Effects of Turn-to-Turn Faults in Air-Core Reactors

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Electrical Engineering

in the

College of Graduate Studies

University of Idaho

by

Rowdy A. Sanford

Approved by:

Major Professor: Joseph Law, Ph.D., P.E.

Committee Members: Brian Johnson, Ph.D., P.E., Hangtian Lei, Ph.D., P.E.

Department Administrator: Joseph Law, Ph.D., P.E.

May 2023

Abstract

Air-Core Reactors are employed in shunt configurations to maintain the line voltage within limits. The reactors consume VARs under lightly loaded conditions, countering the shunt capacitance of transmission lines. Turn-to-Turn faults are commonly observed in the high-side of the reactor windings, and are attributed to the degradation of insulation caused by transients from switching or surge conditions. The fault behavior in these air-core reactors is often observed, but not fully understood. This document is presented as a summary of work developing methods to evaluate the behavior of air-core reactors due to inter-turn shorts. Modeling tools developed for the analysis of turn-to-turn faults, and examples of use are provided, with examples of faults in reactors similar to those used in high-voltage shunt applications.

Contents

List of Figures

List of Tables

Preliminary

Terminology Overview

There are a number of elements of an Air-Core Reactor discussed throughout this document. To reduce the chances of confusion while reading, the following are reactor terminology definitions for your reference.

- **Loop**: A loop is a single turn of a conductor at a specified radius (normally r_{loop}).

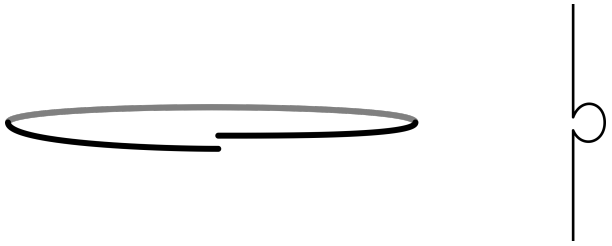


Figure 1: A Loop, with the schematic representation to the right

- **Layer**: A set of N turns connected in series, with the same radius $r_{loop\ 1} = r_{loop\ 2} = ... = r_{loop\ N}$.

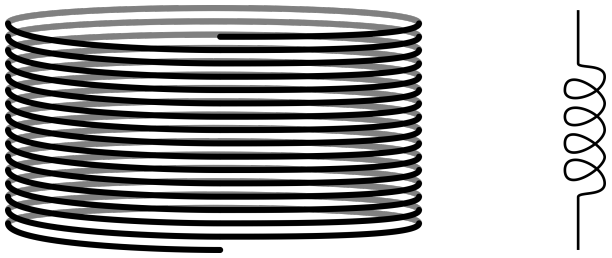


Figure 2: A Layer, with the schematic representation to the right

- **Package**: A contiguous set of parallel layers, usually encapsulated with an epoxy resin.

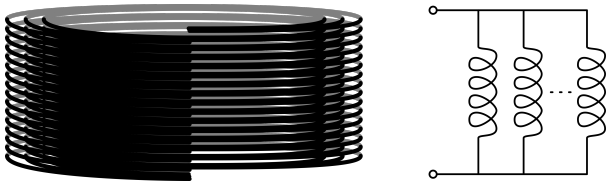


Figure 3: A Package, with the schematic representation to the right

- **Reactor**: When referred to as a element of a circuit is a set of *packages* connected in parallel which are separated by cooling ducts.

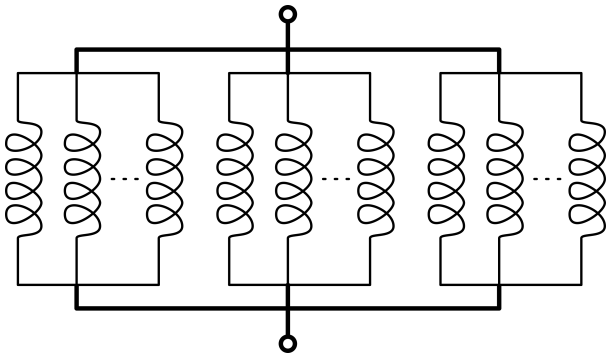


Figure 4: A Reactor, represented as a schematic.

- **Phase**: A series of reactors, usually with two or more stacked coaxially.

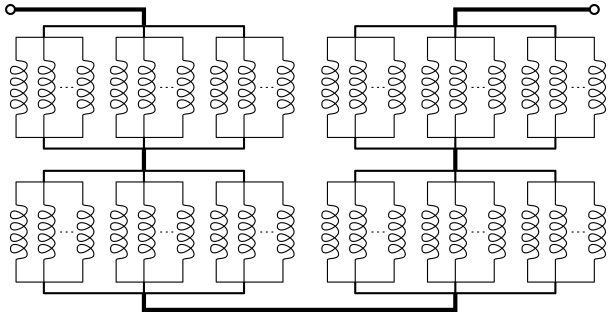


Figure 5: A Phase, represented as a schematic.

Notation

Lowercase variables, i.e. a, b, etc., represent scalars, or individual parameters. Specific parameters are given as uppercase, such as Impedance Z or Current I. Vectors are given with a arrow above the variable, so a current vector would be: $\vec{I}$. Matrices have a bar above the variable, for example an

impedance matrix: $\bar{Z}$. Units are given within square brackets to distinguish from variables, for example meters $\Rightarrow [m]$. In this document the complex variable is $j = \sqrt{-1}$.

List of Symbols and Units

Symbol	units	description
ρ	Ohm-meters, $\frac{\Omega \cdot m^2}{m}$	Resistivity, usually assumed to be Aluminum $2.65 \times 10^{-7} \Omega \cdot m$
ψ	Webers,Wb	Flux
$Z,\ \bar{Z}$	Ohms, Ω	Impedance
X	Ohms, Ω	Reactance
$Y,\ \bar{Y}$	Mhos, $\Omega^{-1} = \mho$	Admittance
$I,\ \vec{I}$	Amps, A	Current
$V,\ \vec{V}$	Volts, V	Voltage
L	Henry, $H = \frac{Wb}{A}$	Inductance
M	Henry, H	Mutual Inductance, M_{ab} is mutual between elements a and b.
$\mathcal{R}$	$\mathcal{R} = H^{-1} \rightarrow \frac{A}{Wb}$	Reluctance
μ	$\frac{Wb}{A \cdot m}$	Relative Permeability, $\mu = \mathbf{B}/\mathbf{H}$
μ_0	$\frac{Wb}{A \cdot m}$	Permeability of Air, $\mu_0 = 4\pi \times 10^{-7}$

Table 1: Symbols and Units

Chapter 1

Introduction

1.1 Introduction

Air-Core Reactors (ACR) are increasingly deployed by utilities in place of oil-immersed reactors, for a number of reasons including the lower capital cost and reduction of maintenance [1]. Surge conditions and transients from switching can damage inter-turn insulation, which is a challenge to detect. When insulation is damaged, shorts can occur between winding turns and cause significant damage before being detected. Models of ACRs under fault conditions are needed to develop protection schemes, and determine if, and for how long, a faulted reactor can be operated with faults in place.

The ACR is modeled with the Biot-Savart Law, and Ampère's Law. Using programs written in Python, the reactor is first designed and prefault conditions are evaluated. Turn-to-Turn faults are a common fault type, and are the focus of this thesis. The development of the modeling method to evaluate the effects of the turn-to-turn faults is presented and demonstrated with a reactor model approximating multi-layered units.

1.2 Motivation

An Air-Core Reactor is a passive component used in operation of the power grid to help maintain line voltage within acceptable limits. Air-Core Reactors can be installed in locations where oil-immersed reactors could pose a risk to the environment. An air-core reactor is cooled via convection, where an oil immersed unit relies on oil for the removal of heat. The oil requires periodic maintenance, and used oil has additional requirements for disposal. Faults in ACRs commonly occur in the high-voltage end of the winding as a result of transients from surges or when energized, these transient conditions can stress and cause breakdown of the insulation and encapsulation of the windings [2].

When the breakdown of insulation progresses sufficiently, it becomes a turn-to-turn fault. Detecting faults for the purpose of protecting Air-Core Reactors is important to utilities and operators.

1.3 Contribution

The focus of the research being presented is the effects of turn-to-turn faults in air-core reactors. In the presentation of this work, the contributions of this thesis are:

- Present a computationally efficient method of modeling and analyzing Air-Core Reactors under normal and faulted conditions.
- Show how the reactor behavior changes due to a fault.
- Present a simplified analogue to describe the fault behavior as an $N:1$ transformer.

Numeric results were attained using programs written in the Python Programming language and the NumPy and SciPy Packages.

1.4 Overview

Each chapter is a general subtopic of the Air-Core Reactor.

The second chapter, *Background*, discusses the theoretical foundations of reactor modeling, starting with the Biot-Savart law, which describes the magnetic field intensity at a point due to a current. It then expands into more specific equations to model series of concentric, coaxial current loops.

The third chapter, *Reactor Modeling*, covers the specific application of the theory presented in the Background chapter to model an air-core reactor.

The fourth chapter, *Fault Modeling*, presents a technique to describe turn-to-turn faults that is computationally efficient. Additionally, examples are given relating a fault in a reactor to an intuitive transformer model, which should be familiar to engineers.

The fifth chapter, *Fault Detection*, is a discussion of fault detection using information from reactor and fault modeling.

The sixth chapter, *Computer Program*, discusses the reasoning behind the computer programs used for design and evaluation of the ACR.

The results in chapter 7, give experimental results from the computer program simulating a reactor model similar to units deployed in industry.

Finally, the conclusions and discussion of future work. The research presented in this document isn't complete, but provides a groundwork for a more targeted study of device behavior.

1.5 Literature Review

1.5.1 Modeling Reactors

Paul published a book in 2010 [3] in which he gives a thorough and detailed explanation of the various methods for computing the flux and inductance of current carrying loops, derived from Gauss' Law, the Biot-Savart law, and Amperes' Law. In chapter 4.4 Paul presents the Neumann integral as a method of calculating the mutual inductance between two concentric, circular loops. The Neumann integral cannot be evaluated to a closed-form solution, but can be evaluated numerically to an acceptable accuracy, but comes with the burden of calculating the mutual inductance between every turn in a reactor.

Burke and Fawzi in 1978 [4] present a method to simplify the self and mutual inductance calculation of N-turn series windings in air-core reactors as a cylindrical sheet with zero thickness. Similar to the Neumann integral presented in Paul's book, the method presented by Fawzi and Burke doesn't have a closed form solution, but can be evaluated numerically. They also went a step further and presented a method of calculating the inductance of a package, in what they refer to as a "coil of finite thickness". This method greatly reduces the number of integrals to be evaluated numerically from N^2 terms to a more manageable N terms, N being the number of layers in a reactor model. Further, in 1988 with the addition of Dahab, Fawzi and Burke [5] published work wherein they modeled a single layer reactor (single package) using the finite-thickness calculation from [4] with the addition of a capacitance network.

Fawzi and Burke published another article in 1991 [6] describing a method of computing the eddy current losses in reactors. The Method provided is useful when simulating transient conditions at various frequencies, but does leave out the capacitance submatrix for the reactor. The Perturbation matrix is frequency dependent and does take into account the skin effects. The method involves "slicing" the reactor into vertical subsections, where the magnetic field intensity is calculated, in the vertical and radial components, this method assumes axial symmetry to simplify calculations.

In 2021, Zaninelli and Bortoni [7] published a paper comparing the results of the methods from Fawzi and Burke's 1978 paper to that of modern Finite Element Analysis (FEA) methods. In the paper, the authors show that Fawzi and Burke's method performs well at low frequencies, but doesn't properly account for eddy-current losses in the windings, as Fawzi and Burke discuss in their 1991 paper [6]. Zaninelli and Bortoni do provide a correction factor for the reactor inductance value at higher frequencies, but for the purposes of this research, the low frequency accuracy will be sufficient.

Nurminen's dissertation from 2008 [8] detailed how he evaluated the thermal design and mechanical stresses of a purpose-built reactor. Using optical fiber to sense temperature they were effectively able to eliminate electrical noise from the currents induced by the reactor while operating. To sense temperature they leveraged the transmission mediums sensitivity to temperature variation, where as

the optic is deferentially heated, there are regions where minuscule amounts of light are reflected back down the fiber which can be detected by the sending unit. Depending on the intensity of the light being sent back down the fiber and the time of travel, Nurminen was able to determine where the reactor was heated unevenly.

Yuan *et al.* 2017 [9] provide insight into the thermal optimization that goes into the design of an ACR. The authors' optimization process has the goal of minimizing temperature rise and minimizing total conductor mass, which will improve the reactor performance while minimizing cost of construction. The initial parameters of the reactor (turn count, layer count, layer radii, etc) being optimized were calculated elsewhere, with an optimization constraint of preserving the inductance of the model. The primary parameters varied by the optimization process were the widths of the cooling ducts (i.e. varying the layer radii), and the conductor radii. The authors validated the results of their work using FEM software to calculate the final inductance and thermal transfer to the surrounding air.

Fiorentin *et al.* 2016 [10] present a method to model the vibrations of an air-core reactor as a function of the frequency and current for a reactor under load. In the development of their analytical vibroacoustic model, the authors relate to electrical energy input to vibrations, using the Biot-Savart law, to the force that produces the vibration modes. This model doesn't directly influence the development of the model for fault analysis, but rather provides insight as to the mechanical forces degrading the insulation and encapsulation of the reactor windings.

Damron in 2016 [1] discussed the application of air-core shunt reactors installed by a utility operator. The focus of the paper is on the non-standard installation of the dry-type shunt reactors, and a challenge in implementing protection schemes due to a lack of available information. Reasoning for the choice of dry-type air-core reactors over the oil-immersed variety was given as the proximity to a waterway which required increased environmental considerations.

1.5.2 Faults in Air-Core Reactors

Haziah's Dissertation [2], 2012, explores the mechanisms by which air-core reactors fail when in use as filter reactors in mechanically-switched capacitor banks with a damping network. This work provides insight as to how these reactors are used in an application and the (comparatively) limited information needed for an operator to perform transients studies. The work builds a lumped-parameter model for use in transients studies. Haziah's work was focused on the effects of the transients around the energization and de-energization of the filter network, particularly, and how daily repeated switching will cause uneven voltage distribution in the reactor degrading the encapsulation materials from heat generated by the series losses. The work, unfortunately, doesn't continue into the failure modes of the reactor.

At the time of this writing, there are few published published works detailing methods where an Air-Core Reactor is subject to fault conditions. However, there is an interesting method presented by

Faridi *et al.* [11]. They propose a model for Continuously Transposed Cables (CTCs) by segmenting the winding as "transpose lays", or "*FELD*" as referred to in the article, where each lay differs from the last by "rotating" around the center insulation. By rotating the conductors through an assumed uniform magnetic field for a turn of the CTC, the self and mutual inductance of turn is evaluated. Continuously Transposed Cables are used in transformers and other electric machines to improve the efficiency by reducing eddy-current losses, and have possible applications in ACRs. The authors model the inductance of the winding as N impedance elements with M branch currents, an impedance matrix is calculated as $B_{M\times(N+1)}Z_{N\times N}B^T_{M\times(N+1)}$, to calculate the current distribution of a winding turn. In application, the self, and mutual inductance is calculated for the positions of the winding before being applied to the entire set of turns in the winding. Faridi *et al.* test the proposed model under normal operation and a faulted condition. The fault is a short between 2 strands of the cable, which causes a 20% change in current distribution between the faulted conductors, but a minimal variation in the others. This finding is particularly interesting for the purposes of cross-layer faults as they may occur in a package of a reactor.

In the 2015 article by Geissler and Leibfreid [12] evaluated the forces continuously transposed cables (CTC) are subjected to during short-circuit conditions. While the article was primarily interested in the mechanical integrity of cellulose insulation, the authors went into detail regarding the structure and geometry of a CTC.

Mohammad *et al.* [13], 2018, proposed of method of detecting turn-to-tun faults for the purpose of developing relay protection schemes. There were two methods of detection, first method was to compare the performance of the two or more reactor banks connected on the same bus, and monitoring the neutral connection between the banks. When a fault occurs, the inductance value should change in the reactor, which causes current to pass on the neutral connection, the neutral current can also be detected in the zero sequence. This first method relies on two reactor banks of near-identical characteristics. The second method relies on the zero sequence current of an independent reactor. Where the operator would record the zero sequence current and bus voltage in the steady-state as a reference, and check the error between the reactors zero sequence values during operation against an "ideal" approximation based on the reference.

Basha and Thompson presented a paper in 2013 [14] in which they list the expected fault types and the ways in which the faults effect the performance of the reactor. The one fault type of interest they discuss is the *Turn-to-Turn* fault. Where a turn-to-turn fault is likely to occur on the high-side of the reactor winding, sometimes due to transients from switching, similar to the findings of [2]. Also in the paper the authors state that the fault can be seen in as a zero-sequence unbalance.

