Rotor Input Data
%-------------------------------------------------------------------------
SHDENS  = 0;                % SHAFT DENSITY:
%-------------------------------------------------------------------------
% Parameters calculated from the design vector
% ------------------------------------------------------------------------
SD      = design(1);     % ROTOR SHAFT DIAMETER
DRC     = design(2);     % DEPTH OF THE ROTOR CORE
HRT     = design(4);     % HEIGHT OF THE ROTOR TOOTH
```

```
G1      = design(5);       % MAIN AIR GAP LENGTH, m
HST     = design(6);       % HEIGHT OF THE STATOR TOOTH
DBS     = design(7);       % STATOR YOKE DEPTH, m
OD      = SD+(DRC+HRT+G1+HST+DBS)*2 ; % STATOR OUTER DIAMETER, m
ROD     = SD+(DRC+HRT)*2;  % ROTOR OUTER DIAMETER, m
GLS     = design(3);       % STATOR STACK LENGTH, m
ID      = ROD + 2*G1;      % STATOR INNER DIAMETER, m
STTW = (ID/2)*(2*pi/SLL)*(1-design(8)); % WIDTH OF STATOR TOOTH TIP, m
tipw    = STTW*design(18); % width of stator tooth tip side
tiph    = HST*design(19);  % height of stator tooth tip
STW     = STTW-2*tipw;     % STATOR TOOTH SHANK WIDTH, m
B0      = (ID/2)*(2*pi/SLL)*design(8); % STATOR SLOT DIMENSION, m
fHRTT   = design(9); % VALUE TO DETERMINE HEIGHT OF ROTOR TOOTH TIP
RPIT    = design(10); % ROTOR POLE PITCH COEFFICIENT
fWRTSH  = design(11); % VALUE TO DETERMINE WIDTH OF ROTOR TOOTH SHANK
CL      = GLS;             % ROTOR CORE LENGTH, m
GLP     = GLS;             % LENGTH OF ROTOR POLE, m

% TURNS VECTOR - DEPENDS ON SLOTS PER POLE PER PHASE
SPPPP = SLL/RP/NPH;
if SPPPP == 1
    Npmax = round(design(12));
    Nphase  = [0 Npmax  0];
elseif SPPPP == 2
    Npmax = round(design(12));
    Nphase = [0 0 Npmax Npmax 0 0];
elseif SPPPP == 3
    Npmax = round(design(12));
    Nphase = [Npmax Npmax Npmax Npmax Npmax Npmax 0 0 0];
elseif SPPPP == 4
    Npmax = round(design(12));
    Nphase = [0 0 0 0 Npmax Npmax Npmax Npmax 0 0 0 0];
elseif rem(SPPPP,1)~=0
    error('There must be an integer number of slots per pole per
phase.')
else
    error('Number of stator slots per pole per phase is unaccounted
for.')
end
frms    = design(14);   % RMS Stator Voltage or current
fph     = design(15);   % Phase of stator voltage or current, degrees
ffld    = design(16);   % FIELD CURRENT or voltage
Nfld    = round(design(13));   % FIELD TURNS
%-----------------------------------------------------------------
% STATOR TOOTH DIMENSIONS
fH3     = 0.95;   % FRACTION OF SLOT HEIGHT OCCUPIED BY WDG
H0      = (OD/2 - DBS - SLTINS - ID/2)*(1-fH3);  % STATOR SLOT HEIGHT
NOT OCCUPIED BY WDG, m
H1      = 0;      % STATOR SLOT HEIGHT DIMENSION, m
H2      = 0;      % STATOR SLOT HEIGHT DIMENSION, m
H3      = (OD/2 - DBS - SLTINS - ID/2)*fH3;      % STATOR SLOT HEIGHT
DIMENSION, m
B1      = (2*pi/SLL)*(ID/2 + H0 + H1) - STW;     % STATOR SLOT WIDTH
DIMENSION, m
```

```
B2      = (2*pi/SLL)*(ID/2 + H0 + H1 + H2) - STW; % STATOR SLOT WIDTH
DIMENSION, m
BS      = (2*pi/SLL)*(ID/2 + H0 + H1 + H2 + H3) - STW; % STATOR SLOT
WIDTH DIMENSION, m
% TURNS
turns = Nphase;
winding = abs(cumsum(turns) - 0.5*sum(turns));
% ROTOR DIMENSIONS
WRTang  = 2*pi*RPIT/RP; % ANGLE AT OUTSIDE EDGE OF ROTOR TOOTH TIP
WRTchord= 2*(ROD/2)*sin(0.5*WRTang); % CHORD LENGTH OF ROTOR TOOTH TIP
WRT     = WRTang*ROD/2; % WIDTH OF ROTOR TOOTH (arc length)
WAIRT   = pi*ROD/RP - WRT; % WIDTH OF AIR BETWEEN ROTOR TEETH (arc
length)
WRTS    = WRT/SPT; % WIDTH OF ROTOR TOOTH SECTION (arc length)
DC      = SD + DRC*2; % ROTOR CORE DIAMETER
WRTSHchord= fWRTSH*WRTchord; % CHORD WIDTH OF ROTOR TOOTH SHANK
yRT     = ROD/2*cos(0.5*WRTang); % VERTICAL HEIGHT TO TOP OF TOOTH TIP
SIDE
yRC     = 0.5*sqrt(DC^2-WRTSHchord^2);  % VERTICAL HEIGHT TO BOTTOM OF
ROTOR TOOTH SHANK SIDE
HRTT    = fHRTT*(yRT-yRC); % VERTICAL HEIGHT OF ROTOR TOOTH TIP SIDE
HRTSH   = (yRT-yRC)*(1-fHRTT); % VERTICAL HEIGHT OF ROTOR TOOTH SHANK
WRTSHang= 2*atan(WRTSHchord/(2*(HRTSH+yRC)));% ANGLE OF ROTOR TOOTH
SHANK AT INSIDE OF ROTOR TOOTH TIP
WRTSHrad= (HRTSH+yRC)/(cos(0.5*WRTSHang)); % RADIUS AT TOP OF ROTOR
TOOTH SHANK
WRTSH   = WRTSHrad*WRTSHang; % WIDTH OF ROTOR TOOTH SHANK (arc length)
WCOILout= (WRTchord-WRTSHchord)/2; % WIDTH OF FIELD COIL AT OUTER EDGE
WCOILin = (pi*DC/RP - WRTSH)/2; % APPROXIMATE WIDTH OF FIELD COIL AT
INNER EDGE
WCOIL   = 0.5*(WCOILout+WCOILin); % AVERAGE WIDTH OF AVAILABLE SPACE
FOR THE FIELD COIL
% ------------------------------------------------------------
% Determination of the number of tangential rotor teeth permeances
(NRrtrt)
ytmid = sqrt((ROD/2)^2-(0.5*WRTSHchord).^2);
lR = (WRTchord-WRTSHchord)/2+min(0.5*(ytmid-(yRT-
HRTT)),0.25*WRTSHchord);
Nsect = lR/(WRTchord/SPT);
NRrtrt = round(Nsect)*(Nsect-floor(Nsect)~=0.5) + floor(Nsect)*(Nsect-
floor(Nsect)==0.5);
NRrtrt = NRrtrt - 1*(2*NRrtrt==SPT);
% CROSS-SECTIONAL AREA OF CONDUCTOR IN THE STATOR AND ROTOR
Ncond       = max(turns);
slotarea    = (0.5*(B1+BS))*H3; % Approximate slot as trapezoid
slotareaf   = WCOIL*HRTSH;
Ac          = 2*1.0403e-6;              % Wire gauge #17, 2 conductors
Acfld       = 2.0865e-6;                % Wire gauge #14
pfs         = 2*Ncond*Ac/slotarea; % STATOR CONDUCTOR PACKING FACTOR
pfr         = Nfld*Acfld/slotareaf; % ROTOR CONDUCTOR PACKING FACTOR
% pfs         = 0.5;      % STATOR CONDUCTOR PACKING FACTOR
% pfr         = 0.6;      % ROTOR CONDUCTOR PACKING FACTOR
% Ac          = slotarea*pfs/Ncond;
% Acfld       = slotareaf*pfr/Nfld;
%-----------------------------------------------------------------------
```

```
% Conductor Characteristics
%-------------------------------------------------------------------------
WIREDENS    = 8900;                     % DENSITY, kg/m^3
sigmac      = 58e6;                     % CONDUCTIVITY of copper
sigalu      = 35e6;                     % Conductivity of aluminium
% WIRE CHARACTERISTICS
SR      = 1/(sigmac*Ac);
RR      = 1/(sigmac*Acfld);
% LENGTH OF STATOR CONDUCTOR AND STATOR RESISTANCE
DZ      = ID + 2*(H0+H1);
DW      = 0.5*(OD-DZ) - SLTINS - DBS;
lslot   = GLS + 2*ESC;
lend    = (2*pi/SLL)*(DZ/2 + DW/2);
lcond   = sum(turns)*lslot*RP + 2*sum(winding)*lend*RP;
RS      = lcond*SR;
% LENGTH OF ROTOR CONDUCTOR AND FIELD RESISTANCE
lcondfld = 2*(GLP + WRTSH + WCOIL*pi/2)*Nfld*RP;
Rfld = lcondfld*RR;

% -------------------------------------------------------------------------
% Resistance of the rotor tooth tip damper bar body
CL_dp = CL; % Damper bar length with extended portion
dp_pos = find(damper_rtip);
Rd_r1 = damper_rtip(dp_pos);
Rd_r2 = flipdim(Rd_r1,1);
if damper_ntip == 0
    Rd_r = [];
elseif dp_pos(1) == 1
    Rd_r = [Rd_r2(1:end-1);Rd_r1];
else
    Rd_r = [Rd_r2;Rd_r1];
end
Rd = CL_dp./(sigmac*pi*Rd_r.^2)*(1+0.004041*47.2);

% Resistance of the rotor tooth tip damper bar end connection
switch damper_ntip
    case 0
        Re_ang = [];
    case 1
        Re_ang = 2*(length(damper_rtip)-
dp_pos(end)+1)*WRTang/(SPT+1)+(2*pi/RP-WRTang);
    case 2
        Re_ang = [(2*dp_pos(1)-1)*WRTang/(SPT+1); ...
                  2*(length(damper_rtip)-
dp_pos(end)+1)*WRTang/(SPT+1)+(2*pi/RP-WRTang)];
    otherwise
        if dp_pos(1) == 1
            Re_ang_1 = zeros(length(dp_pos),1);
            for i = 1:length(dp_pos)-1
                Re_ang_1(i) = (dp_pos(i+1)-
dp_pos(i)+0.5*(i==1))*WRTang/(SPT+1);
            end
            Re_ang_1(end) = 2*(length(damper_rtip)-
dp_pos(end)+1)*WRTang/(SPT+1)+(2*pi/RP-WRTang);
            Re_ang_2 = flipdim(Re_ang_1,1);
```

```
            Re_ang = [Re_ang_2(2:end);Re_ang_1];
        else
            Re_ang_1 = zeros(length(dp_pos)+1,1);
            Re_ang_1(1) = (2*dp_pos(1)-1)*WRTang/(SPT+1);
            for i = 1:length(dp_pos)-1
                Re_ang_1(i+1) = (dp_pos(i+1)-dp_pos(i))*WRTang/(SPT+1);
            end
            Re_ang_1(end) = 2*(length(damper_rtip)-
dp_pos(end)+1)*WRTang/(SPT+1)+(2*pi/RP-WRTang);
            Re_ang_2 = flipdim(Re_ang_1,1);
            Re_ang = [Re_ang_2(2:end-1);Re_ang_1];
        end
end
Re_b = 0.1e-3; % Base value
Re = Re_b*Re_ang/min(Re_ang)*(1+0.004041*47.2);

%-------------------------------------------------------------------------
% Additional Simulation and Optimization Parameters
%-------------------------------------------------------------------------
TOL     = 1e-5;         % Convergence tolerance
PTCmin  = 1e-16;  % Minimum allowed airgap permeance to avoid inf Rag
Pmin    = 1e4;          % Constraint on power output
vdcmax  = 480/sqrt(3);  % Constraint on bus voltage
Jmax    = 7.6*1e6;      % Constraint on current density
slopes  = pi/2;   % Slopes to calculate fringing airgap permeances
%-------------------------------------------------------------------------
% Material data
%-------------------------------------------------------------------------
% Kohler
DENS    = 7437.49;     % Kohler
Bsat    = 2.5;         % Kohler
% Stator steel
matdata.s.K = 4;
matdata.s.mur = 5349.922;
matdata.s.a = [0.12542  0.00019835 0.00019835 0.00019835];
matdata.s.b = [13.14573     0.1971988      129.4606     8.358885];
matdata.s.g = [1.6445      0.01      1.4157      0.58577];
matdata.s.d = matdata.s.a./matdata.s.b;
matdata.s.e = matdata.s.b.*matdata.s.g;
matdata.s.z = 1+exp(matdata.s.e);
matdata.s.alpha = 1.0529;
matdata.s.beta = 1.5969;
matdata.s.kh = 0.33143;
matdata.s.ke = 8.2813e-05;
slB      = 3*SLL/RP;
matdata.s.a = ones(slB,1)*matdata.s.a;
matdata.s.b = ones(slB,1)*matdata.s.b;
matdata.s.d = ones(slB,1)*matdata.s.d;
matdata.s.e = ones(slB,1)*matdata.s.e;
matdata.s.z = ones(slB,1)*matdata.s.z;
% Rotor steel
matdata.r.K = 4;
matdata.r.mur = 5349.922;
matdata.r.a = [0.12542  0.00019835 0.00019835 0.00019835];
matdata.r.b = [13.14573     0.1971988      129.4606     8.358885];
```

```
matdata.r.g = [1.6445      0.01      1.4157      0.58577];
matdata.r.d = matdata.r.a./matdata.r.b;
matdata.r.e = matdata.r.b.*matdata.r.g;
matdata.r.z = 1+exp(matdata.r.e);
matdata.r.alpha = 1.0529;
matdata.r.beta = 1.5969;
matdata.r.kh = 0.33143;
matdata.r.ke = 8.2813e-05;
rlB      = 6+SPT+damper_nshank+SPT+(SPT-1);
matdata.r.a = ones(rlB,1)*matdata.r.a;
matdata.r.b = ones(rlB,1)*matdata.r.b;
matdata.r.d = ones(rlB,1)*matdata.r.d;
matdata.r.e = ones(rlB,1)*matdata.r.e;
matdata.r.z = ones(rlB,1)*matdata.r.z;
% ------------------------------------------------------------------------
% % M19 - PROPERTIES FROM PMMT
% DENS    = 7402;         % DENSITY OF M19
% Bsat    = 1.4311;         % Maximum allowed saturation
% % Stator steel
% matdata.s.K = 4;
% matdata.s.mur = 32685.6784;
% matdata.s.a = [0.098611 0.0014823 0.001435 0.001435];
% matdata.s.b = [69.73973 1.949541 162.2767 3.598553];
% matdata.s.g = [1.399 2.1619 1.2475 2.0377];
% matdata.s.d = matdata.s.a./matdata.s.b;
% matdata.s.e = matdata.s.b.*matdata.s.g;
% matdata.s.z = 1+exp(matdata.s.e);
% matdata.s.alpha = 1.338;
% matdata.s.beta = 1.817;
% matdata.s.kh = 0.09294;
% matdata.s.ke = 0.00005044;
% slB      = 3*SLL/RP;
% matdata.s.a = ones(slB,1)*matdata.s.a;
% matdata.s.b = ones(slB,1)*matdata.s.b;
% matdata.s.d = ones(slB,1)*matdata.s.d;
% matdata.s.e = ones(slB,1)*matdata.s.e;
% matdata.s.z = ones(slB,1)*matdata.s.z;
% % Rotor steel
% matdata.r.K = 4;
% matdata.r.mur = 32685.6784;
% matdata.r.a = [0.098611 0.0014823 0.001435 0.001435];
% matdata.r.b = [69.73973 1.949541 162.2767 3.598553];
% matdata.r.g = [1.399 2.1619 1.2475 2.0377];
% matdata.r.d = matdata.r.a./matdata.r.b;
% matdata.r.e = matdata.r.b.*matdata.r.g;
% matdata.r.z = 1+exp(matdata.r.e);
% matdata.r.alpha = 1.338;
% matdata.r.beta = 1.817;
% matdata.r.kh = 0.09294;
% matdata.r.ke = 0.00005044;
% rlB      = 6+SPT+damper_nshank+SPT+(SPT-1);
% matdata.r.a = ones(rlB,1)*matdata.r.a;
% matdata.r.b = ones(rlB,1)*matdata.r.b;
% matdata.r.d = ones(rlB,1)*matdata.r.d;
% matdata.r.e = ones(rlB,1)*matdata.r.e;
```

```
% matdata.r.z = ones(rlB,1)*matdata.r.z;
%
%-----------------------------------------------------------------------
% Damper data
%-----------------------------------------------------------------------
damperdata.Rd = Rd;
damperdata.Re = Re;
damperdata.Rd_r = Rd_r;
damperdata.damper_rtip = damper_rtip;
damperdata.damper_rshank = damper_rshank;
damperdata.damper_ntip = damper_ntip;
damperdata.damper_nshank = damper_nshank;
damperdata.damper_dtip = damper_dtip;
damperdata.bartype = bartype;
%
% PARS - PARAMETER VECTOR, PRIMARILY GEOMETRY
pars     = zeros(1,63);
pars(1)  = OD;
pars(2)  = ID;
pars(3)  = GLS;
pars(4)  = DBS;
pars(5)  = H0;
pars(6)  = H1;
pars(7)  = H2;
pars(8)  = H3;
pars(9)  = B0;
pars(10) = B1;
pars(11) = B2;
pars(12) = BS;
pars(13) = SLTINS;
pars(14) = G1;
pars(15) = 0; % UNUSED
pars(16) = 0; % UNUSED
pars(17) = ESC;
pars(18) = 0; % UNUSED
pars(19) = 0; % UNUSED
pars(20) = STW;
pars(21) = STTW;
pars(22) = 0; % UNUSED
pars(23) = RS;
pars(24) = ROD;
pars(25) = DC;
pars(26) = CL;
pars(27) = GLP;
pars(28) = RP;
pars(29) = SD;
pars(30) = 0; % UNUSED
pars(31) = 0; % UNUSED
pars(32) = RPIT;
pars(33) = HRT;
pars(34) = WRT;
pars(35) = WAIRT;
pars(36) = WRTS;
pars(37) = DENS;
pars(38) = SHDENS;
```

```
pars(39) = WIREDENS;
pars(40) = Ac;
pars(41) = Nfld;
pars(42) = Acfld;
pars(43) = Rfld;
pars(44) = HRTT;
pars(45) = HRTSH;
pars(46) = WRTSH;
pars(47) = ffld;
pars(48) = pfs;
pars(49) = frms;
pars(50) = fph;
pars(51) = WCOIL;
pars(52) = pfr;
pars(53) = 0; % UNUSED
pars(54) = slopes;
pars(55) = WRTchord; % UNUSED
pars(56) = WRTSHchord;
pars(57) = tipw; % Width of stator teeth tip
pars(58) = tiph; % Height of stator teeth tip
% PARX - SIMULATION PARAMETERS
parx     = zeros(1,30);
parx(1)  = NPH;
parx(2)  = SPT;
parx(3)  = SLL;
parx(4)  = WRMRPM;
parx(5)  = vfreq;
parx(6)  = NCYC; % Number of cycles
parx(7)  = 0; % UNUSED
parx(8)  = 0; % UNUSED
parx(9)  = 0; % UNUSED
parx(10) = TSTART;
parx(11) = TSTOP;
parx(12) = DT;
parx(13) = ALPHAX;
parx(14) = MAXIT;
parx(15) = 0; % 1:Delta connection; 0:Wye connection
parx(16) = scl1; % Scaling factor for stator windings
parx(17) = scl2; % Scaling factor for field windings
parx(18) = DALPHA;
parx(19) = DBETA;
parx(20) = VDROP;
parx(21) = TOL;
parx(22) = PTCmin;
parx(23) = Bsat;
parx(24) = Pmin;
parx(25) = vdcmax;
parx(26) = Jmax;
parx(27) = NRrtrt;
parx(28) = 0; % UNUSED
parx(29) = SPAIR;
parx(30) = ITER;
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% Apr 1, 2013
%-------------------------------------------------------------------------
%
[t,vabc,lamabcpp,lamdamper,iabc,idamper,idc,vdc,vc,torque,qrm,phit,BY,B
T,BTT,nrconverge,saturate,BIRON] =
% wrsmdynamics(parx,pars,turns,damperdata,mudata,qr_init)
%
% Solves the Dynamics of the MEC network.
%
% OUTPUTS: t        - time vector (s)
%          vabcs    - phase voltages (V)
%          lamabcpp - phase flux linkage per pole (Vs)
%          lamdamper - damper flux linkage (Vs)
%          iabcs    - phase currents (A)
%          idamper  - damper bar currents (A)
%          idc      - dc bus currents (A)
%          vdc      - dc bus voltage (V)
%          vc       - dc bus capacitor voltage (V)
%          torque   - torque (Nm)
%          qrm      - mechanical rotor position (radians)
%          phit     - stator teeth flux (Wb)
%          BY,BT,BTT    - flux density in the stator yoke, stator
teeth, and stator tooth tips (T)
%          nrconverge - flag indicating if newton raphson converged
%          saturate   - indicates if the flux density limit is violated
%          BIRON    - flux density in iron (Wb)
%
% INPUTS:  pars     - geometric parameters
%          parx     - simulation parameters
%          turns    - phase winding turns (turn count)
%          damperdata   - information of damper bars
%          mudata   - magnetic material data for finding permeability
%          qr_init  - initial rotor position in electric degree
%-------------------------------------------------------------------------
function
[t,vabc,lamabcpp,lamdamper,iabc,idamper,idc,vdc,vc,torque,qrm,phit,BY,B
T,BTT,nrconverge,saturate,BIRON] =
wrsmdynamics(parx,pars,turns,damperdata,mudata,qr_init)
%-------------------------------------------------------------------------
% INITIALIZE THE SYSTEM
%-------------------------------------------------------------------------
mu0 = pi*4e-7;      % Permeability of free space
RP  = pars(28);     % Poles
S   = parx(3)/RP;   % Number of stator slots per pole
D   = 2*(parx(2));  % Number of rotor pole tip sections per pole pair
Dsl = 4*parx(29);   % Number of inter-polar regions per pole pair
```

```
SPT    = parx(2);  % SECTIONS PER ROTOR TOOTH, including radial and
tangential
NRrtrt = parx(27); % Number of outer pole tip reluctances per pole pair
damper_ntip = damperdata.damper_ntip;     % Number of damper windings
on rotor tip
damper_nshank = damperdata.damper_nshank; % Number of damper windings
on rotor shank
bartype = damperdata.bartype; % Type of damper bars connnection
Rd = damperdata.Rd;     % Damper bar body resistance
Re = damperdata.Re;     % Damper bar end connection resistance
Rload   = 22.81;      % Parallel resistance load
Lload   = 0.0807;     % Parallel resistance load
Cload   = 100e-6;       % Filter capacitance
taus    = 0.1;          % Filter time constant
rs      = pars(23);     % Stator windings resistance
wrm     = parx(4)*2*pi/60;          % Mechanical rotor speed in rad/s
wr      = (pars(28)/2)*wrm;
scl     = parx(16);
ifld    = pars(47);                 % Field current (A)
vrms    = pars(49);                 % rms Stator voltage (V)
vphase  = pars(50);                 % Current phase angle (degrees)
vm      = vrms*sqrt(2);             % Magnitude of vas,vbs,vcs
DT      = parx(12);                 % Time step in s
iter    = parx(30);                 % Number of iterations
vdcmax  = parx(25);                 % Maximum dc voltage
% For machine design with RL load producing rated power ---------------
% Vll_rms = 480;
% pf = 0.8;
% P = parx(24);
% Q = sqrt((P/pf)^2-P^2);
% Rload = 3*(Vll_rms/sqrt(3))^2/P;
% Lload = (Vll_rms/sqrt(3))^2/Q/wr;
% ------------------------------------------------------------------------
% INITIALIZE VARIABLES
slB     = 3*S;                      % Number of iron elements in stator
rlB     = 6+D/2+damper_nshank+SPT+(SPT-1); % Number of iron elements in
rotor
lB      = slB+rlB;                  % Number of iron elements
nriter  = zeros(1,iter);            % Keeps track of N-R iterations
torque  = zeros(1,iter);
PTC     = zeros(S,D+Dsl,iter);      % Matrix of airgap permeances
dPTC    = zeros(S,D+Dsl,iter);
phit    = zeros(S,iter);            % Stator tooth flux
phiiron = zeros(lB,iter);           % Flux in iron
BY      = zeros(S,iter);            % Stator yoke flux density
BT      = zeros(S,iter);            % Stator tooth shank flux density
BTT     = zeros(S,iter);            % Stator tooth tip flux density
BIRON   = zeros(lB,iter);           % Flus density in all iron elements
saturate = ones(1,iter);            % Saturation constraint (is Bsat
violated)
smuiron = get_mur_exp(zeros(slB,1),mudata.s); % Initial permeabilities
of stator
rmuiron = get_mur_exp(zeros(rlB,1),mudata.r); % Initial permeabilities
of rotor
muiron  = [smuiron;rmuiron];        % Initial permeabilities
```

```
TOL     = parx(21);                 % tolerance for convergence of
Newton-Raphson
k       = 1;                        % Simulation step
t(k)    = parx(10);
% ARTIFICIAL ROTOR POSITION MODIFICATION used in the calculation of
airgap
% permeances.----
SLL     = parx(3);
ID      = pars(2);
ROD     = pars(24);
STTW    = pars(21);
WRT     = pars(34);
WAIRT   = pars(35);
shift1  = WRT/(ROD/2);
shift2  = (WAIRT/2)/(ROD/2);
shift3  = 2*pi/SLL;
shift4  = (STTW/2)/(ID/2);
shift5  = (pi/2)/(RP/2);
shift   = shift1 + shift2 - (S/2)*shift3 - shift4 - shift5;
% TIME AND ROTOR POSITION VECTORS
t       = (0:DT:DT*(iter-1))+t(k);
qrm     = t*wrm + qr_init/(RP/2);   % Actual rotor position
qrm_shift =  qrm + shift;           % Angle fed to airgap permeance
function
%--------------------------------------------------------------------------
% CALCULATE VARIABLES/MATRICES WHICH WILL NOT CHANGE DURING SIM
%--------------------------------------------------------------------------
% Variables/matrices to be used in airgap permeance calculation
WRS     = pars(35)/(2*parx(29));
WRTS    = pars(36);
B0      = pars(9);
SPT     = parx(2);
RPIT    = pars(32);
WRTSang = 2*pi*RPIT/RP/SPT;
WRTang  = 2*pi*RPIT/RP;
WRSang  = 2*pi*(1-RPIT)/RP/(Dsl/2);
qs      = STTW/ID*RP;               % Span of stator tooth in
electrical radians
qs1     = B0/ID*RP;                 % Span of stator slot
qrr     = WRTSang*RP/2;             % Span of rotor pole tip section
qrs     = WRSang*RP/2;              % Span of inter-polar section
Gmaxrt = pi*4e-7*pars(3)/(ID-
ROD)*2*(WRTS*(STTW>=WRTS)+STTW*(STTW<WRTS)); % if-else
Gmaxsl = pi*4e-7*pars(3)/(ID-ROD)*2*(WRS*(STTW>=WRS)+STTW*(STTW<WRS));
% if-else
rt      = 1:D; rtsl    = 1:Dsl; st      = (1:S)';
% Matrices defining the angle between every stator tooth and rotor
section
anglert = ones(S,1)*(-mod(rt-1,(D/2))*WRTSang - floor((rt-
1)/(D/2))*2*pi/RP)...
    + ((st-1)*(STTW+B0)/(ID/2))*ones(1,D);
anglesl = ones(S,1)*(-WRTang - mod(rtsl-1,(Dsl/2))*WRSang - ...
    floor((rtsl-1)/(Dsl/2))*2*pi/RP) + ((st-
1)*(STTW+B0)/(ID/2))*ones(1,Dsl);
```

```
% Establish the geometric case for the rotor tooth section
if qrr <= qs1/2
    qrrcs = 1;
elseif (qrr <= qs)
    qrrcs = 2;
elseif (qrr <= qs +qs1/2)
    qrrcs = 3;
elseif (qrr <= qs+qs1)
    qrrcs = 4;
else
    qrrcs = 5;
end
% Establish the geometric case for the rotor slot section
if qrs <= qs1/2
    qrscs = 1;
elseif (qrs <= qs)
    qrscs = 2;
elseif (qrs <= qs +qs1/2)
    qrscs = 3;
elseif (qrs <= qs+qs1)
    qrscs = 4;
else
    qrscs = 5;
end

% ---------------------------------------------------------------------
% turns matrix to be used in system of equations
Natrn   = [-turns turns]';
Nbtrn   = [Natrn(2*SLL/(3*RP)+1:end);Natrn(1:2*SLL/(3*RP))];
Nctrn   = [Natrn(4*SLL/(3*RP)+1:end);Natrn(1:4*SLL/(3*RP))];
Nabc    = [Natrn Nbtrn Nctrn];
Nfld    = pars(41);
Nabcf   = [Nabc(1:S,:) zeros(S,1);0 0 0 Nfld;0 0 0 -Nfld];
% ---------------------------------------------------------------------
% MEC loops with MMF sources
Cvcfixed = (1:S+2)';
%---------------------------------------------------------------------
% Calculate the reluctances
[Rxm,areas,Rair,NPRTS,NPRTB] =
get_reluctances(mu0,parx,pars,damperdata);
Riron = Rxm./muiron;
%---------------------------------------------------------------------
% Identify type of node in rotor tooth and slot
% 1 = node of rotor pole tip radial branch
% 2 = node of rotor pole tip tangential branch
% 3 = rotor slot branch going to rotor edge
% 4 = rotor slot branch going to bottom of rotor pole tip
rtid = [2*ones(NRrtrt,1);ones(D/2-2*NRrtrt,1);2*ones(NRrtrt,1);...
        3*ones(NPRTS,1);4*ones(2*NPRTB,1);3*ones(NPRTS,1);...
        2*ones(NRrtrt,1);ones(D/2-2*NRrtrt,1);2*ones(NRrtrt,1);...
        3*ones(NPRTS,1);4*ones(2*NPRTB,1);3*ones(NPRTS,1)];
% Identify how many RRTOUT branches border the rotor loop
NRBRL = ceil((NRrtrt+1)/2);  % Number of RRTOUT branches Bordering
Rotor Loop
```

```
NRTBD = NRrtrt-NRBRL; % Number of RRTOUT branches with bordering loop
To Be Determined
% ------------------------------------------------------------------
% Define reluctance connections in stator and rotor which do not change
% Stator tooth tip, damper slots, and leakage of damper slots are not
% presented here, but will be derived as postprocess in shape_alg.m
% IRON
% Stator yoke - S
% Stator teeth - S
% Rotor yoke below the slot - 1
% Rotor tooth shank - 1
% Rotor yoke connected to shank - 2
% Rotor tooth tips radial - (D - 4*NRrtrt)
% Rotor tooth to rotor tooth tangential - 4*NRrtrt
% Rotor tooth tangential at sides of tooth tips - 4
% AIR
% Stator tooth leakage - S
% Field winding leakage - 2
% Middle rotor slot leakage - 2
% Fringing permeance from rotor side to airgap boundary - Dsl
% Fringing permeance from rotor slot side to bottom of tooth tip - 4
% RY R RRYSL RRTSH RRYSH RRTIN RRTOUT RRTS RSTL RFDL RRTL RAGFR RFRB
Crcfixed = zeros(2*S+8+D+S+3+Dsl,3);
% RY (all)
Crcfixed(1:S,2)=(1:S)';
% R (all)
Crcfixed(S+1:2*S,2) = [1 2:S]';
Crcfixed(S+1:2*S,3) = [-S 1:S-1]';
% RRYSL (all)
Crcfixed(2*S+1,3) = S+3;
% RRTSH (all)
Crcfixed(2*S+2,2:3) = [S+1 S+2];
% RRYSH (all)
Crcfixed(2*S+2+(1:2)',2) = [S+1;S+2];
% RRTIN (Determined by shape algorithm)
% RRTOUT - One side known if reluctance borders rotor loop
Crcfixed(2*S+2+D-4*NRrtrt+2+(1:4*NRrtrt)',2) = ...

[[zeros(NRTBD,1);ones(NRBRL,1)]*(S+1);[ones(NRBRL,1);zeros(NRTBD,1)]*(S
+2);...
 -[zeros(NRTBD,1);ones(NRBRL,1)]*(S+1);-
[ones(NRBRL,1);zeros(NRTBD,1)]*(S+2)];
% RRTS - (Determined by shape algorithm)
% RSTL (one side known, use shape alg for other)
Crcfixed(2*S+2+D+6+(1:S)',2) = (1:S)';
% RFDL (all)
Crcfixed(2*S+2+D+6+S+(1:2)',2:3) = [-(S+3) S+1;S+2 S+3];
% RRTL (one side known, use shape alg for other)
Crcfixed(2*S+2+D+6+S+2+(1:2)',2) = [S+3;-(S+3)];
% RAGFR - (Determined by shape algorithm)
% RFRB (one side, use shape alg for other)
Crcfixed(2*S+2+D+6+S+4+Dsl+(1:4)',2) = [-(S+3);S+3;S+3;-(S+3)];
%------------------------------------------------------------------
% Initialize variables
if parx(15) == 1 %Delta
```

```
    nio = 3;
    mlam = [0 1 0;-1 0 0;0 0 0];
    m_isil = [-1 0 1;1 -1 0;0 1 -1];
    m_vgvs = 1.5*[1 sqrt(3)/3 0;-sqrt(3)/3 1 0;0 0 0];
else %Wye
    nio = 2;
    mlam = [0 1;-1 0];
    m_isil = -eye(3);
    m_vgvs = [1 0 0;0 1 0];
end
iabc  = zeros(3,iter);
lamabcpp = zeros(3,iter);
vqd0sr = zeros(nio,iter);
iqd0sr = zeros(nio,iter);
lamqd0srpp = zeros(nio,iter+1);
plamqd0srpp = zeros(nio,iter);
idamper = zeros(damper_ntip,iter);
lamdamper = zeros(damper_ntip,iter+1);
plamdamper = zeros(damper_ntip,iter);
index_vect = zeros(damper_ntip,3,iter+1);
flag_vect = ones(damper_ntip,iter+1);
il_qd = zeros(2,iter+1);
pil_qd = zeros(2,iter);
vc = ones(1,iter+1)*vdcmax;
pvc = zeros(1,iter);
idc = ones(1,iter+1)*vdcmax/Rload;
vdc = ones(1,iter+1)*vdcmax;
Ivdc = zeros(1,iter+1);
Ivc = zeros(1,iter+1);

% Calculate the voltages for SSFR test
if wrm>0
    vas = vm*cos((RP/2)*(qrm) + (pi*vphase/180));
    vbs = vm*cos((RP/2)*(qrm) + (pi*vphase/180) - (2*pi/3));
    vcs = vm*cos((RP/2)*(qrm) + (pi*vphase/180) - (4*pi/3));
else
    vfreq = parx(5);
    vas = 2/3*vm*cos(2*pi*vfreq*t);
    vbs = -1/3*vm*cos(2*pi*vfreq*t);
    vcs = -1/3*vm*cos(2*pi*vfreq*t);
end
vabc = [vas;vbs;vcs];

% Initial stator flux linkage per pole values
if wrm > 0
    Ksr_prime = (2/3)*[-sin((RP/2)*(qrm(k))) -sin((RP/2)*(qrm(k))-
2*pi/3) -sin((RP/2)*(qrm(k))+2*pi/3);
        cos((RP/2)*(qrm(k))) cos((RP/2)*(qrm(k))-2*pi/3)
cos((RP/2)*(qrm(k))+2*pi/3)];
    lamqd0srpp(1:2,k) = Ksr_prime*vabc(:,k)/wr/RP;
else
    lamqd0srpp(1:2,k) = [0.00;0.001];
end
%-------------------------------------------------------------------------
% Determine transformation matrix for plamdamper
```

```
if bartype == 1
    % Version-1: No end connection resistance ------------------------
    % For example damper_ntip = 5
    % Tdp = [-rb1 rb2 0 0;0 -rb2 rb3 0;0 0 -rb3 rb4;-rb5 -rb5 -rb5 -
rb5-rb4];

    % if damper_ntip == 2
    %     Tdp = -Rd(1)-Rd(2);
    % else
    %     Tdp = -diag(Rd(1:end-1));
    %     for i = 1:damper_ntip-2
    %         Tdp(i,i+1) = Rd(i+1);
    %     end
    %     Tdp(damper_ntip-1,:) = -Rd(damper_ntip)*ones(1,damper_ntip-
1);
    %     Tdp(damper_ntip-1,damper_ntip-1) = Tdp(damper_ntip-
1,damper_ntip-1)-Rd(damper_ntip-1);
    % end

    % Version-2: With end connection resistance ----------------------
    % Tdp = [-rb1-2*re1 rb2 0 0;
    %        -2*re2 -rb2-2*re2 rb3 0;
    %        -2*re3 -2*re3 -rb3-2*re3 rb4;
    %        -rb5-2*re4 -rb5-2*re4 -rb5-2*re4 -rb5-2*re4-rb4];

    % Re = [0.1 0.1 0.1 0.1 0.1]*1e-3;
    if damper_ntip < 2
        Tdp = [];
    elseif damper_ntip == 2
        Tdp = -Rd(1)-Rd(2)-2*Re(1);
    else
        Tdp = -diag(Rd(1:end-1));
        for i = 1:damper_ntip-2
            Tdp(i,i+1) = Rd(i+1);
        end
        for i = 1:damper_ntip-1
            for j = 1:i
                Tdp(i,j) = Tdp(i,j)-2*Re(i);
            end
        end
        Tdp(damper_ntip-1,:) = Tdp(damper_ntip-1,:)-
Rd(damper_ntip)*ones(1,damper_ntip-1);
    end

elseif bartype == 2
    % Version-1: No end connection resistance ------------------------
    % For example damper_ntip = 5
    % Tdp = [-Rd(1) Rd(2) 0 0 0;0 -Rd(2) Rd(3) 0 0;0 0 -Rd(3) Rd(4) 0;0
0 0 -Rd(4) Rd(5);-Rd(1) 0 0 0 -Rd(5)];

    % if damper_ntip == 1
    %     Tdp = -2*Rd(1);
    % else
    %     Tdp = -diag(Rd(1:end));
```

```
    %     for i = 1:damper_ntip-1
    %         Tdp(i,i+1) = Rd(i+1);
    %     end
    %     Tdp(damper_ntip,1) = -Rd(1);
    % end

    % Version-2: With end connection resistance ----------------------
    % Re = [0.1 0.1 0.1 0.1 0.1]*1e-3;
    % Tdp = -[Rd(1)+Re(1) -Rd(2)-Re(1) -Re(1) -Re(1) -Re(1); ...
    %         Re(2) Rd(2)+Re(2) -Rd(3)-Re(2) -Re(2) -Re(2); ...
    %         Re(3) Re(3) Rd(3)+Re(3) -Rd(4)-Re(3) -Re(3); ...
    %         Re(4) Re(4) Re(4) Rd(4)+Re(4) -Rd(5)-Re(4); ...
    %         Rd(1)+Re(5) Re(5) Re(5) Re(5) Rd(5)+Re(5)];
    %
    % Re = [0.1 0.1 0.1 0.1 1]*1e-3;
    if damper_ntip == 0
        Tdp = [];
    elseif damper_ntip == 1
        Tdp = -2*Rd(1)-2*Re(1);
    else
        Tdp = -diag(Rd(1:end));
        for i = 1:damper_ntip
            for j = 1:damper_ntip
                if j <= i
                    Tdp(i,j) = Tdp(i,j)-Re(i);
                else
                    Tdp(i,j) = Tdp(i,j)+Re(i);
                end
            end
        end
        for i = 1:damper_ntip-1
            Tdp(i,i+1) = Tdp(i,i+1)+Rd(i+1);
        end
        Tdp(damper_ntip,1) = Tdp(damper_ntip,1)-Rd(1);
    end
end
%-------------------------------------------------------------------------
% SOLVING LOOP
%-------------------------------------------------------------------------
nrconverge = 1;
while k <= iter
    % AIR-GAP PERMEANCES
    [PTC(:,:,k),dPTC(:,:,k)] =
get_Pag(qrm_shift(k),pars,parx,Gmaxrt,Gmaxsl,anglert,anglesl,qrrcs,qrsc
s);
    % Shape algorithm - Find the loop topology in the airgap if it has
changed
    if k==1 || sum(sum((PTC(:,:,k-1)~=0)~=(PTC(:,:,k)~=0)))>0
        [Crconn,Cvconn,O,PTCind,d_damper_1,d_damper_2,index,flag] =
shape_alg(PTC(:,:,k),parx,pars,damperdata,Crcfixed,Cvcfixed,rtid,index_
vect(:,:,k),flag_vect(:,k));
        if length(Crconn)~=length([Riron;Rair;PTCind])
            nrconverge = 0;
            break
        end
```

```
    end

    % Obtain list of airgap permeances and their derivatives for this
rotor position
    ptc         = PTC(:,:,k)';
    PTClist     = ptc(PTCind);
    dptc        = dPTC(:,:,k)';
    dPTClist    = dptc(PTCind);

    % Using rotor reference frame
    Ksr = (2/3)*[cos((RP/2)*(qrm(k))) cos((RP/2)*(qrm(k))-2*pi/3)
cos((RP/2)*(qrm(k))+2*pi/3);
                 sin((RP/2)*(qrm(k))) sin((RP/2)*(qrm(k))-2*pi/3)
sin((RP/2)*(qrm(k))+2*pi/3);
                 0.5 0.5 0.5];
    Ksrinv = [cos((RP/2)*(qrm(k))) sin((RP/2)*(qrm(k))) 1;
              cos((RP/2)*(qrm(k))-2*pi/3) sin((RP/2)*(qrm(k))-2*pi/3) 1;
              cos((RP/2)*(qrm(k))+2*pi/3) sin((RP/2)*(qrm(k))+2*pi/3)
1];

    % Find the system of equations and solve for the initial guess
    [A,d] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn,Cvconn);

    % ---------------------------------------------------------------------
    if bartype == 0 || (bartype==1 && damper_ntip<2) || (bartype==2 &&
damper_ntip<1)
        Aaug = [A -scl*d(:,1:3)*Ksrinv(:,1:nio) ;
scl*Ksr(1:nio,:)*d(:,1:3)' zeros(nio,nio)];
        daug = [d(:,4) zeros(length(A),nio) ;zeros(nio,1) eye(nio)];
        if rcond(Aaug)<1e-16
            fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug),k);
        end

        % Solve for vector of loop flux and current
        lam = [ifld;scl*lamqd0srpp(:,k)];
        xg = Aaug\(daug*lam);
        % Identify just the loop fluxes
        fluxm = xg(1:end-nio);
        % NEWTON-RAPHSON SOLVER
        it = 1; % Keeps track of N-R iterations
        NRSOLVE = 1;
        while NRSOLVE
            % DETERMINE FLUXES FOR THE GUESS VECTOR xg
            phi = O*fluxm;
            phiiron(:,k) = phi(1:lB);
            % DETERMINE B-FIELDs
            BIRON(:,k) = phiiron(:,k)./areas;
            % GET PERMEABILITY FOR EACH RESPECTIVE PERM
            [sMU,sdmdb] = get_mur_exp(BIRON(1:slB,k),mudata.s);
            [rMU,rdmdb] = get_mur_exp(BIRON(slB+1:end,k),mudata.r);
            MU = [sMU;rMU];
            dmdb = [sdmdb;rdmdb];
```

```
            % UPDATE MATRICIES
            Riron = Rxm./MU;
            [Ag,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn,Cvconn);
            Aaug = [Ag -scl*d(:,1:3)*Ksrinv(:,1:nio) ;
scl*Ksr(1:nio,:)*d(:,1:3)' zeros(nio,nio)];
            daug = [d(:,4) zeros(length(Ag),nio) ;zeros(nio,1)
eye(nio)];
            if rcond(Aaug)<1e-16
                fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug),k);
            end
            % Pure Newton Raphson Iterator - find Jacobian and update x
            J       =
get_J(Cr(1:lB,:),O(1:lB,:),Aaug,MU,areas,dmdb,xg);
            xnewp = xg -  J\(Aaug*xg - daug*lam);

            % Check for convergence
            if ((sqrt((xnewp-xg)'*(xnewp-
xg))/(length(xg)*max(abs([xnewp;xg])))...
                    < TOL) || (it == parx(14)))
                if (it == parx(14))
                    % Maximum N-R iterations reached
                    disp(['Max Iterations Reached: IT = ' num2str(it)
', Data Point = ' num2str(k)]);
                    nrconverge = 0;
                end
                NRSOLVE = 0;
                nriter(k) = it;
            else
                xg = xnewp;
                fluxm = xg(1:end-nio);
                it = it+1;
            end
        end
        if ~nrconverge
            break
        end
        % Store flux/flux density values after converging
        phit(:,k) = phi(S+1:2*S);
        phiag = phi(4*S+11+D/2+Dsl/2+1+damper_nshank+D/2+2*(SPT-
1):end);
        BY(:,k)    = BIRON(1:S,k);
        BT(:,k)    = BIRON(S+1:2*S,k);
        BTT(:,k)   = BIRON(2*S+1:3*S,k);
        % Calculate torque
        torque(k) = ((RP/2)^2)*sum(phiag.^2.*dPTClist./(PTClist.^2));
        % Phase current calculation
        iqd0sr(:,k) = xg(end-nio+1:end)*scl;
        iabc(:,k) = Ksrinv(:,1:nio)*iqd0sr(:,k);
        % Phase flux linkage calculation
        lamabcpp(:,k) = Ksrinv(:,1:nio)*lamqd0srpp(:,k);

    elseif bartype == 1 % ---------------------------------------------
        % Solve for initial guess of damper flux linkage
```

```
        if k == 1
            Aaug_prime = [A -scl*d(:,1:3)*Ksrinv(:,1:nio) ;
scl*Ksr(1:nio,:)*d(:,1:3)' zeros(nio,nio)];
            daug_prime = [d(:,4) zeros(length(A),nio) ;zeros(nio,1)
eye(nio)];
            lam_prime = [ifld;scl*lamqd0srpp(:,k)];
            xg_prime = Aaug_prime\(daug_prime*lam_prime);
            lamdamper(1:damper_ntip-1,k) = d_damper_2'*xg_prime(1:end-
nio);
        end

        % Solve for vector of loop flux and current
        Aaug = [A -scl*d(:,1:3)*Ksrinv(:,1:nio) -scl*d_damper_1; ...
            scl*Ksr(1:nio,:)*d(:,1:3)' zeros(nio,nio+damper_ntip-1);
...
            scl*d_damper_2' zeros(damper_ntip-1,nio+damper_ntip-1)];
        daug = [d(:,4) zeros(length(A),nio+damper_ntip-1) ; ...
            zeros(nio+damper_ntip-1,1) eye(nio+damper_ntip-1)];
        if rcond(Aaug)<1e-16
            fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug),k);
        end
        lam = [ifld;scl*lamqd0srpp(:,k);scl*lamdamper(1:damper_ntip-
1,k)];
        xg = Aaug\(daug*lam);

        % Identify just the loop fluxes
        fluxm = xg(1:end-nio-damper_ntip+1);
        % NEWTON-RAPHSON SOLVER
        it = 1; % Keeps track of N-R iterations
        NRSOLVE = 1;
        while NRSOLVE
            % DETERMINE FLUXES FOR THE GUESS VECTOR xg
            phi = O*fluxm;
            phiiron(:,k) = phi(1:lB);
            % DETERMINE B-FIELDs
            BIRON(:,k) = phiiron(:,k)./areas;
            % GET PERMEABILITY FOR EACH RESPECTIVE PERM
            [sMU,sdmdb] = get_mur_exp(BIRON(1:slB,k),mudata.s);
            [rMU,rdmdb] = get_mur_exp(BIRON(slB+1:end,k),mudata.r);
            MU = [sMU;rMU];
            dmdb = [sdmdb;rdmdb];
            % UPDATE MATRICIES
            Riron = Rxm./MU;
            [Ag,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn,Cvconn);
            Aaug = [Ag -scl*d(:,1:3)*Ksrinv(:,1:nio) -scl*d_damper_1;
...
                scl*Ksr(1:nio,:)*d(:,1:3)' zeros(nio,nio+damper_ntip-
1); ...
                scl*d_damper_2' zeros(damper_ntip-1,nio+damper_ntip-
1)];
            daug = [d(:,4) zeros(length(Ag),nio+damper_ntip-1) ; ...
                zeros(nio+damper_ntip-1,1) eye(nio+damper_ntip-1)];
            if rcond(Aaug)<1e-16
```

```
                fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug),k);
            end
            % Pure Newton Raphson Iterator - find Jacobian and update x
            J       =
get_J(Cr(1:lB,:),O(1:lB,:),Aaug,MU,areas,dmdb,xg);
            xnewp = xg -  J\(Aaug*xg - daug*lam);

            % Check for convergence
            if ((sqrt((xnewp-xg)'*(xnewp-
xg))/(length(xg)*max(abs([xnewp;xg])))...
                    < TOL) || (it == parx(14)))
                if (it == parx(14))
                    % Maximum N-R iterations reached
                    disp(['Max Iterations Reached: IT = ' num2str(it)
', Data Point = ' num2str(k)]);
                    nrconverge = 0;
                end
                NRSOLVE = 0;
                nriter(k) = it;
            else
                xg = xnewp;
                fluxm = xg(1:end-nio-damper_ntip+1);
                it = it+1;
            end
        end
        if ~nrconverge
            break
        end

        % Store flux/flux density values after converging
        phit(:,k) = phi(S+1:2*S);
        phiag = phi(4*S+11+D/2+Dsl/2+1+damper_nshank+D/2+2*(SPT-
1):end);
        BY(:,k)    = BIRON(1:S,k);
        BT(:,k)    = BIRON(S+1:2*S,k);
        BTT(:,k)   = BIRON(2*S+1:3*S,k);
        % Calculate torque
        torque(k) = ((RP/2)^2)*sum(phiag.^2.*dPTClist./(PTClist.^2));
        % Phase current calculation
        iqd0sr(:,k) = xg(end-nio-damper_ntip+2:end-damper_ntip+1)*scl;
        iabc(:,k) = Ksrinv(:,1:nio)*iqd0sr(:,k); % terminals series
connected
        % Phase flux linkage calculation
        lamabcpp(:,k) = Ksrinv(:,1:nio)*lamqd0srpp(:,k);
        % Damper windings current
        idamper(1:damper_ntip-1,k) = xg(end-damper_ntip+2:end)*scl;
        idamper(damper_ntip,k) = -sum(idamper(1:damper_ntip-1,k));

    elseif bartype == 2 % ---------------------------------------------
        % Solve for initial guess of damper flux linkage
        if k == 1
            Aaug_prime = [A -scl*d(:,1:3)*Ksrinv(:,1:nio) ;
scl*Ksr(1:nio,:)*d(:,1:3)' zeros(nio,nio)];
```

```
            daug_prime = [d(:,4) zeros(length(A),nio) ;zeros(nio,1)
eye(nio)];
            lam_prime = [ifld;scl*lamqd0srpp(:,k)];
            xg_prime = Aaug_prime\(daug_prime*lam_prime);
            lamdamper(:,k) = d_damper_2'*xg_prime(1:end-nio);
        end

        % Solve for vector of loop flux and current
        Aaug = [A -scl*d(:,1:3)*Ksrinv(:,1:nio) -scl*d_damper_1; ...
            scl*Ksr(1:nio,:)*d(:,1:3)' zeros(nio,nio+damper_ntip); ...
            scl*d_damper_2' zeros(damper_ntip,nio+damper_ntip)];
        daug = [d(:,4) zeros(length(A),nio+damper_ntip) ; ...
            zeros(nio+damper_ntip,1) eye(nio+damper_ntip)];
        if rcond(Aaug)<1e-16
            fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug),k);
        end
        lam = [ifld;scl*lamqd0srpp(:,k);scl*lamdamper(:,k)];
        xg = Aaug\(daug*lam);

        % Identify just the loop fluxes
        fluxm = xg(1:end-nio-damper_ntip);
        % NEWTON-RAPHSON SOLVER
        it = 1; % Keeps track of N-R iterations
        NRSOLVE = 1;
        while NRSOLVE
            % DETERMINE FLUXES FOR THE GUESS VECTOR xg
            phi = O*fluxm;
            phiiron(:,k) = phi(1:lB);
            % DETERMINE B-FIELDs
            BIRON(:,k) = phiiron(:,k)./areas;
            % GET PERMEABILITY FOR EACH RESPECTIVE PERM
            [sMU,sdmdb] = get_mur_exp(BIRON(1:slB,k),mudata.s);
            [rMU,rdmdb] = get_mur_exp(BIRON(slB+1:end,k),mudata.r);
            MU = [sMU;rMU];
            dmdb = [sdmdb;rdmdb];
            % UPDATE MATRICIES
            Riron = Rxm./MU;
            [Ag,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn,Cvconn);
            Aaug = [Ag -scl*d(:,1:3)*Ksrinv(:,1:nio) -scl*d_damper_1;
...
                scl*Ksr(1:nio,:)*d(:,1:3)' zeros(nio,nio+damper_ntip);
...
                scl*d_damper_2' zeros(damper_ntip,nio+damper_ntip)];
            daug = [d(:,4) zeros(length(Ag),nio+damper_ntip) ; ...
                zeros(nio+damper_ntip,1) eye(nio+damper_ntip)];
            if rcond(Aaug)<1e-16
                fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug),k);
            end
            % Pure Newton Raphson Iterator - find Jacobian and update x
            J       =
get_J(Cr(1:lB,:),O(1:lB,:),Aaug,MU,areas,dmdb,xg);
            xnewp = xg -  J\(Aaug*xg - daug*lam);
```

```
            % Check for convergence
            if ((sqrt((xnewp-xg)'*(xnewp-
xg))/(length(xg)*max(abs([xnewp;xg])))...
                    < TOL) || (it == parx(14)))
                if (it == parx(14))
                    % Maximum N-R iterations reached
                    disp(['Max Iterations Reached: IT = ' num2str(it)
', Data Point = ' num2str(k)]);
                    nrconverge = 0;
                end
                NRSOLVE = 0;
                nriter(k) = it;
            else
                xg = xnewp;
                fluxm = xg(1:end-nio-damper_ntip);
                it = it+1;
            end
        end
        if ~nrconverge
            break
        end

        % Store flux/flux density values after converging
        phit(:,k) = phi(S+1:2*S);
        phiag = phi(4*S+11+D/2+Dsl/2+1+damper_nshank+D/2+2*(SPT-
1):end);
        BY(:,k)    = BIRON(1:S,k);
        BT(:,k)    = BIRON(S+1:2*S,k);
        BTT(:,k)   = BIRON(2*S+1:3*S,k);
        % Calculate torque
        torque(k) = ((RP/2)^2)*sum(phiag.^2.*dPTClist./(PTClist.^2));
        % Phase current calculation
        iqd0sr(:,k) = xg(end-nio-damper_ntip+1:end-damper_ntip)*scl;
        iabc(:,k) = Ksrinv(:,1:nio)*iqd0sr(:,k); % terminals series
connected
        % Phase flux linkage calculation
        lamabcpp(:,k) = Ksrinv(:,1:nio)*lamqd0srpp(:,k);
        % Damper windings current
        idamper(:,k) = xg(end-damper_ntip+1:end)*scl;
    end
    %-------------------------------------------------------------------
    % External voltage model-----------------------------------------
    % R load
%     vqd0sr(:,k) = -iqd0sr(:,k)*Rload;
    % Parallel RL load
    vqd0sr(:,k) = (-iqd0sr(:,k)-il_qd(:,k))*Rload;
    pil_qd(:,k) = vqd0sr(:,k)/Lload - wr*[0 1;-1 0]*il_qd(:,k);
    il_qd(:,k+1) = il_qd(:,k)+pil_qd(:,k)*DT;

%     abc voltage calculation
    vabc(:,k) = Ksrinv(:,1:nio)*vqd0sr(:,k); % Terminals series
connected
```

```
    % Connected to rectifier with constant vdc
%     iabcl = m_isil*iabc(:,k);
%     [V,idc(k)] = rect(iabcl,vdcmax,parx);
%     vqd0gr = Ksr*V;
%     vqd0sr(:,k) = m_vgvs*vqd0gr;
%     vabc(:,k) = Ksrinv(:,1:nio)*vqd0sr(:,k);
    % Connected to rectifier with RLC load
%     iabcl = m_isil*iabc(:,k);
%     [V,idc(k)] = rect(iabcl,vdc(k),parx);
%     vqd0gr = Ksr*V;
%     vqd0sr(:,k) = m_vgvs*vqd0gr;
%     vabc(:,k) = Ksrinv(:,1:nio)*vqd0sr(:,k);
%     pvc(k) = (idc(k)-vc(k)/Rload)/Cload;
%     vc(k+1) = vc(k)+pvc(k)*DT;
%     Ivc(k+1) = Ivc(k)+(vc(k+1)+vc(k))/2*DT;
%     vdc(k+1) = (-
(Ivdc(k)+vdc(k)*DT/2)+taus*vc(k+1)+Ivc(k+1)+Lload*idc(k))/(taus+DT/2);
%     Ivdc(k+1) = Ivdc(k)+(vdc(k+1)+vdc(k))/2*DT;
    %-----------------------------------------------------------------

    % Forward Euler to solve state model------------------------------
    plamqd0srpp(:,k) = (vqd0sr(:,k) - rs.*iqd0sr(:,k) -
wr*mlam*lamqd0srpp(:,k)*RP)/RP;
    lamqd0srpp(:,k+1) = lamqd0srpp(:,k) + plamqd0srpp(:,k)*DT;

    if bartype == 0
        if damper_ntip > 0
            lamdamper(:,k) = d_damper_2'*xg(1:end-nio);
        end
    elseif bartype == 1
        plamdamper(1:damper_ntip-1,k) = -Tdp*idamper(1:damper_ntip-
1,k);
        lamdamper(:,k+1) = lamdamper(:,k) + plamdamper(:,k)*DT;
    elseif bartype == 2
        plamdamper(:,k) = -Tdp*idamper(:,k);
        lamdamper(:,k+1) = lamdamper(:,k) + plamdamper(:,k)*DT;
    end
    %-----------------------------------------------------------------

    index_vect(:,:,k+1) = index;
    flag_vect(:,k+1) = flag;

    % Increment time/rotor position
    k = k+1;
end

% Check for flux densities above limit
Bsat = parx(23);
maxB = max(abs(BIRON));
saturate(maxB>=Bsat) = 1./(1+abs((maxB(maxB>=Bsat)-Bsat)./(0.1*Bsat)));
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% April 1, 2012
%-------------------------------------------------------------------------
% [Rxm,areas,Rair,NPRTS,NPRTB] =
get_reluctances(mu0,parx,pars,damperdata)
%
% Calculates all terms in the reluctance equation except for the
% relative permeability.  This is done for all iron permeances in the
% stator and rotor.  Calculates cross-sectional area. Calculates all
% reluctances residing in air.
%
% OUTPUTS: Rxm      - iron reluctances times mur
%          areas    - reluctance areas
%          Rair     - reluctances in air
%          NPRTS    - # of permeances connected to the rotor pole tip
side
%          NPRTB    - # of permeances connected to the pole tip bottom
%
% INPUTS:  mu0      - permeability of air
%          parx     - simulation parameters
%          pars     - geometric parameters
%          damperdata - damper properties
%-------------------------------------------------------------------------
function [Rxm,areas,Rair,NPRTS,NPRTB] =
get_reluctances(mu0,parx,pars,damperdata)
% Define reluctance connections in stator and rotor which do not change
% IRON
% Stator yoke - S
% Stator shank - S
% Stator teeth - S
% Rotor yoke below the slot - 1
% Rotor tooth shank - 1
% Damper bar in Rotor tooth shank - damper_nshank
% Rotor yoke connected to shank - 2
% Rotor tooth tips radial - (D - 4*NRrtrt)/2
% Damper windings in Rotor tooth tips radial - (D - 4*NRrtrt)/2
% Rotor tooth to rotor tooth tangential - 4*NRrtrt/2
% Damper windings in Rotor tooth to rotor tooth tangential - 4*NRrtrt/2
% Leakage of damper windings - 2*Nldp
% Rotor tooth tangential at sides of tooth tips - 4/2
% AIR
% Stator tooth leakage - S
% Field winding leakage - 2
% Middle rotor slot leakage - 2/2
% Fringing permeance from rotor side to airgap boundary - Dsl
% Fringing permeance from rotor slot side to bottom of tooth tip - 4/2
```

```
% Machine parameters
G1      = pars(14); % Airgap length, m
DBS     = pars(4);  % stator yoke depth, m
STW     = pars(20); % width of tooth shank, m
CL      = pars(26); % rotor core length, m
SL      = parx(3);  % number of teeth in one mechanical cycle
OD      = pars(1);  % stator outside diameter of yoke, m
ID      = pars(2);  % stator inner diameter (tooth tip to tooth tip), m
WRT     = pars(34);
WRTSH   = pars(46);
SD      = pars(29);
WRTSHchord = pars(56);
NRrtrt  = parx(27);
SPT     = parx(2);
Nrtt    = 2*SPT - 4*NRrtrt;       % Number of radial rotor tooth
branches
GLS     = pars(3);  % Stator stack length, m
H0      = pars(5);  % Stator slot dimension, m
H3      = pars(8);  % Stator slot dimension, m
B0      = pars(9);  % Stator slot dimension, m
B1      = pars(10); % Stator slot dimension, m
B2      = pars(11); % Stator slot dimension, m
BS      = pars(12); % Stator slot dimension, m
GLP     = pars(27); % Rotor stack length, m
HRTT    = pars(44); % Height of rotor tooth tip, m
HRTSH   = pars(45); % Height of rotor tooth shank, m
WCOIL   = pars(51); % Equivalent width of field wdg, m
SPAIR   = parx(29); % Number of rotor sections in half the slot
RPIT    = pars(32); % Rotor pole pitch coefficient
ROD     = pars(24); % Rotor outer diameter, m
RP      = pars(28); % Number of rotor poles
S       = parx(3)/pars(28);  % Number of stator teeth per pole
SPT     = parx(2); % number of rotor sections in the pole tip
DC      = pars(25); % Rotor core diameter, m
tipw    = pars(57);
tiph    = pars(58);
damper_rtip = damperdata.damper_rtip;
damper_rshank = damperdata.damper_rshank;
damper_ntip = damperdata.damper_ntip;
damper_nshank = damperdata.damper_nshank;
damper_dtip = damperdata.damper_dtip;
WRTang  = 2*WRT/ROD;
xout    = sin(WRTang/2)*ROD/2;  % (xout = WRTchord/2)
yb      = cos(WRTang/2)*ROD/2-HRTT; % Vertical height to the bottom of
the rotor tooth tip
xin     = WRTSHchord/2;
WRTS2   = xout*2/SPT; % Horizontal width (not arc width) of the rotor
tooth sections
% yt__ = Vertical height to the top of the rotor tooth tip at a given
"x"
% position
% **Stator yoke
AY = ones(S,1)*GLS*DBS;
RY = (pi*(OD-DBS))/((mu0)*GLS*SL*DBS);
% **Stator tooth shank
```