Chowdhury *et al.* [15] in 2022, look at the practical considerations of the protecting the air-core shunt reactors during faulted conditions. The article looks at 3 types of faults in air-core reactors; a phase (think line-to-line) fault, a ground (analogous to a single-line-to-ground) fault, and a turn (turn-

to-turn) fault. The authors discuss the current transformer selection and protective relay configuration for air-core reactors, and compare to oil-immersed reactors. Interestingly, their modeling method and conclusions regarding turn-to-turn shorts, is that the turn-to-turn fault type is best detected by observing the transient on the neutral line or a rapid change in phase current if the reactor is solidly grounded. The authors provided examples of this method functioning as intended using data from a reactor with a turn-to-turn fault.

Instrumentation of a 35kV reactor was published by Zhigang *et al.* in 2020 [16]. The authors detailed their instrumentation module, and the methods by which air-core reactors are normally checked, primarily by DC resistance measurement and non-contact temperature measurement methods. Their article details the challenges of implementing effective instrumentation for the air-core reactors, which further exemplifies the need to a practical method of determining fault characteristics in air-core reactors.

Guzman, in 2002 [17], presents a method of modeling and simulating energization, steady-state, and fault conditions of conventional iron-core transformers. The transformers are variations of single and three phase units used in transmission and distribution networks. The thesis presents the characteristics of transformer design, considerations and a solution methodology, exemplified in an occurrence matrix, which clearly lays out the equations needed and the solution order used by the program. Guzman's thorough work provides a baseline behavior of the iron core reactor under abnormal conditions, which is useful for comparing findings in ACRs.

1.5.3 Other Materials

These materials were used as a general reference, or as a review of fundamentals to reduce occurrences of "simple" mistakes.

The linear algebra text by Strang [18] provided a background of information regarding the numeric challenges involved in implementing large, dense, matracies and the condition nuber of matracies.

The *Tables of Integrals* by Dwight [19] was used to review the ellipric integral approximations presented by paul [3].

A textbook on the theory of fault modeling by Tleis [20] presents power system modeling and analysis methods. Tleis presents methods of fault modeling using parameters transformed into the sequence domain.

Lammeraner's book on eddy currents [21] is a presentation of eddycurrents in varying applications. Of particular interest is the sections on eddy currents in conductive cylinders and in coil windings.

Kulkarni and Khaparde's book on transformer design [22] provided a background for designing transformers, and analysis methods.

1.5.4 Tools Used in Research

The tools and other materials used (for example, software libraries) in the process of this work include open-source tools built by researchers and enthusiasts and provided free of charge.

- The Python Programming language, an open-source general-purpose interpreted language, version 3.11.
- The NumPy library [23] for the Python programming language, a general purpose scientific data structures and manipulation library.
- The SciPy library [24] for the Python programming language, specifically, the integration library used in the numeric evaluation and analysis.
- MatPlotLib [25], the de facto Python plotting and graphing library.

Chapter 2

Background

2.1 Overview

This chapter provides the theoretical development of the analysis tools used to model Air-Core Reactors, starting with the fundamental equations, and developing the methods used in the modeling and analysis of reactors. The Biot-Savart Law and Ampère's Law are used to evaluate the inductance of the reactor using geometric parameters, and simplifying elements using circuit analysis techniques and then simplifying to a circuits based approach. Then, using the methods presented by Fawzi and Burke [4], to evaluate the inductance of elements of air-core reactors in a computationally efficient method.

2.2 The Biot-Savart Law

The cornerstone of the work done is the use of the Biot-Savart Law [3] to model the flux of the air-core reactor.

$$\mathbf{B} = \frac{\mu_0}{4\pi} \int_v \frac{\vec{J} \times \vec{a}_R}{R^2} dv \tag{2.1}$$

The Biot-Savart law (2.1), is the fundamental law for computing a magnetic field due to a current [3] using a volumetric integral. The current density vector, $\vec{J}$, and $\vec{a}_R$ is a unit vector directed at the point where $\mathbf{B}$ is being computed. The parameter R in the denominator of Eq.2.1 is the distance from the current, $\vec{J}$, to the point where $\mathbf{B}$ is being computed. Similar to Gauss's Law and Ampère's Law, the Biot-Savart Law has an inverse-square relation between distance and intensity. For a circular loop with radius a and wire radius r_w on the xy plane, the flux, ψ, through the surface enclosed by the loop can be written in terms of the z component perpendicular to the enclosed surface:

$$\psi = \int_s \mathbf{B} \cdot ds = \int_{r=0}^{r_a - r_w} \int_{\phi'=0}^{2\pi} B_z \, r \, d\phi' \, dr \tag{2.2}$$

Using the definition of ψ from eqn. 2.2, the z component of the $\mathbf{B}$ field, $\mathbf{B}_z$, at a specific point emanating from a loop with radius r is then B_z can be written using the law of cosines [3]:

$$B_z(r) = \frac{\mu_0 I}{2\pi r} \int_{\phi=0}^{\pi} \frac{r_a^2 cos\phi (a - r\, cos\phi)}{(a^2 + r^2 - 2ar\, cos\phi)^{3/2}} d\phi \tag{2.3}$$

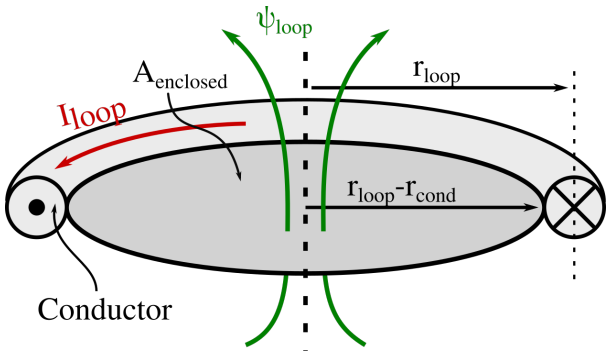


Figure 2.1: A Conductor loop

Equations 2.2 and 2.3 are used to write an expression for the flux passing perpendicular through the area enclosed by a conducting loop, as illustrated in figure 2.1. Substituting the flux perpendicular to surface enclosed by a loop, Eq.2.3, into the definition of ψ from Eq.2.2. The resulting integrals is the evaluation of the of flux passing perpendicular through the enclosed surface:

$$\psi_{loop} = \frac{\mu_0 I}{2\pi} \int_{r=0}^{r_{loop} - r_{cond}} \int_{\phi'=0}^{2\pi} \frac{1}{r} \left[\int_{\phi=0}^{\pi} \frac{r_{loop}^2 cos\phi (r_{loop} - r\, cos\phi)}{(r_{loop}^2 + r^2 + 2\, r_{loop}\, r\, cos\phi)^{3/2}} d\phi \right] r \, dr \, d\phi' \tag{2.4}$$

When ϕ' is integrated out, the 2π from the scaling factor out front of the integral in Eq.2.4 is canceled out:

$$\psi_{loop} = \mu_0 I \int_{\phi=0}^{\pi} \left[\int_{r=0}^{r_{loop} - r_{cond}} \frac{r_{loop}^2\, cos\phi (r_{loop} - r\, cos\phi)}{(r_{loop}^2 + r^2 + 2\, r_{loop}\, r\, cos\phi)^{3/2}} \, dr \right] d\phi \tag{2.5}$$

When the interior integral of Eq.2.5 is evaluated [3], the result is the flux through the area enclosed by a loop, due to a current I_{loop}, as shown in Eq.2.6:

$$\psi_{loop} = \mu_0 \, I \, r_{loop} r_b \int_{\phi=0}^{\pi} \frac{cos\phi}{\sqrt{r_{loop}^2 + r_b^2 - 2\, r_{cond}\, r_b\, cos\phi}} d\phi \tag{2.6}$$

Where $r_b = r_{loop} - r_{cond}$, r_b is the radius of the inner surface of the wire loop. The integral in Eq.2.6 cannot be evaluated to have a closed-form solution, Paul [3] uses a pair of elliptic integrals to approximate a solution. But, Eq.2.6 can be evaluated numerically to a sufficient degree of precision.

2.2.1 Inductance

An issue with using flux is the need to know the current in an element to complete the calculation. A current independent value related to flux, ψ, is inductance L, and the relation is shown in Eq.2.7.

$$\psi = L\,I \Rightarrow \frac{1}{I}\,\psi = L \tag{2.7}$$

When current in one loop gives rise to flux passing through the surface enclosed by another, it is called a *mutual*. The relation between current, flux, and inductance can be described in-terms of a mutual inductance, where current in element a, I_a, gives rise to flux at element b, ψ_b, shown in Eq.2.8:

$$M_{ab} = \frac{\psi_b}{I_a} \tag{2.8}$$

2.3 The Neumann integral

Fundamentally, the Biot-Savart Law (BSL) is used to calculate the inductance (L) values of elements within the reactor. The Neumann Integral, derived from the BSL, is specifically formulated to determine the self inductance of a closed loop carrying a current filament (See [3]), and the mutual to other loops carrying a current filament, see Figure 2.2. The Neumann integral doesn't have a closed-form solution. There is an approximation that uses Elliptic integrals of the first and third kind [19]. However, the integral can be evaluated numerically, and using an algorithm like Gaussian Quadrature [24] the accuracy will be sufficient for our purposes.

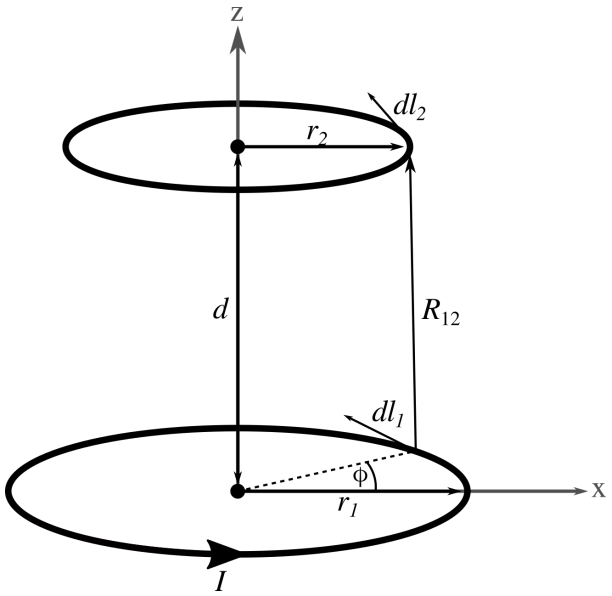


Figure 2.2: Concentric Coaxial Loops

$$L = \mu_0 ab \frac{1}{2} \int_{\phi=0}^{2\pi} \frac{cos\phi}{\sqrt{r_{loop}^2 + r_b^2 + d^2 - 2r_{loop} b cos^2\phi}} d\phi \tag{2.9}$$

Using the Neumann integral (2.9) to evaluate the turn-to-turn inductance will produce a value mutual inductance for *each* turn of the reactor. The result of evaluating the a system of N inductance and current values:

$$V_{terminal} = L_1 \frac{di_1}{dt} + L_2 \frac{di_2}{dt} + ... + L_N \frac{di_N}{dt} + M_{12}\left(\frac{di_1}{dt} + \frac{di_2}{dt}\right) + ... + M_{N\,(N-1)}\left(\frac{di_N}{dt} + \frac{di_{(N-1)}}{dt}\right)$$

The number of elements can be reduced by superimposing the elements that share the same current, i.e. where:$i_1 = i_2 = ... = i_{(N-1)} = i_N$, with N being the number of turns in the layer:

$$V_{terminal} = \frac{di}{dt}\left(L_1 + L_2 + ... + L_N + M_{12} + ... + M_{N\,(N-1)}\right)$$

In this way, the size of the L (and R and Z) matrices can be simplified to a number of elements representing the number of layers in a device. The primary issue with this turn-to-turn method to calculate the inductance is the evaluation time, Using the Gaussian quadrature algorithm for numeric integration evaluating the integrals, to evaluate all the turns in a reactor the runtime is $\mathcal{O}(N^2)$.

2.3.1 Mutual Inductance

Mutual inductance is the a product of the flux in a loop, ψ_m, that is induced by the current in another loop, I_n. So we can use the linear relation:

$$M_{nm} = \frac{\psi_m}{I_n}$$

The Neumann integral (2.10) is the method used to compute the mutual inductance between two loops. Here the current, I, isn't needed since inductance, L, can be represented as: $L = \psi I^{-1}$.

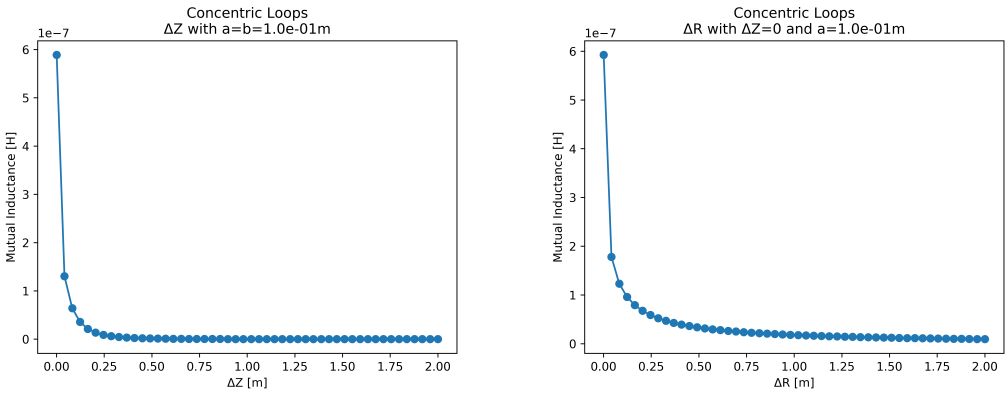
$$M_{nm} = \mu_0\, r_n\, r_b \int_{\phi=0}^{\pi} \frac{cos\phi}{\sqrt{(r_n^2 + r_b^2 - 2\, r_n\, r_b\, cos\phi + d^2)}} d\phi \tag{2.10}$$

Where r_n is the wire loop radius, $r_b = r_m - r_{cond\,m}$ is the radius of the surface being linked by flux enclosed by the second loop, and d is the distance between the parallel planes the loops are on.

Mutual inductances are symmetrical, $M_{nm} = M_{mn}$, so the mutual only needs to be calculated once and any mutual is simply doubled; $M_{nm} + M_{mn} = 2M_{nm}$. Inductances can be super-imposed (summed) with other self and mutual inductances to formulate a net-inductance for a single homogeneous component. Because we're wrapping the loops of our inductor concentrically about the others, we can say that the mutuals are *always* going to be positive (additive-influence), as all loops have the same polarity.

2.4 Integral Evaluation Method

Because the integral for computing mutual inductance, Eq.2.10, doesn't have a closed-form solution, the integral is evaluated numerically. The Python *SciPy* library has a quadrature (quad) integration method, the quad integration method evaluates a given function, in this case the Neumann integral, to a specified error [24]. The error is set by either the available precision of the floating-point representation, or a defined minimum error, which is 10^{-13} by default. In figure 2.3, it is clearly visible that the Biot-Savart flux has an inverse-relation to the loop separation distance. Also visible in the figure is the effect that increasing the radial difference, i.e. the second loop radius r_2 as shown in figure 2.2, the mutual coupling decreases slower than when the loops are separated concentrically, i.e. increasing d.



(a) Coaxial vertical separation

(b) Coaxial radius increase

Figure 2.3: Mutual Inductance vs Distance, with parameters $a = r_1$,$b = r_2 - r_{cond\,2}$,and d described in figure 2.2

2.5 Internal Flux

In the ACR, Flux that doesn't link to any other loop is entirely contained within the conductor and insulation. This flux internal to the wire is a significant component of the reactor.

$$L_{internal} = \mu_0 \left(\frac{r_{loop}}{4} + \frac{r_{cond}}{5} \right) \tag{2.11}$$

Equation 2.11 was developed using (2.2) with the $\mathbf{B}$ field being provided by Ampère's Law rather than the Biot-Savart Law. The derivation is given in appendix A.

2.6 Modeling as a Matrix

The Reactor can be modeled as an $N \times N$ matrix, with elements being the superposition of inductance values that share the same current:

$$\bar{L} = \begin{bmatrix} L_1 & M_{12} & \dots & M_{1n} \\ M_{21} & L_2 & \dots & M_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n1} & M_{n2} & \dots & L_n \end{bmatrix}$$

With the L_n values representing the sum of self and mutual inductance that are due to the currents I_n. The off-diagonal elements, $M_{n\,m}$, are the mutual inductance values that give rise to a current in the layer i_n due to the current in another element i_m. The convention used will denote diagonal elements of the matrices using a single indexing subscript, i.e. $L_n = L_{n\,n}$.

To model the reactor, the inductance is converted to reactance using the angular frequency, $\omega\ [rad/s] = 2\pi f\ [rad \cdot Hz]$, of the steady-state operation:

$$jX\ [\Omega] = j\omega\,\bar{L}\ [\Omega] = j\,2\pi\,f\,\bar{L}\ [\Omega]$$

The reactor is evaluated by solving the linear system for $\vec{I}$:

$$\vec{V} = \bar{Z}\,\vec{I} \Longrightarrow \vec{I} = \bar{Z}^{-1}\,\vec{V}$$

To determine the total impedence of the reactor, the current vector is summed to get the total:

$$I_{total} = \sum_{\forall n} \vec{I}_n$$

The total impedence of the reactor is calcualted:

$$Z_{total} = \frac{V_{term}}{I_{total}}$$

Using these methods, the reactor is evaluated for prefault and postfault behavior. The values given by I_{total} and Z_{total} are the values as would be measured at the terminals of the reactor. For the simulations performed in the work for this thesis, ther terminal voltage V_{term} is the 0° phasor reference.