```
AT_shank = ones(S,1)*STW*GLS;
LT_shank  = (OD/2-DBS/2)-ID/2-tiph;
RT_shank  = LT_shank./(mu0*STW*GLS);
% **Stator tooth tip
AT_tip = ones(S,1)*(STW+2*tipw)*GLS;
RT_tip = tiph./(mu0*(STW+2*tipw)*GLS);
% **Rotor yoke below the slot and connected to shank
ARY     = 0.5*(DC - SD)*CL;
rad     = DC/4+SD/4;
thsh_core = 2*asin(WRTSHchord/DC); % Angle of the rotor shank at the
outside of the rotor core
thsl_core = 2*pi/RP - thsh_core; % Angle of the rotor slot at the
outside of the rotor core
thsl    = thsl_core/2; % Angular length of the rotor yoke reluctance
below the slot
thsh    = thsl_core/4+thsh_core/2; % Angular length of the rotor yoke
reluctance connected to the shank
RRYSL = rad*thsl./(mu0*ARY);
RRYSH = rad*thsh./(mu0*ARY);
%-------------------------------------------------------------------------
% **Rotor tooth tip (inner)
ARTIN = ones(Nrtt/2,1)*GLP*WRTS2;
ymid = (sqrt((ROD/2)^2-(xin).^2)+yb)/2;
ytRTT = sqrt((ROD/2)^2-abs(xout-WRTS2*NRrtrt-WRTS2*((1:Nrtt/2)-
0.5)').^2);
RTTlength = ytRTT - ymid;
RTTlength_IN = zeros(Nrtt/2,1);
for i = 0:(Nrtt/4-1)
    RTTlength_IN(Nrtt/4+i+1) = RTTlength(Nrtt/4+i+1)-
2*damper_rtip(i+1);
    RTTlength_IN(end-(Nrtt/4+i+1)+1) = RTTlength_IN(Nrtt/4+i+1);
end
RRTIN = RTTlength_IN./(mu0*ARTIN);
% damper windings on Rotor tooth tip (inner)
ARD_tip_in = ARTIN;
RRD_tip_in = zeros(Nrtt/2,1);
for i = 0:(Nrtt/4-1)
    if i == 0
        ARD_tip_in(Nrtt/4+i+1) = (WRTS2-damper_rtip(1)/2)*GLP;
        ARD_tip_in(end-(Nrtt/4+i+1)+1) = ARD_tip_in(Nrtt/4+i+1);
        RRD_tip_in(Nrtt/4+i+1) = 2*(-pi/(2*mu0*GLP) +
WRTS2/(mu0*GLP*sqrt(WRTS2^2-damper_rtip(1)^2)) ...
                    *(pi/2+atan(damper_rtip(1)/sqrt(WRTS2^2-
damper_rtip(1)^2))));
        RRD_tip_in(Nrtt/4+i+1) =
RRD_tip_in(Nrtt/4+i+1)*(RRD_tip_in(Nrtt/4+i+1)>0.01*min(RRTIN));
        RRD_tip_in(end-(Nrtt/4+i+1)+1) = RRD_tip_in(Nrtt/4+i+1);
    else
        ARD_tip_in(Nrtt/4+i+1) = (WRTS2-damper_rtip(i+1))*GLP;
        ARD_tip_in(end-(Nrtt/4+i+1)+1) = ARD_tip_in(Nrtt/4+i+1);
        RRD_tip_in(Nrtt/4+i+1) = -pi/(2*mu0*GLP) +
WRTS2/(mu0*GLP*sqrt(WRTS2^2-4*damper_rtip(i+1)^2)) ...
                    *(pi/2+atan(2*damper_rtip(i+1)/sqrt(WRTS2^2-
4*damper_rtip(i+1)^2)));
```

```
        RRD_tip_in(Nrtt/4+i+1) =
RRD_tip_in(Nrtt/4+i+1)*(RRD_tip_in(Nrtt/4+i+1)>0.01*min(RRTIN));
        RRD_tip_in(end-(Nrtt/4+i+1)+1) = RRD_tip_in(Nrtt/4+i+1);
    end
end

%-------------------------------------------------------------------------
% **Rotor tooth shank
ARTSH = GLP*WRTSH;
l     = ymid - SD/2 - (DC-SD)/4;
RRTSH = (l-2*damper_nshank*damper_rshank)/(mu0*ARTSH);
% damper windings on Rotor tooth shank
ARD_shank = ones(damper_nshank,1)*GLP*(WRTSH-damper_rshank);
RRD_shank = ones(damper_nshank,1)*(-pi/(2*mu0*GLP) +
WRTSH/(mu0*GLP*sqrt(WRTSH^2-4*damper_rshank^2)) ...
                      *(pi/2+atan(2*damper_rshank/sqrt(WRTSH^2-
4*damper_rshank^2))));
%-------------------------------------------------------------------------
% **Rotor tooth section to rotor tooth section permeance
damper_rtip_out_2 = damper_rtip(end-NRrtrt+1:end);
damper_rtip_out_1 = flipdim(damper_rtip_out_2,1);
damper_rtip_out = [damper_rtip_out_1;damper_rtip_out_2];
ytend   = sqrt((ROD/2)^2-(xout-WRTS2/4)^2);
ARTOUT = zeros(2*NRrtrt,1);
for jj = 1:NRrtrt
    ytNR = sqrt((ROD/2)^2-(xout-WRTS2*jj).^2);
    ARTOUT(jj) = (ytNR-yb)*GLP;
    ARTOUT(end-jj+1) = (ytNR-yb)*GLP;
end
lR     = xout - xin + min(0.5*xin,(ymid-yb)); % Total length of the
estimated tangential reluctance from the side of the rotor tooth tip
WRTSIN = lR - (NRrtrt-1)*WRTS2 - WRTS2/2; % Adjusted length of the
inner rotor tooth tip tangential reluctance
lRRTOUT = [WRTS2*ones((NRrtrt-1),1);WRTSIN;WRTSIN;WRTS2*ones((NRrtrt-
1),1)];
lRRTOUT = lRRTOUT-2*damper_rtip_out;
RRTOUT = lRRTOUT./(mu0*ARTOUT);
% damper windings on Rotor tooth section to rotor tooth section
ARD_tip_out = zeros(2*NRrtrt,1);
RRD_tip_out = zeros(2*NRrtrt,1);
for jj = 1:NRrtrt
    ytNR = sqrt((ROD/2)^2-(xout-WRTS2*jj).^2);
    ARD_tip_out(jj) = (ytNR-yb-damper_rtip_out(jj))*GLP;
    ARD_tip_out(end-jj+1) = ARD_tip_out(jj);
    RRD_tip_out(jj) = -pi/(2*mu0*GLP) + (ytNR-yb)/(mu0*GLP*sqrt((ytNR-
yb)^2-4*damper_rtip_out(jj)^2)) ...
                      *(pi/2+atan(2*damper_rtip_out(jj)/sqrt((ytNR-
yb)^2-4*damper_rtip_out(jj)^2)));
    RRD_tip_out(jj) =
RRD_tip_out(jj)*(RRD_tip_out(jj)>0.01*min(RRTOUT));
    RRD_tip_out(end-jj+1) = RRD_tip_out(jj);
end
%-------------------------------------------------------------------------
% Leakage reluctance of damper windings in iron
% Leakage components on tangential path
```

```
ARD_ldp_out = zeros(2*NRrtrt,1);
RRD_ldp_out = zeros(2*NRrtrt,1);
for i = 1:2*NRrtrt
    if damper_rtip_out(i) == 0
        ARD_ldp_out(i) = ARTOUT(i)/2;
        RRD_ldp_out(i) = lRRTOUT(i)/(mu0*ARD_ldp_out(i));
    else
        ARD_ldp_out(i) = (ARTOUT(i)/GLP-
2*damper_rtip_out(i))*damper_dtip*GLP;
        RRD_ldp_out(i) =
2*pi/(mu0*GLP*log((ARD_ldp_out(i)/GLP+damper_rtip_out(i))/damper_rtip_o
ut(i)));
    end
end
% Leakage components on radial path
damper_rtip_in_2 = damper_rtip(1:Nrtt/4);
damper_rtip_in_1 = flipdim(damper_rtip_in_2,1);
damper_rtip_in = [damper_rtip_in_1;damper_rtip_in_2];
ARD_ldp_in = (RTTlength-2*damper_rtip_in)*GLP*damper_dtip;
RRD_ldp_in =
2*pi./(mu0*GLP*log((ARD_ldp_in/GLP+damper_rtip_in)./damper_rtip_in));
% Combine the tangential and radial components
ARD_ldp =
[ARD_ldp_out(1:end/2);ARD_ldp_in(1:end/2);ARD_ldp_in(end/2+2:end);ARD_l
dp_out(end/2+1:end)];
RRD_ldp =
[RRD_ldp_out(1:end/2);RRD_ldp_in(1:end/2);RRD_ldp_in(end/2+2:end);RRD_l
dp_out(end/2+1:end)];
% Correction of RRTOUT due to leakage
RRTOUT = 1./(1./(RRTOUT+RRD_tip_out)-1./RRD_ldp_out)-RRD_tip_out;

%------------------------------------------------------------------------
% **side rotor tangential reluctances
ARTRTS  = (ytend-yb)*GLP*ones(2,1);  % Area of the side rotor
tangential reluctances
lRTRTS = WRTS2/2*ones(2,1); % length of the side rotor tangential
reluctances
RRTRTS = lRTRTS./(mu0*ARTRTS);

% AREAS AND RELUCTANCES*MUR FOR IRON ELEMENTS
areas =
[AY;AT_shank;AT_tip;ARY;ARTSH;ARD_shank;ARY;ARY;ARTIN;ARD_tip_in;ARTOUT
;ARD_tip_out;ARD_ldp;ARTRTS];
Rxm =
[RY*ones(S,1);RT_shank*ones(S,1);RT_tip*ones(S,1);RRYSL;RRTSH;RRD_shank
;RRYSH*ones(2,1);RRTIN;RRD_tip_in;RRTOUT;RRD_tip_out;RRD_ldp;RRTRTS];
%------------------------------------------------------------------------
% **Stator tooth tip leakage
P012 = mu0*H0/(B1-B0)*log(B1/B0);
beta = B2/BS;
P3 = mu0*(H3/BS)*(( (beta^2) - ((beta^4)*0.25) - log(beta) - 0.75 )/(
(1-beta)*((1-beta^2)^2) ));
RSTL = 1/((P012 + P3)*GLS);
% **Field wdg leakage permeance
RFDL = 3*HRTSH/(mu0*GLP*WCOIL);
```

```
% Geometry calculations needed for determining rotor fringing
permeances
WRTang  = 2*pi*RPIT/RP;                     % ANGLE AT OUTSIDE EDGE OF
ROTOR TOOTH TIP
WRTchord= (ROD)*sin(0.5*WRTang);            % CHORD LENGTH OF ROTOR
TOOTH TIP
Rint    = WRTchord/(2*sin(pi/RP));          % Radius at the point where
a line extended from the rotor tooth side intersects with a line
through the center of the rotor slot
halfWAIRTchd = ROD*sin(0.5*pi/RP*(1-RPIT)); % Chord length of the arc
encompassing half the outer rotor slot
theta_Rfr = asin((ROD/2-Rint)/halfWAIRTchd*sin(pi/RP)); % Angle between
the rotor tooth side and the line halfWAIRTchd
WAIRTSchd  = halfWAIRTchd/(SPAIR);          % Width of a fringing flux
tube
WRTTS   = (WRTchord - pars(56))/2;          % Width of one side of the
rotor tooth tip not including the rotor shank
lRinttoROD = sqrt(halfWAIRTchd^2+(ROD/2-Rint)^2-2*(ROD/2-
Rint)*halfWAIRTchd*cos(pi-pi/RP-theta_Rfr)); % length of the line
extending from the rotor tooth tip side to the intersection point in
the middle of the rotor slot
% ** Fringing permeance from rotor slot to rotor bottom
WRTB2   = WRTTS;                            % Ending radius of RFRB
flux tube
WRTB1   = max(min([WRTchord/SPT/4 HRTSH/2 WRTTS/2]), 0.0001); %
Starting radius of RFRB flux tube
RFRB = 1./(mu0*GLP*2/pi*log(WRTB2/WRTB1));
% ** Fringing permeance from airgap to rotor side
if halfWAIRTchd < (HRTSH+HRTT)
    % Uniform flux tube widths can be used
    NPRTS   = max(ceil((HRTT+WRTB1)/WAIRTSchd),1);
    NPRTS = NPRTS*(NPRTS<SPAIR) + SPAIR*(NPRTS>=SPAIR); % if-else
    NPRTB = (SPAIR-NPRTS)*(NPRTS<SPAIR); % if-else
    lPAGFR = theta_Rfr*(0.5*WAIRTSchd+(0:WAIRTSchd:WAIRTSchd*(SPAIR-
1))'); % for-loop
    % Length of flux tube overlapping side and bottom
    lPAGFR(NPRTS) = lPAGFR(NPRTS)+WRTB1/WAIRTSchd*(WRTB1/2*pi/2);
    RAGFR = lPAGFR./(mu0*WAIRTSchd*GLP);
    % **Middle rotor slot leakage
    lmeanRTSL = 2*sin(pi/RP)*(lRinttoROD-halfWAIRTchd);
    wRTSL = (ROD/2-DC/2)/3;
    RRTL = lmeanRTSL/(mu0*GLP*wRTSL);
else
    % Flux tubes with decreasing width must be used
    theta_Rfr2 = min(acos((HRTSH+HRTT)/halfWAIRTchd),theta_Rfr);
%Portion of theta_Rfr where the flux tube is a triangle with non-
uniform width
    theta_Rfr1 = theta_Rfr - theta_Rfr2; %Portion of theta_Rfr where
the flux tube is still an arc with uniform width
    WRFR2 = HRTSH+HRTT;     % Total width of the flux tubes at the
rotor and field winding side
    WRFRs = WRFR2/SPAIR;    % Width of an individual flux tube at the
small end
    WRFRavg = (WRFRs*theta_Rfr2+WAIRTSchd*theta_Rfr1)/theta_Rfr; %
Average width of the flux tubes
```

```
    NPRTS   = max(ceil((HRTT+WRTB1)/WRFRs),1);
    NPRTS = NPRTS*(NPRTS<SPAIR) + SPAIR*(NPRTS>=SPAIR); % if-else
    NPRTB = (SPAIR-NPRTS)*(NPRTS<SPAIR); % if-elsE
    lPAGFR = theta_Rfr*(0.5*WAIRTSchd+(0:WAIRTSchd:WAIRTSchd*(SPAIR-
1))'); % for-loop
    % Length of flux tube overlapping side and bottom
    lPAGFR(NPRTS) = lPAGFR(NPRTS)+WRTB1/WRFRs*(WRTB1/2*pi/2);
    RAGFR = lPAGFR./(mu0*WRFRavg*GLP);
    % **Middle rotor slot leakage
    lmeanRTSL = 2*sin(pi/RP)*(lRinttoROD-WRFR2);
    wRTSL = (ROD/2-DC/2)/3;
    RRTL = lmeanRTSL/(mu0*GLP*wRTSL);
end

% Leakage reluctance of damper windings in air
Rair_ldp_out = 1e16*ones(2*NRrtrt,1);
for i = 1:2*NRrtrt
    if damper_rtip_out(i) > 0
        Rair_ldp_out(i) = 1./(mu0*GLP/8/pi + mu0*GLP/2* ...

log((sqrt(2*G1*(ARD_ldp_out(i)/GLP+damper_rtip_out(i))+G1^2) ...

+G1+ARD_ldp_out(i)/GLP+damper_rtip_out(i))./(ARD_ldp_out(i)/GLP+damper_
rtip_out(i))));
    end
end
Rair_ldp_in = 1./(mu0*GLP/8/pi + mu0*GLP/2* ...
            log((sqrt(2*G1*(ARD_ldp_in/GLP+damper_rtip_in)+G1^2) ...

+G1+ARD_ldp_in/GLP+damper_rtip_in)./(ARD_ldp_in/GLP+damper_rtip_in)));
Rair_ldp =
[Rair_ldp_out(1:end/2);Rair_ldp_in(1:end/2);Rair_ldp_in(end/2+2:end);Ra
ir_ldp_out(end/2+1:end)];

% **Air permeances
Rair =
[RSTL*ones(S,1);RFDL*ones(2,1);RRTL;RAGFR;flipud(RAGFR);RFRB*ones(2,1);
Rair_ldp];
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Michelle Bash, Steven D. Pekarek, Hamza Derbas
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% Feb, 2010
%-------------------------------------------------------------------------
% [G,dG] =
% get_Pag(theta_rm,pars,parx,Gmaxrt,Gmaxsl,anglert,anglesl,qrrcs,qrscs)
%
% Determines the airgap permeance between each rotor tooth/slot section
and
% stator tooth.
%
% OUTPUTS: G        - matrix of airgap permeances, size S x D+Dslot
%          dG       - derivative of G w.r.t. thetar, size S x D+Dslot
%
% INPUTS:  theta_rm - mechanical rotor position (shifted to work
herein)
%          pars     - parameters
%          parx     - simulation parameters
%          Gmaxrt   - permeance when a rotor tooth section is
completely
%          under a stator tooth
%          Gmaxsl   - permeance when a rotor slot section is completely
%          under a stator tooth
%          anglert  - angle between each rotor tooth section and stator
%          tooth
%          anglesl  - angle between each rotor slot section and stator
%          tooth
%          qrrcs    - geometric case for rotor tooth section
%          qrscs    - geometric case for rotor slot section
%-------------------------------------------------------------------------
function [G,dG] =
get_Pag(theta_rm,pars,parx,Gmaxrt,Gmaxsl,anglert,anglesl,qrrcs,qrscs)
%DIMENSIONS & PARAMETERS
ID      = pars(2);      % Stator inner diameter, m
GLS     = pars(3);      % Stator stack length, m
ROD     = pars(24);     % Rotor outer diameter, m
RP      = pars(28);     % Number of rotor poles
STTW    = pars(21);     % Width of stator tooth, m
RPIT    = pars(32);     % Rotor pole pitch coefficient, m
B0      = pars(9);      % Stator slot width, m
g       = pars(14);     % Airgap length, m
slope   = pars(54);     % Used to calculate airgap permeance, rad
SPT     = parx(2);      % Number of rotor tooth sections
D       = SPT*2;        % Number of rotor tooth sections over a pole
pair
S1P     = parx(3)/RP;   % Number of stator teeth per pole
Dslot   = 4*parx(29);   % Number of rotor slot sections over a pole
pair
```

```
mu0     = pi*4e-7;      % Permeability of free space
% Relevant angular spans
WRTSang = 2*pi*RPIT/RP/SPT; % Angular width of rotor tooth section
WRSang  = 2*pi*(1-RPIT)/RP/(Dslot/2); % Angular width of rotor slot
section
qs      = STTW/ID*RP; % Electrical angular width of stator tooth
qs1     = B0/ID*RP; % Electrical angular width of stator slot
qrr     = WRTSang*RP/2; % Electrical angular width of rotor tooth
section
qrs     = WRSang*RP/2; % Electrical angular width of rotor slot section
aoff    = 1e-13; % Angular offset used to avoid numerical errors
% Initialize matrices
Grt     = zeros(S1P,D);
dGrt    = zeros(S1P,D);
Gsl     = zeros(S1P,Dslot);
dGsl    = zeros(S1P,Dslot);
% position (Electrical) of rotor tooth and rotor
% slot sections in relation to stator teeth
posrt = mod(RP/2*(theta_rm+anglert),2*pi); % defined as shown below
possl = mod(RP/2*(theta_rm+anglesl),2*pi); % defined as shown below
% Calculate airgap permeances over the rotor tooth (pole)
% Common terms in permeances and derivative calculations
Pm1 = (mu0*GLS/slope);
Pm2 = ROD/RP*slope;
dPm1 = (mu0*GLS*ROD/RP);
switch qrrcs
  case 1 % qrr <= qs1/2
    Gedges = [0 qrr qs1/2 qs1/2+qrr-aoff (2*S1P-1)*(qs+qs1)+qs1/2+aoff
...
      (2*S1P-1)*(qs+qs1)+qs1/2+qrr (2*S1P-1)*(qs+qs1)+qs1 ...
      (2*S1P-1)*(qs+qs1)+qs1+qrr 2*pi];
    [ncs,Gcs] = histc(posrt,Gedges,2);
    % calculate permeances for non-zero cases (Case 4 P=0)
    % Case 1
    Grt(Gcs==1) = (Gmaxrt*(qrr - posrt(Gcs==1))/qrr) + ...
        Pm1*log((posrt(Gcs==1)*Pm2 + g)/g);
    dGrt(Gcs==1) = -Gmaxrt/qrr + dPm1./(posrt(Gcs==1)*Pm2 + g);
    % Case 2
    Grt(Gcs==2) = Pm1*log((posrt(Gcs==2)*Pm2+g)./((posrt(Gcs==2)-
qrr)*Pm2+g));
    dGrt(Gcs==2) = dPm1*(1./(posrt(Gcs==2)*Pm2 + g) - ...
        1./((posrt(Gcs==2)-qrr)*Pm2 + g));
    % Case 3
    Grt(Gcs==3) = Pm1*(log(qs1/2*Pm2 + g) - log((posrt(Gcs==3)-qrr)*Pm2
+ g));
    dGrt(Gcs==3) = dPm1*(-1./((posrt(Gcs==3)-qrr)*Pm2 + g));
    % Case 5
    Grt(Gcs==5) = Pm1*(log(qs1/2*Pm2 + g) - ...
        log((2*pi - posrt(Gcs==5) - qs)*Pm2 + g));
    dGrt(Gcs==5) = dPm1*(1./((2*pi - posrt(Gcs==5) - qs)*Pm2 + g));
    % Case 6
    Grt(Gcs==6) = Pm1*(log((2*pi-posrt(Gcs==6)-qs+qrr)*Pm2 + g) - ...
        log((2*pi-posrt(Gcs==6)-qs)*Pm2 + g));
    dGrt(Gcs==6) = dPm1*(-1./((2*pi-posrt(Gcs==6)-qs+qrr)*Pm2 + g) +
...
```

```
        1./((2*pi-posrt(Gcs==6)-qs)*Pm2 + g));
    % Case 7
    Grt(Gcs==7) = Gmaxrt*(posrt(Gcs==7)-2*pi+qs)/qrr + ...
        Pm1*log(((2*pi-qs-posrt(Gcs==7)+qrr)*Pm2 + g)/g);
    dGrt(Gcs==7) = Gmaxrt/qrr+dPm1*(-1./((2*pi-qs-
posrt(Gcs==7)+qrr)*Pm2+g));
    % Case 8
    Grt(Gcs>=8) = Gmaxrt;
    dGrt(Gcs>=8) = 0;
  case 2 % (qrr > qs1/2) && (qrr <= qs)
    Gedges = [0 qs1/2 qrr qrr+qs1/2-aoff (2*S1P-1)*(qs+qs1) +
qs1/2+aoff ...
        (2*S1P-1)*(qs+qs1) + qs1 (2*S1P-1)*(qs+qs1) + qs1/2 + qrr ...
        (2*S1P-1)*(qs+qs1) + qs1 + qrr max(2*S1P*(qs+qs1),2*pi)];
    [ncs,Gcs] = histc(posrt,Gedges,2);
    % calculate permeances for non-zero cases (Case 4 P=0)
    % Case 1
    Grt(Gcs==1) = (Gmaxrt*(qrr - posrt(Gcs==1))/qrr) + ...
        Pm1*log((posrt(Gcs==1)*Pm2 + g)/g);
    dGrt(Gcs==1) = -Gmaxrt/qrr + dPm1./(posrt(Gcs==1)*Pm2 + g);
    % Case 2
    Grt(Gcs==2) = (Gmaxrt*(qrr-posrt(Gcs==2))/qrr)+Pm1*log((qs1/2*Pm2 +
g)/g);
    dGrt(Gcs==2) = -Gmaxrt/qrr;
    % Case 3
    Grt(Gcs==3) = Pm1*(log(qs1/2*Pm2 + g) - log((posrt(Gcs==3)-
qrr)*Pm2+ g));
    dGrt(Gcs==3) = -dPm1./((posrt(Gcs==3) - qrr)*Pm2 + g);
    % Case 5
    Grt(Gcs==5) = Pm1*(log(qs1/2*Pm2+g)-log((2*pi-posrt(Gcs==5)-qs)*Pm2
+ g));
    dGrt(Gcs==5) = dPm1./((2*pi - posrt(Gcs==5) - qs)*Pm2 + g);
    % Case 6
    Grt(Gcs==6)=(Gmaxrt*(posrt(Gcs==6)+qs-
2*pi)/qrr)+Pm1*log((qs1/2*Pm2+g)/g);
    dGrt(Gcs==6) = Gmaxrt/qrr;
    % Case 7
    Grt(Gcs==7) = (Gmaxrt*(posrt(Gcs==7) + qs - 2*pi)/qrr) + ...
        Pm1*log(((2*pi - posrt(Gcs==7) - qs + qrr)*Pm2 + g)/g);
    dGrt(Gcs==7) = Gmaxrt/qrr - dPm1./((2*pi-posrt(Gcs==7)-qs+qrr)*Pm2
+ g);
    % Case 8
    Grt(Gcs>=8) = Gmaxrt;
    dGrt(Gcs>=8) = 0;
  case 3 % (qrr > qs) && (qrr <= qs +qs1/2)
    Gedges = [0 qrr-qs qs1/2 qrr qrr+qs1/2-aoff ...
        (2*S1P-1)*(qs+qs1)+qs1/2+aoff (2*S1P-1)*(qs+qs1)+qs1...
        (2*S1P-1)*(qs+qs1)+qs1/2+qrr max(2*S1P*(qs+qs1),2*pi)];
    [ncs,Gcs] = histc(posrt,Gedges,2);
    % calculate permeances for non-zero cases (Case 5 P=0)
    % Case 1
    Grt(Gcs==1) = Gmaxrt+Pm1*(log((posrt(Gcs==1)*ROD/RP)*slope+g)-
log(g))+...
            Pm1*(log((qrr-qs-posrt(Gcs==1))*Pm2 + g) - log(g));
    dGrt(Gcs==1) = dPm1*(-1./((qrr-qs-posrt(Gcs==1))*Pm2 + g) +...
```

```
        1./((posrt(Gcs==1))*Pm2 + g));
    % Case 2
    Grt(Gcs==2) = Gmaxrt*(qrr - posrt(Gcs==2))/qs + ...
        Pm1*(log((posrt(Gcs==2)*ROD/RP)*slope + g) - log(g));
    dGrt(Gcs==2) = -Gmaxrt/qs + dPm1*(1./(posrt(Gcs==2)*Pm2 + g));
    % Case 3
    Grt(Gcs==3) = Gmaxrt*(qrr - posrt(Gcs==3))/qs + ...
        Pm1*(log((qs1/2*ROD/RP)*slope + g) - log(g));
    dGrt(Gcs==3) = -Gmaxrt/qs;
    % Case 4
    Grt(Gcs==4) = Pm1*(log((qs1/2*ROD/RP)*slope + g) - ...
        log((posrt(Gcs==4)-qrr)*Pm2 + g));
    dGrt(Gcs==4) = dPm1*(-1./((posrt(Gcs==4)-qrr)*Pm2 + g));
    % Case 6
    Grt(Gcs==6) = Pm1*(log((qs1/2*ROD/RP)*slope + g) - ...
        log((2*pi-qs-posrt(Gcs==6))*Pm2 + g));
    dGrt(Gcs==6) = dPm1*(1./((2*pi-qs-posrt(Gcs==6))*Pm2 + g));
    % Case 7
    Grt(Gcs==7) = Gmaxrt*(qs-(2*pi-posrt(Gcs==7)))/qs + ...
        Pm1*(log((qs1/2*ROD/RP)*slope + g) - log(g));
    dGrt(Gcs==7) = Gmaxrt/qs;
    % Case 8
    Grt(Gcs>=8) = Gmaxrt*(qs-(2*pi-posrt(Gcs>=8)))/qs + ...
        Pm1*(log((2*pi-qs-posrt(Gcs>=8)+qrr)*Pm2 + g) - log(g));
    dGrt(Gcs>=8) = Gmaxrt/qs + dPm1*(-1./((2*pi-qs-
posrt(Gcs>=8)+qrr)*Pm2+g));
  case 4 % (qrr > qs+qs1/2) && (qrr <= qs+qs1)
    Gedges = [0 qrr-qs-qs1/2 qs1/2 qrr-qs qrr qs1/2+qrr-aoff ...
        (2*S1P-1)*(qs+qs1)+qs1/2+aoff (2*S1P-1)*(qs+qs1)+qs1 2*pi];
    [ncs,Gcs] = histc(posrt,Gedges,2);
    % calculate permeances for non-zero cases (Case 6 P=0)
    % Case 1
    Grt(Gcs==1) = Gmaxrt + Pm1*(log((posrt(Gcs==1))*Pm2 + g) - log(g))
+ ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGrt(Gcs==1) = dPm1*(1./((posrt(Gcs==1))*Pm2 + g));
    % Case 2
    Grt(Gcs==2) = Gmaxrt + Pm1*(log((posrt(Gcs==2))*Pm2 + g) - log(g))
+ ...
        Pm1*(log((qrr-posrt(Gcs==2)-qs)*Pm2 + g) - log(g));
    dGrt(Gcs==2) = dPm1*(1./((posrt(Gcs==2))*Pm2 + g) - ...
        1./((qrr-posrt(Gcs==2)-qs)*Pm2 + g));
    % Case 3
    Grt(Gcs==3) = Gmaxrt + Pm1*(log(qs1/2*Pm2 + g) - log(g)) + ...
        Pm1*(log((qrr-posrt(Gcs==3)-qs)*Pm2 + g) - log(g));
    dGrt(Gcs==3) = dPm1*(-1./((qrr-posrt(Gcs==3)-qs)*Pm2 + g));
    % Case 4
    Grt(Gcs==4) = Gmaxrt*(qrr-posrt(Gcs==4))/qs + Pm1*(log(qs1/2*Pm2 +
g) ...
        - log(g));
    dGrt(Gcs==4) = -Gmaxrt/qs;
    % Case 5
    Grt(Gcs==5) = Pm1*(log(qs1/2*Pm2 + g) - log((posrt(Gcs==5)-qrr)*Pm2
+ g));
    dGrt(Gcs==5) = dPm1*(-1./((posrt(Gcs==5)-qrr)*Pm2 + g));
```

```
    % Case 7
    Grt(Gcs==7) = Pm1*(log(qs1/2*Pm2 + g)-log((2*pi-posrt(Gcs==7)-
qs)*Pm2+g));
    dGrt(Gcs==7) = dPm1*(1./((2*pi-posrt(Gcs==7)-qs)*Pm2 + g));
    % Case 8
    Grt(Gcs>=8) = Gmaxrt*(posrt(Gcs>=8)-2*pi+qs)/qs + ...
        Pm1*(log(qs1/2*Pm2+g) - log(g));
    dGrt(Gcs>=8) = Gmaxrt/qs;
  case 5 % (qrr > qs+qs1)
    Gedges = [0 qs1/2 qrr-qs-qs1/2 qrr-qs qrr qs1/2+qrr-aoff ...
        (2*S1P-1)*(qs+qs1)+qs1/2+aoff (2*S1P-1)*(qs+qs1)+qs1 2*pi];
    [ncs,Gcs] = histc(posrt,Gedges,2);
    % calculate permeances for non-zero cases (Case 6 P=0)
    % Case 1
    Grt(Gcs==1) = Gmaxrt + Pm1*(log((posrt(Gcs==1))*Pm2 + g) - log(g))
+ ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGrt(Gcs==1) = dPm1*(1./((posrt(Gcs==1))*Pm2 + g));
    % Case 2
    Grt(Gcs==2) = Gmaxrt + Pm1*(log(qs1/2*Pm2 + g) - log(g)) + ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGrt(Gcs==2) = 0;
    % Case 3
    Grt(Gcs==3) = Gmaxrt + Pm1*(log(qs1/2*Pm2 + g) - log(g)) + ...
        Pm1*(log((qrr-posrt(Gcs==3)-qs)*Pm2 + g) - log(g));
    dGrt(Gcs==3) = dPm1*(-1./((qrr-posrt(Gcs==3)-qs)*Pm2 + g));
    % Case 4
    Grt(Gcs==4) = Gmaxrt*(qrr-posrt(Gcs==4))/qs + ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGrt(Gcs==4) = -Gmaxrt/qs;
    % Case 5
    Grt(Gcs==5) = Pm1*(log(qs1/2*Pm2 + g) - log((posrt(Gcs==5)-qrr)*Pm2
+ g));
    dGrt(Gcs==5) = dPm1*(-1./((posrt(Gcs==5)-qrr)*Pm2 + g));
    % Case 7
    Grt(Gcs==7) = Pm1*(log(qs1/2*Pm2+g) - log((2*pi-posrt(Gcs==7)-
qs)*Pm2+g));
    dGrt(Gcs==7) = dPm1*(1./((2*pi-posrt(Gcs==7)-qs)*Pm2 + g));
    % Case 8
    Grt(Gcs>=8) = Gmaxrt*(posrt(Gcs>=8)-2*pi+qs)/qs + ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGrt(Gcs>=8) = Gmaxrt/qs;
end
% Calculate airgap permeances over the rotor slot (inter-polar region)
switch qrscs
  case 1 % qrs <= qs1/2
    Gedges = [0 qrs qs1/2 qs1/2+qrs-aoff (2*S1P-1)*(qs+qs1)+qs1/2+aoff
...
        (2*S1P-1)*(qs+qs1)+qs1/2+qrs (2*S1P-1)*(qs+qs1)+qs1...
        (2*S1P-1)*(qs+qs1)+qs1+qrs 2*pi];
    [ncs,Gcs] = histc(possl,Gedges,2);
    % calculate permeances for non-zero cases (Case 4 P=0)
    % Case 1
    Gsl(Gcs==1) = (Gmaxsl*(qrs - possl(Gcs==1))/qrs) + ...
        Pm1*log((possl(Gcs==1)*Pm2 + g)/g);
```

```
    dGsl(Gcs==1) = -Gmaxsl/qrs + dPm1./(possl(Gcs==1)*Pm2 + g);
    % Case 2
    Gsl(Gcs==2) = Pm1*log((possl(Gcs==2)*Pm2+g)./((possl(Gcs==2)-
qrs)*Pm2+g));
    dGsl(Gcs==2) = dPm1*(1./(possl(Gcs==2)*Pm2 + g) - ...
        1./((possl(Gcs==2)-qrs)*Pm2 + g));
    % Case 3
    Gsl(Gcs==3) = Pm1*(log(qs1/2*Pm2 + g) - log((possl(Gcs==3)-qrs)*Pm2
+ g));
    dGsl(Gcs==3) = dPm1*(-1./((possl(Gcs==3)-qrs)*Pm2 + g));
    % Case 5
    Gsl(Gcs==5) = Pm1*(log(qs1/2*Pm2+g) - log((2*pi-possl(Gcs==5)-
qs)*Pm2+g));
    dGsl(Gcs==5) = dPm1*(1./((2*pi - possl(Gcs==5) - qs)*Pm2 + g));
    % Case 6
    Gsl(Gcs==6) = Pm1*(log((2*pi-possl(Gcs==6)-qs+qrs)*Pm2 + g) - ...
        log((2*pi-possl(Gcs==6)-qs)*Pm2 + g));
    dGsl(Gcs==6) = dPm1*(-1./((2*pi-possl(Gcs==6)-qs+qrs)*Pm2 + g) +
...
        1./((2*pi-possl(Gcs==6)-qs)*Pm2 + g));
    % Case 7
    Gsl(Gcs==7) = Gmaxsl*(possl(Gcs==7)-2*pi+qs)/qrs + ...
        Pm1*log(((2*pi-qs-possl(Gcs==7)+qrs)*Pm2 + g)/g);
    dGsl(Gcs==7) = Gmaxsl/qrs+dPm1*(-1./((2*pi-qs-
possl(Gcs==7)+qrs)*Pm2+g));
    % Case 8
    Gsl(Gcs>=8) = Gmaxsl;
    dGsl(Gcs>=8) = 0;
  case 2 % (qrs > qs1/2) && (qrs <= qs)
    Gedges = [0 qs1/2 qrs qrs+qs1/2-aoff (2*S1P-1)*(qs+qs1) +
qs1/2+aoff ...
        (2*S1P-1)*(qs+qs1) + qs1 (2*S1P-1)*(qs+qs1) + qs1/2 + qrs ...
        (2*S1P-1)*(qs+qs1) + qs1 + qrs max(2*S1P*(qs+qs1),2*pi)];
    [ncs,Gcs] = histc(possl,Gedges,2);
    % calculate permeances for non-zero cases (Case 4 P=0)
    % Case 1
    Gsl(Gcs==1) = (Gmaxsl*(qrs - possl(Gcs==1))/qrs) + ...
        Pm1*log((possl(Gcs==1)*Pm2 + g)/g);
    dGsl(Gcs==1) = -Gmaxsl/qrs + dPm1./(possl(Gcs==1)*Pm2 + g);
    % Case 2
    Gsl(Gcs==2) = (Gmaxsl*(qrs-possl(Gcs==2))/qrs) +
Pm1*log((qs1/2*Pm2+g)/g);
    dGsl(Gcs==2) = -Gmaxsl/qrs;
    % Case 3
    Gsl(Gcs==3) = Pm1*(log(qs1/2*Pm2 + g) - log((possl(Gcs==3)-qrs)*Pm2
+ g));
    dGsl(Gcs==3) = -dPm1./((possl(Gcs==3) - qrs)*Pm2 + g);
    % Case 5
    Gsl(Gcs==5) = Pm1*(log(qs1/2*Pm2+g) - log((2*pi-possl(Gcs==5)-
qs)*Pm2+g));
    dGsl(Gcs==5) = dPm1./((2*pi - possl(Gcs==5) - qs)*Pm2 + g);
    % Case 6
    Gsl(Gcs==6) = (Gmaxsl*(possl(Gcs==6) + qs - 2*pi)/qrs) + ...
        Pm1*log((qs1/2*Pm2 + g)/g);
    dGsl(Gcs==6) = Gmaxsl/qrs;
```

```
    % Case 7
    Gsl(Gcs==7) = (Gmaxsl*(possl(Gcs==7) + qs - 2*pi)/qrs) + ...
        Pm1*log(((2*pi - possl(Gcs==7) - qs + qrs)*Pm2 + g)/g);
    dGsl(Gcs==7) = Gmaxsl/qrs - dPm1./((2*pi-possl(Gcs==7)-qs+qrs)*Pm2
+ g);
    % Case 8
    Gsl(Gcs>=8) = Gmaxsl;
    dGsl(Gcs>=8) = 0;
  case 3 % (qrs > qs) && (qrs <= qs +qs1/2)
    Gedges = [0 qrs-qs qs1/2 qrs qrs+qs1/2-aoff ...
        (2*S1P-1)*(qs+qs1)+qs1/2+aoff (2*S1P-1)*(qs+qs1)+qs1 ...
        (2*S1P-1)*(qs+qs1)+qs1/2+qrs max(2*S1P*(qs+qs1),2*pi)];
    [ncs,Gcs] = histc(possl,Gedges,2);
    % calculate permeances for non-zero cases (Case 5 P=0)
    % Case 1
    Gsl(Gcs==1) = Gmaxsl+Pm1*(log((possl(Gcs==1)*ROD/RP)*slope+g)-
log(g))+...
            Pm1*(log((qrs-qs-possl(Gcs==1))*Pm2 + g) - log(g));
    dGsl(Gcs==1) = dPm1*(-1./((qrs-qs-possl(Gcs==1))*Pm2 + g) + ...
        1./((possl(Gcs==1))*Pm2 + g));
    % Case 2
    Gsl(Gcs==2) = Gmaxsl*(qrs - possl(Gcs==2))/qs + ...
        Pm1*(log((possl(Gcs==2)*ROD/RP)*slope + g) - log(g));
    dGsl(Gcs==2) = -Gmaxsl/qs + dPm1*(1./(possl(Gcs==2)*Pm2 + g));
    % Case 3
    Gsl(Gcs==3) = Gmaxsl*(qrs - possl(Gcs==3))/qs + ...
        Pm1*(log((qs1/2*ROD/RP)*slope + g) - log(g));
    dGsl(Gcs==3) = -Gmaxsl/qs;
    % Case 4
    Gsl(Gcs==4) = Pm1*(log((qs1/2*ROD/RP)*slope + g) - ...
        log((possl(Gcs==4)-qrs)*Pm2 + g));
    dGsl(Gcs==4) = dPm1*(-1./((possl(Gcs==4)-qrs)*Pm2 + g));
    % Case 6
    Gsl(Gcs==6) = Pm1*(log((qs1/2*ROD/RP)*slope + g) - ...
        log((2*pi-qs-possl(Gcs==6))*Pm2 + g));
    dGsl(Gcs==6) = dPm1*(1./((2*pi-qs-possl(Gcs==6))*Pm2 + g));
    % Case 7
    Gsl(Gcs==7) = Gmaxsl*(qs-(2*pi-possl(Gcs==7)))/qs + ...
        Pm1*(log((qs1/2*ROD/RP)*slope + g) - log(g));
    dGsl(Gcs==7) = Gmaxsl/qs;
    % Case 8
    Gsl(Gcs>=8) = Gmaxsl*(qs-(2*pi-possl(Gcs>=8)))/qs + ...
        Pm1*(log((2*pi-qs-possl(Gcs>=8)+qrs)*Pm2 + g) - log(g));
    dGsl(Gcs>=8) = Gmaxsl/qs + dPm1*(-1./((2*pi-qs-
possl(Gcs>=8)+qrs)*Pm2+g));
  case 4 % (qrs > qs+qs1/2) && (qrs <= qs+qs1)
    Gedges = [0 qrs-qs-qs1/2 qs1/2 qrs-qs qrs qs1/2+qrs-aoff ...
        (2*S1P-1)*(qs+qs1)+qs1/2+aoff (2*S1P-1)*(qs+qs1)+qs1 2*pi];
    [ncs,Gcs] = histc(possl,Gedges,2);
    % calculate permeances for non-zero cases (Case 6 P=0)
    % Case 1
    Gsl(Gcs==1) = Gmaxsl + Pm1*(log((possl(Gcs==1))*Pm2 + g) - log(g))
+ ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGsl(Gcs==1) = dPm1*(1./((possl(Gcs==1))*Pm2 + g));
```

```
    % Case 2
    Gsl(Gcs==2) = Gmaxsl + Pm1*(log((possl(Gcs==2))*Pm2 + g) - log(g))
+ ...
        Pm1*(log((qrs-possl(Gcs==2)-qs)*Pm2 + g) - log(g));
    dGsl(Gcs==2) = dPm1*(1./((possl(Gcs==2))*Pm2 + g) - ...
        1./((qrs-possl(Gcs==2)-qs)*Pm2 + g));
    % Case 3
    Gsl(Gcs==3) = Gmaxsl + Pm1*(log(qs1/2*Pm2 + g) - log(g)) + ...
        Pm1*(log((qrs-possl(Gcs==3)-qs)*Pm2 + g) - log(g));
    dGsl(Gcs==3) = dPm1*(-1./((qrs-possl(Gcs==3)-qs)*Pm2 + g));
    % Case 4
    Gsl(Gcs==4) = Gmaxsl*(qrs-possl(Gcs==4))/qs + ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGsl(Gcs==4) = -Gmaxsl/qs;
    % Case 5
    Gsl(Gcs==5) = Pm1*(log(qs1/2*Pm2 + g) - log((possl(Gcs==5)-qrs)*Pm2
+ g));
    dGsl(Gcs==5) = dPm1*(-1./((possl(Gcs==5)-qrs)*Pm2 + g));
    % Case 7
    Gsl(Gcs==7) = Pm1*(log(qs1/2*Pm2+g) - log((2*pi-possl(Gcs==7)-
qs)*Pm2+g));
    dGsl(Gcs==7) = dPm1*(1./((2*pi-possl(Gcs==7)-qs)*Pm2 + g));
    % Case 8
    Gsl(Gcs>=8) = Gmaxsl*(possl(Gcs>=8)-2*pi+qs)/qs + ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGsl(Gcs>=8) = Gmaxsl/qs;
  case 5 % (qrs > qs+qs1)
    Gedges = [0 qs1/2 qrs-qs-qs1/2 qrs-qs qrs qs1/2+qrs-aoff ...
        (2*S1P-1)*(qs+qs1)+qs1/2+aoff (2*S1P-1)*(qs+qs1)+qs1 2*pi];
    [ncs,Gcs] = histc(possl,Gedges,2);
    % calculate permeances for non-zero cases (Case 6 P=0)
    % Case 1
    Gsl(Gcs==1) = Gmaxsl + Pm1*(log((possl(Gcs==1))*Pm2 + g) - log(g))
+ ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGsl(Gcs==1) = dPm1*(1./((possl(Gcs==1))*Pm2 + g));
    % Case 2
    Gsl(Gcs==2) = Gmaxsl + Pm1*(log(qs1/2*Pm2 + g) - log(g)) + ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGsl(Gcs==2) = 0;
    % Case 3
    Gsl(Gcs==3) = Gmaxsl + Pm1*(log(qs1/2*Pm2 + g) - log(g)) + ...
        Pm1*(log((qrs-possl(Gcs==3)-qs)*Pm2 + g) - log(g));
    dGsl(Gcs==3) = dPm1*(-1./((qrs-possl(Gcs==3)-qs)*Pm2 + g));
    % Case 4
    Gsl(Gcs==4) = Gmaxsl*(qrs-possl(Gcs==4))/qs+Pm1*(log(qs1/2*Pm2+g)-
log(g));
    dGsl(Gcs==4) = -Gmaxsl/qs;
    % Case 5
    Gsl(Gcs==5) = Pm1*(log(qs1/2*Pm2 + g) - log((possl(Gcs==5)-qrs)*Pm2
+ g));
    dGsl(Gcs==5) = dPm1*(-1./((possl(Gcs==5)-qrs)*Pm2 + g));
    % Case 7
    Gsl(Gcs==7) = Pm1*(log(qs1/2*Pm2+g) - log((2*pi-possl(Gcs==7)-
qs)*Pm2+g));
```

```
    dGsl(Gcs==7) = dPm1*(1./((2*pi-possl(Gcs==7)-qs)*Pm2 + g));
    % Case 8
    Gsl(Gcs>=8) = Gmaxsl*(possl(Gcs>=8)-2*pi+qs)/qs + ...
        Pm1*(log(qs1/2*Pm2 + g) - log(g));
    dGsl(Gcs>=8) = Gmaxsl/qs;
end
% Construct final matrices
G = [Grt(:,1:SPT) Gsl(:,1:Dslot/2) Grt(:,SPT+1:D)
Gsl(:,Dslot/2+1:Dslot)];
dG=[dGrt(:,1:SPT) dGsl(:,1:Dslot/2) dGrt(:,SPT+1:D)
dGsl(:,Dslot/2+1:Dslot)];
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% May 1, 2009
%-------------------------------------------------------------------------
% J = get_J(Crfe,Ofe,A,mus,areas,dm_dbs,x)
%
% Determines the Jacobian.
%
% OUTPUTS: J      - Jacobian
%
% INPUTS:  Crfe   - Reluctance connection matrix
%          Ofe    - orientation matrix
%          A      - A * x = F
%          mus    - relative permeability corresponding to iron
branches
%          areas  - areas of the iron branches
%          dm_dbs - derivative of relative permeabilities
%          x      - mesh fluxes
%-------------------------------------------------------------------------
function J = get_J(Crfe,Ofe,A,mus,areas,dm_dbs,x)
% IRON
% Stator yoke - S
% Stator teeth - S
% Rotor yoke below the slot - 1
% Rotor tooth shank - 1
% Rotor yoke connected to shank - 2
% Rotor tooth tips radial - (D - 4*NRrtrt)
% Rotor tooth to rotor tooth tangential - 4*NRrtrt
% Rotor tooth tangential at sides of tooth tips - 4
% AIR
% Stator tooth leakage - S
% Field winding leakage - 2
% Middle rotor slot leakage - 1
% Fringing permeance from rotor side to airgap boundary - Dsl
% Remaining air gap terms - Nam --air
% Build Jacobian using building algorithm
pA = zeros(size(A));
ind1 = abs(Crfe(:,2));
ind2 = abs(Crfe(:,3));
dRdPhi = -Crfe(:,1).*dm_dbs./(mus.*areas);
for i=1:length(Crfe)
    if ind1(i)*ind2(i)>0 %&& ind2(i)>0
        neg = sign(Crfe(i,2)*Crfe(i,3));
        pA(ind1(i),ind1(i)) = pA(ind1(i),ind1(i)) + ...
          dRdPhi(i)*Ofe(i,ind1(i))*(x(ind1(i)) - neg*x(ind2(i)));
        pA(ind2(i),ind2(i)) = pA(ind2(i),ind2(i)) + ...
          dRdPhi(i)*Ofe(i,ind2(i))*(x(ind2(i)) - neg*x(ind1(i)));
        pA(ind1(i),ind2(i)) = pA(ind1(i),ind2(i)) + ...
```

```
          dRdPhi(i)*Ofe(i,ind2(i))*(x(ind1(i)) - neg*x(ind2(i)));
        pA(ind2(i),ind1(i)) = pA(ind1(i),ind2(i));
    elseif ind1(i)>0
        neg = sign(Crfe(i,2));
        pA(ind1(i),ind1(i)) = pA(ind1(i),ind1(i)) + ...
          dRdPhi(i)*Ofe(i,ind1(i))*neg*x(ind1(i));
    elseif ind2(i)>0
        neg = sign(Crfe(i,3));
        pA(ind2(i),ind2(i)) = pA(ind2(i),ind2(i)) + ...
          dRdPhi(i)*Ofe(i,ind2(i))*neg*x(ind2(i));
    end
end
J = A+pA;
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% Apr 1, 2013
%-------------------------------------------------------------------------
% [A,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Cr,Cvconn)
%
% Builds the matrices A and d used to solve for flux. Outputs Cr for
use by
% get_J.m
%
% OUTPUTS: A,d      - matrices describing the MEC system, A*x =
d*current
%          Cr       - connection matrix complete with reluctances
%
% INPUTS:  Rair     - air reluctances
%          PTClist  - air gap permeances
%          Riron    - iron reluctance
%          parx     - simulation parameters
%          pars     - geometry parameters
%          Nabcf    - matrix of stator and rotor conductor turns
%          Cr       - Reluctance connection matrix
%          Cvconn   - mmf source connection matrix
%-------------------------------------------------------------------------
function [A,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Cr,Cvconn)
% ------------------------------------------------------------------------
%PARAMETERS
S     = parx(3)/pars(28); % Number of stator teeth per pole
SPT   = parx(2);
Dsl   = 4*parx(29);
Nldp  = SPT-1; % Number of damper leakage meshes
Nm    = 3 + S + length(PTClist) + Nldp;  % Total number of meshes
% ---------------------------------------
% Determine connection matrix size
% Nrym = 3;               % Number of rotor yoke meshes
% Nsm = S;                % Number of stator tooth meshes
% Nam = length(PTClist);  % Number of air gap meshes
% Connection matrix reluctances
% IRON
% Stator yoke - S
% Stator shank - S
% Stator teeth - S
% Rotor yoke below the slot - 1
% Rotor tooth shank - 1
% Damper bar in Rotor tooth shank - damper_nshank
% Rotor yoke connected to shank - 2
% Rotor tooth tips radial - (D - 4*NRrtrt)/2
```

```
% Damper windings in Rotor tooth tips radial - (D - 4*NRrtrt)/2
% Rotor tooth to rotor tooth tangential - 4*NRrtrt/2
% Damper windings in Rotor tooth to rotor tooth tangential - 4*NRrtrt/2
% Leakage of rotor pole tip - Nldp
% Rotor tooth tangential at sides of tooth tips - 4/2
% AIR
% Stator tooth leakage - S
% Field winding leakage - 2
% Middle rotor slot leakage - 2/2
% Fringing permeance from rotor side to airgap boundary - Dsl/2
% Fringing permeance from rotor slot side to bottom of tooth tip - 4/2
% Airgap - Nam
% ------------------------------------------------------------------
% Combine the rotor pole tip leakage in the iron and air
Riron(end-2-Nldp+1:end-2) = ...
    1./(1./Riron(end-2-Nldp+1:end-2) + 1./Rair(end-Nldp+1:end));
Rair(end-Nldp+1:end) = Rair(end-Nldp+1:end)*0;
%-----------------------------------
% Add reluctances to connection matrix
RTC = 1./PTClist;
Cr(:,1) = [Riron;Rair;RTC];
% ------------------------------------------------------------------
% Find A using building algorithm
A = zeros(Nm);
ind1 = abs(Cr(:,2));
ind2 = abs(Cr(:,3));
pm   = sign(Cr(:,3).*Cr(:,2));
for i=1:length(Cr)
    if ind1(i)*ind2(i)>0
        A(ind1(i),ind1(i)) = A(ind1(i),ind1(i))+Cr(i,1);
        A(ind2(i),ind2(i)) = A(ind2(i),ind2(i))+Cr(i,1);
        A(ind1(i),ind2(i)) = A(ind1(i),ind2(i))-pm(i)*Cr(i,1);
        A(ind2(i),ind1(i)) = A(ind2(i),ind1(i))-pm(i)*Cr(i,1);
    elseif ind1(i)>0
        A(ind1(i),ind1(i)) = A(ind1(i),ind1(i))+Cr(i,1);
    elseif ind2(i)>0
        A(ind2(i),ind2(i)) = A(ind2(i),ind2(i))+Cr(i,1);
    end
end
% ------------------------------------
% Find d: d = zeros(Nm,NPH+1);
d = zeros(Nm,4);
d(1:S+2,:) = Nabcf;
d(S+1,:) = d(S+1,:)*sign(Cvconn(S+1));
d(S+2,:) = d(S+2,:)*sign(Cvconn(S+2));
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% Apr 1, 2013
%-------------------------------------------------------------------------
% [Crconn,Cvconn,O,PTCind,d_damper_1,d_damper_2,index,flag] =
%
shape_alg(PTC,parx,pars,damperdata,Crcfixed,Cvcfixed,rtid,index_old,fla
g_old)
%
% Determines the mesh connections for each reluctance and mmf source
for a
% given rotor position.  The first column of the connection matrics is
left
% as zero and is later updated with the specific reluctance/source
value.
%
% OUTPUTS: Crconn - reluctance connections for the MEC mesh
%          Cvconn - mmf source connections for the MEC mesh
%          O      - orientation matrix: FLUX = O * mesh_flux, where
FLUX is
%          the flux through a reluctance
%          PTCind - ordered indices of the relevant airgap permeances
%          d_damper_1 - represents MMF of damper currents
%          d_damper_2 - relates loop fluxes and the flux linkage
crossing each of two dampers
%          index, flag - identify poles crossing
%
% INPUTS:  PTC    - Permeances in the air gap (S x D)
%          parx   - machine parameters
%          pars   - machine parameters
%          damperdata - informations of damper bars
%          Crcfixed - reluctance connections that do not change
%          Cvcfixed - mmf source connections
%          rtid   - vector identifying type of node in each rotor
section
%-------------------------------------------------------------------------
function [Crconn,Cvconn,O,PTCind,d_damper_1,d_damper_2,index,flag] =
shape_alg(PTC,parx,pars,damperdata,Crcfixed,Cvcfixed,rtid,index_old,fla
g_old)
%PARAMETERS
SPT     = parx(2);
SL      = parx(3);
RP      = pars(28);
S       = SL/RP;
D       = 2*SPT;
SPAIR   = parx(29);
Dsl     = 4*parx(29);
NRrtrt  = parx(27);
```

```
Nrtt    = D - 4*NRrtrt;
damper_rtip = damperdata.damper_rtip;
damper_ntip = damperdata.damper_ntip;
damper_nshank = damperdata.damper_nshank;
bartype = damperdata.bartype;
% ------------------------------------------------------------------------
% Build source connections
Cvconn = Cvcfixed;
% ------------------------------------------------------------------------
% Based on airgap permeances, determine if a reordering is necessary
and if
% the rotor source mmf is negative. (Necessary because of single pole
% model.)
% st contains list of stator teeth with connections in order
% and rt is the corresponding list of rotor teeth sections.
% PTCind contains the indices of the non-zero permeances in the correct
% order.
[rt,st] = find(PTC');
rtup    = sort(rt,1,'ascend');
rtdown  = sort(rt,1,'descend');
if ~isequal(rt,rtup) && ~isequal(rt,rtdown)
    PTCnew = [PTC(:,(D+Dsl)/2+1:Dsl+D) PTC(:,1:(D+Dsl)/2)];
    [rt,st] = find(PTCnew');
    PTCind1 = find([zeros(S,(D+Dsl)/2) PTC(:,(D+Dsl)/2+1:D+Dsl)]');
    PTCind2 = find([PTC(:,1:(D+Dsl)/2) zeros(S,(D+Dsl)/2)]');
    PTCind = [PTCind1;PTCind2];
    Cvconn(S+1) = -(S+1);
    Cvconn(S+2) = -(S+2);
elseif(rt(1)==(D+Dsl)/2+1)
    Cvconn(S+1) = -(S+1);
    Cvconn(S+2) = -(S+2);
    PTCind = find(PTC');
else
    PTCind = find(PTC');
end
% ------------------------------------------------------------------------
% Build reluctance connections
% Determine connection matrix size
% Nrym = 3;               % Number of rotor yoke meshes
% Nsm = S;                % Number of stator tooth meshes
Nam = length(rt);  % Number of air gap meshes
Nldp = SPT-1; % Number of damper leakage meshes
Nm = 3 + S + Nam + Nldp;  % Total number of meshes
% Initialize matrix
Crconn = [Crcfixed;zeros(Nam,3)];
% Connection matrix reluctances
% IRON
% Stator yoke - S
% Stator teeth - S
% Rotor yoke below the slot - 1
% Rotor tooth shank - 1
% Rotor yoke connected to shank - 2
% Rotor tooth tips radial - (D - 4*NRrtrt)
% Rotor tooth to rotor tooth tangential - 4*NRrtrt
% Rotor tooth tangential at sides of tooth tips - 4
```

```
% AIR
% Stator tooth leakage - S
% Field winding leakage - 2
% Middle rotor slot leakage - 2
% Fringing permeance from rotor side to airgap boundary - Dsl
% Fringing permeance from rotor slot side to bottom of tooth tip - 4
% Remaining air gap terms - Nam
% Indexing variables and other terms used in algorithm
rtcs = S*3+16+D+Dsl;    % start index for air gap reluct
rt1s = 2*S+4;           % start index for radial rotor tooth reluct
rt2s = 2*S+4+D-4*NRrtrt;% start index for tangential rotor tooth reluct
rt34s = S*3+12+D;       % start index for fringing permeances
rtrts = 2*S+4+D;        % start index for side tangential rotor tooth
reluct
rfrbs = 3*S+12+D+Dsl;   % start index for rotor fringing to the bottom
of the tooth tip
rtposs = 1:D+Dsl;       % List of possible rotor nodes connecting to
airgap reluctances
rt1 = rtposs(rtid==1);  % Rotor nodes corresponding to RRTIN
rt2 = rtposs(rtid==2);  % Rotor nodes corresponding to RRTOUT
rt34 = rtposs(rtid>2);  % Rotor nodes corresponding to RAGFR
rt5 = rt2([1 2*NRrtrt 2*NRrtrt+1 end]); % Rotor nodes corresponding to
RRTS
NRTBD = length(find(Crconn(rt2s+1:rt2s+NRrtrt,2)==0)); % number of
RRTOUT branches with both meshes unknown
% Connections in stator and rotor which depend on airgap config but do
not
% rely on shape algorithm
% RSTL connection (stator tooth leakage)
Crconn(3*S+8+D,3) = S+3+Nam;
% RAG Connections (Air gap reluctances)
Crconn(rtcs+1:rtcs+Nam,3) = (S+4:Nam+S+3)';
Crconn(rtcs+1:rtcs+Nam,2) = [-(Nam+S+3) S+4:Nam+S+2]';
%-----------------------------------------------------------------
% PROCESS SHAPES
for i=1:Nam-1
% Current mesh (loop flux) to be assigned to a reluctance
currm = S+3+i;
% Condition for the reluctance to be connected to a negative loop
neg = 1-2*(rt(1)+(D+Dsl)/2==rt(i+1));
  if st(i)==st(i+1) && rt(i)+1==rt(i+1)
    % Base down triangle ------------------------------------------
    bdt_cs = rtid(rt(i))*10 + rtid(rt(i+1));
    switch bdt_cs
      case 11
          % Connecting to 2 radial rotor tooth branches
          Crconn(rt1s+find(rt1==rt(i)),3) = currm;
          Crconn(rt1s+find(rt1==rt(i+1)),2) = neg*currm;
      case 12
          % Connecting to a radial branch and a tangential branch
          Crconn(rt1s+find(rt1==rt(i)),3) = currm;
          Crconn(rt2s+find(rt2==rt(i+1)),3) = currm;
      case 22
          % Connecting to 2 tangential branches
          rt2off = rt(i)>D/4&&rt(i)<(D+Dsl)/2 || rt(i)>D*3/4+Dsl/2;
```

```
            Crconn(rt2s+find(rt2==rt(i))+rt2off,3) = currm;
        case 21
            % Connecting to a tangential branch and a radial branch
            Crconn(rt2s+find(rt2==rt(i)),3) = currm;
            Crconn(rt1s+find(rt1==rt(i+1)),2) = neg*currm;
        case 23
            % Connecting to tangential branch & fringing to outer edge
            Crconn(rtrts+find(rt5==rt(i)),3) = currm;
            Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
        case 33
            % Connecting to 2 fringing branch
            rt3i = find(rt34==rt(i));
            if mod(rt3i,2*SPAIR) == SPAIR
                Crconn(3*S+10+D+ceil(rt3i/(2*SPAIR)),3) =  currm;
                if rt3i<2*SPAIR
                    Crconn(rt2s+(2*NRrtrt-NRTBD+1:2*NRrtrt+NRTBD)',2) =
currm*ones(2*NRTBD,1);
                    Crconn(rt2s+(4*NRrtrt-NRTBD+1:4*NRrtrt)',2) = -
currm*ones(NRTBD,1);
                    Crconn(rt2s+(1:NRTBD)',2) = -currm*ones(NRTBD,1);
                    Crconn(rtrts+(2:3)',2) = [currm;currm];
                    Crconn(rtrts+[1;4],2) = [-currm;-currm];
                    Crconn(rfrbs+(2:3)',3) = [currm;currm];
                else
                    Crconn(rt2s+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                    Crconn(rt2s+(4*NRrtrt-NRTBD+1:4*NRrtrt)',2) =
currm*ones(NRTBD,1);
                    Crconn(rt2s+(2*NRrtrt-NRTBD+1:2*NRrtrt+NRTBD)',2) = -
currm*ones(2*NRTBD,1);
                    Crconn(rtrts+[1;4],2) = [currm;currm];
                    Crconn(rtrts+(2:3)',2) = [-currm;-currm];
                    Crconn(rfrbs+[1;4]',3) = [currm;currm];
                end
            end
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
        case 34
            % Connecting to fringing going to edge and bottom
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
            whichtt = ceil(find(rt34==rt(i))/SPAIR)+1;
            whichtt = whichtt*(whichtt<=4)+(whichtt>4); % if-else
            Crconn(rtrts+whichtt,2) = currm;
            Crconn(rfrbs+whichtt,3) = currm;
            if whichtt==1
                Crconn(rt2s+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+3,2) = -currm;
            elseif whichtt==2
                Crconn(rt2s+2*NRrtrt-NRTBD+(1:NRTBD)',2) =
currm*ones(NRTBD,1);
                Crconn(rt2s+4*NRrtrt-NRTBD+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+4,2) = -currm;
```