2.7 Turn-to-Turn Method

To model an air-core reactor using the turn-to-turn method is a straightforward implementation of the Neumann integral (Eq.2.10), with the internal flux given by Eq.2.11. Knowing the radius from the center of the loop to the center of the conductor, r_{loop}, the conductor radius, r_{cond}, and the vertical

separation, d, between two loops, including $d = 0$ for the self value, Eq.2.10 can be evaluated numerically. These evaluations are repeated until every self and mutual inductance has been calculated. Using superposition, the loops that are in the same layer have their inductance values summed, to consolidate the number of unknowns when solving for the current in each layer.

2.8 Thin Sheets Method

Fawzi and Burke in 1978 [4], proposed a more computationally efficient method for calculating the inductance of a series of turns, and the mutual between two series turns. The elaboration of the Neumann Integral resulted in the expression used to compute the induction of concentric coils:

$$M = 2\pi\mu_0(R_1R_2)^{3/2}n_1n_2[Ci(R_1, R_2, z_1) - Ci(R_1, R_2, z_2) + Ci(R_1, R_2, z_3) - Ci(R_1, R_2, z_4)] \quad (2.12)$$

Where:

$$n_1 = \frac{N_1}{h_1}\ \frac{[turns]}{[m]} \quad (2.13)$$

$$n_2 = \frac{N_2}{h_2}\ \frac{[turns]}{[m]} \quad (2.14)$$

$$z_1 = l_1 + l_2 + s\ [m] \quad (2.15)$$

$$z_2 = l_1 - l_2 + s\ [m] \quad (2.16)$$

$$z_3 = -l_1 - l_2 + s\ [m] \quad (2.17)$$

$$z_4 = -l_1 + l_2 + s\ [m] \quad (2.18)$$

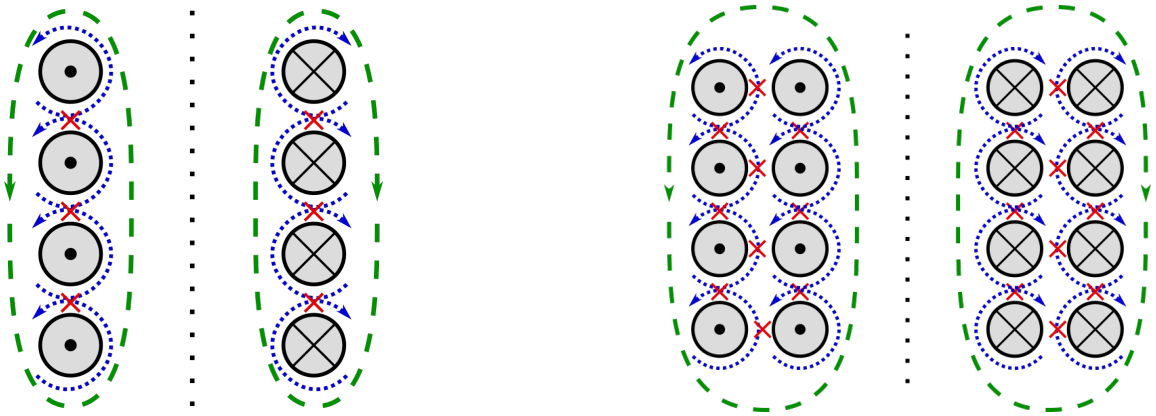
With: $l_1 = \frac{h_1}{2}\ [m]\ l_2 = \frac{h_2}{2}\ [m]$, and

$$Ci(R_1, R_2, z) = \frac{\sqrt{R_1R_2}}{2\pi}\int_{\psi=0}^{\pi}\frac{\sqrt{R_1^2 + R_2^2 + z^2 - 2R_1R_2cos\psi}}{R_1^2 + R_2^2 - 2R_1R_2cos\psi}sin^2\psi\, d\psi \quad (2.19)$$

The Ci Eq.(2.19), like the Neumann integral doesn't have a closed form solution and needs to be evaluated numerically, Gaussian Quadrature algorithm from the SciPy Python library [24] as used for the low time, and low error advantages of the technique.

2.9 Cylindrical Shells Method

Fawzi and Burke elaborated further on their thin-sheets method so the reactor packages could be represented as cylindrical shells of finite thickness. This is done by integrating the thin sheets mutual between the thickness of both packages or layers:



(a) Flux around a single reactor layer

(b) Flux around a 2-layer reactor package

Figure 2.4: Flux interaction between turns resulting in flux around an entire layer or package

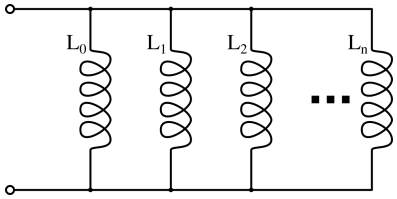


Figure 2.5: Schematic representation of an Air-Core Reactor, with elements represented as parallel inductors

$$M_{shell} = \int_{r_1=R_1-t_1/2}^{R_1+t_1/2} \int_{r_2=R_2-t_2/2}^{R_2+t_2/2} n_1 n_2 M(r_1, r_2) dr_2 dr_1 \tag{2.20}$$

$$= 2\pi\mu_0 n_1^2 n_2^2 \int_{r_1=R_1-t_1/2}^{R_1+t_1/2} \int_{r_2=R_2-t_2/2}^{R_2+t_2/2} (r_1 r_2)^{3/4} Ci(r_1, r_2, z_1, ..., z_4) dr_2 dr_1 \tag{2.21}$$

With $Ci(r_1, r_2, z_1, ..., z_4) = [Ci(r_1, r_2, z_1) - Ci(r_1, r_2, z_2) + Ci(r_1, r_2, z_3) - Ci(r_1, r_2, z_4)]$ in Eq.2.21. This cylindrical shell inductance model appears to account for the flux that links subsets of the turns in each layer, and how the overlapping fields would effectively cancel-out the that of it's neighbors immediately to the inside and outside of the loop, leaving only the links that pass round the loops as a whole, see figure 2.4b.

2.10 Reactor Construction

In most literature, the reactor is described as each layer connected in parallel, as shown in figure 2.5. The parallel arrangement of the reactor layers means each layer of a package effectively links the same flux. Because the Neumann integral is describing the effect of the flux produced by a current in one circular loop passing through an area enclosed by another coaxial loop, the effect of increasing

the radius will be an increased self inductance. Ass more layers are added to the reactor, the increase in radius will necessitate a decreased number of turns to ensure a similar inductance to that of the layers with a smaller radius.

Ideally each layer of the reactor would have a similar self inductance and a mutual coupling to the other layers, and the current in each layer would be approximately equal in magnitude and phase angle. Another way to say it, would be to eliminate currents circulating in the reactor, which generate heat without any performance gain. Through careful consideration, it was deemed impractical to design a reactor in which the currents are of equal magnitude, and of similar phase, in reference to terminal voltage. Instead, to eliminate circulating currents within the reactor, the current magnitude is allowed to vary, but the phase angle of the currents in each layer should be as close to in-phase as possible.

2.11 Notes on Computation

The modeling of a reactor consists of 2 main parts that can be considered independently, these are the turns of elements of the reactor, and the geometric components of the reactor. Looking at the matrix $\bar{L}$ with elements computed using Fawzi and Burke's method, we can see the matrix can be rewritten as a scaled Hadamard product of 2 matrices:

$$\bar{L} = 2\pi\mu_0 \cdot \bar{N} \circ \bar{G} \tag{2.22}$$

Which will be $\bar{N}$ with elements representing the product of the turns of two layers, and a matrix $\bar{G}$ representing the values dependent of the geometry of the reactor, and computed using the Ci function (eqn 2.19) from [4].

$$\bar{N} = \begin{bmatrix} N_1^2 & N_1N_2 & \dots & N_1N_n \\ N_2N_1 & N_2^2 & \dots & N_2N_n \\ \vdots & \vdots & \ddots & \vdots \\ N_nN_1 & N_nN_2 & \dots & N_n^2 \end{bmatrix} \tag{2.23}$$

$$\bar{G} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{n,1} & g_{n,2} & \cdots & g_{n,n} \end{bmatrix} \tag{2.24}$$

$$= \begin{bmatrix} C(R_1,R_1,h_1,h_1,s) & C(R_1,R_2,h_1,h_2,s) & \dots & C(R_1,R_n,h_1,h_n,s) \\ C(R_2,R_1,h_2,h_1,s) & C(R_2,R_2,h_2,h_2,s) & \dots & C(R_2,R_n,h_2,h_n,s) \\ \vdots & \vdots & \ddots & \vdots \\ C(R_n,R_1,h_n,h_1,s) & C(R_n,R_2,h_n,h_2,s) & \dots & C(R_n,R_n,h_n,h_n,s) \end{bmatrix} \tag{2.25}$$

Where:

$$C(R_a, R_b, h_a, h_b, s) \Rightarrow \frac{1}{h_a h_b}\left[(Ci(R_a, R_b, z_1) - Ci(R_a, R_b, z_2)) + (Ci(R_a, R_b, z_3) - Ci(R_a, R_b, z_4))\right] \tag{2.26}$$

with z_1,z_2,z_3, and z_4 are defined as in equations 2.15, 2.16, 2.17, and 2.18.

When evaluating the condition of the turns and geometry matrices separately, the evaluation of the geometric portion requires the evaluation of 2 or more integrals, shown in 2.26 in the more general form where 4 integrals are evaluated to evaluate the geometric component of the mutual coupling. To save computation time, a geometric configuration can be evaluated to generate the $\bar{G}$ matrix, then iterate the $\bar{N}$ matrix to produce the $\bar{L}$ and $\bar{Z}$ matrices. The parameters driving the $\bar{G}$ matrix would be driven by thermal or other mechanical constraints.

Chapter 3

Reactor Modeling

3.1 Introduction

The following chapter is a presentation of the methods and assumptions used to generate a prefault model reactor.

3.2 Modeling Software

The software for modeling the reactor and the effects of faults was written in Python, using the NumPy library for matrix manipulation functionality [23], and the SciPy Integrate library was used for an efficient implementation of the quadrature integration algorithm [24]. Development of analysis tools started with the turn-to-turn method of calculating mutual inductance using the Neumann integral (2.9), with each flux linkage between turns evaluated numerically using a series of integration methods and ultimately the quadrature algorithm. Using the turn-to-turn method was slow, requiring minutes to hours to cacualte the inductance of a reactor configuration. Then, using the method by Fawzi and Burke in their 1978 paper [4], the computation time was greatly reduced

3.3 Turn-to-Turn Method

The turn-to-turn method is a direct implementation using the Neumann intergral (2.10), and the internal inductance of a cylindrical conductor. Each turn of the reactor is evaluated to determine the self inductance, due to the flux enclosed by the loop and due to the flux internal to the conductor, and the coupling to each of the other turns in the reactor. With the turn-to-turn method, the inductance calculation can take some time, originally the program was extended to use hyper threading, which has a different set of challenges, especially memory management. In later versions of the project,

with the thin-sheets method implemented, the need for the added complexity of hyper threading was unnecessary.

3.4 Thin-Sheets Method

The implementation of the methods presented by Fawzi and Burke [4] is likewise straightforward. Taking the layer or package radii, turns, and height, the self and mutual inductance values can be evaluated numerically. In the modeling of the reactor, there is a caveat in the accuracy of the mutual, this comes down to where the two radii are being evaluated. Using the turn-to-turn method, the mutual was evaluated to the inner contour of the conductor loop, with the thin sheets method, the mutual is being evaluated to, effectivl, the same loop radius. The difference is obvious when turn count and radius are relatively small, to reduce the error further the second radius of the sheet self inductance should be the loop radius minus the radius of the conductor: $r_b = r_{loop} - r_{cond}$.

3.5 Reactor Modeling Parameters

Turns, layer/package radius, difference in radius of coupled packages, package height, wire diameter, number of layers in a package (package thickness). All these things have significant impact on the inductance and mutual coupling. The values listed in table 3.1 are gathered from a variety of sources for the purpose of accurately modeling the component behavior of the air-core reactors. The known values from the manufacturer reports are used as a baseline in the design process.

For the model reactor used in fault analysis, because there is missing information regarding the mean radius or diameter of each package of the reactor, initially the outer diameter and the inner and outer turn counts are used to determine the innermost package diameter, and the 8 packages between inner and outer are assumed to be evenly distributed between the inner and outer layers.

parameter	value	source
Reactor Height	$3.1m$	Drawings
Reactor Outer Diameter	$2.301m$	Drawings
Average Turns	985.8	Communications
Minimum Turns	841	Communications
Maximum Turns	1285	Communications
Measured DC Resistance	0.9394Ω	Test Reports
Measured AC Resistance	1.1168Ω	Test Reports
Measured Inductance	$694.44mH$	Test Reports
DC Power Dissipation	$13821.8W$	Test Reports
Testing Current	$121.3A$	Test Reports
Testing Frequency	$60Hz$	Test Reports

Table 3.1: Reactor parameters from manufacturer testing reports, mechanical drawings, and e-mail communications, these parameters are used as the baseline for developing a reasonable approximation of a commercial reactor.

3.6 Conditioning

3.6.1 Condition Analysis

Because the fundamental equations of the reactor are not of the closed-form variety, the additional steps need to be take to ensure a minimum of error introduced into the reactor model.

The Condition number (3.1) of the $\bar{Z}$ matrix is a representation of the matrix sensitivity to small perturbations, and an indicator of the computational error in the $\bar{Y}$ matrix during the inversion process [18]. The relative error for a solution to a linear system $\vec{x} = \bar{A}^{-1} \cdot \vec{b}$ is bounded by (3.2)

$$c = \frac{\lambda_n}{\lambda_1} = \frac{\lambda_{max}}{\lambda_{min}} \tag{3.1}$$

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\lambda_{max}}{\lambda_{min}} \frac{\|\delta b\|}{\|b\|} \tag{3.2}$$

The condition number of the $\bar{Z}$ matrix is increases proportionally with the dimension of the matrix

Using the component matrices $\bar{N}$ (2.23) and $\bar{G}$ (2.24), the condition number of the turns component matrix, $\bar{N}$, is very large, meaning that $\bar{N}$ is very poorly conditioned. However, The condition of the geometry component matrix is on the same order as the square of the matrix dimension, and when the component matrices are element-wise multiplied (Hadamard product), the condition of $\bar{N} \circ \bar{G}$ has

a condition number less than the condition of $\bar{G}$, more formally:

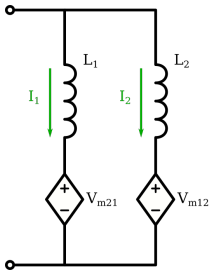
$$\text{cond}(\bar{N} \circ \bar{G}) < \text{cond}(\bar{G}) << \text{cond}(\bar{N})$$

Therefore, when designing a reactor, or developing a model for evaluation, time should be spent evaluating the geometry of the reactor to minimize error. Keeping in-mind that the mutual coupling of the elements is the product of turns-squared and the relative positioning of the sheets being evaluated, or hollow cylinders if the finite-thickness representation is being used.

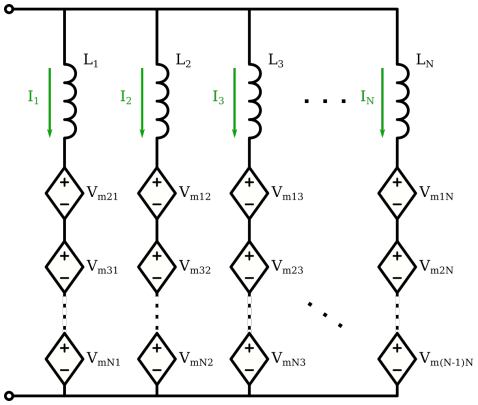
In the process of modeling the reactor, the condition number of the reactor matrix provides an insight into the performance of the model. By minimizing the condition number of the reactor, the accuracy of the model increases by reducing the error bounded by the condition. Minimizing the condition number of the impedance matrix can be achieved by reducing the number of elements in the matrix, or by reducing the mutual coupling between reactor elements.

3.7 Prefault Behavior

The design of a reactor is challenging, and critical, part to properly understanding the fault behavior of and ACR. As shown in table 3.1, the parameters of the reactor are In the design of the reactor, the turns of each layer is optimized to prevent circulating currents, this constraint can be interpreted as a dependent voltage source, $V_{m\,ab} = I_b \cdot jX_{m\,ab}$, where $jX_{m\,ab}$ is the mutual reactence between layers a and b.



(a) 2 Layer Model



(b) N Layer Model

Figure 3.1: Prefault Reactor Models, with Mutuals shown as Dependent Voltages

The design process for the reactors is intended to produce self and mutual terms that will produce a balanced set of currents. Where the currents are all approximatly in-phase, and no one laye rhas significantly larger currents than the others.