```
            elseif whichtt==3
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                Crconn(rt2s+(1:NRTBD)',2) = -currm*ones(NRTBD,1);
                Crconn(rtrts+1,2) = -currm;
            else
                Crconn(rt2s+4*NRrtrt-NRTBD+(1:NRTBD)',2) =
currm*ones(NRTBD,1);
                Crconn(rt2s+2*NRrtrt-NRTBD+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+2,2) = -currm;
            end
        case 32
            % Connecting to fringing and tangential branch
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rtrts+find(rt5==rt(i+1)),3) = currm;
        case 44
            % Connecting to 2 fringing paths both going to bottom
            rt4i = find(rt34==rt(i));
            if mod(rt4i,2*SPAIR) == SPAIR
                Crconn(3*S+10+D+ceil(rt4i/(2*SPAIR)),3) =  currm;
            end
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
        case 43
            % Connecting to fringing to bottom and edge
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
            whichtt = ceil(find(rt34==rt(i))/SPAIR)+1;
            whichtt = whichtt*(whichtt<=4)+(whichtt>4); % if-else
            Crconn(rtrts+whichtt,2) = currm;
            Crconn(rfrbs+whichtt,3) = currm;
            if whichtt==1
                Crconn(rt2s+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+3,2) = -currm;
            elseif whichtt==2
                Crconn(rt2s+2*NRrtrt-NRTBD+(1:NRTBD)',2) =
currm*ones(NRTBD,1);
                Crconn(rt2s+4*NRrtrt-NRTBD+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+4,2) = -currm;
            elseif whichtt==3
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                Crconn(rt2s+(1:NRTBD)',2) = -currm*ones(NRTBD,1);
                Crconn(rtrts+1,2) = -currm;
            else
                Crconn(rt2s+4*NRrtrt-NRTBD+(1:NRTBD)',2) =
currm*ones(NRTBD,1);
                Crconn(rt2s+2*NRrtrt-NRTBD+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+2,2) = -currm;
            end
    end
  elseif rt(i)==rt(i+1) && st(i)+1==st(i+1)
```

```
    % Base up triangle ------------------------------------------
    Crconn(2*S+8+D+st(i),3) = currm;
  elseif rt(i)+1==rt(i+1) && st(i+1)>=st(i)+1
    % Four-sided polygon -----------------------------------------
    bdt_cs = rtid(rt(i))*10 + rtid(rt(i+1));
    switch bdt_cs
      case 11
          % Connecting to 2 radial rotor tooth branches
          Crconn(rt1s+find(rt1==rt(i)),3) = currm;
          Crconn(rt1s+find(rt1==rt(i+1)),2) = neg*currm;
      case 12
          % Connecting to a radial branch and a tangential branch
          Crconn(rt1s+find(rt1==rt(i)),3) = currm;
          Crconn(rt2s+find(rt2==rt(i+1)),3) = currm;
      case 22
          % Connecting to 2 tangential branches
          rt2off = rt(i)>D/4&&rt(i)<(D+Dsl)/2 || rt(i)>D*3/4+Dsl/2;
          Crconn(rt2s+find(rt2==rt(i))+rt2off,3) = currm;
      case 21
          % Connecting to a tangential branch and a radial branch
          Crconn(rt2s+find(rt2==rt(i)),3) = currm;
          Crconn(rt1s+find(rt1==rt(i+1)),2) = neg*currm;
      case 23
          % Connecting to tangential branch & fringing to outer edge
          Crconn(rtrts+find(rt5==rt(i)),3) = currm;
          Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
      case 33
          % Connecting to 2 fringing branch
          rt3i = find(rt34==rt(i));
          if mod(rt3i,2*SPAIR) == SPAIR
              Crconn(3*S+10+D+ceil(rt3i/(2*SPAIR)),3) =  currm;
              if rt3i<2*SPAIR
                  Crconn(rt2s+(2*NRrtrt-NRTBD+1:2*NRrtrt+NRTBD)',2) =
currm*ones(2*NRTBD,1);
                  Crconn(rt2s+(4*NRrtrt-NRTBD+1:4*NRrtrt)',2) = -
currm*ones(NRTBD,1);
                  Crconn(rt2s+(1:NRTBD)',2) = -currm*ones(NRTBD,1);
                  Crconn(rtrts+(2:3)',2) = [currm;currm];
                  Crconn(rtrts+[1;4],2) = [-currm;-currm];
                  Crconn(rfrbs+(2:3)',3) = [currm;currm];
              else
                  Crconn(rt2s+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                  Crconn(rt2s+(4*NRrtrt-NRTBD+1:4*NRrtrt)',2) =
currm*ones(NRTBD,1);
                  Crconn(rt2s+(2*NRrtrt-NRTBD+1:2*NRrtrt+NRTBD)',2) = -
currm*ones(2*NRTBD,1);
                  Crconn(rtrts+[1;4],2) = [currm;currm];
                  Crconn(rtrts+(2:3)',2) = [-currm;-currm];
                  Crconn(rfrbs+[1;4]',3) = [currm;currm];
              end
          end
          Crconn(rt34s+find(rt34==rt(i)),3) = currm;
          Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
      case 34
          % Connecting to fringing going to edge and bottom
```

```
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
            whichtt = ceil(find(rt34==rt(i))/SPAIR)+1;
            whichtt = whichtt*(whichtt<=4)+(whichtt>4); % if-else
            Crconn(rtrts+whichtt,2) = currm;
            Crconn(rfrbs+whichtt,3) = currm;
            if whichtt==1
                Crconn(rt2s+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+3,2) = -currm;
            elseif whichtt==2
                Crconn(rt2s+2*NRrtrt-NRTBD+(1:NRTBD)',2) =
currm*ones(NRTBD,1);
                Crconn(rt2s+4*NRrtrt-NRTBD+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+4,2) = -currm;
            elseif whichtt==3
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                Crconn(rt2s+(1:NRTBD)',2) = -currm*ones(NRTBD,1);
                Crconn(rtrts+1,2) = -currm;
            else
                Crconn(rt2s+4*NRrtrt-NRTBD+(1:NRTBD)',2) =
currm*ones(NRTBD,1);
                Crconn(rt2s+2*NRrtrt-NRTBD+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+2,2) = -currm;
            end
        case 32
            % Connecting to fringing and tangential branch
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rtrts+find(rt5==rt(i+1)),3) = currm;
        case 44
            % Connecting to 2 fringing paths both going to bottom
            rt4i = find(rt34==rt(i));
            if mod(rt4i,2*SPAIR) == SPAIR
                Crconn(3*S+10+D+ceil(rt4i/(2*SPAIR)),3) =  currm;
            end
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
        case 43
            % Connecting to fringing to bottom and edge
            Crconn(rt34s+find(rt34==rt(i)),3) = currm;
            Crconn(rt34s+find(rt34==rt(i+1)),2) = neg*currm;
            whichtt = ceil(find(rt34==rt(i))/SPAIR)+1;
            whichtt = whichtt*(whichtt<=4)+(whichtt>4); % if-else
            Crconn(rtrts+whichtt,2) = currm;
            Crconn(rfrbs+whichtt,3) = currm;
            if whichtt==1
                Crconn(rt2s+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+3,2) = -currm;
            elseif whichtt==2
```

```
                Crconn(rt2s+2*NRrtrt-NRTBD+(1:NRTBD)',2) =
currm*ones(NRTBD,1);
                Crconn(rt2s+4*NRrtrt-NRTBD+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+4,2) = -currm;
            elseif whichtt==3
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = currm*ones(NRTBD,1);
                Crconn(rt2s+(1:NRTBD)',2) = -currm*ones(NRTBD,1);
                Crconn(rtrts+1,2) = -currm;
            else
                Crconn(rt2s+4*NRrtrt-NRTBD+(1:NRTBD)',2) =
currm*ones(NRTBD,1);
                Crconn(rt2s+2*NRrtrt-NRTBD+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rtrts+2,2) = -currm;
            end
        end
        Crconn(2*S+8+D+st(i):2*S+8+D+st(i+1)-1,3) = currm;
    end
end
% PROCESS BOUNDARY SHAPE
currm = S+3+Nam; % final airgap loop
if rt(1)+D/2+Dsl/2~=rt(Nam) % First and last airgap reluct not
connected to the same rotor tooth
    % Base-down triangle or four-sided polygon
    neg = -1;
    bdtbound_cs = rtid(rt(Nam))*10 + rtid(rt(1));
    switch bdtbound_cs
        case 11
            % Connecting to 2 radial rotor tooth branches
            Crconn(rt1s+find(rt1==rt(Nam)),3) = currm;
            Crconn(rt1s+find(rt1==rt(1)),2) = neg*currm;
        case 12
            % Connecting to a radial branch and a tangential branch
            Crconn(rt1s+find(rt1==rt(Nam)),3) = currm;
            Crconn(rt2s+find(rt2==rt(1)),3) = neg*currm;
        case 22
            % Connecting to 2 tangential branches
            rt2off = rt(Nam)>D/4&&rt(Nam)<(D+Dsl)/2 ||
rt(Nam)>D*3/4+Dsl/2;
            Crconn(rt2s+find(rt2==rt(Nam))+rt2off,3) = currm;
        case 21
            % Connecting to a tangential branch and a radial branch
            Crconn(rt2s+find(rt2==rt(Nam)),3) = currm;
            Crconn(rt1s+find(rt1==rt(1)),2) = neg*currm;
        case 23
            % Connecting to tangential branch & fringing to outer edge
            Crconn(rtrts+find(rt5==rt(Nam)),3) = currm;
            Crconn(rt34s+find(rt34==rt(1)),2) = neg*currm;
        case 33
            % Connecting to 2 fringing branch
            rt3i = find(rt34==rt(Nam));
            if mod(rt3i,2*SPAIR) == SPAIR
                Crconn(3*S+10+D+ceil(rt3i/(2*SPAIR)),3) =  currm;
                Crconn(rfrbs+[1;4],3) = [currm;currm];
```

```
                Crconn(rtrts+[1;4],2) = [currm;currm];
                Crconn(rtrts+[2;3],2) = [-currm;-currm];
                Crconn(rt2s+2*NRrtrt+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
                Crconn(rt2s+(4*NRrtrt-NRTBD+1:4*NRrtrt)',2) =
currm*ones(NRTBD,1);
            end
            Crconn(rt34s+find(rt34==rt(Nam)),3) = currm;
            Crconn(rt34s+find(rt34==rt(1)),2) = neg*currm;
        case 34
            % Connecting to fringing going to edge and bottom
            Crconn(rt34s+find(rt34==rt(Nam)),3) = currm;
            Crconn(rt34s+find(rt34==rt(1)),2) = neg*currm;
            whichtt = ceil(find(rt34==rt(Nam))/SPAIR)+1;
            whichtt = whichtt*(whichtt<=4)+(whichtt>4); % if-else
            Crconn(rtrts+whichtt,2) = currm;
            Crconn(rfrbs+whichtt,3) = currm;
            Crconn(rt2s+(4*NRrtrt-NRTBD+1:4*NRrtrt)',2) =
currm*ones(NRTBD,1);
        case 32
            % Connecting to fringing and tangential branch
            Crconn(rt34s+find(rt34==rt(Nam)),3) = currm;
            if rt(1)>(D+Dsl)/2
                Crconn(rtrts+find(rt5==(rt(1)-(D+Dsl)/2)),3) = currm;
            else
                Crconn(rtrts+find(rt5==(rt(1)+(D+Dsl)/2)),3) = currm;
            end
        case 44
            % Connecting to 2 fringing paths both going to bottom
            rt4i = find(rt34==rt(Nam));
            if mod(rt4i,2*SPAIR) == SPAIR
                Crconn(3*S+10+D+ceil(rt4i/(2*SPAIR)),3) =  currm;
            end
            Crconn(rt34s+find(rt34==rt(Nam)),3) = currm;
            Crconn(rt34s+find(rt34==rt(1)),2) = neg*currm;
        case 43
            % Connecting to fringing to bottom and edge
            Crconn(rt34s+find(rt34==rt(Nam)),3) = currm;
            Crconn(rt34s+find(rt34==rt(1)),2) = neg*currm;
            whichtt = ceil(find(rt34==rt(1))/SPAIR)+1;
            whichtt = whichtt*(whichtt<=4)+(whichtt>4);
            Crconn(rtrts+whichtt,2) = neg*currm;
            Crconn(rfrbs+whichtt,3) = neg*currm;
            Crconn(rt2s+(whichtt-1)*NRrtrt+(1:NRTBD)',2) = -
currm*ones(NRTBD,1);
    end
else
    % Base-up triangle
    if rtid(rt(1)) == 1
        Crconn(rt1s+find(rt1==rt(1)),2) =
Crconn(rt1s+find(rt1==rt(Nam)),2);
    elseif rtid(rt(1)) > 2
        Crconn(rt34s+find(rt34==rt(1)),2) =
Crconn(rt34s+find(rt34==rt(Nam)),2);
    end
```

```
end
% GET RID OF UNUSED ROTOR TEETH RELUCTANCES IN Crconn (D/2 positions)
% Cut down rotor reluctances to one pole instead of a pole pair
remov = (Crconn(:,2)==0 | Crconn(:,3) ==0) &
[zeros(2*S+4,1);ones(D+4+S+8+Dsl+Nam,1)];
firstpole = [zeros(2*S+4,1);ones(D/2-2*NRrtrt,1);zeros(D/2-
2*NRrtrt,1);ones(2*NRrtrt,1);zeros(2*NRrtrt,1);zeros(S+2+4+2,1);ones(Ds
l/2,1);zeros(Dsl/2+4+Nam,1)];
secpole = [zeros(2*S+4,1);zeros(D/2-2*NRrtrt,1);ones(D/2-
2*NRrtrt,1);zeros(2*NRrtrt,1);ones(2*NRrtrt,1);zeros(S+6+2,1);zeros(Dsl
/2,1);ones(Dsl/2,1);zeros(4+Nam,1)];
Crconn(firstpole&remov,:)=Crconn(secpole&~remov,:);
Crconn(secpole&secpole|remov&~firstpole,:) = [];
% Crconn is ordered such that the flux through a reluctance branch is
equal
% to the loop flux in column 2 - loop flux in column 3

% ------------------------------------------------------------------
% Crconn matrix postprocess to incorporate the branches of
% stator tooth tip, damper slots, and leakage of damper slots
% ------------------------------------------------------------------
% Add branches for stator tooth tip ---------------------------------
Crconn_stt = Crconn(S+1:2*S,:);
Crconn_temp_1 = Crconn(1:2*S,:);
Crconn_temp_2 = Crconn(2*S+1:end,:);
Crconn = [Crconn_temp_1;Crconn_stt;Crconn_temp_2];

% Add branches for damper slots in shank ----------------------------
if damper_nshank == 0
    Crconn_shank = [];
else
    for i = 1:damper_nshank
        Crconn_shank(i,:) = Crconn(3*S+2,:);
    end
end
Crconn_temp_1 = Crconn(1:3*S+2,:);
Crconn_temp_2 = Crconn(3*S+3:end,:);
Crconn = [Crconn_temp_1;Crconn_shank;Crconn_temp_2];

% Add branches for damper slots on tip ------------------------------
Crconn_in = Crconn(3*S+4+damper_nshank+1:3*S+4+damper_nshank+Nrtt/2,:);
Crconn_out =
Crconn(3*S+4+damper_nshank+Nrtt/2+1:3*S+4+damper_nshank+D/2,:);
Crconn_temp_1 = Crconn(1:3*S+4+damper_nshank+Nrtt/2,:);
Crconn_temp_2 = Crconn(3*S+4+damper_nshank+Nrtt/2+1:end,:);
Crconn = [Crconn_temp_1;Crconn_in;Crconn_out;Crconn_temp_2];

% Add branches for leakage path of rotor pole tip iron--------------
rt1s = 3*S+4+damper_nshank;
ldp_start = 3*S+4+damper_nshank+D;
Crconn_ldp = zeros(Nldp,3);

for i = 1:ceil(Nldp/2)
    if i <= Nrtt/4
```

```
        c1 = sign(Crconn_in(Nrtt/4-i+1,3))*(S+3+Nam+ceil(Nldp/2)-i+1);
        c2 = sign(Crconn_in(Nrtt/4-i+2,2))*(S+3+Nam+ceil(Nldp/2)-i+1);
        Crconn(rt1s+Nrtt/4-i+1,3) = c1;
        Crconn(rt1s+Nrtt/4-i+2,2) = c2;
        Crconn_ldp(ceil(Nldp/2)-i+1,2) = c1;
        Crconn_ldp(ceil(Nldp/2)-i+1,3) = Crconn_in(Nrtt/4-i+1,3);

        c1 = sign(Crconn_in(Nrtt/4+i-1,3))*(S+3+Nam+floor(Nldp/2)+i);
        c2 = sign(Crconn_in(Nrtt/4+i,2))*(S+3+Nam+floor(Nldp/2)+i);
        Crconn(rt1s+Nrtt/4+i-1,3) = c1;
        Crconn(rt1s+Nrtt/4+i,2) = c2;
        Crconn_ldp(floor(Nldp/2)+i,2) = c1;
        Crconn_ldp(floor(Nldp/2)+i,3) = Crconn_in(Nrtt/4+i-1,3);
    else
        c1 = sign(Crconn_out(length(damper_rtip)-
i+1,3))*(S+3+Nam+ceil(Nldp/2)-i+1);
        Crconn(rt1s+D-4*NRrtrt+length(damper_rtip)-i+1,3) = c1;
        Crconn_ldp(ceil(Nldp/2)-i+1,2) = c1;
        Crconn_ldp(ceil(Nldp/2)-i+1,3) =
Crconn_out(length(damper_rtip)-i+1,3);

        c1 = sign(Crconn_out(2*NRrtrt-(length(damper_rtip)-
i),3))*(S+3+Nam+floor(Nldp/2)+i);
        Crconn(rt1s+D-4*NRrtrt+2*NRrtrt-(length(damper_rtip)-i),3) =
c1;
        Crconn_ldp(floor(Nldp/2)+i,2) = c1;
        Crconn_ldp(floor(Nldp/2)+i,3) = Crconn_out(2*NRrtrt-
(length(damper_rtip)-i),3);

        if i == Nrtt/4+1
            Crconn(rt1s+1,2) =
sign(Crconn_in(1,2))*Crconn_ldp(ceil(Nldp/2)-i+1,2);
            Crconn(rt1s+Nrtt/2,3) =
sign(Crconn_in(Nrtt/2,3))*Crconn_ldp(floor(Nldp/2)+i,2);
        end
    end
end
Crconn(rt1s+Nrtt/2+(1:Nrtt/2),:) = Crconn(rt1s+(1:Nrtt/2),:);
Crconn(rt1s+D-2*NRrtrt+(1:2*NRrtrt),:) = Crconn(rt1s+D-
4*NRrtrt+(1:2*NRrtrt),:);
Crconn_temp_1 = Crconn(1:ldp_start,:);
Crconn_temp_2 = Crconn(ldp_start+1:end,:);
Crconn = [Crconn_temp_1;Crconn_ldp;Crconn_temp_2];

% Add branches for leakage path of rotor pole tip air-----------------
Crconn_temp_1 = Crconn(1:ldp_start+Nldp+S+Dsl/2+7,:);
Crconn_temp_2 = Crconn(ldp_start+Nldp+S+Dsl/2+7+1:end,:);
Crconn = [Crconn_temp_1;Crconn_ldp;Crconn_temp_2];

% Final output connection matrix Crconn ------------------------------
% IRON
% Stator yoke - S
% Stator shank - S
% Stator teeth - S
```

```
% Rotor yoke below the slot - 1
% Rotor tooth shank - 1
% Damper bar in Rotor tooth shank - damper_nshank
% Rotor yoke connected to shank - 2
% Rotor tooth tips radial - (D - 4*NRrtrt)/2
% Damper windings in Rotor tooth tips radial - (D - 4*NRrtrt)/2
% Rotor tooth to rotor tooth tangential - 4*NRrtrt/2
% Damper windings in Rotor tooth to rotor tooth tangential - 4*NRrtrt/2
% Leakage of rotor pole tip - Nldp
% Rotor tooth tangential at sides of tooth tips - 4/2
% AIR
% Stator tooth leakage - S
% Field winding leakage - 2
% Middle rotor slot leakage - 2/2
% Fringing permeance from rotor side to airgap boundary - Dsl/2
% Fringing permeance from rotor slot side to bottom of tooth tip - 4/2
% Airgap - Nam

% ------------------------------------------------------------------------
% Create a matrix O such that [branch flux] = O*[loop flux]
Osize = [length(Crconn),Nm];
O = zeros(Osize)';
% vec_ind used to convert indexing to one long vector, instead of using
% (row,col) indexing
vec_ind = (0:Osize(2):Osize(2)*(Osize(1)-1))';
Ocols = ([vec_ind vec_ind]+abs(Crconn(:,2:3))).*(Crconn(:,2:3)~=0);
% find fluxes in the positive column which are actually negative
because of
% symmetry conditions, place -1 in O
oposopp = find((Crconn(:,2)<0)==1);
O(Ocols(oposopp)) = -1;
Ocols(oposopp) = 0;
% find fluxes in the neg column which are actually pos, place 1 in O
onegopp = find((Crconn(:,3)<0)==1);
O(Ocols(onegopp,2)) = 1;
Ocols(onegopp,2) = 0;
% add a -1 in O for the remaining fluxes in neg column
Oneg = Ocols(Ocols(:,2)~=0,2);
O(Oneg) = O(Oneg)-1;
% add a +1 in O for remaining fluxes in the pos column
Opos = Ocols(Ocols(:,1)~=0,1);
O(Opos) = O(Opos)+1;
O = O';

%------------------------------------------------------------------------
% Update turn matrix for damper windings
% d_damper_1 represents MMF of damper currents
% d_damper_2 relates loop fluxes and the flux linkage crossing each of
two dampers
damper_rtip_prime = flipdim(damper_rtip,1);
damper_rtip_full = [damper_rtip_prime;damper_rtip(2:end)];
dp_pos = find(damper_rtip_full);
index = Crconn_ldp(dp_pos,:);
flag = flag_old;
% Damper bars disconnected between poles
```

```
if bartype == 1
    if damper_ntip < 2
        d_damper_1 = [];
        d_damper_2 = [];
    else
        d_damper_1 = zeros(Nm,damper_ntip-1);
        d_damper_2 = zeros(Nm,damper_ntip-1);
        for i = 1:damper_ntip-1
            if abs(index(i,3)-index_old(i,3)) > Nam/2
                flag(i) = -flag(i);
            end
            d_damper_1(abs(index(i,2)),i) = flag(i);
        end
        if abs(index(i+1,3)-index_old(i+1,3)) > Nam/2
            flag(i+1) = -flag(i+1);
        end
        d_damper_1(abs(index(i+1,2)),:) = -ones(1,damper_ntip-
1)*flag(i+1);

        for i = 1:damper_ntip-1
            d_damper_2(abs(index(i,2)),i) = -flag(i);
            d_damper_2(abs(index(i+1,2)),i) = flag(i+1);
        end
    end
% Damper bars connected between poles or no connection
elseif bartype == 2 || bartype == 0
    if damper_ntip < 1
        d_damper_1 = [];
    else
        d_damper_1 = zeros(Nm,damper_ntip);
        d_damper_2 = zeros(Nm,damper_ntip);
        for i = 1:damper_ntip
            if abs(index(i,3)-index_old(i,3)) > Nam/2
                flag(i) = -flag(i);
            end
            d_damper_1(abs(index(i,2)),i) = flag(i);
        end
    end

    if damper_ntip == 0
        d_damper_2 = [];
    elseif damper_ntip == 1
        d_damper_2(abs(index(1,2)),1) = -2*flag(1);
    else
        for i = 1:damper_ntip-1
            d_damper_2(abs(index(i,2)),i) = -flag(i);
            d_damper_2(abs(index(i+1,2)),i) = flag(i+1);
        end
        d_damper_2(abs(index(damper_ntip,2)),damper_ntip) = -
flag(damper_ntip);
        d_damper_2(abs(index(1,2)),damper_ntip) = -flag(1);
    end
end
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% May 1, 2009
%-------------------------------------------------------------------------
% [mur,pmur] = get_mur_exp(B,mubd)
%
% Calculate mur and pmur from exponential curve fit formulation in
PMMT.
%
% OUTPUTS: mur    - relative permeability
%          pmur   - derivative of the relative permeability
%
% INPUTS:  B      - flux density (T)
%          mubd   - structure containing curve fit parameters
%-------------------------------------------------------------------------
function [mur,pmur] = get_mur_exp(B,mubd)
B_w_sign = sign(B);
B_w_sign(B==0) = 1;
% Flux density is copied into a matrix to enable calculation without a
for
% loop.  The parameters are already in matrix form.
B = abs(B)*ones(1,mubd.K);
fofB    = mubd.mur/(mubd.mur-1) + ...
    sum(mubd.a.*B+mubd.d.*log((1+exp(-mubd.b.*B+mubd.e))./mubd.z),2);
dfofBdB = B_w_sign.*sum(mubd.a./(1+exp(-mubd.b.*B+mubd.e)),2);
% Relative permeability and its derivative
mur     = fofB./(fofB-1);
pmur    = -dfofBdB./((fofB-1).^2);
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% June 1, 2012
%-------------------------------------------------------------------------
% [WSTT,WST,WSY,WRT,WRY,WSW,WRW,WTOT] = get_mass(pars,parx,trns)
%
% Calculates the weight of the machine.
%
% OUTPUTS: WSTT     - stator teeth tip weight
%          WST      - stator teeth shank weight
%          WSY      - stator yoke weight
%          WRT      - rotor teeth weight
%          WRSH     - rotor shank weight
%          WRY      - rotor yoke weight
%          WSW,WRW  - stator and rotor copper weight
%          WTOT     - total weight
%
% INPUTS:  pars     - geometric parameters
%          parx     - simulation parameters
%          trns     - conductor turns
%-------------------------------------------------------------------------
function [WSTT,WST,WSY,WRT,WRSH,WRY,WSW,WRW,WTOT] =
get_mass(pars,parx,trns,damperdata)
% Parameters
OD          = pars(1);
ID          = pars(2);
GLS         = pars(3);
DBS         = pars(4);
H0          = pars(5);
H1          = pars(6);
SLTINS      = pars(13);
ESC         = pars(17);
STW         = pars(20);
DC          = pars(25);
CL          = pars(26);
GLP         = pars(27);
RP          = pars(28);
DENS        = pars(37);
SHDENS      = pars(38);
SD          = pars(29);
WIREDENS    = pars(39);
Ac          = pars(40);
Nfld        = pars(41);
Acfld       = pars(42);
ROD         = pars(24);
RPIT        = pars(32);
HRTT        = pars(44);
WRTSH       = pars(46);
```

```
NPH         = parx(1);
SL          = parx(3);
WRTang      = 2*pi*RPIT/RP; % ANGLE AT OUTSIDE EDGE OF ROTOR TOOTH TIP
WRTchord    = 2*(ROD/2)*sin(0.5*WRTang); % CHORD LENGTH OF ROTOR TOOTH
TIP
WRTSHchord  = pars(56); % CHORD WIDTH OF ROTOR TOOTH SHANK
yRT         = ROD/2*cos(0.5*WRTang);     % VERTICAL HEIGHT TO TOP OF
TOOTH TIP SIDE
yRC         = 0.5*sqrt(DC^2-WRTSHchord^2);  % VERTICAL HEIGHT TO BOTTOM
OF ROTOR TOOTH SHANK SIDE
HRTSH       = pars(45);          % VERTICAL HEIGHT OF ROTOR TOOTH SHANK
WCOIL       = pars(51);
tipw        = pars(57); % Width of stator teeth tip
tiph        = pars(58); % Height of stator teeth tip
damper_rtip = damperdata.damper_rtip;
damper_rshank = damperdata.damper_rshank;
damper_nshank = damperdata.damper_nshank;
%%%STATOR WEIGHT
%STATOR T00TH SHANK WEIGHT
rb = OD/2 - DBS; % Radius to back iron
rsi = ID/2; % Inner stator radius
% STW is the tooth arc width at the inner stator radius
thetats = 0.5*STW/rsi;
STWchd = sin(thetats)*rsi*2; %linear width of tooth
thetatb = asin((STWchd/2)/rb);
a1 = thetatb*(rb^2);
a3 = rb*rsi*sin(thetats-thetatb)/2;
a2 = thetats*(rsi^2);
area_stator_tooth_shank = a1 + 2*a3 - a2 - STW*tiph;
WST = DENS*(GLS*area_stator_tooth_shank)*SL;
%STATOR T00TH TIP WEIGHT
area_stator_tooth_tip = (2*tipw+STW)*tiph;
WSTT = DENS*(GLS*area_stator_tooth_tip)*SL;
%STATOR YOKE WEIGHT
volume_stator_outer_slice = GLS*pi*(OD/2)^2;
volume_stator_inner_slice = GLS*pi*(OD/2-DBS)^2;
WSY = DENS*(volume_stator_outer_slice - volume_stator_inner_slice);
% STATOR WEIGHT
SWEIGHT = WST + WSY + WSTT;
%%%ROTOR WEIGHT
%ROTOR CORE WEIGHT
volume_rotor_core_yoke = CL*pi*((DC/2)^2 - (SD/2)^2);
volume_shaft = CL*pi*(SD/2)^2;
WRY = DENS*volume_rotor_core_yoke + SHDENS*volume_shaft;
%ROTOR POLE TIP WEIGHT
artslice = WRTang/2*(ROD/2)^2;
arttri = WRTchord/2*yRT;
apt = HRTT*WRTchord;
area_damper_tip = sum(pi*damper_rtip.^2);
area_rotor_tip = artslice-arttri + apt - area_damper_tip;
volume_rotor_tip = GLP*RP*area_rotor_tip;
WRT = DENS*volume_rotor_tip;
% ROTOR POLE SHANK WEIGHT
apb = WRTSHchord*HRTSH;
arcslice = (DC/2)^2*asin(WRTSHchord/DC);
```

```
arctri = WRTSHchord/2*yRC;
area_damper_shank = damper_nshank*pi*damper_rshank.^2;
area_rotor_shank = apb - (arcslice-arctri) - area_damper_shank;
volume_rotor_shank = GLP*RP*area_rotor_shank;
WRSH = DENS*volume_rotor_shank;

% ROTOR WEIGHT
RWEIGHT = WRY + WRT + WRSH;
%%%COPPER WEIGHT
winding = abs(cumsum(trns) - 0.5*sum(trns));
DZ      = ID + 2*(H0+H1);
DW      = 0.5*(OD-DZ) - SLTINS - DBS;
lslot   = GLS + 2*ESC;
lend    = (2*pi/SL)*(DZ/2 + DW/2);
lcond   = sum(trns)*lslot*RP + 2*sum(winding)*lend*RP;
volume_stator_copper = Ac*lcond*NPH;
WSW = WIREDENS*volume_stator_copper;
%ROTOR WINDINGS
lcondfld = 2*(GLP + WRTSH + WCOIL*pi/2)*Nfld;
volume_rotor_copper = Acfld*lcondfld*RP;
WRW = WIREDENS*volume_rotor_copper;
% Copper weight
CUWEIGHT = WSW + WRW;
%TOTAL WEIGHT
WTOT = SWEIGHT + RWEIGHT + CUWEIGHT;
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% November 1, 2009
%-------------------------------------------------------------------------
% Pld = coreloss(B,f,DT,matdata)
%
% Calculates the core loss of the iron sections for any given material.
%
% OUTPUTS: Pld      - Volumetric power loss density
%
% INPUTS:  B        - flux density
%          f        - fundamental frequency
%          DT       - time step
%          matdata  - structure containing material data
%-------------------------------------------------------------------------
function Pld = coreloss(B,f,DT,matdata)
% Bb = 1;
deltB = max(B) - min(B);
% Coefficients for magnetic material
alp     = matdata.alpha;
beta    = matdata.beta;
kh      = matdata.kh;
ke      = matdata.ke;
% DEFINE NUMBER OF POINTS FOR ONE CYCLE
num_pts=round((1/f)/DT);
% LENGTH OF DATA VECTORS
n=length(B);
% OBTAIN WAVEFORM PORTION OF INTEREST
B_1=B(n-num_pts:n);
npts = length(B_1);
% NUMERICALLY DIFFERENTIATE
dBdt = (B_1(2:npts) - B_1(1:npts-1))./DT;
dBdt(npts) = dBdt(1);
dBdt2 = dBdt.*dBdt;
% INTEGRATE dB/dt^2
int_0toT = DT*(sum(dBdt2(1:npts-1))/2 + sum(dBdt2(2:npts))/2);
% EQUIVALENT FREQUENCY
feq = 2/(deltB^2*pi*pi)*int_0toT;
% POWER LOSS DENSITY
Pld = kh*feq^(alp-1)*max(B)^beta*f + ke*f*int_0toT;
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Wang Xiaoqi, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% August 1, 2013
%-------------------------------------------------------------------------
% damper_losses = calc_dploss(idamper, damperdata, pars, parx)
%
% Calculates the damper loss.
%
% OUTPUTS: damper_losses   -  damper loss
%
% INPUTS:  idamper      - damper currents
%          damperdata   - information of damper bars
%          pars         - geometric parameters
%          parx         - simulation parameters
%-------------------------------------------------------------------------

function [damper_losses] = calc_dploss(idamper, damperdata, pars, parx)

synfreq   = (pars(28)/2)*parx(4)/60;  % Fundamental frequency
DT        = parx(12);                 % Time step in s
RP        = pars(28);                 % Number of rotor poles
damper_ntip = damperdata.damper_ntip; % Number of damper windings on
rotor tip
Rd = damperdata.Rd; % Resistance of damper windings on rotor tip
Re = damperdata.Re; % Resistance of damper windings connection
bartype = damperdata.bartype; % Type of damper bars connnection

if damper_ntip == 0
    damper_losses = 0;
else
    % Current in the bars
    idp_rms = zeros(damper_ntip,1);
    for i = 1:damper_ntip
        idp_rms(i) = tools('tool_rms',idamper(i,:),1,synfreq,DT);
    end
    % Current in the end connections
    dp_conn = tril(ones(damper_ntip-(bartype==1),damper_ntip-
(bartype==1)),-1) ...
        + diag(ones(damper_ntip-(bartype==1),1));
    idp_end = dp_conn*idamper(1:damper_ntip-(bartype==1),:);
    idp_end_rms = zeros(damper_ntip-(bartype==1),1);
    for i = 1:damper_ntip-(bartype==1)
        idp_end_rms(i) = tools('tool_rms',idp_end(i,:),1,synfreq,DT);
    end
    % Calculate loss
    damper_losses = RP*(sum(Rd.*idp_rms.^2,1)+sum(Re(1:damper_ntip-
(bartype==1)).*idp_end_rms.^2,1));
end
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% July 1, 2012
%-------------------------------------------------------------------------
% wrsmpostprocess.m - NOT A FUNCTION
%
% Calculates postprocessing values (voltage, flux linkage, etc.) after
% modeling a machine.
%-------------------------------------------------------------------------
qr  = (RP/2)*(qrm); % Electrical rotor position
% qd current, flux linkage, and voltage
iqsr = (2/3)*(ias.*cos(qr) + ibs.*cos(qr - 2*pi/3) + ics.*cos(qr -
4*pi/3));
idsr = (2/3)*(ias.*sin(qr) + ibs.*sin(qr - 2*pi/3) + ics.*sin(qr -
4*pi/3));
i0sr = (2/3)*(ias*0.5 + ibs*0.5 + ics*0.5);
lamqsr = (2/3)*(lamabcpp(1,:).*cos(qr) + lamabcpp(2,:).*cos(qr -
2*pi/3) + lamabcpp(3,:).*cos(qr - 4*pi/3))*RP;
lamdsr = (2/3)*(lamabcpp(1,:).*sin(qr) + lamabcpp(2,:).*sin(qr -
2*pi/3) + lamabcpp(3,:).*sin(qr - 4*pi/3))*RP;
lam0sr = (2/3)*(lamabcpp(1,:)*0.5 + lamabcpp(2,:)*0.5 +
lamabcpp(3,:)*0.5)*RP;
% Vqd method 1
vqsr = (2/3)*(vabc(1,:).*cos(qr) + vabc(2,:).*cos(qr - 2*pi/3) +
vabc(3,:).*cos(qr - 4*pi/3));
vdsr = (2/3)*(vabc(1,:).*sin(qr) + vabc(2,:).*sin(qr - 2*pi/3) +
vabc(3,:).*sin(qr - 4*pi/3));
% Vqd method 2
% vqsr = rs*iqsr + wr*lamdsr+[0 (lamqsr(2:end)-lamqsr(1:end-1))]/DT;
% vdsr = rs*idsr - wr*lamqsr+[0 (lamdsr(2:end)-lamdsr(1:end-1))]/DT;

% Torque based on qd
torque_qd = mean(3/2*RP/2*(lamdsr(floor((NCYC-
1)/NCYC*end)+1:end).*iqsr(floor((NCYC-1)/NCYC*end)+1:end) ...
    -lamqsr(floor((NCYC-1)/NCYC*end)+1:end).*idsr((floor((NCYC-
1)/NCYC*end)+1:end))));
% compute reactive power
Qelec = 3/2*(mean(vqsr(floor((NCYC-
1)/NCYC*end)+1:end).*idsr(floor((NCYC-1)/NCYC*end)+1:end)) ...
    -mean(vdsr(floor((NCYC-1)/NCYC*end)+1:end).*iqsr(floor((NCYC-
1)/NCYC*end)+1:end)));
Pelec = 3/2*(mean(vqsr(floor((NCYC-
1)/NCYC*end)+1:end).*iqsr(floor((NCYC-1)/NCYC*end)+1:end)) ...
    +mean(vdsr(floor((NCYC-1)/NCYC*end)+1:end).*idsr(floor((NCYC-
1)/NCYC*end)+1:end)));
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% June 1, 2012
%-------------------------------------------------------------------------
% plothand = plotwrsm(pars,parx,pos,fign)
%
% Depicts the machine topology in a plot.
%
% OUTPUTS: plothand - handle to the plot created
%
% INPUTS:  pars     - geometric parameters
%          parx     - simulation parameters
%          pos      - rotor position in radians
%          fign     - figure number for the plot (optional)
%-------------------------------------------------------------------------
function plothand = plotwrsm(pars,parx,damperdata,pos,fign)
% INITIALIZE FIGURE
if nargin==4
    plothand = figure(fign);
else
    plothand = figure;
end
plot(0,0)
axis square
hold on
% MACHINE PARAMETERS
mtomm = 1000;
OD      = pars(1)*mtomm;
ID      = pars(2)*mtomm;
GLS     = pars(3)*mtomm;
DBS     = pars(4)*mtomm;
H0      = pars(5)*mtomm;
H1      = pars(6)*mtomm;
H2      = pars(7)*mtomm;
H3      = pars(8)*mtomm;
B0      = pars(9)*mtomm;
B1      = pars(10)*mtomm;
B2      = pars(11)*mtomm;
BS      = pars(12)*mtomm;
G1      = pars(14)*mtomm;
STW     = pars(20)*mtomm;
STTW    = pars(21)*mtomm;
ROD     = pars(24)*mtomm;
DC      = pars(25)*mtomm;
RP      = pars(28);
SD      = pars(29)*mtomm;
HRT     = pars(33)*mtomm;
WRT     = pars(34)*mtomm;
```

```
WAIRT   = pars(35)*mtomm;
HRTT    = pars(44)*mtomm;
HRTSH   = pars(45)*mtomm;
WRTSH   = pars(46)*mtomm;
RPIT    = pars(32);
WRTSHchord = pars(56)*mtomm;
NPH = parx(1);
SPT = parx(2);
SLL = parx(3);
NRrtrt  = parx(27);
Nrtt    = 2*SPT - 4*NRrtrt;       % Number of radial rotor tooth
branches
tipw    = pars(57)*mtomm; % width of stator teeth tip
tiph    = pars(58)*mtomm; % height of stator teeth tip
damper_rtip   = damperdata.damper_rtip;
damper_rshank = damperdata.damper_rshank;
damper_ntip   = damperdata.damper_ntip;
damper_nshank = damperdata.damper_nshank;
damper_dtip   = damperdata.damper_dtip;

% Plot stator ------------------------------------------
% Plot outer diameter
theta = 0:0.1:2*pi+0.1;
polar(theta,repmat(OD/2,1,length(theta)))
% Initialize terms used to plot stator teeth
angoff = 0; % angle offset of next tooth
strep = 2*pi/SLL; % angle between adjacent teeth
sistart = 0.5*STW/(OD/2-DBS); % angle associated with inner slot
boundary
siend = 2*pi/SLL - STW/(OD/2-DBS)+sistart;
tostart = -0.5*STTW/(ID/2); % angle associated with outer tooth
boundary
toend = STTW/(ID/2)+tostart;
tistart = -0.5*STTW/(ID/2+tiph); % angles associated with the left &
right inner tooth boundary
tiend = -0.5*STW/(ID/2+tiph);
tioff = (STW+tipw)/(ID/2+tiph);
tilango = 0.5*STW/(ID/2+tiph); % angles associated with the left &
right inner tooth edge
tirango = -0.5*STW/(ID/2+tiph);
tilangi = 0.5*STW/(OD/2-DBS);
tirangi = -0.5*STW/(OD/2-DBS);
tolango = 0.5*STTW/(ID/2); % angles associated with the left & right
outer tooth edge
torango = -0.5*STTW/(ID/2);
tolangi = 0.5*STTW/(ID/2+tiph);
torangi = -0.5*STTW/(ID/2+tiph);
% Plot stator teeth/slots
for st = 1:SLL
    % Plot "curved" portions
    arang = (sistart:(siend-sistart)/10:siend)+angoff;
    polar(arang,repmat(OD/2-DBS,1,length(arang)))
    arang = (tostart:(toend-tostart)/10:toend)+angoff;
    polar(arang,repmat(ID/2,1,length(arang)))
    arang = (tistart:(tiend-tistart)/10:tiend)+angoff;
```

```
    polar(arang,repmat((ID/2+tiph),1,length(arang)))
    arang = (tistart:(tiend-tistart)/10:tiend)+angoff+tioff;
    polar(arang,repmat((ID/2+tiph),1,length(arang)))
    % Plot radial portions
    polar([tilango+angoff tilangi+angoff],[ID/2+tiph OD/2-DBS])
    polar([tirango+angoff tirangi+angoff],[ID/2+tiph OD/2-DBS])
    polar([tolango+angoff tolangi+angoff],[ID/2 ID/2+tiph])
    polar([torango+angoff torangi+angoff],[ID/2 ID/2+tiph])
    % Increment angle offset to plot next tooth
    angoff = angoff+strep;
end
% PLOT ROTOR ----------------------------------------
%Plot shaft
theta = 0:0.1:2*pi+0.1;
polar(theta,repmat(SD/2,1,length(theta)))
% Initialize terms used to plot rotor
WRTang  = 2*pi*RPIT/RP; % ANGLE AT OUTSIDE EDGE OF ROTOR TOOTH TIP
WRTchord= 2*(ROD/2)*sin(0.5*WRTang); % CHORD LENGTH OF ROTOR TOOTH TIP
yRC     = 0.5*sqrt(DC^2-WRTSHchord^2); % VERTICAL HEIGHT TO BOTTOM OF
ROTOR TOOTH SHANK SIDE
WRTSHang= 2*atan(WRTSHchord/(2*(HRTSH+yRC))); % ANGLE OF ROTOR TOOTH
SHANK AT INSIDE OF ROTOR TOOTH TIP
WRTSHrad= (HRTSH+yRC)/(cos(0.5*WRTSHang));  % RADIUS AT TOP OF ROTOR
TOOTH SHANK
RTToutrad = sqrt((WRTchord*0.5)^2+(HRTSH+yRC)^2); % Radius at bottom of
outer tooth tip edge
WRTinang = 2*asin(WRTchord/(2*RTToutrad)); % INNER ANGLE OF ROTOR TOOTH
TIP
WRTSHinang = 2*asin(WRTSHchord/DC);
% angles associated with the inner rotor slot boundary (inter-polar
region)
rsistrt = pos - (2*pi/RP - WRTSHinang)/2;
rsiend = rsistrt + (2*pi/RP - WRTSHinang);
% angles associated with the rotor pole tip and pole body
rtostrt = pos + (WAIRT/(ROD/2))/2;
rtoend = rtostrt + WRT/(ROD/2);
rttistrt = pos + (2*pi/RP - WRTinang)/2;
rttiend = rttistrt + 0.5*(WRTinang-WRTSHang);
rttioff = (WRTSHang + 0.5*(WRTinang-WRTSHang));
rtrep = 2*pi/RP; % angle between adjacent rotor poles
angoff = 0;
for rt = 1:RP
    % Plot curved portions
    arang = (rsistrt:(rsiend-rsistrt)/10:rsiend)+angoff;
    polar(arang,repmat(DC/2,1,length(arang)))
    arang = (rtostrt:(rtoend-rtostrt)/10:rtoend)+angoff;
    polar(arang,repmat(ROD/2,1,length(arang)))
    % Plot straight portions
    polar([rttistrt rttiend]+angoff,[RTToutrad WRTSHrad])
    polar([rttistrt rttiend]+angoff+rttioff,[WRTSHrad RTToutrad])
    polar([rttiend+rttioff rtoend]+angoff,[RTToutrad ROD/2])
    polar([rttistrt rtostrt]+angoff,[RTToutrad ROD/2])
    polar([rsiend rttiend]+angoff,[DC/2 WRTSHrad])
    polar([rsistrt+rtrep rttistrt+rttioff]+angoff,[DC/2 WRTSHrad])
    % Increment offset angle to plot next tooth
```

```
    angoff = angoff+rtrep;
end

% Plot rotor pole tip dampers ------------------------------------
WRTang  = 2*WRT/ROD;
xout    = sin(WRTang/2)*ROD/2; % (xout = WRTchord/2)
yb      = cos(WRTang/2)*ROD/2-HRTT; % Vertical height to the bottom of
the rotor tooth tip
xin     = WRTSHchord/2;
WRTS2   = xout*2/SPT; % Horizontal width (not arc width) of the rotor
tooth sections
ymid = (sqrt((ROD/2)^2-(xin).^2)+yb)/2;
ytRTT = sqrt((ROD/2)^2-abs(xout-WRTS2*NRrtrt-WRTS2*((1:Nrtt/2)-
0.5)').^2);
ytNR = sqrt((ROD/2)^2-(xout-WRTS2*(1:NRrtrt)).^2);
angoff = 0;
dplength = length(damper_rtip)*2-1;
dpmid = length(damper_rtip);
dpx = zeros(1,dplength);
dpy = zeros(1,dplength);
dpr = [flipud(damper_rtip(2:end));damper_rtip]*mtomm;
for rt = 1:RP
    for k = 0:dpmid-1
        if k == 0
            dpy(dpmid) = 0;
            dpx(dpmid) = (ROD/2-ymid-2*dpr(dpmid))*(1-
damper_dtip)+dpr(dpmid)+ymid;
            [THETA,RHO] = cart2pol(dpx(dpmid),dpy(dpmid));
            [dpx(dpmid),dpy(dpmid)] = pol2cart(THETA-
pi/RP+pos+angoff,RHO);
        elseif k < Nrtt/4
            dpy(dpmid+k) = WRTS2*(k+0.5);
            dpx(dpmid+k) = (ytRTT(Nrtt/4-k)-ymid-2*dpr(dpmid+k))*(1-
damper_dtip)+dpr(dpmid+k)+ymid;
            dpy(dpmid-k) = -dpy(dpmid+k);
            dpx(dpmid-k) = dpx(dpmid+k);
            [THETA,RHO] = cart2pol(dpx(dpmid+k),dpy(dpmid+k));
            [dpx(dpmid+k),dpy(dpmid+k)] = pol2cart(THETA-
pi/RP+pos+angoff,RHO);
            [THETA,RHO] = cart2pol(dpx(dpmid-k),dpy(dpmid-k));
            [dpx(dpmid-k),dpy(dpmid-k)] = pol2cart(THETA-
pi/RP+pos+angoff,RHO);
        else
            dpy(dplength-k+Nrtt/4) = xout-WRTS2*(k-Nrtt/4+1);
            dpx(dplength-k+Nrtt/4) = (ytNR(k-Nrtt/4+1)-yb-2*dpr(k-
Nrtt/4+1))*(1-damper_dtip)+dpr(k-Nrtt/4+1)+yb;
            dpy(k-Nrtt/4+1) = -dpy(dplength-k+Nrtt/4);
            dpx(k-Nrtt/4+1) = dpx(dplength-k+Nrtt/4);
            [THETA,RHO] = cart2pol(dpx(k-Nrtt/4+1),dpy(k-Nrtt/4+1));
            [dpx(k-Nrtt/4+1),dpy(k-Nrtt/4+1)] = pol2cart(THETA-
pi/RP+pos+angoff,RHO);
            [THETA,RHO] = cart2pol(dpx(dplength-k+Nrtt/4),dpy(dplength-
k+Nrtt/4));
            [dpx(dplength-k+Nrtt/4),dpy(dplength-k+Nrtt/4)] =
pol2cart(THETA-pi/RP+pos+angoff,RHO);
```

```
        end
    end

    for k = 1:dplength
        if dpr(k) > 0
            x = linspace(dpr(k),-dpr(k),100);
            y = sqrt(dpr(k)^2-x.^2);
            x_new = [x+dpx(k) -x+dpx(k)];
            y_new = [y -y]+dpy(k);
            plot(x_new,y_new)
        end
    end
    angoff = angoff+2*pi/RP;
end
% Plot rotor pole shank dampers ---------------------------------------
angoff = 0;
l    = ymid - SD/2 - (DC-SD)/4;
shank_sec = l/(2*damper_nshank);
dpx2 = zeros(1,damper_nshank);
dpy2 = zeros(1,damper_nshank);
dpr2 = damper_rshank*mtomm;
for rt = 1:RP
    angmid = (rtoend+rtostrt)/2+angoff;
    for k = 1:damper_nshank
        if k == 1
            [dpx2(k),dpy2(k)] = pol2cart(angmid,SD/2+(DC-SD)/4+2*dpr2);
        elseif k == damper_nshank
            [dpx2(k),dpy2(k)] = pol2cart(angmid,ymid-2*dpr2);
        else
            [dpx2(k),dpy2(k)] = pol2cart(angmid,((2*k-
1)*shank_sec+SD/2+(DC-SD)/4));
        end
        if dpr2 > 0
            x = linspace(dpr2,-dpr2,100);
            y = sqrt(dpr2^2-x.^2);
            x_new = [x+dpx2(k) -x+dpx2(k)];
            y_new = [y -y]+dpy2(k);
            plot(x_new,y_new)
        end
    end
    angoff = angoff+2*pi/RP;
end
% LENGTH ----------------------------------------
plot([OD/2+0.03*OD OD/2+0.03*OD],[-GLS/2 GLS/2],'r')
% Format plot
xlabel('x (mm)')
ylabel('y (mm)')
title('WRSM geometry')
axlim = max(GLS/2+0.1*GLS/2,OD/2+0.1*OD/2);
axis([-axlim axlim -axlim axlim])
box on
hold off
```

```
%-----------------------------------------------------------------------
% AUTHORS:  Michelle Bash, Steven D. Pekarek
%-----------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-----------------------------------------------------------------------
% Version 1 - May 1, 2009
%-----------------------------------------------------------------------
% [V,idc] = rect(iabcl,vdc,parx)
%
% Calculates the rectifier voltages based on the rectifier currents.
%
% OUTPUTS: V        - vector of rectifier voltages (vag,vbg,vcg)
%          idc      - dc bus current
%
% INPUTS:  iabcl    - rectifier currents
%          parx     - simulation parameters
%          vdc      - dc bus voltage
%-----------------------------------------------------------------------
function [V,idc] = rect(iabcl,vdc,parx)

dalpha = parx(18);
dbeta  = parx(19);
eps = 0.005;
vdend = 1/dbeta*log(eps/dalpha + 1);
i1 = 0.0; i3 = 0.0; i5 = 0.0;
% Rectifier phase currents
ial = iabcl(1);
ibl = iabcl(2);
icl = iabcl(3);

if (ial <= -eps)
   vag = -1/dbeta*log(abs(ial)/dalpha + 1);
elseif (ial >= eps)
   vag = vdc + 1/dbeta*log(abs(ial)/dalpha + 1);
elseif(ial < eps && ial > -eps)
   vag = ((vdc + 2*vdend)/(2*eps))*ial + vdc/2;
end

if (ibl <= -eps)
   vbg = -1/dbeta*log(abs(ibl)/dalpha + 1);
elseif (ibl >= eps)
   vbg = vdc + 1/dbeta*log(abs(ibl)/dalpha + 1);
elseif(ibl < eps && ibl > -eps)
   vbg = ((vdc + 2*vdend)/(2*eps))*ibl + vdc/2;
end

if (icl <= -eps)
   vcg = -1/dbeta*log(abs(icl)/dalpha + 1);
elseif(icl >= eps)
   vcg = vdc + 1/dbeta*log(abs(icl)/dalpha + 1);
```

```
elseif(icl < eps && icl > -eps)
   vcg = ((vdc + 2*vdend)/(2*eps))*icl + vdc/2;
end

% Calculate idc
if (ial > 0.0)
    i1 = ial;
end

if (ibl > 0.0)
    i3 = ibl;
end

if (icl > 0.0)
    i5 = icl;
end

idc = i1 + i3 + i5;
V = [vag;vbg;vcg];
```

```
%-----------------------------------------------------------------------
% AUTHORS:  Michelle Bash, Steven D. Pekarek
%-----------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-----------------------------------------------------------------------
% May 1, 2009
%-----------------------------------------------------------------------
% [xrms,xavg,xrip]=tools(which_one, x, cycles, f, DT)
%
% Finds the average value, rms value, and/or ripple of a given signal.
%
% OUTPUTS: xrms      - rms value of the signal
%          xavg      - average value of the signal
%          xrip      - ripple value of the signal
%
% INPUTS:  which_one - tool_rms,tool_avg,tool_rip, or tool_all
%          x         - signal to be analyzed
%          cycles    - number of cycles to use in analysis
%          f         - fundamental freq
%          DT        - sampling period
%-----------------------------------------------------------------------
function [xrms,xavg,xrip]=tools(which_one, x, cycles, f, DT)
switch which_one
    case 'tool_rms'
        xrms = tool_rms(x,cycles,f,DT);
        xavg = 0;
        xrip = 0;
    case 'tool_avg'
        xrms = 0;
        xavg = tool_avg(x,cycles,f,DT);
        xrip = 0;
    case 'tool_rip'
        xrms = 0;
        xavg = 0;
        xrip = tool_rip(x,cycles,f,DT);
    case 'tool_all'
        xrms = tool_rms(x,cycles,f,DT);
        xavg = tool_avg(x,cycles,f,DT);
        xrip = tool_rip(x,cycles,f,DT);
end
%-----------------------------------------------------------------------
% TOOL_RMS
%-----------------------------------------------------------------------
function x_rms = tool_rms(x,cycles,f,DT)
%DEFINE NUMBER OF CYCLES OF AC WAVEFORM TO USE
num_cycles = cycles*round((1/f)/DT);
%LENGTH OF DATA VECTORS
n = length(x);
%OBTAIN WAVEFORM PORTION OF INTEREST
x_1 = x(n-num_cycles:n);
%RMS CALCULATION
```

```
px_rms = (f/cycles)*x_1.*x_1;
x_rms = 0;
for i = 1:num_cycles
    x_rms = x_rms + px_rms(i+1)*DT;
end
x_rms = sqrt(x_rms);
%-------------------------------------------------------------------------
% TOOL_AVG
%-------------------------------------------------------------------------
function x_avg = tool_avg(x,cycles,f,DT)
%DEFINE NUMBER OF CYCLES OF AC WAVEFORM TO USE
num_cycles = cycles*round((1/f)/DT);
%LENGTH OF DATA VECTORS
n = length(x);
%OBTAIN WAVEFORM PORTION OF INTEREST
x_1 = x(n-num_cycles:n);
%AVG CALCULATION
px_avg = (f/cycles)*x_1;
x_avg = 0;
for i = 1:num_cycles
    x_avg = x_avg + px_avg(i+1)*DT;
end
%-------------------------------------------------------------------------
% TOOL_RIP
%-------------------------------------------------------------------------
function x_rip = tool_rip(x,cycles,f,DT)
%DEFINE NUMBER OF CYCLES OF AC WAVEFORM TO USE
num_cycles = cycles*round((1/f)/DT);
%LENGTH OF DATA VECTORS
n = length(x);
%OBTAIN WAVEFORM PORTION OF INTEREST
x_1 = x(n-num_cycles:n) - tool_avg(x,cycles,f,DT);
%RIPPLE CALCULATION
xmin = 0;
xmax = 0;
for i = 1:num_cycles+1
    if x_1(i) >= xmax
        xmax = x_1(i);
    end
    if x_1(i) <= xmin
        xmin = x_1(i);
    end
end
x_rip = abs(xmax) + abs(xmin);
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% Apr 1, 2013
%-------------------------------------------------------------------------
%
[t,vabc,lamabcpp,lamdamper,iabc,idamper,idc,vdc,vc,torque,qrm,phit,BY,B
T,BTT,nrconverge,saturate,BIRON] =
%wrsmdynamics_multislice(parx,pars,turns,damperdata,mudata,qr_init)
%
% Solves the Dynamics of the MEC network.
%
% OUTPUTS: t        - time vector (s)
%          vabcs    - phase voltages (V)
%          lamabcpp - phase flux linkage per pole (Vs)
%          lamdamper - damper flux linkage (Vs)
%          iabcs    - phase currents (A)
%          idamper  - damper bar currents (A)
%          idc      - dc bus currents (A)
%          vdc      - dc bus voltage (V)
%          vc       - dc bus capacitor voltage (V)
%          torque   - torque (Nm)
%          qrm      - mechanical rotor position (radians)
%          phit     - stator teeth flux (Wb)
%          BY,BT,BTT    - flux density in the stator yoke, stator
teeth, and stator tooth tips (T)
%          nrconverge - flag indicating if newton raphson converged
%          saturate   - indicates if the flux density limit is violated
%          BIRON    - flux density in iron (Wb)
%
% INPUTS:  pars     - geometric parameters
%          parx     - simulation parameters
%          turns    - phase winding turns (turn count)
%          damperdata   - information of damper bars
%          mudata   - magnetic material data for finding permeability
%          qr_init  - initial rotor position in electric degree
%-------------------------------------------------------------------------
----
function
[t,vabc,lamabcpp,lamdamper,iabc,idamper,idc,vdc,vc,torque,qrm,phit,BY,B
T,BTT,nrconverge,saturate,BIRON] = wrsmdynamics_multislice
(parx,pars,turns,damperdata,mudata,qr_init)
%-------------------------------------------------------------------------
% INITIALIZE THE SYSTEM
%-------------------------------------------------------------------------
mu0     = pi*4e-7;      % Permeability of free space
RP      = pars(28);     % Poles
S       = parx(3)/RP;   % Number of stator slots per pole
```

```
D       = 2*(parx(2));  % Number of rotor pole tip sections per pole
pair
Dsl     = 4*parx(29);   % Number of inter-polar regions per pole pair
SPT     = parx(2);      % SECTIONS PER ROTOR TOOTH, including radial
and tangential
NRrtrt  = parx(27);     % Number of outer pole tip reluctances per pole
pair
damper_ntip = damperdata.damper_ntip;     % Number of damper windings
on rotor tip
damper_nshank = damperdata.damper_nshank; % Number of damper windings
on rotor shank
bartype = damperdata.bartype;             % Type of damper bars
connnection
Rd = damperdata.Rd;     % Damper bar body resistance
Re = damperdata.Re;     % Damper bar end connection resistance
Rload   = 22.81 ;      % Parallel resistance load
Lload   = 0.0807 ;     % Parallel resistance load
Cload   = 100e-6;       % Filter capacitance
taus    = 0.1;          % Filter time constant
rs      = pars(23);     % Stator windings resistance
wrm     = parx(4)*2*pi/60;          % Mechanical rotor speed in rad/s
wr      = (pars(28)/2)*wrm;
scl     = parx(16);
ifld    = pars(47);                 % Field current (A)
vrms    = pars(49);                 % rms Stator voltage (V)
vphase  = pars(50);                 % Current phase angle (degrees)
vm      = vrms*sqrt(2);             % Magnitude of vas,vbs,vcs
DT      = parx(12);                 % Time step in s
iter    = parx(30);                 % Number of iterations
vdcmax  = parx(25);                 % Maximum dc voltage
NPTS    = parx(7);                  % NUMBER OF DATA POINTS PER CYCLE
skew_angle = pars(30);              % Electrical skew angle, rad
stack_num = pars(31);               % Number of stack for skew
% For machine design with RL load producing rated power ---------------
----
% Vll_rms = 480;
% pf = 0.8;
% P = parx(24);
% Q = sqrt((P/pf)^2-P^2);
% Rload = 3*(Vll_rms/sqrt(3))^2/P;
% Lload = (Vll_rms/sqrt(3))^2/Q/wr;
% ---------------------------------------------------------------------
----
% INITIALIZE VARIABLES
slB     = 3*S;                      % Number of iron elements in stator
rlB     = 6+D/2+damper_nshank+SPT+(SPT-1); % Number of iron elements in
rotor
lB      = slB+rlB;                  % Number of iron elements
nriter  = zeros(1,iter);            % Keeps track of N-R iterations
torque  = zeros(1,iter);
PTC     = zeros(S,D+Dsl,iter);      % Matrix of airgap permeances
dPTC    = zeros(S,D+Dsl,iter);
phit    = zeros(S,iter,stack_num); % Stator tooth flux
phiiron = zeros(lB,iter); % Flux in iron
BY      = zeros(S,iter,stack_num); % Stator yoke flux density
```

```
BT      = zeros(S,iter,stack_num); % Stator tooth shank flux density
BTT     = zeros(S,iter,stack_num); % Stator tooth tip flux density
BIRON   = zeros(lB,iter,stack_num); % Flus density in all iron elements
saturate = ones(1,iter);           % Saturation constraint (is Bsat
violated)
smuiron = get_mur_exp(zeros(slB,1),mudata.s); % Initial permeabilities
of stator
rmuiron = get_mur_exp(zeros(rlB,1),mudata.r); % Initial permeabilities
of rotor
muiron  = [smuiron;rmuiron];       % Initial permeabilities
TOL     = parx(21);                % tolerance for convergence of
Newton-Raphson
k       = 1;                       % Simulation step
t(k)    = parx(10);
% ARTIFICIAL ROTOR POSITION MODIFICATION used in the calculation of
airgap
% permeances.----
SLL     = parx(3);
ID      = pars(2);
ROD     = pars(24);
STTW    = pars(21);
WRT     = pars(34);
WAIRT   = pars(35);
shift1  = WRT/(ROD/2);
shift2  = (WAIRT/2)/(ROD/2);
shift3  = 2*pi/SLL;
shift4  = (STTW/2)/(ID/2);
shift5  = (pi/2)/(RP/2);
shift   = shift1 + shift2 - (S/2)*shift3 - shift4 - shift5;
% TIME AND ROTOR POSITION VECTORS
t       = (0:DT:DT*(iter-1))+t(k);
qrm     = t*wrm + qr_init/(RP/2);  % Actual rotor position
qrm_shift =  qrm + shift;          % Angle fed to airgap permeance
function
%--------------------------------------------------------------------
% CALCULATE VARIABLES/MATRICES WHICH WILL NOT CHANGE DURING SIM
%--------------------------------------------------------------------
% Variables/matrices to be used in airgap permeance calculation
WRS     = pars(35)/(2*parx(29));
WRTS    = pars(36);
B0      = pars(9);
SPT     = parx(2);
RPIT    = pars(32);
WRTSang = 2*pi*RPIT/RP/SPT;
WRTang  = 2*pi*RPIT/RP;
WRSang  = 2*pi*(1-RPIT)/RP/(Dsl/2);
qs      = STTW/ID*RP;              % Span of stator tooth in
electrical radians
qs1     = B0/ID*RP;                % Span of stator slot
qrr     = WRTSang*RP/2;            % Span of rotor pole tip section
qrs     = WRSang*RP/2;             % Span of inter-polar section
Gmaxrt = pi*4e-7*pars(3)/stack_num/(ID-
ROD)*2*(WRTS*(STTW>=WRTS)+STTW*(STTW<WRTS)); % if-else
Gmaxsl = pi*4e-7*pars(3)/stack_num/(ID-
ROD)*2*(WRS*(STTW>=WRS)+STTW*(STTW<WRS)); % if-else
```