3.8 Model Validation

3.8.1 Example: 2 Element Reactor

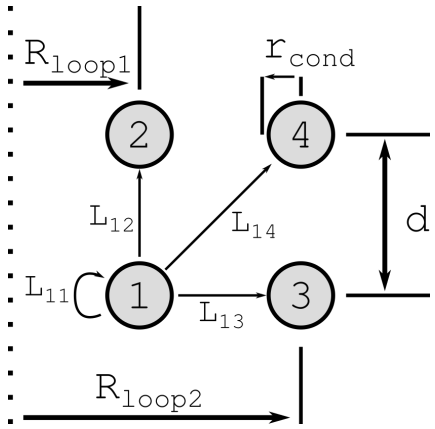


Figure 3.2: 2 Layer, 4 Loop Reactor Turns Diagram, with relevant parameters listed

Using the diagram in figure 3.2, we'll go through the process of calculating the self and mutual inductance for a reactor, and the consolidation

1. First, the self inductance is calculated, L_{11} in the diagram using the Neumann Integral, equation 2.10, for mutual inductance to itself with $a = R_{loop1}$, $b = R_{loop1} - r_{cond}$, and $d = 0$. Using equation 2.11, the loop radius and conductor are plugged-in, and added to the self mutual to get the total self inductance. For the turns numbered 2 and 4, the self inductance is the same, i.e. $L_{11} = L_{22}$, and $L_{33} = L_{44}$, so the calculation only needs to be perform once per pair. This process is repeated for for the second layer, with R_{loop2} substituted in, these are the diagonal elements of the inductance matrix.

2. Next, the mutual inductance is calculated, for the mutual inductance values along the first :

$$L_{12} = M_{Neumann}(a = R_{loop1}, b = R_{loop1} - r_{cond}, d \neq 0)$$

$$L_{13} = M_{Neumann}(a = R_{loop1}, b = R_{loop2} - r_{cond}, d = 0)$$

$$L_{14} = M_{Neumann}(a = R_{loop1}, b = R_{loop2} - r_{cond}, d \neq 0)$$

   Since, the mutual values between turns are the same, i.e. $L_{12} = L_{21}$, the values can be placed in the corresponding row,column pairs of the inductance matrix. These steps are repeated to calculate L_{23}, L_{24}, and L_{34}, so for a 4×4 matrix, 6 mutual values are computed. The resulting

inductance matrix is then:

$$L = \begin{bmatrix} l_{11} & l_{12} & l_{13} & l_{14} \\ l_{12} & l_{22} & l_{23} & l_{24} \\ l_{13} & l_{23} & l_{33} & l_{34} \\ l_{14} & l_{24} & l_{34} & l_{44} \end{bmatrix} \tag{3.3}$$

3. Then to simplify the 4×4 matrix, assume $i_{t1} = i_{t2}$, and $i_{t3} = i_{t4}$

$$\Psi_{11} = l_{11}i_1 + l_{22}i_2 + l_{12}(i_1 + i_2) \Rightarrow L_{11} = l_{11} + l_{22} + 2 * l_{12}$$

$$\Psi_{33} = l_{33}i_3 + l_{44}i_4 + l_{34}(i_3 + i_4) \Rightarrow L_{22} = l_{33} + l_{44} + 2 * l_{34}$$

$$\Psi_{13} = l_{13}i_1 + l_{14}i_1 + l_{23}i_2 + l_{24}i_2 \Rightarrow L_{12} = L_{21} = l_{13} + l_{14} + l_{23} + l_{24}$$

4. Simplify the $\bar{L}$ matrix using the consolidated elements:

$$\bar{L} = \begin{bmatrix} L_{11} & L_{12} \\ L_{12} & L_{22} \end{bmatrix}$$

To perform the same calculation using Fawzi and Burke's thin-sheets method, equation 2.12:

1. The turns are computed as a density: $n_1 = n_2 = \frac{2}{(2 * r_{cond} + d)} \frac{[turns]}{[m]}$,

2. The Layer heights are $n * r_{cond} +$ (turn pitch) $= 2 * r_{cond} + d\ [m]$

3. Evaluate 3 mutuals:

$$L_{11} = M_{FB}(n1 = n2, R1 = R2, h1 = h2)$$

$$L_{22} = M_{FB}(n1 = n2, R1 = R2, h1 = h2)$$

$$L_{12} = L_{21} = M_{FB}(n1 \neq n2, R1 \neq R2, h1 \neq h2)$$

4. Form the inductance matrix $\bar{L}$:

$$\bar{L} = \begin{bmatrix} L_{11} & L_{12} \\ L_{12} & L_{22} \end{bmatrix}$$

3.8.2 Comparing Methods Numerically

Using the example reactor shown in figure 3.2, applying the following parameters:

$$r_{cond} = 0.25\ [mm]$$

$$c_{ins} = 0.01\ [mm]$$

$$d = 2 * (r_{cond} + c_{ins})$$

$$R_{loop1} = 0.1[m]$$

$$R_{loop2} = 0.103[m]$$

The addition of c_ins is the radial insulation thickness on the conductor, is not strictly needed for this example, but practically needed if this were to be physically realized. When the parameters above are applied to to the elements of the matrix described in equation 3.3, the result is 3.4. A 4×4 inductance matrix matrix using the turn-to-turn method.

$$L_{t2t\ 4\times4} = \begin{bmatrix} 7.618E-07 & 6.568E-07 & 4.697E-07 & 4.675E-07 \\ 6.568E-07 & 7.618E-07 & 4.675E-07 & 4.697E-07 \\ 4.697E-07 & 4.675E-07 & 7.885E-07 & 6.804E-07 \\ 4.675E-07 & 4.697E-07 & 6.804E-07 & 7.885E-07 \end{bmatrix} \tag{3.4}$$

Looking at the coupling coefficients of the 4×4 reactor, 3.5, the coupling between turns of the same radius are "strong", with $K_{12} = 86.2\%$ and $K_{34} = 86.3\%$ of the self elements.

$$K_{t2t\ 4\times4} = \begin{bmatrix} 1.000 & 0.862 & 0.606 & 0.603 \\ 0.862 & 1.000 & 0.603 & 0.606 \\ 0.606 & 0.603 & 1.000 & 0.863 \\ 0.603 & 0.606 & 0.863 & 1.000 \end{bmatrix} \tag{3.5}$$

When simplified from a 4×4 to a 2×2 the inductance becomes:

$$L_{t2t\ 2\times2} = \begin{bmatrix} 2.837E-06 & 1.874E-06 \\ 1.874E-06 & 2.938E-06 \end{bmatrix} \tag{3.6}$$

$$K_{t2t\ 2\times2} = \begin{bmatrix} 1.000 & 0.649 \\ 0.649 & 1.000 \end{bmatrix} \tag{3.7}$$

Now, looking at the equivalent 2×2 matrix produced by the *thin sheets* method:

$$L_{sheet} = \begin{bmatrix} 2.796E-06 & 1.873E-06 \\ 1.873E-06 & 2.895E-06 \end{bmatrix} \tag{3.8}$$

With the corresponding coupling coefficients matrix:

$$K_{sheet} = \begin{bmatrix} 1.000 & 0.658 \\ 0.658 & 1.000 \end{bmatrix} \tag{3.9}$$

Looking at the difference between the result given by the turn-to-turn method and the method given by Fawzi and Burke [4]:

$$L_{diff} = L_{t2t\ 2\times2} - L_{sheet} = \begin{bmatrix} 4.169E-08 & 1.211E-09 \\ 1.211E-09 & 4.294E-08 \end{bmatrix} \tag{3.10}$$

Restating the difference in 3.10 as a percentage:

$$\frac{L_{t2t\ 2\times2} - L_{sheet}}{L_{t2t\ 2\times2}} \times 100\% = \begin{bmatrix} 1.469 & 0.065 \\ 0.065 & 1.462 \end{bmatrix} \% \tag{3.11}$$

The difference in methods numeric results given in 3.10, and the relative difference given as a percentage 3.11.

3.8.3 Physical tests of Reactors

To validate the modeling methodology and design considerations, small scale test reactors were constructed. These test reactors were wound around 3d printed forms to ensure a specified layer radius, and testing was performed while the layers were installed in a jig to ensure concentricity, see figure 3.4.

Reactor Layer	Turns	Radius	Height	Position	Note
Layer 1p	41	$50mm$	$23mm$	innermost layer	prefault layer 1
Layer 1f	41	$50mm$	$23mm$	innermost layer	turn 21 shorted
Layer 2a	41	$54mm$	$23mm$	middle layer	”untuned” case
Layer 3a	41	$58mm$	$23mm$	outermost layer	”untuned” case
Layer 2b	37	$54mm$	$23mm$	middle layer	”tuned” turn count
Layer 3b	37	$58mm$	$23mm$	outermost layer	”tuned” turn count

Table 3.2: Parameters for the test reactor, all wound using 24AWG enameled solid copper

3.8.4 41 Turn Test

Using a 41 turn Reactor of 50mm diameter, 24awg enameled magnet wire, and a 3d printed form, the accuracy of the modeling method using the turn-to-turn method from section 2.3 and the sheets method from section 2.8 were tested.

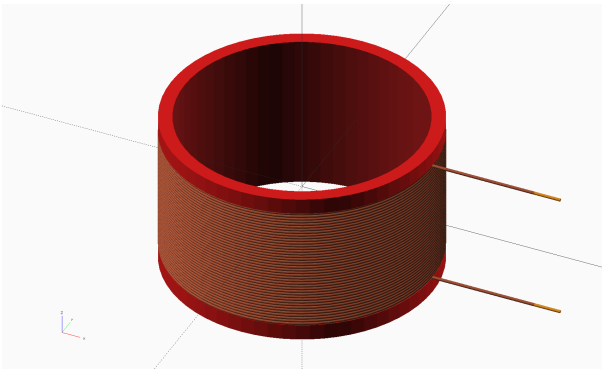


Figure 3.3: Rendering of a 41 turn simple test reactor

The simple test reactor shown in figure 3.3, which has the parameters of "layer 1p" in table 3.2, was calculated to have the following inductance values:

$$L_{bs} = 87.9365[nH]$$

$$L_{fb} = 88.6318[nH]$$

$$L_{fbt} = 74.5048[nH]$$

When the Biot-Savart loop-to-loop method is compared to Fawzi and Burke's more computational efficient methods, i.e. the thin sheets L_{fb}, and cylindrical shells L_{fbt} methods,

$$\frac{L_{bs}}{L_{fb}} = 0.9921561086787597 \Rightarrow \frac{L_{bs} - L_{fb}}{L_{bs}} = 0.791\% \text{ Difference}$$

$$\frac{L_{bs}}{L_{fbt}} = 1.180280437602568 \Rightarrow \frac{L_{bs} - L_{fbt}}{L_{bs}} = 15.27\% \text{ Difference}$$

Here we can see that for a small wire dimer the finite thickness, cylindrical shells method, is not as accurate for small reactors. The script to generate this test can be found in appendix D.5.

Testing results using the parameters: current sensing resistor: $R_i = 10\Omega$, and testing frequency: $f_{testing} = 10000[Hz]$:

$$Z_{bs} = 10.5284 + 5.5252j[\Omega]$$

$$Z_{fb} = 10.5284 + 5.5689j[\Omega]$$

$$Z_{fbt} = 10.5284 + 4.6813j[\Omega]$$

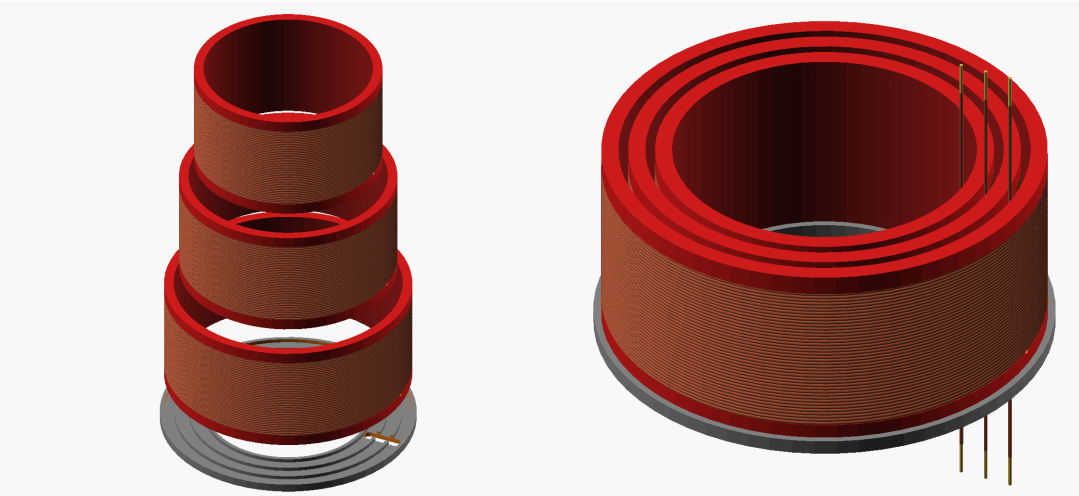
$$I_{bs} = 8.915 \times 10^{-02} \angle -27.690^{\circ}$$

$$I_{fb} = 8.900 \times 10^{-02} \angle -27.876^{\circ}$$

$$I_{fbt} = 9.200 \times 10^{-02} \angle -23.972^{\circ}$$

The results from testing this single layer reactor with a $1\ V$, $10\ kHz$ sinusoidal excitation yielded a $7.8 \angle 27.3^{\circ}\ [mA]$ current, measured across a $10\ \Omega \pm 1\%$ resistor. An the oscilloscope screen capture of this test can be found in appendix B, figure B.14.

3.8.5 Multilayer Models



(a) Expanded (b) Nested

Figure 3.4: Rendering of the 3-layer testing reactor, with parameters listed in table 3.2

Expanding the testing to include multiple layers, the reactor would appear as in figure 3.4. The 3-layer model will use layer 1p, 2a, and 3a parameters from table 3.2. In the following multilayer model tests, the layer current is measured as the voltage across a $10\Omega \pm 1\%$ resistor in series with the layer winding. Using the same turn count for each layer of the reactor would result in an unbalanced reactor model, however, with the resistors used to measure the current of the layers the effective layer resistance is increased. With the artificially larger per-layer resistance from the instrumentation resistors, the reactors won't exhibit a circulating current behavior, where the angle of the current phasor is less than $-90°$.

$$Z_{ut} = \begin{bmatrix} 10.53 + j5.733 & j5.247 & j4.849 \\ j5.247 & 10.57 + j6.441 & j5.901 \\ j4.849 & j5.901 & 10.61 + j7.171 \end{bmatrix} \tag{3.12}$$

In equation 3.12, the untuned impedance matrix,

$$I_{ut} = \begin{bmatrix} 2.097E-02 - j2.630E-02 \\ 1.602E-02 - j2.853E-02 \\ 1.498E-02 - j2.861E-02 \end{bmatrix} = \begin{bmatrix} 3.364E-02\angle -51.432° \\ 3.272E-02\angle -60.684° \\ 3.229E-02\angle -62.370° \end{bmatrix} \tag{3.13}$$

Equation 3.14 is the total total current of the physical test reactor:

$$I_{total\ ut} = 5.1971E-02 - j8.3444E-02 = 9.8305E-02\angle -58.085° \tag{3.14}$$

The total impedance of the untuned reactor, equation 3.15:

$$Z_{total\ ut} = 3.5494 + j5.6989 = 6.7138\angle 58.085° \tag{3.15}$$

Looking at a tuned variant of the 3-layer model, using layer 1p, 2b, and 3b parameters from table 3.2.

$$Z_t = \begin{bmatrix} 1.053E+01+j5.733E+00 & 0+j4.833E+00 & 0+j4.456E+00 \\ 0+j4.833E+00 & 1.051E+01+j5.522E+00 & 0+j5.022E+00 \\ 0+j4.456E+00 & 0+j5.022E+00 & 1.055E+01+j6.139E+00 \end{bmatrix} \tag{3.16}$$

Equation 3.17 is the current vector of the

$$I_t = \begin{bmatrix} 1.976E-02-j2.703E-02 \\ 1.879E-02-j2.757E-02 \\ 1.806E-02-j2.779E-02 \end{bmatrix} = \begin{bmatrix} 3.348E-02\angle -53.833^\circ \\ 3.337E-02\angle -55.722^\circ \\ 3.314E-02\angle -56.986^\circ \end{bmatrix} \tag{3.17}$$

Equation 3.18 is the total current of the tuned model.

$$I_{total\ t} = 5.6603E-02-j8.2384E-02 = 9.9955E-02\angle -55.508^\circ \tag{3.18}$$

Equation 3.19 is the total impedance of the tuned reactor model.

$$Z_{total\ t} = 3.5125E+00+j5.1124E+00 = 6.2028E+00\angle 55.508^\circ \tag{3.19}$$

3.8.6 Impact of Tuning

The untuned reactor current phasors each have the argument:

$$arg(I_{l1p}) = -51.432^\circ$$

$$arg(I_{l2a}) = -60.684^\circ$$

$$arg(I_{l3a}) = -62.370^\circ$$

Where the difference in argument are relatively large in the untuned case. Without the 10Ω measurement resistor the untuned arguments would be nearer to -90°

$$arg(I_{l1p}) = -53.833^\circ$$

$$arg(I_{l2b}) = -55.722^\circ$$

$$arg(I_{l3b}) = -56.986^\circ$$

The result of tuning the reactor can be seen in the argument of the current phasors, where the layer currents are closer to going in the same direction. The magnitudes of the currents also move into a reasonable range, where the magnitudes of adjacent layers become closer to an average of the magnitudes, as seen when comparing equation 3.13 to 3.17.