```
rt      = 1:D; rtsl    = 1:Dsl; st      = (1:S)';
% Matrices defining the angle between every stator tooth and rotor
section
anglert = ones(S,1)*(-mod(rt-1,(D/2))*WRTSang - floor((rt-
1)/(D/2))*2*pi/RP)...
    + ((st-1)*(STTW+B0)/(ID/2))*ones(1,D);
anglesl = ones(S,1)*(-WRTang - mod(rtsl-1,(Dsl/2))*WRSang - ...
    floor((rtsl-1)/(Dsl/2))*2*pi/RP) + ((st-
1)*(STTW+B0)/(ID/2))*ones(1,Dsl);

% Establish the geometric case for the rotor tooth section
if qrr <= qs1/2
    qrrcs = 1;
elseif (qrr <= qs)
    qrrcs = 2;
elseif (qrr <= qs +qs1/2)
    qrrcs = 3;
elseif (qrr <= qs+qs1)
    qrrcs = 4;
else
    qrrcs = 5;
end
% Establish the geometric case for the rotor slot section
if qrs <= qs1/2
    qrscs = 1;
elseif (qrs <= qs)
    qrscs = 2;
elseif (qrs <= qs +qs1/2)
    qrscs = 3;
elseif (qrs <= qs+qs1)
    qrscs = 4;
else
    qrscs = 5;
end

% ---------------------------------------------------------------------
% turns matrix to be used in system of equations
Natrn   = [-turns turns]';
Nbtrn   = [Natrn(2*SLL/(3*RP)+1:end);Natrn(1:2*SLL/(3*RP))];
Nctrn   = [Natrn(4*SLL/(3*RP)+1:end);Natrn(1:4*SLL/(3*RP))];
Nabc    = [Natrn Nbtrn Nctrn];
Nfld    = pars(41);
Nabcf   = [Nabc(1:S,:) zeros(S,1);0 0 0 Nfld;0 0 0 -Nfld];
% ---------------------------------------------------------------------
% MEC loops with MMF sources
Cvcfixed = (1:S+2)';
%---------------------------------------------------------------------
% Calculate the reluctances
[Rxm,areas,Rair,NPRTS,NPRTB] =
get_reluctances(mu0,parx,pars,damperdata);
Riron = Rxm./muiron;
%---------------------------------------------------------------------
% Identify type of node in rotor tooth and slot
% 1 = node of rotor pole tip radial branch
% 2 = node of rotor pole tip tangential branch
```

```
% 3 = rotor slot branch going to rotor edge
% 4 = rotor slot branch going to bottom of rotor pole tip
rtid = [2*ones(NRrtrt,1);ones(D/2-2*NRrtrt,1);2*ones(NRrtrt,1);...
        3*ones(NPRTS,1);4*ones(2*NPRTB,1);3*ones(NPRTS,1);...
        2*ones(NRrtrt,1);ones(D/2-2*NRrtrt,1);2*ones(NRrtrt,1);...
        3*ones(NPRTS,1);4*ones(2*NPRTB,1);3*ones(NPRTS,1)];
% Identify how many RRTOUT branches border the rotor loop
NRBRL = ceil((NRrtrt+1)/2);  % Number of RRTOUT branches Bordering
Rotor Loop
NRTBD = NRrtrt-NRBRL; % Number of RRTOUT branches with bordering loop
To Be Determined
% ------------------------------------------------------------------
% Define reluctance connections in stator and rotor which do not change
% Stator tooth tip, damper slots, and leakage of damper slots are not
% presented here, but will be derived as postprocess in shape_alg.m
% IRON
% Stator yoke - S
% Stator teeth - S
% Rotor yoke below the slot - 1
% Rotor tooth shank - 1
% Rotor yoke connected to shank - 2
% Rotor tooth tips radial - (D - 4*NRrtrt)
% Rotor tooth to rotor tooth tangential - 4*NRrtrt
% Rotor tooth tangential at sides of tooth tips - 4
% AIR
% Stator tooth leakage - S
% Field winding leakage - 2
% Middle rotor slot leakage - 2
% Fringing permeance from rotor side to airgap boundary - Dsl
% Fringing permeance from rotor slot side to bottom of tooth tip - 4
% RY R RRYSL RRTSH RRYSH RRTIN RRTOUT RRTS RSTL RFDL RRTL RAGFR RFRB
Crcfixed = zeros(2*S+8+D+S+3+Dsl,3);
% RY (all)
Crcfixed(1:S,2)=(1:S)';
% R (all)
Crcfixed(S+1:2*S,2) = [1 2:S]';
Crcfixed(S+1:2*S,3) = [-S 1:S-1]';
% RRYSL (all)
Crcfixed(2*S+1,3) = S+3;
% RRTSH (all)
Crcfixed(2*S+2,2:3) = [S+1 S+2];
% RRYSH (all)
Crcfixed(2*S+2+(1:2)',2) = [S+1;S+2];
% RRTIN (Determined by shape algorithm)
% RRTOUT - One side known if reluctance borders rotor loop
Crcfixed(2*S+2+D-4*NRrtrt+2+(1:4*NRrtrt)',2) = ...

[[zeros(NRTBD,1);ones(NRBRL,1)]*(S+1);[ones(NRBRL,1);zeros(NRTBD,1)]*(S
+2);...
 -[zeros(NRTBD,1);ones(NRBRL,1)]*(S+1);-
[ones(NRBRL,1);zeros(NRTBD,1)]*(S+2)];
% RRTS - (Determined by shape algorithm)
% RSTL (one side known, use shape alg for other)
Crcfixed(2*S+2+D+6+(1:S)',2) = (1:S)';
% RFDL (all)
```

```
Crcfixed(2*S+2+D+6+S+(1:2)',2:3) = [-(S+3) S+1;S+2 S+3];
% RRTL (one side known, use shape alg for other)
Crcfixed(2*S+2+D+6+S+2+(1:2)',2) = [S+3;-(S+3)];
% RAGFR - (Determined by shape algorithm)
% RFRB (one side, use shape alg for other)
Crcfixed(2*S+2+D+6+S+4+Dsl+(1:4)',2) = [-(S+3);S+3;S+3;-(S+3)];
%-------------------------------------------------------------------------
----
% Initialize variables
if parx(15) == 1 %Delta
    nio = 3;
    mlam = [0 1 0;-1 0 0;0 0 0];
    m_isil = [-1 0 1;1 -1 0;0 1 -1];
    m_vgvs = 1.5*[1 sqrt(3)/3 0;-sqrt(3)/3 1 0;0 0 0];
else %Wye
    nio = 2;
    mlam = [0 1;-1 0];
    m_isil = -eye(3);
    m_vgvs = [1 0 0;0 1 0];
end
iabc  = zeros(3,iter);
lamabcpp = zeros(3,iter);
vqd0sr = zeros(nio,iter);
iqd0sr = zeros(nio,iter);
lamqd0srpp = zeros(nio,iter+1);
plamqd0srpp = zeros(nio,iter);
idamper = zeros(damper_ntip,iter);
lamdamper = zeros(damper_ntip,iter+1);
plamdamper = zeros(damper_ntip,iter);
il = zeros(3,iter+1);
pil = zeros(3,iter);
vc = ones(1,iter+1)*vdcmax;
pvc = zeros(1,iter);
idc = ones(1,iter+1)*vdcmax/Rload;
vdc = ones(1,iter+1)*vdcmax;
Ivdc = zeros(1,iter+1);
Ivc = zeros(1,iter+1);
index_vect = zeros(damper_ntip,3,iter+1,stack_num);
flag_vect = ones(damper_ntip,iter+1,stack_num);

% Calculate the voltages for SSFR test
if wrm>0
    vas = vm*cos((RP/2)*(qrm) + (pi*vphase/180));
    vbs = vm*cos((RP/2)*(qrm) + (pi*vphase/180) - (2*pi/3));
    vcs = vm*cos((RP/2)*(qrm) + (pi*vphase/180) - (4*pi/3));
else
    vfreq = parx(5);
    vas = 2/3*vm*cos(2*pi*vfreq*t);
    vbs = -1/3*vm*cos(2*pi*vfreq*t);
    vcs = -1/3*vm*cos(2*pi*vfreq*t);
end
vabc = [vas;vbs;vcs];

% Initial stator flux linkage per pole values
if wrm > 0
```

```
    Ksr_prime = (2/3)*[-sin((RP/2)*(qrm(k))) -sin((RP/2)*(qrm(k))-
2*pi/3) -sin((RP/2)*(qrm(k))+2*pi/3);
        cos((RP/2)*(qrm(k))) cos((RP/2)*(qrm(k))-2*pi/3)
cos((RP/2)*(qrm(k))+2*pi/3)];
    lamqd0srpp(1:2,k) = Ksr_prime*vabc(:,k)/wr/RP;
else
    lamqd0srpp(1:2,k) = [0.00;0.001];
end
%-----------------------------------------------------------------------
% Determine transformation matrix for plamdamper
if bartype == 1
    % Version-1: No end connection resistance -------------------------
    % For example damper_ntip = 5
    % Tdp = [-rb1 rb2 0 0;0 -rb2 rb3 0;0 0 -rb3 rb4;-rb5 -rb5 -rb5 -
rb5-rb4];

    % if damper_ntip == 2
    %     Tdp = -Rd(1)-Rd(2);
    % else
    %     Tdp = -diag(Rd(1:end-1));
    %     for i = 1:damper_ntip-2
    %         Tdp(i,i+1) = Rd(i+1);
    %     end
    %     Tdp(damper_ntip-1,:) = -Rd(damper_ntip)*ones(1,damper_ntip-
1);
    %     Tdp(damper_ntip-1,damper_ntip-1) = Tdp(damper_ntip-
1,damper_ntip-1)-Rd(damper_ntip-1);
    % end

    % Version-2: With end connection resistance -----------------------
    % Tdp = [-rb1-2*re1 rb2 0 0;
    %        -2*re2 -rb2-2*re2 rb3 0;
    %        -2*re3 -2*re3 -rb3-2*re3 rb4;
    %        -rb5-2*re4 -rb5-2*re4 -rb5-2*re4 -rb5-2*re4-rb4];

    % Re = [0.1 0.1 0.1 0.1 0.1]*1e-3;
    if damper_ntip < 2
        Tdp = [];
    elseif damper_ntip == 2
        Tdp = -Rd(1)-Rd(2)-2*Re(1);
    else
        Tdp = -diag(Rd(1:end-1));
        for i = 1:damper_ntip-2
            Tdp(i,i+1) = Rd(i+1);
        end
        for i = 1:damper_ntip-1
            for j = 1:i
                Tdp(i,j) = Tdp(i,j)-2*Re(i);
            end
        end
        Tdp(damper_ntip-1,:) = Tdp(damper_ntip-1,:)-
Rd(damper_ntip)*ones(1,damper_ntip-1);
    end
```

```
elseif bartype == 2
    % Version-1: No end connection resistance ------------------------
    % For example damper_ntip = 5
    % Tdp = [-Rd(1) Rd(2) 0 0 0;0 -Rd(2) Rd(3) 0 0;0 0 -Rd(3) Rd(4) 0;0
0 0 -Rd(4) Rd(5);-Rd(1) 0 0 0 -Rd(5)];

    % if damper_ntip == 1
    %     Tdp = -2*Rd(1);
    % else
    %     Tdp = -diag(Rd(1:end));
    %     for i = 1:damper_ntip-1
    %         Tdp(i,i+1) = Rd(i+1);
    %     end
    %     Tdp(damper_ntip,1) = -Rd(1);
    % end

    % Version-2: With end connection resistance ----------------------
    % Re = [0.1 0.1 0.1 0.1 0.1]*1e-3;
    % Tdp = -[Rd(1)+Re(1) -Rd(2)-Re(1) -Re(1) -Re(1) -Re(1); ...
    %         Re(2) Rd(2)+Re(2) -Rd(3)-Re(2) -Re(2) -Re(2); ...
    %         Re(3) Re(3) Rd(3)+Re(3) -Rd(4)-Re(3) -Re(3); ...
    %         Re(4) Re(4) Re(4) Rd(4)+Re(4) -Rd(5)-Re(4); ...
    %         Rd(1)+Re(5) Re(5) Re(5) Re(5) Rd(5)+Re(5)];
    %
    % Re = [0.1 0.1 0.1 0.1 1]*1e-3;
    if damper_ntip == 0
        Tdp = [];
    elseif damper_ntip == 1
        Tdp = -2*Rd(1)-2*Re(1);
    else
        Tdp = -diag(Rd(1:end));
        for i = 1:damper_ntip
            for j = 1:damper_ntip
                if j <= i
                    Tdp(i,j) = Tdp(i,j)-Re(i);
                else
                    Tdp(i,j) = Tdp(i,j)+Re(i);
                end
            end
        end
        for i = 1:damper_ntip-1
            Tdp(i,i+1) = Tdp(i,i+1)+Rd(i+1);
        end
        Tdp(damper_ntip,1) = Tdp(damper_ntip,1)-Rd(1);
    end
end

%-------------------------------------------------------------------------
% SOLVING LOOP
%-------------------------------------------------------------------------
nrconverge = 1;
if stack_num == 1
    stack_span = 0;
else
    stack_span = floor(skew_angle/(2*pi)*NPTS/(stack_num-1));
```

```
end
% AIR-GAP PERMEANCES
for i = 1:iter
    [PTC(:,:,i),dPTC(:,:,i)] =
get_Pag(qrm_shift(i),pars,parx,Gmaxrt,Gmaxsl,anglert,anglesl,qrrcs,qrsc
s);
end
[l,m,n] = size(PTC);
PTC_prime = zeros(l,m,n,stack_num);
dPTC_prime = zeros(l,m,n,stack_num);
for i = 1:stack_num
    PTC_prime(:,:,1:(i-1)*stack_span,i) = PTC(:,:,end-(i-
1)*stack_span+1:end);
    PTC_prime(:,:,(i-1)*stack_span+1:end,i) = PTC(:,:,1:end-(i-
1)*stack_span);
    dPTC_prime(:,:,1:(i-1)*stack_span,i) = dPTC(:,:,end-(i-
1)*stack_span+1:end);
    dPTC_prime(:,:,(i-1)*stack_span+1:end,i) = dPTC(:,:,1:end-(i-
1)*stack_span);
end

while k <= iter
    % Using rotor reference frame
    Ksr = (2/3)*[cos((RP/2)*(qrm(k))) cos((RP/2)*(qrm(k))-2*pi/3)
cos((RP/2)*(qrm(k))+2*pi/3);
        sin((RP/2)*(qrm(k))) sin((RP/2)*(qrm(k))-2*pi/3)
sin((RP/2)*(qrm(k))+2*pi/3);
        0.5 0.5 0.5];
    Ksrinv = [cos((RP/2)*(qrm(k))) sin((RP/2)*(qrm(k))) 1;
        cos((RP/2)*(qrm(k))-2*pi/3) sin((RP/2)*(qrm(k))-2*pi/3) 1;
        cos((RP/2)*(qrm(k))+2*pi/3) sin((RP/2)*(qrm(k))+2*pi/3) 1];
    for i = 1:stack_num
        if k==1 || sum(sum((PTC_prime(:,:,k-
1,i)~=0)~=(PTC_prime(:,:,k,i)~=0)))>0
            [Crconn,Cvconn,O,PTCind,d_damper_1,d_damper_2,index,flag]
...
                =
shape_alg(PTC_prime(:,:,k,i),parx,pars,damperdata,Crcfixed,Cvcfixed,rti
d,index_vect(:,:,k,i),flag_vect(:,k,i));
            if length(Crconn)~=length([Riron;Rair;PTCind])
                nrconverge = 0;
                break
            end
            % Save variables
            [row_Crconn(i),col_Crconn(i)] = size(Crconn);
            [row_O(i),col_O(i)] = size(O);
            [row_PTCind(i),col_PTCind(i)] = size(PTCind);
            [row_d_damper_1(i),col_d_damper_1(i)] = size(d_damper_1);
            [row_d_damper_2(i),col_d_damper_2(i)] = size(d_damper_2);
            if i == 1 && k == 1
                Crconn_prime = -
1e12*ones(row_Crconn(i)+5,col_Crconn(i),stack_num);
                O_prime = -1e12*ones(row_O(i)+5,col_O(i)+5,stack_num);
                PTCind_prime = -1e12*ones(row_PTCind(i)+5,stack_num);
```

```
                d_damper_1_prime = -
1e12*ones(row_d_damper_1(i)+5,col_d_damper_1(i),stack_num);
                d_damper_2_prime = -
1e12*ones(row_d_damper_2(i)+5,col_d_damper_2(i),stack_num);
            end
            Cvconn_prime(:,i) = Cvconn;
            Crconn_prime(1:row_Crconn(i),:,i) = Crconn;
            O_prime(1:row_O(i),1:col_O(i),i) = O;
            PTCind_prime(1:row_PTCind(i),i) = PTCind;
            d_damper_1_prime(1:row_d_damper_1(i),:,i) = d_damper_1;
            d_damper_2_prime(1:row_d_damper_2(i),:,i) = d_damper_2;
        end

        % Obtain list of airgap permeances and their derivatives for
this rotor position
        ptc           = PTC_prime(:,:,k,i)';
        PTClist       = ptc(PTCind_prime(1:row_PTCind(i),i));
        dptc          = dPTC_prime(:,:,k,i)';
        dPTClist      = dptc(PTCind_prime(1:row_PTCind(i),i));
        % Find the system of equations and solve for the initial guess
        [A,d] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn_prime(1:row_
Crconn(i),:,i),Cvconn_prime(:,i));
        % Total number of meshes
        Nm(i) = 3 + S + length(PTClist) + (SPT-1);

        % Save variables
        [row_PTClist(i),col_PTClist(i)] = size(PTClist);
        [row_dPTClist(i),col_dPTClist(i)] = size(dPTClist);
        [row_A(i),col_A(i)] = size(A);
        [row_d(i),col_d(i)] = size(d);
        if i == 1 && k == 1
            PTClist_prime = -1e12*ones(row_PTClist(i)+5,stack_num);
            dPTClist_prime = -1e12*ones(row_dPTClist(i)+5,stack_num);
            A_prime = -1e12*ones(row_A(i)+5,col_A(i)+5,stack_num);
            d_prime = -1e12*ones(row_d(i)+5,col_d(i),stack_num);
        end
        PTClist_prime(1:row_PTClist(i),i) = PTClist;
        dPTClist_prime(1:row_dPTClist(i),i) = dPTClist;
        A_prime(1:row_A(i),1:col_A(i),i) = A;
        d_prime(1:row_d(i),:,i) = d;
        index_vect(:,:,k+1,i) = index;
        flag_vect(:,k+1,i) = flag;
    end

    % ---------------------------------------------------------------------
    if bartype == 0 || (bartype==1 && damper_ntip<2) || (bartype==2 &&
damper_ntip<1)
        for i = 1:stack_num
            if i == 1
                A_multi = A_prime(1:row_A(i),1:col_A(i),i);
                d1_multi = -
scl*d_prime(1:row_d(i),1:3,i)*Ksrinv(:,1:nio);
                d2_multi = scl*Ksr(1:nio,:)*d_prime(1:row_d(i),1:3,i)';
                d3_multi = d_prime(1:row_d(i),4,i);
```

```
            else
                A_multi =
blkdiag(A_multi,A_prime(1:row_A(i),1:col_A(i),i));
                d1_multi = [d1_multi;-
scl*d_prime(1:row_d(i),1:3,i)*Ksrinv(:,1:nio)];
                d2_multi = [d2_multi
scl*Ksr(1:nio,:)*d_prime(1:row_d(i),1:3,i)'];
                d3_multi = blkdiag(d3_multi,d_prime(1:row_d(i),4,i));
            end
        end
        Aaug_multi = [A_multi d1_multi;d2_multi zeros(nio,nio)];
        daug_multi = blkdiag(d3_multi,eye(nio));
        if rcond(Aaug_multi)<1e-16
            fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug_multi),k);
        end
        % Solve for vector of loop flux and current
        lam_multi = [ifld*ones(stack_num,1);scl*lamqd0srpp(:,k)];
        xg_multi = Aaug_multi\(daug_multi*lam_multi);
        % Identify just the loop fluxes
        fluxm_multi = xg_multi(1:end-nio);
        % NEWTON-RAPHSON SOLVER
        it = 1; % Keeps track of N-R iterations
        NRSOLVE = 1;
        while NRSOLVE
            xg_multi_temp = xg_multi;
            fluxm_multi_temp = fluxm_multi;
            for i = 1:stack_num
                % Assign variables
                Cvconn = Cvconn_prime(:,i);
                Crconn = Crconn_prime(1:row_Crconn(i),:,i);
                O = O_prime(1:row_O(i),1:col_O(i),i);
                d_damper_1 = d_damper_1_prime(1:row_d_damper_1(i),:,i);
                d_damper_2 = d_damper_2_prime(1:row_d_damper_2(i),:,i);
                PTClist = PTClist_prime(1:row_PTClist(i),i);
                dPTClist = dPTClist_prime(1:row_dPTClist(i),i);
                % Find xg and fluxm for each stack
                xg = [xg_multi_temp(1:Nm(i));xg_multi_temp(end-
nio+1:end)];
                xg_multi_temp = removerows(xg_multi_temp,1:Nm(i));
                fluxm = fluxm_multi_temp(1:Nm(i));
                fluxm_multi_temp =
removerows(fluxm_multi_temp,1:Nm(i));
                % DETERMINE FLUXES FOR THE GUESS VECTOR xg
                phi = O*fluxm;
                phiiron(:,k) = phi(1:lB);
                % DETERMINE B-FIELDs
                BIRON(:,k,i) = phiiron(:,k)./areas;
                % Store flux/flux density values after converging
                phit(:,k,i) = phi(S+1:2*S);
                phiag =
phi(4*S+11+D/2+Dsl/2+1+damper_nshank+D/2+2*(SPT-1):end);
                BY(:,k,i) = BIRON(1:S,k,i);
                BT(:,k,i) = BIRON(S+1:2*S,k,i);
                BTT(:,k,i) = BIRON(2*S+1:3*S,k,i);
```

```
                % GET PERMEABILITY FOR EACH RESPECTIVE PERM
                [sMU,sdmdb] = get_mur_exp(BIRON(1:slB,k,i),mudata.s);
                [rMU,rdmdb] =
get_mur_exp(BIRON(slB+1:end,k,i),mudata.r);
                MU = [sMU;rMU];
                dmdb = [sdmdb;rdmdb];
                % UPDATE MATRICIES
                Riron = Rxm./MU;
                [Ag,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn,Cvconn);
                % Pure Newton Raphson Iterator - find Jacobian and
update x for each stack
                Aaug = [Ag -
scl*d(:,1:3)*Ksrinv(:,1:nio);scl*Ksr(1:nio,:)*d(:,1:3)'
zeros(nio,nio)];
                J = get_J(Cr(1:lB,:),O(1:lB,:),Aaug,MU,areas,dmdb,xg);
                DR = J-Aaug;
                DR = DR(1:Nm(i),1:Nm(i));

                if i == 1
                    Ag_multi = Ag;
                    DR_multi = DR;
                    torque(k) =
((RP/2)^2)*(sum(phiag.^2.*dPTClist./(PTClist.^2)));
                else
                    Ag_multi = blkdiag(Ag_multi,Ag);
                    DR_multi = blkdiag(DR_multi,DR);
                    torque(k) = torque(k) +
((RP/2)^2)*(sum(phiag.^2.*dPTClist./(PTClist.^2)));
                end
            end
            % Solve the multi-stack system equations
            Aaug_multi = [Ag_multi d1_multi;d2_multi zeros(nio,nio)];
            daug_multi = blkdiag(d3_multi,eye(nio));
            if rcond(Aaug_multi)<1e-16
                fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug_multi),k);
            end
            DR_multi = blkdiag(DR_multi,zeros(nio,nio));
            J_multi = Aaug_multi+DR_multi;
            xnewp = xg_multi -  J_multi\(Aaug_multi*xg_multi -
daug_multi*lam_multi);

            % Check for convergence
            if ((sqrt((xnewp-xg_multi)'*(xnewp-
xg_multi))/(length(xg_multi)*max(abs([xnewp;xg_multi]))))...
                    < TOL) || (it == parx(14)))
                if (it == parx(14))
                    % Maximum N-R iterations reached
                    disp(['Max Iterations Reached: IT = ' num2str(it)
', Data Point = ' num2str(k)]);
                    nrconverge = 0;
                end
                NRSOLVE = 0;
                nriter(k) = it;
```

```
            else
                xg_multi = xnewp;
                fluxm_multi = xg_multi(1:end-nio);
                it = it+1;
            end
        end
        if ~nrconverge
            break
        end

        % Phase current calculation
        iqd0sr(:,k) = xg_multi(end-nio+1:end)*scl;
        iabc(:,k) = Ksrinv(:,1:nio)*iqd0sr(:,k);
        % Phase flux linkage calculation
        lamabcpp(:,k) = Ksrinv(:,1:nio)*lamqd0srpp(:,k);

    elseif bartype == 1 % ---------------------------------------------
        for i = 1:stack_num
            if i == 1
                A_multi = A_prime(1:row_A(i),1:col_A(i),i);
                d1_multi = -
scl*[d_prime(1:row_d(i),1:3,i)*Ksrinv(:,1:nio) ...
                    d_damper_1_prime(1:row_d_damper_1(i),:,i)];
                d2_multi =
scl*[Ksr(1:nio,:)*d_prime(1:row_d(i),1:3,i)'; ...
                    d_damper_2_prime(1:row_d_damper_2(i),:,i)'];
                d3_multi = d_prime(1:row_d(i),4,i);
            else
                A_multi =
blkdiag(A_multi,A_prime(1:row_A(i),1:col_A(i),i));
                d1_multi = [d1_multi;-
scl*[d_prime(1:row_d(i),1:3,i)*Ksrinv(:,1:nio) ...
                    d_damper_1_prime(1:row_d_damper_1(i),:,i)]];
                d2_multi = [d2_multi
scl*[Ksr(1:nio,:)*d_prime(1:row_d(i),1:3,i)'; ...
                    d_damper_2_prime(1:row_d_damper_2(i),:,i)']];
                d3_multi = blkdiag(d3_multi,d_prime(1:row_d(i),4,i));
            end
        end
        Aaug_multi = [A_multi d1_multi;d2_multi zeros(nio+damper_ntip-
1,nio+damper_ntip-1)];
        daug_multi = blkdiag(d3_multi,eye(nio+damper_ntip-1));
        if rcond(Aaug_multi)<1e-16
            fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug_multi),k);
        end
        lam_multi =
[ifld*ones(stack_num,1);scl*lamqd0srpp(:,k);scl*lamdamper(1:damper_ntip
-1,k)];
        xg_multi = Aaug_multi\(daug_multi*lam_multi);
        % Identify just the loop fluxes
        fluxm_multi = xg_multi(1:end-nio-damper_ntip+1);
        % NEWTON-RAPHSON SOLVER
        it = 1; % Keeps track of N-R iterations
        NRSOLVE = 1;
```

```
        while NRSOLVE
            xg_multi_temp = xg_multi;
            fluxm_multi_temp = fluxm_multi;
            for i = 1:stack_num
                % Assign variables
                Cvconn = Cvconn_prime(:,i);
                Crconn = Crconn_prime(1:row_Crconn(i),:,i);
                O = O_prime(1:row_O(i),1:col_O(i),i);
                d_damper_1 = d_damper_1_prime(1:row_d_damper_1(i),:,i);
                d_damper_2 = d_damper_2_prime(1:row_d_damper_2(i),:,i);
                PTClist = PTClist_prime(1:row_PTClist(i),i);
                dPTClist = dPTClist_prime(1:row_dPTClist(i),i);
                % Find xg and fluxm for each stack
                xg = [xg_multi_temp(1:Nm(i));xg_multi_temp(end-nio-
damper_ntip+2:end)];
                xg_multi_temp = removerows(xg_multi_temp,1:Nm(i));
                fluxm = fluxm_multi_temp(1:Nm(i));
                fluxm_multi_temp =
removerows(fluxm_multi_temp,1:Nm(i));
                % DETERMINE FLUXES FOR THE GUESS VECTOR xg
                phi = O*fluxm;
                phiiron(:,k) = phi(1:lB);
                % DETERMINE B-FIELDs
                BIRON(:,k,i) = phiiron(:,k)./areas;
                % Store flux/flux density values after converging
                phit(:,k,i) = phi(S+1:2*S);
                phiag =
phi(4*S+11+D/2+Dsl/2+1+damper_nshank+D/2+2*(SPT-1):end);
                BY(:,k,i) = BIRON(1:S,k,i);
                BT(:,k,i) = BIRON(S+1:2*S,k,i);
                BTT(:,k,i) = BIRON(2*S+1:3*S,k,i);
                % GET PERMEABILITY FOR EACH RESPECTIVE PERM
                [sMU,sdmdb] = get_mur_exp(BIRON(1:slB,k,i),mudata.s);
                [rMU,rdmdb] =
get_mur_exp(BIRON(slB+1:end,k,i),mudata.r);
                MU = [sMU;rMU];
                dmdb = [sdmdb;rdmdb];
                % UPDATE MATRICIES
                Riron = Rxm./MU;
                [Ag,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn,Cvconn);
                % Pure Newton Raphson Iterator - find Jacobian and
update x for each stack
                Aaug = [Ag -scl*d(:,1:3)*Ksrinv(:,1:nio) -
scl*d_damper_1; ...
                    scl*Ksr(1:nio,:)*d(:,1:3)'
zeros(nio,nio+damper_ntip-1); ...
                    scl*d_damper_2' zeros(damper_ntip-
1,nio+damper_ntip-1)];
                J = get_J(Cr(1:lB,:),O(1:lB,:),Aaug,MU,areas,dmdb,xg);
                DR = J-Aaug;
                DR = DR(1:Nm(i),1:Nm(i));

                if i == 1
                    Ag_multi = Ag;
```

```
                    DR_multi = DR;
                    torque(k) =
((RP/2)^2)*(sum(phiag.^2.*dPTClist./(PTClist.^2)));
                else
                    Ag_multi = blkdiag(Ag_multi,Ag);
                    DR_multi = blkdiag(DR_multi,DR);
                    torque(k) = torque(k) +
((RP/2)^2)*(sum(phiag.^2.*dPTClist./(PTClist.^2)));
                end
            end

            % Solve the multi-stack system equations
            Aaug_multi = [Ag_multi d1_multi;d2_multi
zeros(nio+damper_ntip-1,nio+damper_ntip-1)];
            daug_multi = blkdiag(d3_multi,eye(nio+damper_ntip-1));
            if rcond(Aaug_multi)<1e-16
                fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug_multi),k);
            end
            DR_multi = blkdiag(DR_multi,zeros(nio+damper_ntip-
1,nio+damper_ntip-1));
            J_multi = Aaug_multi+DR_multi;
            xnewp = xg_multi -  J_multi\(Aaug_multi*xg_multi -
daug_multi*lam_multi);

            % Check for convergence
            if ((sqrt((xnewp-xg_multi)'*(xnewp-
xg_multi))/(length(xg_multi)*max(abs([xnewp;xg_multi])))...
                    < TOL) || (it == parx(14)))
                if (it == parx(14))
                    % Maximum N-R iterations reached
                    disp(['Max Iterations Reached: IT = ' num2str(it)
', Data Point = ' num2str(k)]);
                    nrconverge = 0;
                end
                NRSOLVE = 0;
                nriter(k) = it;
            else
                xg_multi = xnewp;
                fluxm_multi = xg_multi(1:end-nio-damper_ntip+1);
                it = it+1;
            end
        end
        if ~nrconverge
            break
        end

        % Phase current calculation
        iqd0sr(:,k) = xg_multi(end-nio-damper_ntip+2:end-
damper_ntip+1)*scl;
        iabc(:,k) = Ksrinv(:,1:nio)*iqd0sr(:,k); % terminals series
connected
        % Phase flux linkage calculation
        lamabcpp(:,k) = Ksrinv(:,1:nio)*lamqd0srpp(:,k);
        % Damper windings current
```