Chapter 4

Fault Modeling

4.1 Introduction

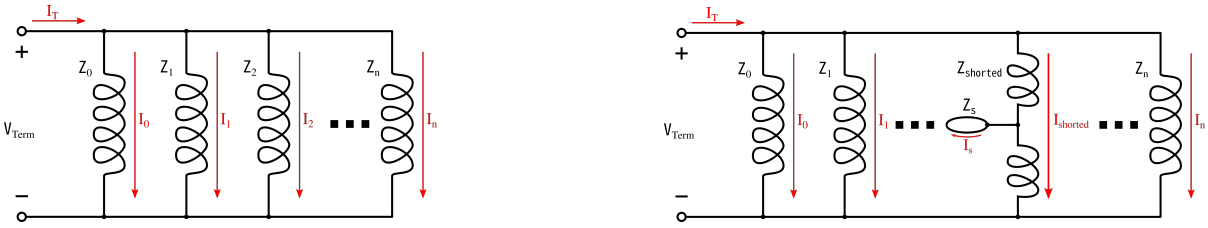
The purpose of this chapter is to present the method developed and used to evaluate faults, specifically turn to turn, in air core reactors. presented here is the application of the theory, and the parametrization of the faults that they can be evaluated using an extension of the theory presented in previous chapters.

4.2 Fault Background

To evaluate the fault using *Fawzi and Burke's* cylindrical shell methods the physical space of shorted loop needs to be evaluated. When using the cylindrical shell with zero thickness, i.e. the thin-sheet method, turn mass is evaluated separately from the turns density, as in equation 2.12, $n_1 = \frac{N_1}{h_1}$, $n_2 = \frac{N_2}{h_2}$.

The density of the turns, $\frac{N}{h}$, will be the same as in the layer the fault occurs in, this maintains the integrity of cylindrical sheets approximation of equation (2.12). The height of the fault is evaluated as part of the height elements, z_1,z_2,z_3,and z_4, where the height is evaluated as $\frac{1}{N} \cdot h$. That is to say, the height of a turn-to-turn fault is calculated as the fraction of the height of the layer or package in which the now closed loop occupies.

To evaluate fault current the impedance matrix is formed and inverted as usual, but to calculate currents we assume a potential of zero volts ($0.0\ [V]$) in the column vector corresponding to the indices were the faults self occurs: $\vec{V} = [V_1, V_2, ..., V_N, V_f]^T = [V_{term}, V_{term}, ..., V_{term}, 0]^T$. When the current vector is solved for, $\vec{I} = \bar{Z}^{-1}\ \vec{V}$, the fault won't contribute a self flux produced by a current in the faulted loop. Instead, the faulted turn will have current induced by the mutual flux of the current-carrying elements in the reactor, so the fault current will lag 90° behind the layer and terminal currents.



(a) A Prefault Reactor Schematic

(b) A Faulted Reactor Schematic

Figure 4.1: Currents in a Reactor, (a) prefault, (b) with faulted turns

4.3 Representing a Fault Between Turns

The perturbation matrices will start with the unfaulted NxN matrix, with with an additional row/-column appended to the right for each fault.

$$Z_{pref} = \begin{bmatrix} Z_{00} & jX_{01} & jX_{02} & \dots & jX_{0n} \\ jX_{10} & Z_{11} & jX_{12} & \dots & jX_{1n} \\ jX_{20} & jX_{21} & Z_{22} & \dots & jX_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ jX_{n0} & jX_{n1} & jX_{n2} & \dots & Z_{nn} \end{bmatrix}$$

The Perturbation, a fault in $Z1$ in this example, effects the impedance Z here proportionally to the short impedance Z_{S1}, and the mutuals to other layers. Where Z_{S_1} here is the impedance of the shorted turn(s).

$$Z_{pert} = \begin{bmatrix} 0 & -jX_{0S_1} & 0 & \dots & 0 & jX_{nS_1} \\ -jX_{S_10} & -Z_{S_1} - jX_{S1} & -jX_{S_12} & \dots & -jX_{S_1n} & jX_{1S_1} \\ 0 & -jX_{2S_1} & 0 & \dots & 0 & jX_{2S_1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & -jX_{nS_1} & 0 & \dots & 0 & jX_{nS_1} \\ jX_{S_10} & jX_{S_11} & jX_{S_12} & \dots & jX_{S_1n} & Z_{S_1} \end{bmatrix}$$

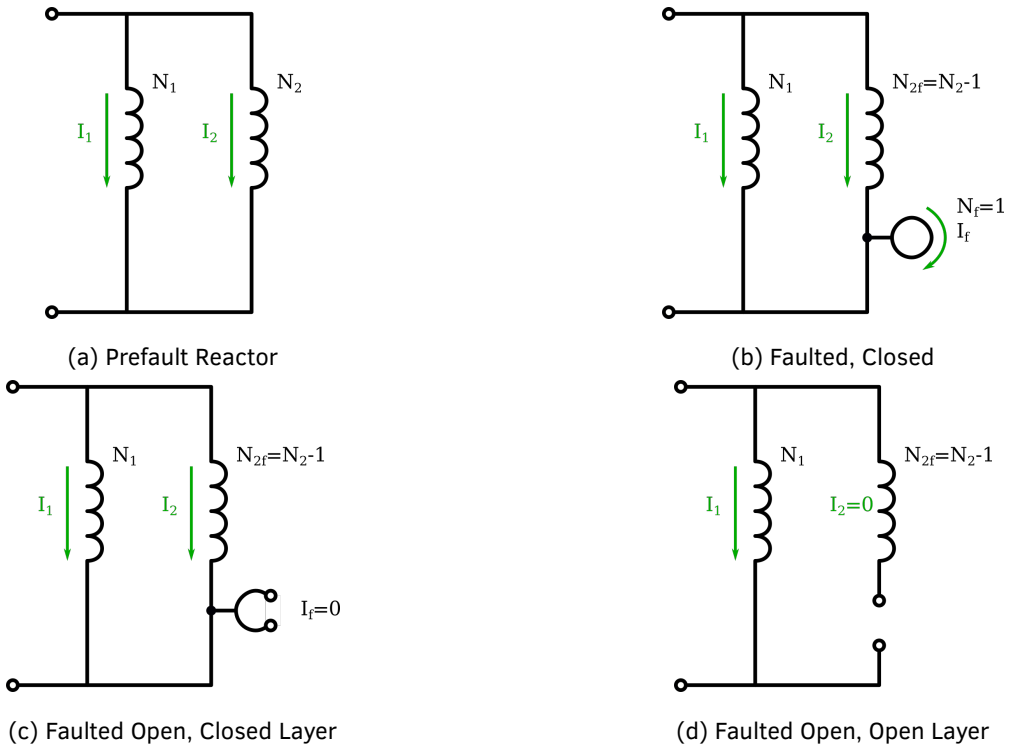
This results in an N+S square matrix:

$$Z_{faulted} = \begin{bmatrix} Z_{00} & jX_{01} & jX_{02} & \dots & jX_{0n} & jX_{0S_1} \\ jX_{10} & Z_{11} & jX_{12} & \dots & jX_{1n} & jX_{1S_1} \\ jX_{20} & jX_{21} & Z_{22} & \dots & jX_{2n} & jX_{2S_1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ jX_{n0} & jX_{n1} & jX_{n2} & \dots & Z_{nn} & jX_{nS_1} \\ jX_{S_10} & jX_{S_11} & jX_{S_12} & \dots & jX_{S_1n} & Z_{S_1S_1} \end{bmatrix}$$

There will be additional elements, augments, to represent the fault self and contribution to the other elements of the reactor: Z_{S_1} The mutuals $M = \{\, jX_{S_10},\, jX_{S_11},\, jX_{S_12},\, \cdots,\, jX_{S_1n} \,\}$

This representation of a fault as a perturbed matrix preserves the computation that went into the prefault condition while explicitly quantifying the effects of the fault on the reactor in-terms of the change in self and mutuals. The new row / column added to the perturbation matrix are the same magnitude of the difference from the prefault case, proportional to the loss of the turn.

When the faulted loop progresses to an open, the extra row and column can be removed to represent the loss of turn as reduction in self inductance, and the proportional losses in the mutual that turn contributes to the package. The open fault is then represented as a loss of inductance of the self, and the turns contribution to the mutual linkages to the other elements within the reactor.



(a) Prefault Reactor

(b) Faulted, Closed

(c) Faulted Open, Closed Layer

(d) Faulted Open, Open Layer

Figure 4.2: Possible Fault states of an Air-Core Reactor

4.4 Faulted Reactor as a Transformer

To better understand the behavior of the fault in an ACR, the fault can be thought of as a $N:1$ ideal transformer. In this transformer analogue, the change in layer currents can be described in-terms of the fault resistance, R_f, being referred to the primary side of the transformer via a $N^2:1$ ratio, R'_f. As will be demonstrated in section 4.4.1, the resistance of the fault turn is significantly larger than inductance of the faulted turn, so the fault resistance dominates the fault behavior. When the fault resistance is referred to the primary, it will be relatively large depending on the turns ratio, and will be in parallel with the winding resistance and reactance. The larger resistance will pass a small amount of current and would present as a slight positive phase shift toward $0°$, there is also an increase in the current magnitude as measurable from the terminal.

4.4.1 2 Layer Fault Example

Each fault case described in the example will have a correspondence to the fault states illustrated in figure 4.2. For this example, the reactor radii (layers 1 and 2) and height were chosen arbitrarily, and the layer 1 turn count also chosen arbitrarily. Layer 2 turn count was driven by the desire to have the currents of each layer approximately in-phase, and thus have a reduced turn count. The reactor

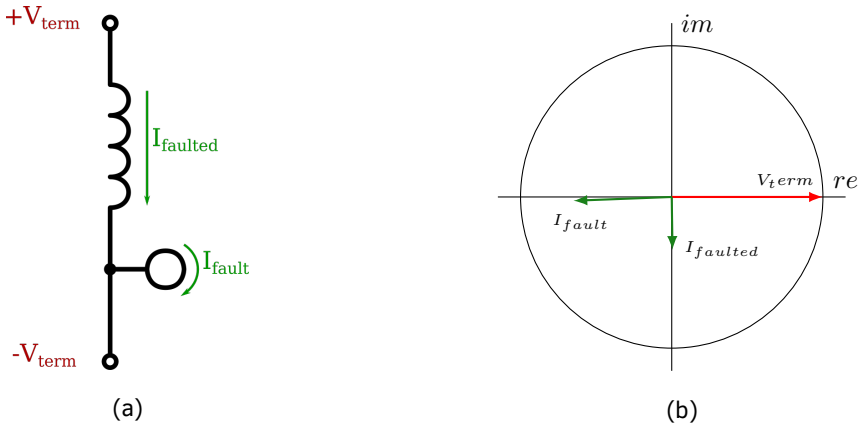


Figure 4.3: Example Fault Schematic and Phasor Diagram

parameter	layer 1	layer 2	fault
radius $[m]$	0.50	0.55	0.55
height $[m]$	0.50	0.50	0.000532
turns	1000	940	1

Table 4.1: Parameters for the 2-layer example reactor.

parameters listed in table 4.1, during fault conditions, the layer 2 will effectively have 1 fewer turn, but this is modeled in the perturbation L_p which is the effect the fault has on the prefault reactor, shown in equation (4.12). The total DC resistance was desired to be around 0.9Ω, and real components of the diagonal of the impedence matrix (4.4) were chosen to achieve that final resistance as seen in (4.7).

$$N = \begin{bmatrix} 1000000 & 940000 \\ 940000 & 883600 \end{bmatrix} \tag{4.1}$$

$$G = \begin{bmatrix} 0.131 & 0.119 \\ 0.119 & 0.152 \end{bmatrix} \tag{4.2}$$

$$L = \begin{bmatrix} 1.037 & 0.881 \\ 0.881 & 1.060 \end{bmatrix} \tag{4.3}$$

$$Z = \begin{bmatrix} 1.800E+00+j3.911E+02 & 0+j3.320E+02 \\ 0+j3.320E+02 & 1.800E+00+j3.998E+02 \end{bmatrix} \tag{4.4}$$

$$I = \begin{bmatrix} 6.337E-06-j1.470E-03 \\ 5.038E-07-j1.281E-03 \end{bmatrix} = \begin{bmatrix} 1.470E-03\angle -89.753^\circ \\ 1.281E-03\angle -89.977^\circ \end{bmatrix} \tag{4.5}$$

$$I_T = 6.841E-06-j2.750E-03 = 2.751E-03\angle -89.857^\circ \tag{4.6}$$

$$Z_T = 9.042E-01 + j3.636E+02 = 3.636E+02\angle 89.857^\circ \tag{4.7}$$

As a reminder; N and G are provided as they are used to compute L by way of a Hadamard product and scaled by the value $2\pi\mu_0$, for more see equation (2.22). The prefault conditions in (4.1) through (4.7) give us a bit of information:

1. The difference in self inductance of the Layer minor, $< 13mH$,
2. The mutual elements of (4.3) are between 85% and 83% of the self values, looking at the difference in area enclosed by the two layers, this makes sense since: $1-\frac{A_2-A_1}{A_2} = 1-\frac{((.55m)^2-(.5m^2))}{(.55m)^2} \approx 0.826 = 82.6\%$, and as stated in a previous chapter inductance is proportional to the flux through the surface enclosed by a loop.
3. Both currents are on of the same order of magnitude

Equation 4.8 gives the turns matrix of the faulted reactor:

$$N_f = \begin{bmatrix} 1000000 & 940000 & 1000 \\ 940000 & 883600 & 940 \\ 1000 & 940 & 1 \end{bmatrix} \tag{4.8}$$

Equation 4.9 is the geometry matrix of the faulted reactor with the input parameters given in table 4.1.

$$G_f = \begin{bmatrix} 0.131 & 0.119 & 1.394E-04 \\ 0.119 & 0.152 & 1.774E-04 \\ 1.394E-04 & 1.774E-04 & 8.441E-07 \end{bmatrix} \tag{4.9}$$

Equation 4.10 is the reactor inductance with a single loop fault. Note that $L_{3,3}$ is significantly smaller than the layer the fault is in, approximately $1/940^2$ smaller.

$$L_f = \begin{bmatrix} 1.037 & 0.881 & 1.101E-06 \\ 0.881 & 1.060 & 1.316E-06 \\ 1.101E-06 & 1.316E-06 & 6.665E-12 \end{bmatrix} \tag{4.10}$$

Equation 4.11 is the faulted impedance matrix, with the perturbation added.

$$Z_f = \begin{bmatrix} 1.800 + j3.911E+02 & 0 + j3.320E+02 & 0 + j4.150E-04 \\ 0 + j3.320E+02 & 1.798 + j3.998E+02 & 0 + j4.963E-04 \\ 0 + j4.150E-04 & 0 + j4.963E-04 & 1.915E-03 + j2.513E-09 \end{bmatrix} \tag{4.11}$$

Equation 4.12 inductance perterbation matrix for this example, this is how the fault effects the self and mutual elements of the layer.

$$L_p = \begin{bmatrix} 0 & -1.101E-06 & 0 \\ -1.101E-06 & -1.316E-06 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.12}$$

Equation 4.13 is the faulted loop current vector, the loop is clopsed and still able to carry a current.

$$I_{f\ cl} = \begin{bmatrix} 6.355E-06-j1.470E-03 \\ 4.840E-07-j1.281E-03 \\ 0-j1.502E-06 \end{bmatrix} = \begin{bmatrix} 1.470E-03\angle -89.752^\circ \\ 1.281E-03\angle -89.978^\circ \\ 6.505E-04\angle -179.868^\circ \end{bmatrix} \tag{4.13}$$

Equation 4.14 shows the total current, as would be seen by a current probe at a terminal.

$$I_{T\ cl} = -6.437E-04-j2.752E-03 = 2.826E-03\angle -103.164^\circ \tag{4.14}$$

Equation 4.15 the total impedance, as measurable at the terminals.

$$Z_{T\ cl} = 9.042E-01+j3.636E+02 = 3.636E+02\angle 89.857^\circ \tag{4.15}$$

Equation 4.16 faulted current loop vector, where the faulted loop has opened and is no longer carrying a current.

$$I_{f\ ol} = \begin{bmatrix} 6.355E-06-j1.470E-03 \\ 4.832E-07-j1.281E-03 \end{bmatrix} = \begin{bmatrix} 1.470E-03\angle -89.752^\circ \\ 1.281E-03\angle -89.978^\circ \end{bmatrix} \tag{4.16}$$

Equation 4.17 is the faulted-open total current as would be seen from a terminal.

$$I_{T\ ol} = 6.838E-06-j2.750E-03 = 2.751E-03\angle -89.858^\circ \tag{4.17}$$

Equation 4.18 is the total impedance of the reactor with the faulted turn opened up.

$$Z_{T\ ol} = 9.038E-01+j3.636E+02 = 3.636E+02\angle 89.858^\circ \tag{4.18}$$

Equation 4.19 shows the faulted current when the layer containing the fault opens, and the entire layer is effectivly removed from the reactor

$$I_{f\ o} = 1.177E-05-j2.557E-03 = 2.557E-03\angle -89.736^\circ \tag{4.19}$$

Equation 4.20 is the total current of the open-layer condition, effectivly just the current on layer 1 (inner layer)

$$I_{T\ o} = 1.177E-05-j2.557E-03 = 2.557E-03\angle -89.736^\circ \tag{4.20}$$

Equation 4.21 is the total impedance, which is the same as the impedance of the 1st layer because the 2nd is open.

$$Z_{T\ o} = 1.800E+00+j3.911E+02 = 3.911E+02\angle 89.736^\circ \tag{4.21}$$

In the results above 4.13 shows the current vector with the fault as the last element, and it can be see the angle of the current in the closed loop is lagging $90°$ behind either layer 1 or layer 2. When the faulted loop opens up in equation 4.16, the current returns to a normal angle, but the inductance has changed by 1 turn and there is more real current flowing due to the reduced resistance.

The large influence of the faulted loop on the angle of the total current can be attributed to the relatively small system, in this example, a single faulted loop is $\frac{1\text{ turn}}{1940\text{turns}}$. In production reactors, there are between $10k$ and $40k$ turns or more, which reduces the effects of a single fault on the system.

4.5 Fault Model Validation

This section goes over the analysis of the fault, and validation of fault modeling methods logically by using a transformer analogue to describe behavior.

Using the parameters from the previous chapter, the model reactors described in table 3.2 are used, with faults inserted on the innermost layer.

4.5.1 Single Layer, Single Turn Fault

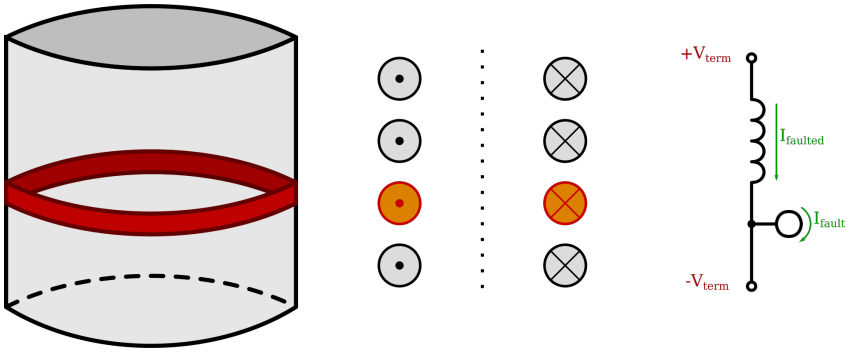
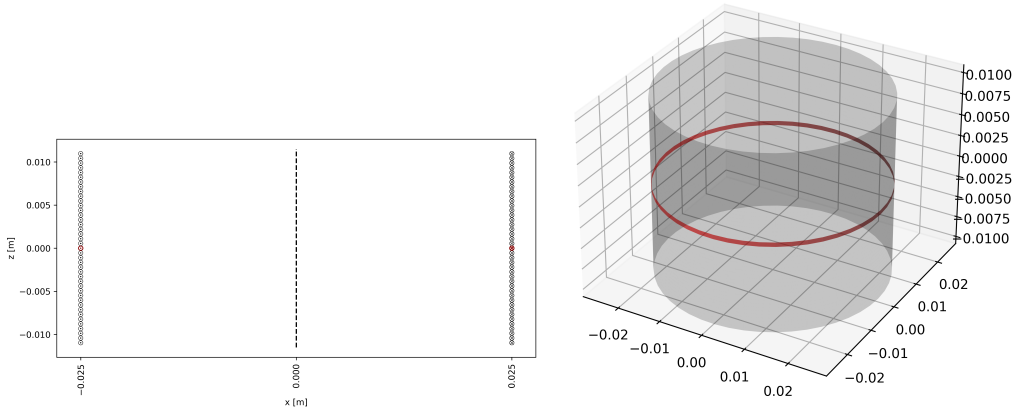


Figure 4.4: Faulted turn in a cylindrical shell (red band), the

To test the transformer analogue using test data gathered using the test reactors from the previous chapter, the single-layer test reactor is built, and a short is inserted at turn 21, the turn-to-turn visualization and sheets visualized in figure 4.5. The reactor is described in table 3.2, as *Layer 1f*, it is a faulted equivalent to the reactor shown in figure 3.3. The test reactor is in series with a 10Ω resistor to measure the current in the faulted reactor.

Using the Biot-Savart methods to evaluate the turn-to-turn inductance of the reactor in the faulted state, the inductance matrix can be seen in 4.22, this result



(a) Turns Representation

(b) Sheets Representation

Figure 4.5: Visualization of a fault in a single-layer reactor. The fault is highlighted by the red turn and band.

$$L_{bs} = \begin{bmatrix} 8.338E-05 & 2.204E-06 \\ 2.204E-06 & 1.535E-07 \end{bmatrix} \tag{4.22}$$

$$Z_{bs} = \begin{bmatrix} 1.052E+01+j5.239 & 0+j1.385E-01 \\ 0+j1.385E-01 & 1.289E-02+j9.646E-03 \end{bmatrix} \tag{4.23}$$

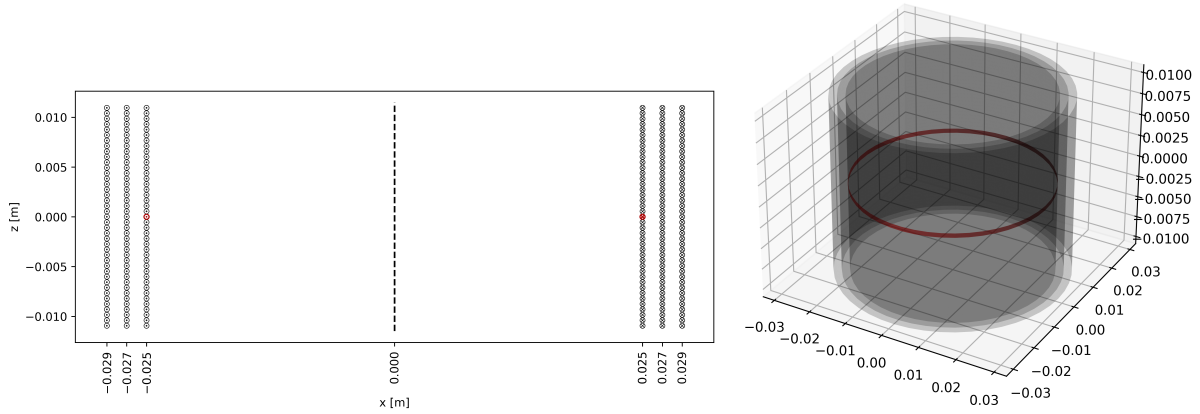
$$I_{bs} = \begin{bmatrix} 7.997E-02-j3.155E-02 \\ 0-j3.881E-01 \end{bmatrix} = \begin{bmatrix} 8.597E-02\angle -21.531^\circ \\ 7.395E-01\angle -148.345^\circ \end{bmatrix} \tag{4.24}$$

$$L_{fb} = \begin{bmatrix} 8.609E-05 & 2.439E-06 \\ 2.439E-06 & 1.689E-07 \end{bmatrix} \tag{4.25}$$

$$Z_{fb} = \begin{bmatrix} 1.052E+01+j5.409 & 0+j1.532E-01 \\ 0+j1.532E-01 & 1.289E-02+j1.061E-02 \end{bmatrix} \tag{4.26}$$

$$I_{fb} = \begin{bmatrix} 7.935E-02-j3.088E-02 \\ 0-j3.820E-01 \end{bmatrix} = \begin{bmatrix} 8.515E-02\angle -21.265^\circ \\ 7.815E-01\angle -150.735^\circ \end{bmatrix} \tag{4.27}$$

When Compared to a physical test of a 41 turn reactor with turn 21 faulted, the result in the first element, representing the current in the layer, it closly matches the result seen in physical testing, the results of the testing can be found in appendix B.2. When compared to the prefault model from the previous chapter: $I_{bs} = 8.915 \times 10^{-02}\angle -27.690^\circ$ and $I_{fb} = 8.900 \times 10^{-02}\angle -27.876^\circ$, the resulting change due to the fault is minor, both methods show a change in current as $\approx 3.56\%$ using the turn-to-turn methods, and $\approx 4.33\%$ from the sheets method. Considering the change in turns is 1, and the fault is $1/40 = 2.5\%$ of the total turns in the reactor, this appears to be a valid



(a) Turns Depiction

(b) Layers Depiction

Figure 4.6: Visualization of a "Untuned" multilayer reactor model. The fault is highlighted by the red turn and band.

Comparing the values of the current, in either method, we see there is approximately an order of magnitude difference in the current between the prefault and faulted cases. The Python program to simulate these tests can be found in appendix D.6.

4.5.2 Multiple Layer, Single Turn Fault

Starting with the 3-layer example model from the previous chapter, the innermost layer the example fault from the previous section. The following section will demonstrate the modeling and analysis method for a multi-layer reactor.

Equation 4.28 impedance matrix of the untuned prefault reactor.

$$Z_{ut} = \begin{bmatrix} 10.53 + j5.733 & 0 + j5.247 & 0 + j4.849 \\ 0 + j5.247 & 10.57 + j6.441 & 0 + j5.901 \\ 0 + j4.849 & 0 + j5.901 & 10.61 + j7.171 \end{bmatrix} \tag{4.28}$$

Equation 4.29 is the prefault current vector for an untuned reactor.

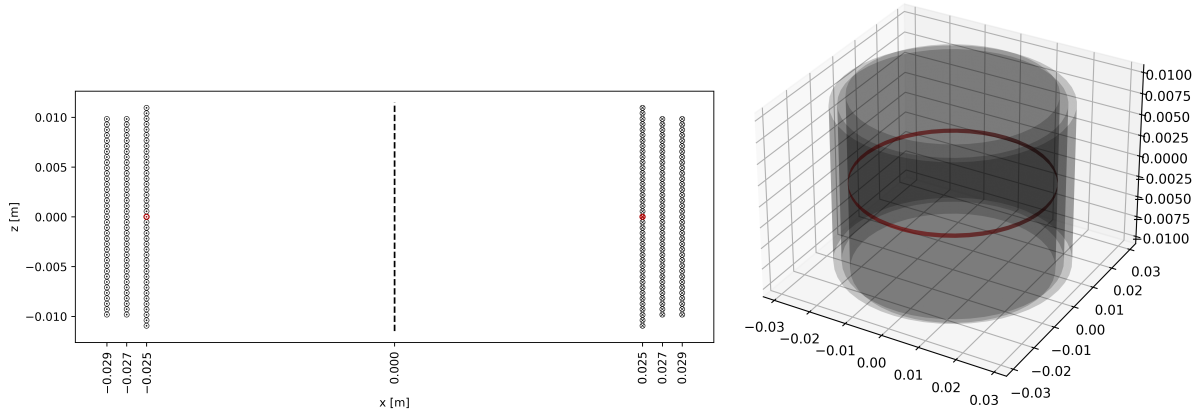
$$I_{ut} = \begin{bmatrix} 2.097E-02 - j2.630E-02 \\ 1.602E-02 - j2.853E-02 \\ 1.498E-02 - j2.861E-02 \end{bmatrix} = \begin{bmatrix} 3.364E-02\angle -51.432^\circ \\ 3.272E-02\angle -60.684^\circ \\ 3.229E-02\angle -62.370^\circ \end{bmatrix} \tag{4.29}$$

Equation 4.30 is the total current for the prefault untuned reactor.

$$I_{total\ ut} = 5.1971E-02 - j8.3444E-02 = 9.8305E-02\angle -58.085^\circ \tag{4.30}$$

Equation 4.31 is the total impedance of the untuned reactor, prefault.

$$Z_{total\ ut} = 3.5494 + j5.6989 = 6.7138\angle 58.085^\circ \tag{4.31}$$



(a) Turns Depiction

(b) Layers Depiction

Figure 4.7: Visualization of a "Tuned" multilayer reactor model. The fault is highlighted by the red turn and band.

Equation 4.32 is the impedance matrix of the untuned test reactor, with a fault on turn 21 of the innermost layer, layer 1.

$$Z_{utf} = \begin{bmatrix} 10.53 + j5.733 & 0 + j5.247 & 0 + j4.849 & 0 + j0.1398 \\ 0 + j5.247 & 10.57 + j6.441 & 0 + j5.901 & 0 + j0.1280 \\ 0 + j4.849 & 0 + j5.901 & 10.61 + j7.171 & 0 + j0.1183 \\ 0 + j0.1398 & 0 + j0.1280 & 0 + j0.1183 & 0.01289 + j3.410E - 3 \end{bmatrix} \tag{4.32}$$

Equation 4.33 is the current vector of the untuned test reactor, here it can be seen that the fault current is around $90°$ lagging the average layer current.

$$I_{utf} = \begin{bmatrix} 2.306E - 02 - j2.012E - 02 \\ 1.917E - 02 - j2.387E - 02 \\ 1.845E - 02 - j2.486E - 02 \\ 0 - j4.009E - 01 \end{bmatrix} = \begin{bmatrix} 3.060E - 02\angle - 41.107° \\ 3.061E - 02\angle - 51.235° \\ 3.096E - 02\angle - 53.417° \\ 8.855E - 01\angle - 153.080° \end{bmatrix} \tag{4.33}$$

$$I_{total\,utf} = 6.0683E - 02 - j6.8857E - 02 = 9.1781E - 02\angle - 48.611° \tag{4.34}$$

$$Z_{total\,utf} = 4.7545 + j5.3950 = 7.1910\angle 48.611° \tag{4.35}$$

Equation 4.34 is the total current of the untuned test reactor, excluding the fault element I_{utf}. Comparing the faulted case, eqn. 4.34, to the prefault case, eqn. 3.15, the total current has decreased. When looking at the total impedence of the test reactor, eqn. 4.35, shows that the total impedance of the untuned test reactor has increased.

When the simulated values are compared to a physical test of prefault and faulted cases, we see the fault behavior of the simulated model relativly accuratly captures the effects. Equation 4.36 shows

the measured untuned prefault currents in the 3-Layer model reactor, and equation 4.37 shows the measured untuned currents in the faulted 3-Layer model reactor.

$$I_{meas.\ ut} = \begin{bmatrix} 3.400E-02\angle -42.25^\circ \\ 3.200E-02\angle -69.15^\circ \\ 3.000E-02\angle -66.03^\circ \end{bmatrix} \tag{4.36}$$

$$I_{meas.\ utf} = \begin{bmatrix} 3.200E-02\angle -39.44^\circ \\ 3.000E-02\angle -52.55^\circ \\ 3.000E-02\angle -55.92^\circ \end{bmatrix} \tag{4.37}$$

It is worth mentioning that the physical test reactors were excited at $10kHz$, which does have a non-negligable amount of skin effect that the simulation doesn't account for, and makes for part of the difference between simulated and experimental values.

Now, to look into the case where the example reactor layers have been tuned for balanced currents: Equation 4.38 is the prefault impedance matrix of the tuned test reactor.

$$Z_t = \begin{bmatrix} 10.53+j5.733 & 0+j4.833 & 0+j4.456 \\ 0+j4.833 & 10.51+j5.522 & 0+j5.022 \\ 0+j4.456 & 0+j5.022 & 10.55+j6.139 \end{bmatrix} \tag{4.38}$$

Equation 4.39 current vector of the prefault tuned reactor.

$$I_t = \begin{bmatrix} 1.976E-02-j2.703E-02 \\ 1.879E-02-j2.757E-02 \\ 1.806E-02-j2.779E-02 \end{bmatrix} = \begin{bmatrix} 3.348E-02\angle -53.833^\circ \\ 3.337E-02\angle -55.722^\circ \\ 3.314E-02\angle -56.986^\circ \end{bmatrix} \tag{4.39}$$

Equation 4.40 is the total current of the prefault tuned test reactor.

$$I_{total\,t} = 5.6603E-02-j8.2384E-02 = 9.9955E-02\angle -55.508^\circ \tag{4.40}$$

Equation 4.41 is the prefault total impedance, with the 10Ω resistors inserted in series with the layer:

$$Z_{total\,t} = 3.5125+j5.1124 = 6.2028\angle 55.508^\circ \tag{4.41}$$

$$Z_{tuned\,f} = \begin{bmatrix} 10.53+j5.733 & 0+j5.247 & 0+j4.849 & 0+j0.1398 \\ 0+j5.247 & 10.57+j6.441 & 0+j5.901 & 0+j0.1280 \\ 0+j4.849 & 0+j5.901 & 10.61+j7.171 & 0+j0.1183 \\ 0+j0.1398 & 0+j0.1280 & 0+j0.1183 & 1.289E-2+j3.410E-3 \end{bmatrix} \tag{4.42}$$

$$I_{tf} = \begin{bmatrix} 2.306E-02-j2.012E-02 \\ 1.917E-02-j2.387E-02 \\ 1.845E-02-j2.486E-02 \\ 0-j4.009E-01 \end{bmatrix} = \begin{bmatrix} 3.060E-02\angle -41.107^\circ \\ 3.061E-02\angle -51.235^\circ \\ 3.096E-02\angle -53.417^\circ \\ 8.855E-01\angle -153.080^\circ \end{bmatrix} \tag{4.43}$$

Equation 4.44 is the total current of the faulted condition:

$$I_{total\,tf} = 6.0683E-02 - j6.8857E-02 = 9.1781E-02\angle -48.611^\circ \tag{4.44}$$

Equation 4.45 is the total impedance of the test reactor with the shorted loop #21 of layer 0:

$$Z_{total\,tf} = 4.7545 + j5.3950 = 7.1910\angle 48.611^\circ \tag{4.45}$$

As with the untuned variant of the physical test reactors, Equation 4.46 shows the tuned prefault 3-layer model reactor, and equation 4.47 is the faulted model reactor:

$$I_{meas.\,t} = \begin{bmatrix} 3.400E-02\angle -49.87^\circ \\ 3.400E-02\angle -55.65^\circ \\ 3.400E-02\angle -60.91^\circ \end{bmatrix} \tag{4.46}$$

$$I_{meas.\,tf} = \begin{bmatrix} 3.400E-02\angle -39.78^\circ \\ 3.400E-02\angle -50.24^\circ \\ 3.400E-02\angle -55.36^\circ \end{bmatrix} \tag{4.47}$$

The Python program to simulate these tests can also be found in appendix D.7. Oscilloscope screencaptures for the measured results are provided in appendix B.1.