```
        idamper(1:damper_ntip-1,k) = xg_multi(end-
damper_ntip+2:end)*scl;
        idamper(damper_ntip,k) = -sum(idamper(1:damper_ntip-1,k));

    elseif bartype == 2 % ---------------------------------------------
        for i = 1:stack_num
            if i == 1
                A_multi = A_prime(1:row_A(i),1:col_A(i),i);
                d1_multi = -
scl*[d_prime(1:row_d(i),1:3,i)*Ksrinv(:,1:nio) ...
                    d_damper_1_prime(1:row_d_damper_1(i),:,i)];
                d2_multi =
scl*[Ksr(1:nio,:)*d_prime(1:row_d(i),1:3,i)'; ...
                    d_damper_2_prime(1:row_d_damper_2(i),:,i)'];
                d3_multi = d_prime(1:row_d(i),4,i);
            else
                A_multi =
blkdiag(A_multi,A_prime(1:row_A(i),1:col_A(i),i));
                d1_multi = [d1_multi;-
scl*[d_prime(1:row_d(i),1:3,i)*Ksrinv(:,1:nio) ...
                    d_damper_1_prime(1:row_d_damper_1(i),:,i)]];
                d2_multi = [d2_multi
scl*[Ksr(1:nio,:)*d_prime(1:row_d(i),1:3,i)'; ...
                    d_damper_2_prime(1:row_d_damper_2(i),:,i)']];
                d3_multi = blkdiag(d3_multi,d_prime(1:row_d(i),4,i));
            end
        end
        Aaug_multi = [A_multi d1_multi;d2_multi
zeros(nio+damper_ntip,nio+damper_ntip)];
        daug_multi = blkdiag(d3_multi,eye(nio+damper_ntip));
        if rcond(Aaug_multi)<1e-16
            fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug_multi),k);
        end
        lam_multi =
[ifld*ones(stack_num,1);scl*lamqd0srpp(:,k);scl*lamdamper(1:damper_ntip
,k)];
        xg_multi = Aaug_multi\(daug_multi*lam_multi);

        % Identify just the loop fluxes
        fluxm_multi = xg_multi(1:end-nio-damper_ntip);
        % NEWTON-RAPHSON SOLVER
        it = 1; % Keeps track of N-R iterations
        NRSOLVE = 1;
        while NRSOLVE
            xg_multi_temp = xg_multi;
            fluxm_multi_temp = fluxm_multi;
            for i = 1:stack_num
                % Assign variables
                Cvconn = Cvconn_prime(:,i);
                Crconn = Crconn_prime(1:row_Crconn(i),:,i);
                O = O_prime(1:row_O(i),1:col_O(i),i);
                d_damper_1 = d_damper_1_prime(1:row_d_damper_1(i),:,i);
                d_damper_2 = d_damper_2_prime(1:row_d_damper_2(i),:,i);
                PTClist = PTClist_prime(1:row_PTClist(i),i);
```

```
                dPTClist = dPTClist_prime(1:row_dPTClist(i),i);
                % Find xg and fluxm for each stack
                xg = [xg_multi_temp(1:Nm(i));xg_multi_temp(end-nio-
damper_ntip+1:end)];
                xg_multi_temp = removerows(xg_multi_temp,1:Nm(i));
                fluxm = fluxm_multi_temp(1:Nm(i));
                fluxm_multi_temp =
removerows(fluxm_multi_temp,1:Nm(i));
                % DETERMINE FLUXES FOR THE GUESS VECTOR xg
                phi = O*fluxm;
                phiiron(:,k) = phi(1:lB);
                % DETERMINE B-FIELDs
                BIRON(:,k,i) = phiiron(:,k)./areas;
                % Store flux/flux density values after converging
                phit(:,k,i) = phi(S+1:2*S);
                phiag =
phi(4*S+11+D/2+Dsl/2+1+damper_nshank+D/2+2*(SPT-1):end);
                BY(:,k,i) = BIRON(1:S,k,i);
                BT(:,k,i) = BIRON(S+1:2*S,k,i);
                BTT(:,k,i) = BIRON(2*S+1:3*S,k,i);
                % GET PERMEABILITY FOR EACH RESPECTIVE PERM
                [sMU,sdmdb] = get_mur_exp(BIRON(1:slB,k,i),mudata.s);
                [rMU,rdmdb] =
get_mur_exp(BIRON(slB+1:end,k,i),mudata.r);
                MU = [sMU;rMU];
                dmdb = [sdmdb;rdmdb];
                % UPDATE MATRICIES
                Riron = Rxm./MU;
                [Ag,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn,Cvconn);
                % Pure Newton Raphson Iterator - find Jacobian and
update x for each stack
                Aaug = [Ag -scl*d(:,1:3)*Ksrinv(:,1:nio) -
scl*d_damper_1; ...
                    scl*Ksr(1:nio,:)*d(:,1:3)'
zeros(nio,nio+damper_ntip); ...
                    scl*d_damper_2'
zeros(damper_ntip,nio+damper_ntip)];
                J = get_J(Cr(1:lB,:),O(1:lB,:),Aaug,MU,areas,dmdb,xg);
                DR = J-Aaug;
                DR = DR(1:Nm(i),1:Nm(i));

                if i == 1
                    Ag_multi = Ag;
                    DR_multi = DR;
                    torque(k) =
((RP/2)^2)*(sum(phiag.^2.*dPTClist./(PTClist.^2)));
                else
                    Ag_multi = blkdiag(Ag_multi,Ag);
                    DR_multi = blkdiag(DR_multi,DR);
                    torque(k) = torque(k) +
((RP/2)^2)*(sum(phiag.^2.*dPTClist./(PTClist.^2)));
                end
            end
```

```
            % Solve the multi-stack system equations
            Aaug_multi = [Ag_multi d1_multi;d2_multi
zeros(nio+damper_ntip,nio+damper_ntip)];
            daug_multi = blkdiag(d3_multi,eye(nio+damper_ntip));
            if rcond(Aaug_multi)<1e-16
                fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Aaug_multi),k);
            end
            DR_multi =
blkdiag(DR_multi,zeros(nio+damper_ntip,nio+damper_ntip));
            J_multi = Aaug_multi+DR_multi;
            xnewp = xg_multi -  J_multi\(Aaug_multi*xg_multi -
daug_multi*lam_multi);

            % Check for convergence
            if ((sqrt((xnewp-xg_multi)'*(xnewp-
xg_multi))/(length(xg_multi)*max(abs([xnewp;xg_multi])))...
                    < TOL) || (it == parx(14)))
                if (it == parx(14))
                    % Maximum N-R iterations reached
                    disp(['Max Iterations Reached: IT = ' num2str(it)
', Data Point = ' num2str(k)]);
                    nrconverge = 0;
                end
                NRSOLVE = 0;
                nriter(k) = it;
            else
                xg_multi = xnewp;
                fluxm_multi = xg_multi(1:end-nio-damper_ntip);
                it = it+1;
            end
        end
        if ~nrconverge
            break
        end

        % Phase current calculation
        iqd0sr(:,k) = xg_multi(end-nio-damper_ntip+1:end-
damper_ntip)*scl;
        iabc(:,k) = Ksrinv(:,1:nio)*iqd0sr(:,k); % terminals series
connected
        % Phase flux linkage calculation
        lamabcpp(:,k) = Ksrinv(:,1:nio)*lamqd0srpp(:,k);
        % Damper windings current
        idamper(:,k) = xg_multi(end-damper_ntip+1:end)*scl;
    end
    %-------------------------------------------------------------------
    % External voltage model-----------------------------------------
    % R load
%     vabc(:,k) = -iabc(:,k)*Rload;
    % Parallel RL load
    vabc(:,k) = (-iabc(:,k)-il(:,k))*Rload;
    pil(:,k) = vabc(:,k)/Lload;
    il(:,k+1) = il(:,k)+pil(:,k)*DT;
```

```
    % qd voltage calculation
    vqd0sr(:,k) = Ksr(1:nio,:)*vabc(:,k); % Terminals series connected

    % Connected to rectifier with constant vdc
    %     iabcl = m_isil*iabc(:,k);
    %     [V,idc(k)] = rect(iabcl,vdcmax,parx);
    %     vqd0gr = Ksr*V;
    %     vqd0sr(:,k) = m_vgvs*vqd0gr;
    %     vabc(:,k) = Ksrinv(:,1:nio)*vqd0sr(:,k);
    % Connected to rectifier with RLC load
    %     iabcl = m_isil*iabc(:,k);
    %     [V,idc(k)] = rect(iabcl,vdc(k),parx);
    %     vqd0gr = Ksr*V;
    %     vqd0sr(:,k) = m_vgvs*vqd0gr;
    %     vabc(:,k) = Ksrinv(:,1:nio)*vqd0sr(:,k);
    %     pvc(k) = (idc(k)-vc(k)/Rload)/Cload;
    %     vc(k+1) = vc(k)+pvc(k)*DT;
    %     Ivc(k+1) = Ivc(k)+(vc(k+1)+vc(k))/2*DT;
    %     vdc(k+1) = (-
(Ivdc(k)+vdc(k)*DT/2)+taus*vc(k+1)+Ivc(k+1)+Lload*idc(k))/(taus+DT/2);
    %     Ivdc(k+1) = Ivdc(k)+(vdc(k+1)+vdc(k))/2*DT;
    %-----------------------------------------------------------------

    % Forward Euler to solve state model------------------------------
    plamqd0srpp(:,k) = (vqd0sr(:,k) - rs.*iqd0sr(:,k) -
wr*mlam*lamqd0srpp(:,k)*RP)/RP;
    lamqd0srpp(:,k+1) = lamqd0srpp(:,k) + plamqd0srpp(:,k)*DT;

    if bartype == 1
        plamdamper(1:damper_ntip-1,k) = -Tdp*idamper(1:damper_ntip-
1,k);
        lamdamper(:,k+1) = lamdamper(:,k) + plamdamper(:,k)*DT;
    elseif bartype == 2
        plamdamper(:,k) = -Tdp*idamper(:,k);
        lamdamper(:,k+1) = lamdamper(:,k) + plamdamper(:,k)*DT;
    end
    %-----------------------------------------------------------------

    % Increment time/rotor position
    k = k+1;
end

% Check for flux densities above limit
Bsat = parx(23);
maxB = max(abs(BIRON));
saturate(maxB>=Bsat) = 1./(1+abs((maxB(maxB>=Bsat)-Bsat)./(0.1*Bsat)));

end
```

```
%-------------------------------------------------------------------------
% AUTHORS:  Xiaoqi Wang, Michelle Bash, Steven D. Pekarek
%-------------------------------------------------------------------------
% CONTACT:  School of Electrical & Computer Engineering
%           Purdue University
%           465 Northwestern Ave.
%           West Lafayette, IN 47907
%           765-494-3434, spekarek@ecn.purdue.edu
%-------------------------------------------------------------------------
% April 1, 2012
%-------------------------------------------------------------------------
% [t,ias,ibs,ics,torque,qrm,phit,BY,BT,nrconverge,saturate,BIRON] =
% wrsmdynamics_ss_multislice(parx,pars,turns,mudata)
%
% Solves the MEC network.
%
% OUTPUTS: t        - time vector (s)
%          ias,ibs,ics  - phase currents (s)
%          torque   - torque (Nm)
%          qrm      - mechanical rotor position (radians)
%          phit     - stator teeth flux (Wb)
%          BY,BT,BTT    - flux density in the stator yoke, stator
teeth, and stator tooth tips (T)
%          nrconverge - flag indicating if newton raphson converged
%          saturate   - indicates if the flux density limit is violated
%          BIRON    - flux density in iron (Wb)
%
% INPUTS:  pars     - geometric parameters
%          parx     - simulation parameters
%          turns    - phase winding turns (turn count)
%          mudata   - magnetic material data for finding permeability
%-------------------------------------------------------------------------
function
[t,ias,ibs,ics,torque,qrm,phit,BY,BT,BTT,nrconverge,saturate,BIRON] =
wrsmdynamics_ss_multislice (parx,pars,turns,damperdata,mudata,qr_init)
%-------------------------------------------------------------------------
% INITIALIZE THE SYSTEM
%-------------------------------------------------------------------------
DT      = parx(12);                 % Time step in s
iter    = parx(30);                 % Number of iterations
wrm     = parx(4)*2*pi/60;          % Mechanical rotor speed in rad/s
ifld    = pars(47);                 % Field current (A)
irms    = pars(49);                 % rms Stator current (A)
iph     = pars(50);                 % Current phase angle (degrees)
im      = irms*sqrt(2);             % Magnitude of ias,ibs,ics
mu0     = pi*4e-7;      % Permeability of free space
RP      = pars(28);     % Poles
S       = parx(3)/RP;   % Number of stator slots per pole
D       = 2*(parx(2));  % Number of rotor pole tip sections per pole
pair
Dsl     = 4*parx(29);   % Number of inter-polar regions per pole pair
SPT     = parx(2);      % SECTIONS PER ROTOR TOOTH, including radial
and tangential
NRrtrt  = parx(27);     % Number of outer pole tip reluctances per pole
pair
```

```
damper_ntip = damperdata.damper_ntip;     % Number of damper windings
on rotor tip
damper_nshank = damperdata.damper_nshank; % Number of damper windings
on rotor shank
bartype = damperdata.bartype; % Type of damper bars connnection
Rd = damperdata.Rd;     % Damper bar body resistance
Re = damperdata.Re;     % Damper bar end connection resistance
NPTS    = parx(7);                  % NUMBER OF DATA POINTS PER CYCLE
skew_angle = pars(30);              % Electrical skew angle, rad
stack_num = pars(31);               % Number of stack for skew
% INITIALIZE VARIABLES
slB     = 3*S; % Number of iron elements in stator
rlB     = 6+D/2+damper_nshank+SPT+(SPT-1); % Number of iron elements in
rotor
lB      = slB+rlB; % Number of iron elements
nriter  = zeros(1,iter); % Keeps track of N-R iterations
torque  = zeros(1,iter);
PTC     = zeros(S,D+Dsl,iter); % Matrix of airgap permeances
dPTC    = zeros(S,D+Dsl,iter);
phit    = zeros(S,iter,stack_num); % Stator tooth flux
phiiron = zeros(lB,iter); % Flux in iron
BY      = zeros(S,iter,stack_num); % Stator yoke flux density
BT      = zeros(S,iter,stack_num); % Stator tooth shank flux density
BTT     = zeros(S,iter,stack_num); % Stator tooth tip flux density
BIRON   = zeros(lB,iter,stack_num); % Flus density in all iron elements
saturate = ones(1,iter); % Saturation constraint (is Bsat violated)
smuiron = get_mur_exp(zeros(slB,1),mudata.s); % Initial permeabilities
of stator
rmuiron = get_mur_exp(zeros(rlB,1),mudata.r); % Initial permeabilities
of rotor
muiron  = [smuiron;rmuiron]; % Initial permeabilities
TOL     = parx(21); % tolerance for convergence of Newton-Raphson
k       = 1; % Simulation step
t(k)    = parx(10);
% ARTIFICIAL ROTOR POSITION MODIFICATION used in the calculation of
airgap
% permeances.----
SLL     = parx(3);
ID      = pars(2);
ROD     = pars(24);
STTW    = pars(21);
WRT     = pars(34);
WAIRT   = pars(35);
shift1  = WRT/(ROD/2);
shift2  = (WAIRT/2)/(ROD/2);
shift3  = 2*pi/SLL;
shift4  = (STTW/2)/(ID/2);
shift5  = (pi/2)/(RP/2);
shift   = shift1 + shift2 - (S/2)*shift3 - shift4 - shift5;
% TIME AND ROTOR POSITION VECTORS
t       = (0:DT:DT*(iter-1))+t(k);
qrm     = t*wrm + qr_init/(RP/2) ; % Actual rotor position
qrm_shift =  qrm + shift +pi/12+pi/4; % Angle fed to airgap permeance
function
%-------------------------------------------------------------------
```

```
% CALCULATE VARIABLES/MATRICES WHICH WILL NOT CHANGE DURING SIM
%-------------------------------------------------------------------------
% Variables/matrices to be used in airgap permeance calculation
WRS     = pars(35)/(2*parx(29));
WRTS    = pars(36);
B0      = pars(9);
SPT     = parx(2);
RPIT    = pars(32);
WRTSang = 2*pi*RPIT/RP/SPT;
WRTang  = 2*pi*RPIT/RP;
WRSang  = 2*pi*(1-RPIT)/RP/(Dsl/2);
qs      = STTW/ID*RP; % Span of stator tooth in electrical radians
qs1     = B0/ID*RP; % Span of stator slot
qrr     = WRTSang*RP/2; % Span of rotor pole tip section
qrs     = WRSang*RP/2;  % Span of inter-polar section
Gmaxrt = pi*4e-7*pars(3)/stack_num/(ID-
ROD)*2*(WRTS*(STTW>=WRTS)+STTW*(STTW<WRTS)); % if-else
Gmaxsl = pi*4e-7*pars(3)/stack_num/(ID-
ROD)*2*(WRS*(STTW>=WRS)+STTW*(STTW<WRS)); % if-else
rt      = 1:D; rtsl    = 1:Dsl; st      = (1:S)';
% Matrices defining the angle between every stator tooth and rotor
section
anglert = ones(S,1)*(-mod(rt-1,(D/2))*WRTSang - floor((rt-
1)/(D/2))*2*pi/RP)...
    + ((st-1)*(STTW+B0)/(ID/2))*ones(1,D);
anglesl = ones(S,1)*(-WRTang - mod(rtsl-1,(Dsl/2))*WRSang - ...
    floor((rtsl-1)/(Dsl/2))*2*pi/RP) + ((st-
1)*(STTW+B0)/(ID/2))*ones(1,Dsl);

% Establish the geometric case for the rotor tooth section
if qrr <= qs1/2
    qrrcs = 1;
elseif (qrr <= qs)
    qrrcs = 2;
elseif (qrr <= qs +qs1/2)
    qrrcs = 3;
elseif (qrr <= qs+qs1)
    qrrcs = 4;
else
    qrrcs = 5;
end
% Establish the geometric case for the rotor slot section
if qrs <= qs1/2
    qrscs = 1;
elseif (qrs <= qs)
    qrscs = 2;
elseif (qrs <= qs +qs1/2)
    qrscs = 3;
elseif (qrs <= qs+qs1)
    qrscs = 4;
else
    qrscs = 5;
end

% -------------------------------------------------------------------------
```

```
% turns matrix to be used in system of equations
Natrn   = [-turns turns]';
Nbtrn   = [Natrn(2*SLL/(3*RP)+1:end);Natrn(1:2*SLL/(3*RP))];
Nctrn   = [Natrn(4*SLL/(3*RP)+1:end);Natrn(1:4*SLL/(3*RP))];
Nabc    = [Natrn Nbtrn Nctrn];
Nfld    = pars(41);
Nabcf   = [Nabc(1:S,:) zeros(S,1);0 0 0 Nfld;0 0 0 -Nfld];
% ------------------------------------------------------------------------
% MEC loops with MMF sources
Cvcfixed = (1:S+2)';
%-------------------------------------------------------------------
% Calculate the reluctances
[Rxm,areas,Rair,NPRTS,NPRTB] =
get_reluctances(mu0,parx,pars,damperdata);
Riron = Rxm./muiron;
%-------------------------------------------------------------------
% Identify type of node in rotor tooth and slot
% 1 = node of rotor pole tip radial branch
% 2 = node of rotor pole tip tangential branch
% 3 = rotor slot branch going to rotor edge
% 4 = rotor slot branch going to bottom of rotor pole tip
rtid = [2*ones(NRrtrt,1);ones(D/2-2*NRrtrt,1);2*ones(NRrtrt,1);...
        3*ones(NPRTS,1);4*ones(2*NPRTB,1);3*ones(NPRTS,1);...
        2*ones(NRrtrt,1);ones(D/2-2*NRrtrt,1);2*ones(NRrtrt,1);...
        3*ones(NPRTS,1);4*ones(2*NPRTB,1);3*ones(NPRTS,1)];
% Identify how many RRTOUT branches border the rotor loop
NRBRL = ceil((NRrtrt+1)/2);  % Number of RRTOUT branches Bordering
Rotor Loop
NRTBD = NRrtrt-NRBRL; % Number of RRTOUT branches with bordering loop
To Be Determined
% ------------------------------------------------------------------------
% Define reluctance connections in stator and rotor which do not change
% Stator tooth tip, damper slots, and leakage of damper slots are not
% presented here, but will be derived as postprocess in shape_alg.m
% IRON
% Stator yoke - S
% Stator teeth - S
% Rotor yoke below the slot - 1
% Rotor tooth shank - 1
% Rotor yoke connected to shank - 2
% Rotor tooth tips radial - (D - 4*NRrtrt)
% Rotor tooth to rotor tooth tangential - 4*NRrtrt
% Rotor tooth tangential at sides of tooth tips - 4
% AIR
% Stator tooth leakage - S
% Field winding leakage - 2
% Middle rotor slot leakage - 2
% Fringing permeance from rotor side to airgap boundary - Dsl
% Fringing permeance from rotor slot side to bottom of tooth tip - 4
% RY R RRYSL RRTSH RRYSH RRTIN RRTOUT RRTS RSTL RFDL RRTL RAGFR RFRB
Crcfixed = zeros(2*S+8+D+S+3+Dsl,3);
% RY (all)
Crcfixed(1:S,2)=(1:S)';
% R (all)
Crcfixed(S+1:2*S,2) = [1 2:S]';
```

```
Crcfixed(S+1:2*S,3) = [-S 1:S-1]';
% RRYSL (all)
Crcfixed(2*S+1,3) = S+3;
% RRTSH (all)
Crcfixed(2*S+2,2:3) = [S+1 S+2];
% RRYSH (all)
Crcfixed(2*S+2+(1:2)',2) = [S+1;S+2];
% RRTIN (Determined by shape algorithm)
% RRTOUT - One side known if reluctance borders rotor loop
Crcfixed(2*S+2+D-4*NRrtrt+2+(1:4*NRrtrt)',2) = ...

[[zeros(NRTBD,1);ones(NRBRL,1)]*(S+1);[ones(NRBRL,1);zeros(NRTBD,1)]*(S
+2);...
 -[zeros(NRTBD,1);ones(NRBRL,1)]*(S+1);-
[ones(NRBRL,1);zeros(NRTBD,1)]*(S+2)];
% RRTS - (Determined by shape algorithm)
% RSTL (one side known, use shape alg for other)
Crcfixed(2*S+2+D+6+(1:S)',2) = (1:S)';
% RFDL (all)
Crcfixed(2*S+2+D+6+S+(1:2)',2:3) = [-(S+3) S+1;S+2 S+3];
% RRTL (one side known, use shape alg for other)
Crcfixed(2*S+2+D+6+S+2+(1:2)',2) = [S+3;-(S+3)];
% RAGFR - (Determined by shape algorithm)
% RFRB (one side, use shape alg for other)
Crcfixed(2*S+2+D+6+S+4+Dsl+(1:4)',2) = [-(S+3);S+3;S+3;-(S+3)];
%-----------------------------------------------------------------
% Initialization
index_vect = zeros(damper_ntip,3,iter+1,stack_num);
flag_vect = ones(damper_ntip,iter+1,stack_num);
% Calculate the currents
ias = im*cos((RP/2)*(qrm) + (pi*iph/180));
ibs = im*cos((RP/2)*(qrm) + (pi*iph/180) - (2*pi/3));
ics = im*cos((RP/2)*(qrm) + (pi*iph/180) - (4*pi/3));
curr = [ias;ibs;ics;ifld*ones(1,iter)];

%-------------------------------------------------------------------------
% SOLVING LOOP
%-------------------------------------------------------------------------
nrconverge = 1;
if stack_num == 1
    stack_span = 0;
else
    stack_span = floor(skew_angle/(2*pi)*NPTS/(stack_num-1));
end
% AIR-GAP PERMEANCES
for i = 1:iter
    [PTC(:,:,i),dPTC(:,:,i)] =
get_Pag(qrm_shift(i),pars,parx,Gmaxrt,Gmaxsl,anglert,anglesl,qrrcs,qrsc
s);
end
[l,m,n] = size(PTC);
PTC_prime = zeros(l,m,n,stack_num);
dPTC_prime = zeros(l,m,n,stack_num);
for i = 1:stack_num
```

```
    PTC_prime(:,:,1:(i-1)*stack_span,i) = PTC(:,:,end-(i-
1)*stack_span+1:end);
    PTC_prime(:,:,(i-1)*stack_span+1:end,i) = PTC(:,:,1:end-(i-
1)*stack_span);
    dPTC_prime(:,:,1:(i-1)*stack_span,i) = dPTC(:,:,end-(i-
1)*stack_span+1:end);
    dPTC_prime(:,:,(i-1)*stack_span+1:end,i) = dPTC(:,:,1:end-(i-
1)*stack_span);
end

while k <= iter
    % Shape algorithm - Find the loop topology in the airgap if it has
changed
    for i = 1:stack_num
        if k==1 || sum(sum((PTC_prime(:,:,k-
1,i)~=0)~=(PTC_prime(:,:,k,i)~=0)))>0
            [Crconn,Cvconn,O,PTCind,d_damper_1,d_damper_2,index,flag]
...
                =
shape_alg(PTC_prime(:,:,k,i),parx,pars,damperdata,Crcfixed,Cvcfixed,rti
d,index_vect(:,:,k,i),flag_vect(:,k,i));
            if length(Crconn)~=length([Riron;Rair;PTCind])
                nrconverge = 0;
                break
            end
            % Save variables
            [row_Crconn(i),col_Crconn(i)] = size(Crconn);
            [row_O(i),col_O(i)] = size(O);
            [row_PTCind(i),col_PTCind(i)] = size(PTCind);
            if i == 1 && k == 1
                Crconn_prime = -
1e12*ones(row_Crconn(i)+5,col_Crconn(i),stack_num);
                O_prime = -1e12*ones(row_O(i)+5,col_O(i)+5,stack_num);
                PTCind_prime = -1e12*ones(row_PTCind(i)+5,stack_num);
            end
            Cvconn_prime(:,i) = Cvconn;
            Crconn_prime(1:row_Crconn(i),:,i) = Crconn;
            O_prime(1:row_O(i),1:col_O(i),i) = O;
            PTCind_prime(1:row_PTCind(i),i) = PTCind;
        end
        % Obtain list of airgap permeances and their derivatives for
this rotor position
        ptc           = PTC_prime(:,:,k,i)';
        PTClist       = ptc(PTCind_prime(1:row_PTCind(i),i));
        dptc          = dPTC_prime(:,:,k,i)';
        dPTClist      = dptc(PTCind_prime(1:row_PTCind(i),i));
        % Find the system of equations and solve for the initial guess
        [A,d] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn_prime(1:row_
Crconn(i),:,i),Cvconn_prime(:,i));
        % Total number of meshes
        Nm(i) = 3 + S + length(PTClist) + (SPT-1);
        % Save variables
        [row_PTClist(i),col_PTClist(i)] = size(PTClist);
        [row_dPTClist(i),col_dPTClist(i)] = size(dPTClist);
```

```
        if i == 1 && k == 1
            PTClist_prime = -1e12*ones(row_PTClist(i)+5,stack_num);
            dPTClist_prime = -1e12*ones(row_dPTClist(i)+5,stack_num);
        end
        PTClist_prime(1:row_PTClist(i),i) = PTClist;
        dPTClist_prime(1:row_dPTClist(i),i) = dPTClist;
        index_vect(:,:,k+1,i) = index;
        flag_vect(:,k+1,i) = flag;

        if i == 1
            A_multi = A;
            d_multi = d;
        else
            A_multi = blkdiag(A_multi,A);
            d_multi = [d_multi;d];
        end
    end
    xg_multi = A_multi\(-d_multi*curr(:,k));
    % NEWTON-RAPHSON SOLVER
    it = 1; % Keeps track of N-R iterations
    NRSOLVE = 1;
    while NRSOLVE
        xg_multi_temp = xg_multi;
        for i = 1:stack_num
            % Assign variables
            Cvconn = Cvconn_prime(:,i);
            Crconn = Crconn_prime(1:row_Crconn(i),:,i);
            O = O_prime(1:row_O(i),1:col_O(i),i);
            PTClist = PTClist_prime(1:row_PTClist(i),i);
            dPTClist = dPTClist_prime(1:row_dPTClist(i),i);
            % Find xg and fluxm for each stack
            xg = xg_multi_temp(1:Nm(i));
            xg_multi_temp = removerows(xg_multi_temp,1:Nm(i));
            % DETERMINE FLUXES FOR THE GUESS VECTOR xg
            phi = O*xg;
            phiiron(:,k) = phi(1:lB);
            % DETERMINE B-FIELDs
            BIRON(:,k,i) = phiiron(:,k)./areas;
            % Store flux/flux density values after converging
            phit(:,k,i) = phi(S+1:2*S);
            phiag = phi(4*S+11+D/2+Dsl/2+1+damper_nshank+D/2+2*(SPT-
1):end);
            BY(:,k,i) = BIRON(1:S,k,i);
            BT(:,k,i) = BIRON(S+1:2*S,k,i);
            BTT(:,k,i) = BIRON(2*S+1:3*S,k,i);
            % GET PERMEABILITY FOR EACH RESPECTIVE PERM
            [sMU,sdmdb] = get_mur_exp(BIRON(1:slB,k,i),mudata.s);
            [rMU,rdmdb] = get_mur_exp(BIRON(slB+1:end,k,i),mudata.r);
            MU = [sMU;rMU];
            dmdb = [sdmdb;rdmdb];
            % UPDATE MATRICIES
            Riron = Rxm./MU;
            [Ag,d,Cr] =
get_meshmatrices(Rair,PTClist,Riron,parx,pars,Nabcf,Crconn,Cvconn);
```

```
            % Pure Newton Raphson Iterator - find Jacobian and update x
for each stack
            J = get_J(Cr(1:lB,:),O(1:lB,:),Ag,MU,areas,dmdb,xg);
            DR = J-Ag;

            if i == 1
                Ag_multi = Ag;
                d_multi = d;
                DR_multi = DR;
                torque(k) =
((RP/2)^2)*(sum(phiag.^2.*dPTClist./(PTClist.^2)));
            else
                Ag_multi = blkdiag(Ag_multi,Ag);
                d_multi = [d_multi;d];
                DR_multi = blkdiag(DR_multi,DR);
                torque(k) = torque(k) +
((RP/2)^2)*(sum(phiag.^2.*dPTClist./(PTClist.^2)));
            end
        end
        % Solve the multi-stack system equations
        if rcond(Ag_multi)<1e-16
            fprintf('Warning: rcond(Aaug) = %d at
k=%i.\n',rcond(Ag_multi),k);
        end
        J_multi = Ag_multi+DR_multi;
        xnewp = xg_multi -  J_multi\(Ag_multi*xg_multi +
d_multi*curr(:,k));
        % Check for convergence
        if ((sqrt((xnewp-xg_multi)'*(xnewp-
xg_multi))/(length(xg_multi)*max(abs([xnewp;xg_multi])))...
                < TOL) || (it == parx(14)))
            if (it == parx(14))
                % Maximum N-R iterations reached
                disp(['Max Iterations Reached: IT = ' num2str(it) ',
Data Point = ' num2str(k)]);
                nrconverge = 0;
            end
            NRSOLVE = 0;
            nriter(k) = it;
        else
            xg_multi = xnewp;
            it = it+1;
        end
    end
    if ~nrconverge
        break
    end
    % Increment time/rotor position
    k = k+1;
end
% Check for flux densities above limit
Bsat = parx(23);
maxB = max(abs(BIRON));
saturate(maxB>=Bsat) = 1./(1+abs((maxB(maxB>=Bsat)-Bsat)./(0.1*Bsat)));
```

VITA

VITA

Xiaoqi (Ron) Wang, obtained his Bachelor degree at the Electrical Engineering department in Zhejiang University, China. He is currently pursuing his Ph.D. in the area of Power and Energy Devices and Systems at the ECE department in Purdue University. As a research assistant, he is responsible for multiple design, modeling, and simulation projects in the areas of Electromechanical Devices, Power Electronics, Design of Electric Machines, and Control of Drives.