Chapter 5

Fault Detection

Turn to turn faults in air core reactors are a common mode of failure, repeated high-voltage transients from switching the reactor into service and the mechanical stresses produced can degrade the reactor insulation over time [2]. Modeling of turn-to-turn faults has been the primary goal of the research presented.

There is a method presented in literature, [13] and [15], both articles use an approach based on the relative angle between the zero and negative sequence current on the bus the reactors are connected. The idea behind this approach is that as a faulted loop occurs, and progresses, The faulted turns will be producing a flux counter to the primary flux generated by the reactor, which presents in the negative and zero sequence currents in reference to the phase which it is connected [15]. The issue with turn to turn faults is the high degree of sensitivity needed to detect fault.

Mohammad *et. al.* [13], propose a method for determining the presence of a turn-to-turn fault in an ACR relying on the negative and zero sequence currents measured by a relay. Using the argument of the negative and zero sequence currents, $\theta_0 = \arg(I_0)$ and $\theta_2 = \arg(I_2)$, the difference is used for evaluating presence of a fault: $\Delta\theta = \theta_0 - \theta_2$. The zero sequence current argument, θ_0, is used as a reference, and when a turn-to-turn fault is present the negative sequence argument, θ_2, will change from being approximately the same as θ_0 to where $\Delta\theta = 120^\circ$. Chowdhury *et. al.* [15], like Mohammad *et. al.*, uses the deviation in argument of the negative and zero sequence current.

The use of negative and zero sequence current looks to be a standard method of detecting turn-to-turn faults. The work of Mohammad *et. al.* [13] and Chowdhury *et. al.* [15] rely on the symmetrical components to determine the presence of the fault.

Although it would be impractical for production units, Nurminen's thesis [8] presented a novel method to evaluate the design of a reactor that could be applied to fault detection. The method using optical fibers embedded within the reactor layers for temperature measurements and hotspot detection. The change in refractive index can be calculated from the time it takes a light pulse to travel

the length of the fiber. Hotspots can be detected with a combination of time and back scatter, to detect the position of the hotspot. In the case of a turn-to-turn fault, the insulation and encapsulation would degrade from excess heat prior to a fault, which would be seen as a hotspot. would provide sufficient information to the operator to take action before a fault or potential fault becomes a larger issue.

The result of the work performed in the course of the modeling and fault analysis chapters shows the ACR is relatively insensitive to a few, between 1 and 10 turns, faults. These observations are in-line with the issues stated by Chowdhury *et. al.* [15]. As the effects of faults in ACRs is challenging to detect using terminal measurements, aside from adding additional hardware to the system, there isn't a better method of detecting faults than those already in-use.

Chapter 6

Computer Program

6.1 Introduction

The computer program used to construct a model of a reactor, and evaluate the fault behavior was where a significant amount of time and effort went into for the duration of the research presented in this document. The computer program evolved from a computation intense loop to loop method, to a more efficient method through the application of Fawzi and Burke's 1978 paper. This chapter will look at the general practices and optimizations used in the development of the computer program during the course of the research. Starting with thee loop to loop implementation and thee techniques used to reduce compute time,Going to the more competition efficient thin sheets method and how that method is used to design, or tune, reactors to achieve a desired pre-fault state.

6.2 Loop to Loop Implementation

Initially the computer program to evaluate air core reactors and faults use an object oriented approach. With an object class for layer elements, packages, and reactor objects, the use of object or intend programming seemed necessary to manages the process and data in concise ways color by packaging methods with data the mutual inductance between layer objects which would form packages. Because the mutual between every turn and every other turn and the reactor was necessary, it made the object oriented approach convenient. Also due to the number of mutuals that needed to be computed, a multithreaded computation method was necessary to leverage modern multicore compute power primarily to reduce the amount of time it took to calculate the mutual inductance between all the terms and the reactor, for since the processes used were $\mathcal{O}(N^2)$. Once the more computationally efficient method by Fawzi and Burke was implemented it made the object oriented approach unnecessary Because the amount of data and computation was significantly reduced. This

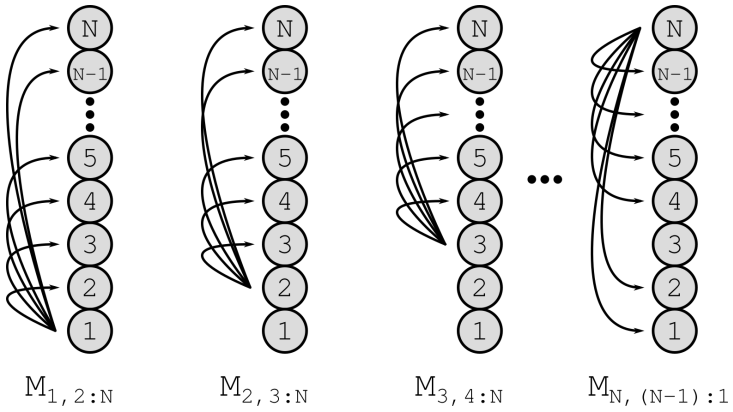


Figure 6.1: The behavior used to calculate mutual inductance of a layer once, due to regular spacing of turns.

eliminated the complexity of the object oriented approach, and eliminated the need for multithreaded programming which was prone to memory weeks and allocation errors which would hold the evaluation effort and cause the program to crash.

Initially the project started using individual turns as the lowest level object, this approach led to a hierarchy of objects with the lowest level being a turn continuing onto groups of turns as layers which were turns connected in series, and packages consisting of multiple layers in parallel. This object oriented approach was highly granular where turns could be added and removed from layers relatively easily recompute the total inductance of the reactor,

The mutual inductance between turns in a layer can be evaluated in $\mathcal{O}(N)$ time, assuming turns are of the same radius and distributed over a regular interval. With the assumption of a regular and consistent turn pitch, the mutual inductance can be calculated *once*, and the resultant values can be re-used for turns that are of the same distance apart. Figure 6.1 illustrates the reasoning behind the process, the mutual links are evaluated from turn 1 to each turn $2, 3, ..., (N-1), N$. Then, evaluate the mutual inductance between turn 2 and turns $3, ..., (N-1), N$, the mutuals become: $M_{2,3} = M_{1,2}$, $M_{2,4} = M_{1,3}$, ..., $M_{2,(N-1)} = M_{1,(N-2)}$, $M_{2,N} = M_{1,(N-1)}$, where $M_{1,2 \to N}$ were evaluated on the first pass. The mutual $M_{2,1}$ is the same as the mutual $M_{2,3}$ when the regularity assumption is applied, and both $M_{2,1}$, $M_{2,3} = M_{1,2}$.

Because all the turns in a single layer shared the same current superposition could be used to consolidate values of the layers. Layers place into a package will not have the same current magnitude so superposition cannot be used to simplify that element. The entire react you be represented by a square matrix whose dimensions are driven by the number of layers in the reactor total.

As layers are placed in parallel and packages who position could not be used as we cannot assume the same current magnitude in every layer of the package. So when forming the matrix to represent the reactor model there would be a single element for every layer in every package. And because every

layer in the package consisted at the same number of turns, the only thing that changed between layers is the radius of the layer and since the Biot-Savart law, and specifically the Newman integral is highly sensitive to the area enclosed by the current filament (i.e. the turns). This means that the mutual and self terms of the submatrix that represents the package would all be very similar. That means the submatrix representing to package would be close to singular, and as discussed in a previous section the condition number of that matrix would be large, so we would have high error when inverting the matrix. this poor conditioning can be extended out to the larger matrix, and since the condition number is the ratio of the largest eigen value over the smallest, the lower bound on error when inverted the matrix for the entire reactor would be large.

The procedure for implementing a fault in one of these reactors, which is described by components, is to remove the turn and any mutual between the faulted turn from every element of the reactor. Then compose a new element for the reactor which consists of that single faulted turn and every layer element within that matrix. This would add a new row and column to the reactor model, whose self element would be the self inductance of the turn and the flux internal to that turn, and the of diagonal elements would be the mutual between the faulted turn and the layer, for every layer in the reactor.

This approach to computation preserved the values already computed for the prefault case, since the only element that changed was that turn. And since the turn that was faulted in the prefault case is physically located in the same space the values could simply be copied over to the new element representing that fault.

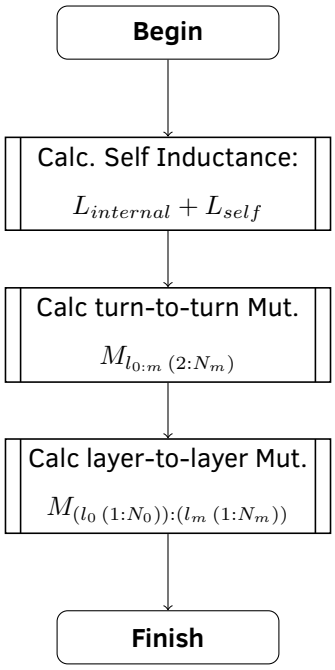


Figure 6.2: Turn-to-Turn Reactor Model Calculation, N: Number of Turns, m: Number of Layers in the Reactor

The drawback to this approach is that it was computationally intense to solve. Since the mutual had to be computed from every turn to every other, turn this approach required an N^2 time to solve. The advantage to this approach is that we have extreme granularity and we can pick and choose what we want to change and when we can do it on the fly the initial computation time and effort can be preserved.

To compute the inductance value of a layer "object" the main thing to remember is the symmetry of the flux linkages. This symmetry means that we can effectively halve our computation effort, when computing the self inductance of a layer used the torch turn method because the linkages are symmetrical we don't need to calculates the inductance going backwards, that is, we don't need to calculate inductions going towards the zero index, we only need to calculate the inductance towards the nth index. Was we had the inductance for each value from the zero to the end index it is a trivial operation to simply double it to represent going from the nth index to the zeroth index. A similar principle to this applies when computing the mutual between turns in different layers, where the different layers have different current values flowing through them, where we only need to calculate the mutual inductance from a layer a to a layer b and those values can be reflected (mirrored) on layer b. A nice mental visualization of this would be something like a slighting scale on set of er near calipers, the inductance in a lair is only dependent on its for total distance (assuming the coupled loops are coaxial).

6.3 Sheets Implementation

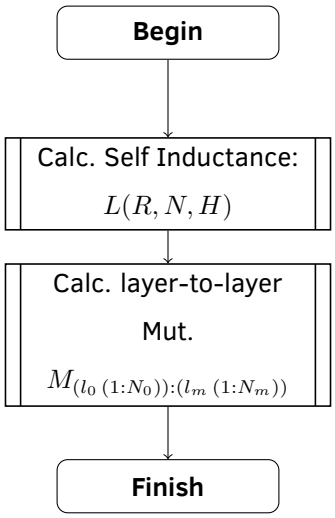


Figure 6.3: Thin-Sheets and Cylindrical-Shells Reactor Model Calculation, N: Number of Turns, m: Number of Layers in the Reactor

The sheet's method as described by Fawzi and Burke's 1978 publication is an extrapolation of the Neumann Integral which generally lumps the parameters from the turn to turn method into a single unit known as a sheet. Implementing this method greatly reduces the time to compute the inductance of a series of turns, and the mutual inductance between series of turns. This method of simplifying a series of turns as a sheet makes the smallest element of a reactor a layer or package instead of a turn, as with the turned to turn method. This method greatly simplifies the reactor in terms of data structures, and reducing the time it takes to compute elements of the structure.

The computation time used to calculate the inductance values of an air corrector using deter to turn method verses the sheets method is still on the order of $\mathcal{O}(N^2)$, with N being the number of elements in the reactor. But with $N \approx 10$ in this case, rather than $N \approx 10^4$ for the turn-to-turn method. Evaluating the Newman integral takes up the majority of the computation time, rather than having to evaluate N_{turns}^2 integrals per layer / package, the four integrals of the `Ci` function per layer / package requires considerably less time to complete.

The method of valuating fault effects follows the same principles as the turn to turn method of faults, but there is a difference in that we don't have individual turns to evaluate. Instead the space in which the fault occupies is evaluated using the turns density of the layer that it is contained within and the equivalent height for the number of turns involved in the fault. This procedure will effectively create another cylindrical shell / sheet and its self and mutual contributions represent the self an mutual contributions of the fault to the rest of the reactor. This contribution is subtracted from the diagonal element representing the layer that the fault occurs in as well as its mutual contribution to that layer, this contribution will be also subtracted from the mutuals to other layers in the reactor. When the mutuals are placed on the outer row and column these values represent the mutual contributions of the fault to the rest the reactor, and the self of the fault is added to the $N+1^{th}$ place of the perturbation matrix.

6.4 Tuning Components For Balance

The process of tuning, more generally referred to as design, of the reactors took an iterative approach. There are two algorithms developed to tune the reactor, both algorithms seek approximately the same current magnitude distributed evenly across layers, and require all currents be flowing in roughly the same direction, i.e. the argument of the current phasor be approximately equal. The thin sheets reactor is a relatively straightforward implementation, but the finite thickens reactor has a few extra steps, as there are more parameters to tune. Both methods assume that the diameters of the lairs or packages have already been determined, and these methods are changing either the turns or package thickness to achieve a desired result.

6.4.1 Tuning a Thin-Sheets Reactor

Tuning the thin-sheets reactor is a time consuming process due to the number of elements in the reactor model. The method discussed here will focus on optimizing the number of turns to achieve a current phase angle on every layer that are approximately the same, to put another way we want all the layer currents to have approximately the same phase angle. The process is described in the flowchart shown in figure 6.4. Assuming the layer radii r and heights (h) are known, an initial guess as to the number of turns for each layer is given N, and the desired thevenin equivalent resistance the reactor would exhibit if measured across the terminals with all letters connected in parallel. To save time, the matrix representing the influence of the geometric parameters, $\bar{G}$ (equation 2.24), is calculated since these values are assumed not change. The matrix representing the influence of the turns, $\bar{N}$ (equation 2.23), can be evaluated for each iteration of turns without the need to evaluate the integrals that make the elements of the geometry matrix. The inductance matrix, $\bar{L}$, is then calculated as the scaled Hadamard product of the turns and geometry matrix (equation 2.22) The resistance of each layer is calculated to be proportional to the number of turns so that the final DC thevenin in resistance is the same as the desired input R_T, knowing that the DC resistance of each layer is proportional to the number of turns in that layer: $R_k \propto N_k$.

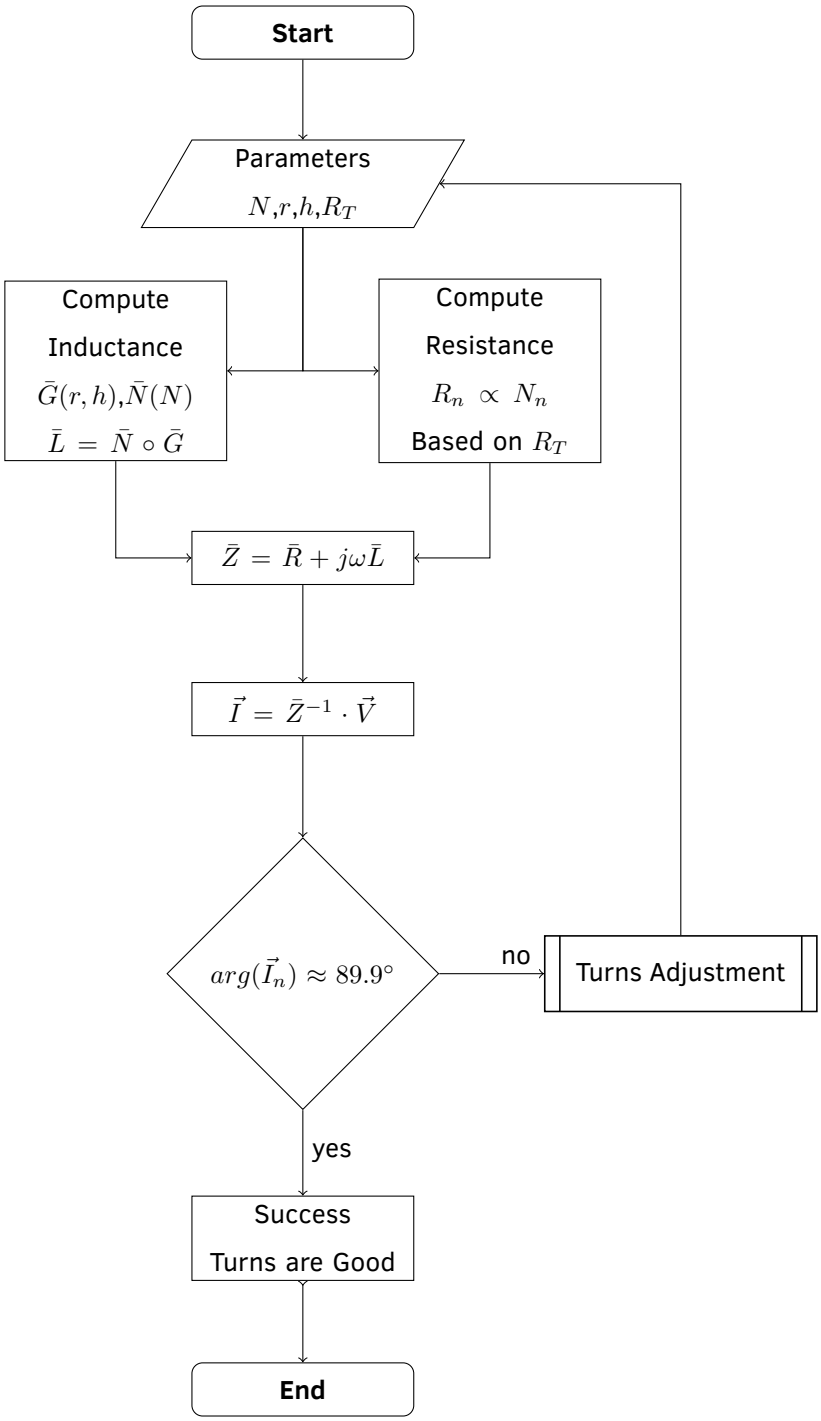


Figure 6.4: Tuning turns for a thin-sheets reactor model

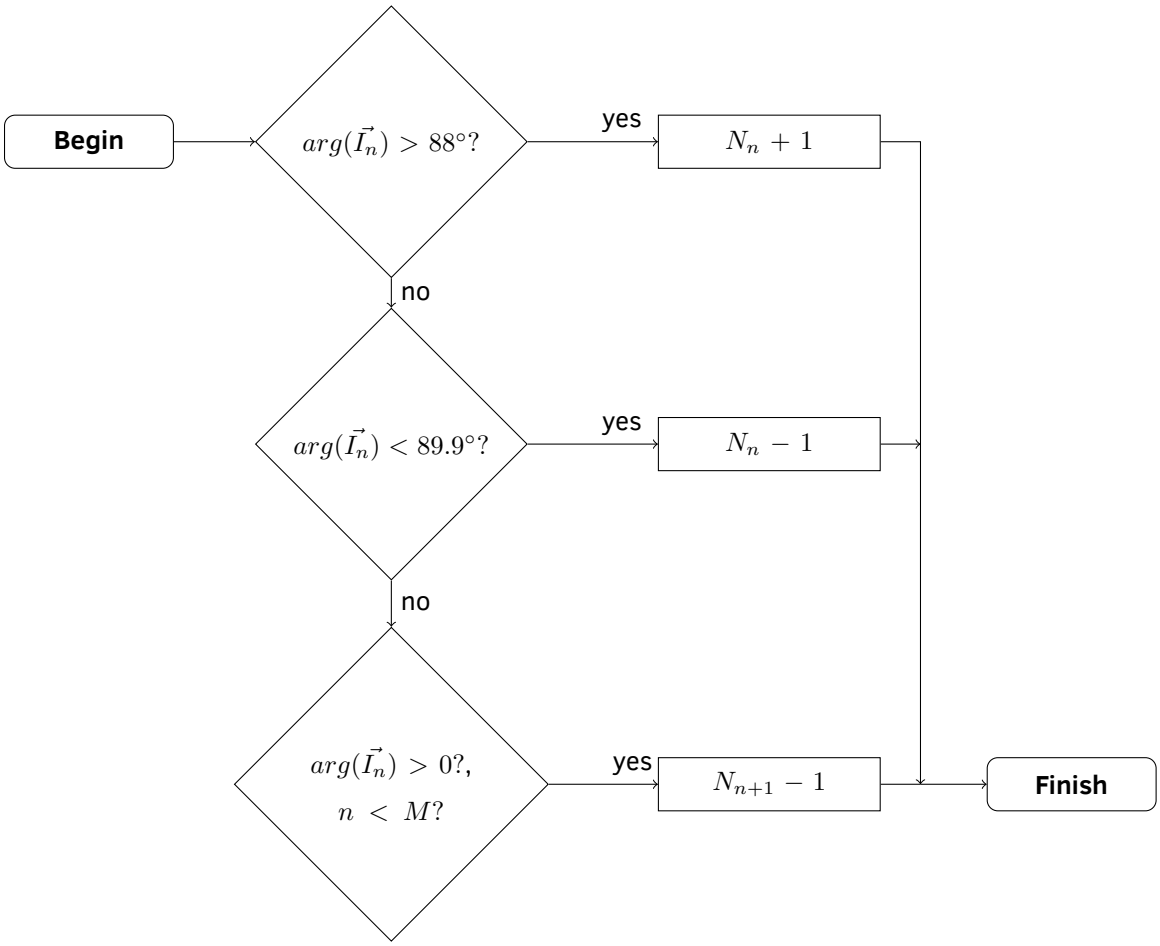


Figure 6.5: Turns Adjustment Subprocess

6.4.2 Tuning a Cylindrical-Shell Reactor

Tuning the finite-thickness, or cylindrical-shell reactor is similar to the process of tuning the turn counts of a thin sheets reactor. The primary difference in this case is that we're taking the result of the thin sheets reactor tuning process, and from there we will add thickness to the reactor elements using the same principles of minimizing circulating currents. If necessary a turns tuning process can be performed after the thickness has been determined, to further reduce circulating currents if needed.

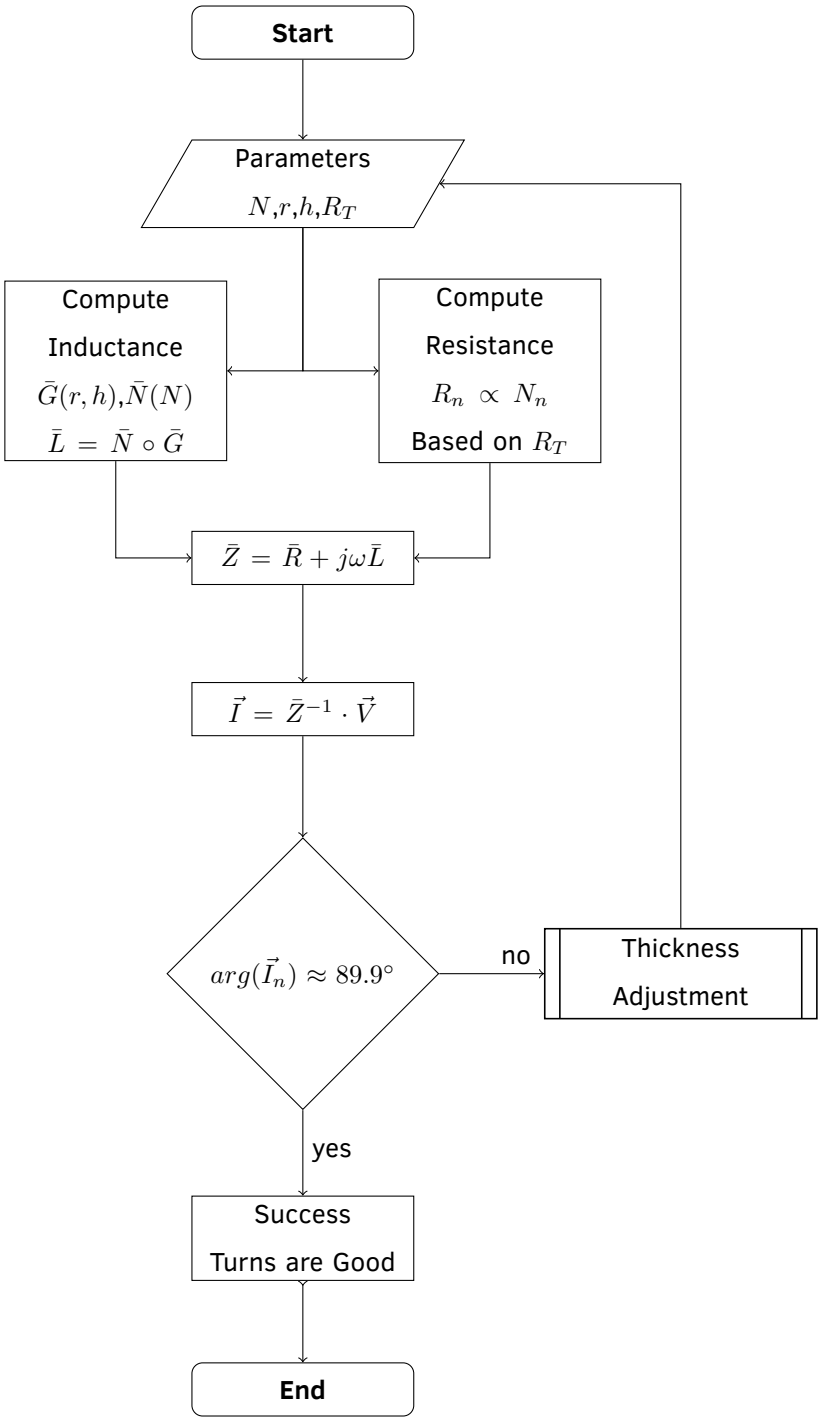


Figure 6.6: Tuning turns and thickness for a finite-thickness reactor model

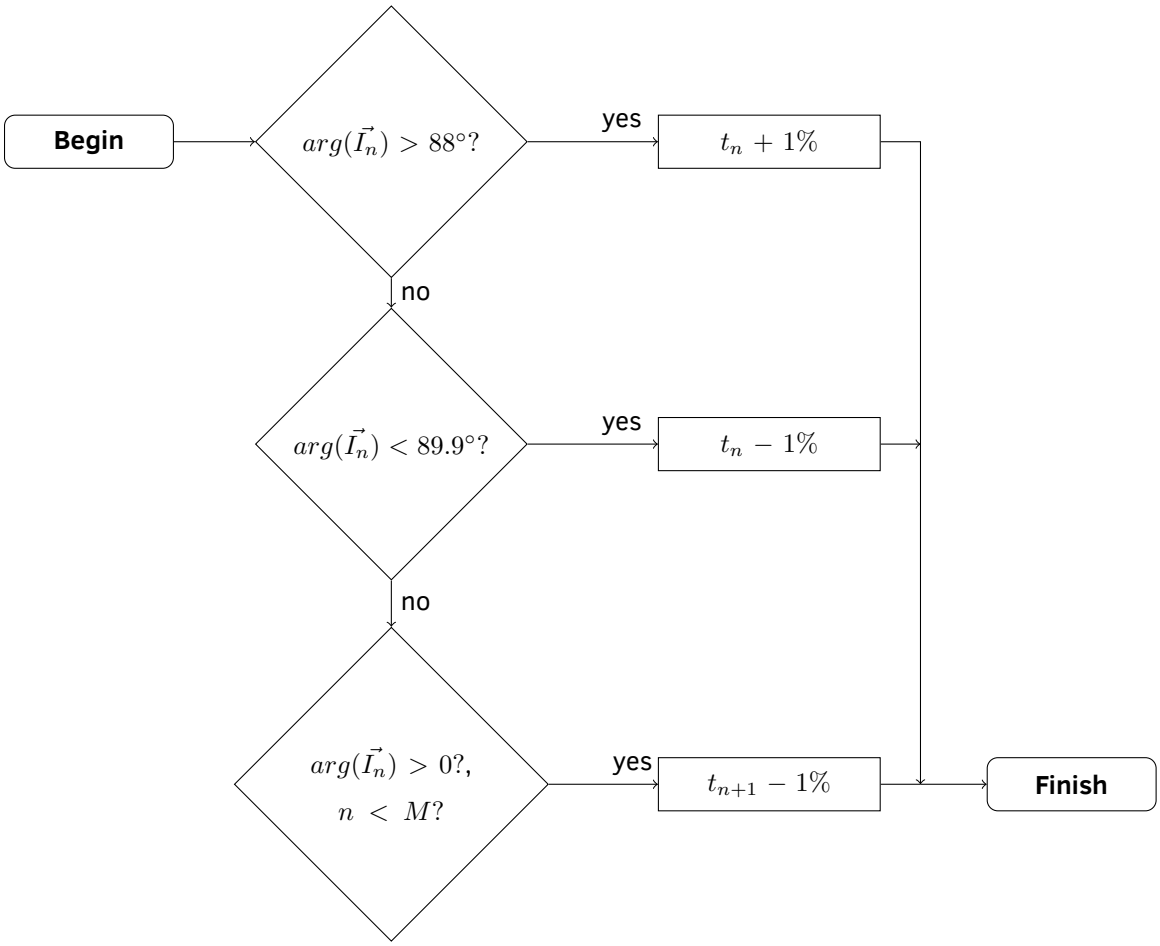


Figure 6.7: Thickness Adjustment Subprocess

Chapter 7

Results

7.1 Introduction

This chapter discusses the results of the simulation, which is the culmination of the work in this thesis. Here we will discuss the reactor model, which is the result of the iterative designed process discussed in previous chapters, and will be used in the fault analysis. A number of fault conditions will be discussed, starting with single turns, and how the location of those turns effects the reactor's behavior. And then we will discuss multiple faults, clustering as though a single turn fault has progressed to a multi turn fault through the breakdown of insulation and encapsulation media.

7.2 Reactor Modeling

The design of a reactor is a challenging, but critical step. In the development of the reactor modeling process, the ideal reactor would have currents of varying magnitudes, with each layer having the same X/R ratio. The goal in reactor design is to minimize the circulating currents between layers in the reactor. The tuning of a reactor design is achieved by varying the turn counts and conductor diameters. Varying the turn count will adjust the inductance values, and the layer resistnace. Changing the conductor diameter will vary the resistance of the layer with little impact on the inductance. Practically, the manuacturer of the reactor will have a specified set of conductors to wind the layers to balance the reactor layer currents.

For the models devoloped for the purpose of fault analysis, the design wasn't perfect, there are small circulating currents in the layers, further refinements could be achieved by adjusting the layer resisntance values. The design of the model reactor achieved a minimized set of circulating currents, which will be sufficient for the purtposes of evaluating faults. Effects of faults are measured as the change in model reactor current as would be measured at the reactor terminals, with the phasor

Package	1	2	3	4	5	6	7	8	9	10
package turns	1327	1187	1093	1029	985	956	938	929	928	935
average radius [m]	0.7	0.75	0.8	0.85	0.9	0.95	1.0	1.05	1.1	1.15
package height [m]	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1

Table 7.1: Cylindrical Sheet modeled Reactor Parameters, with package 1 being the innermost , and 10 being the outermost

reference being the excitation voltage at an angle of $0°$.

7.3 Reactor Model

The result of tuning a reactor to minimize circulating currents using the cylindrical sheets modeling method, with parameters listed in table 7.1. The vector provided in Eqn. 7.2 is the relatively balanced current vector for a reactor of 10 packages represented as cylindrical sheets, with the resistances being massaged to approximate what they would be if calculated using the conductor diameter and number of turns so that the resistances measured at the terminal are similar to that provided in the testing reports of the reactor being modeled. The summing of the current vector, as in Eqn. 7.3, gives the total current of the device as if measured at the terminals, with Eqn. 7.4 being the total impedance of the reactor as if measured across the terminals.

$$L = \begin{bmatrix} 0.916 & 0.803 & 0.726 & 0.672 & 0.632 & 0.603 & 0.582 & 0.567 & 0.557 & 0.552 \\ 0.803 & 0.831 & 0.750 & 0.693 & 0.652 & 0.622 & 0.600 & 0.584 & 0.574 & 0.569 \\ 0.726 & 0.750 & 0.792 & 0.731 & 0.687 & 0.655 & 0.632 & 0.615 & 0.604 & 0.599 \\ 0.672 & 0.693 & 0.731 & 0.783 & 0.735 & 0.700 & 0.675 & 0.657 & 0.645 & 0.639 \\ 0.632 & 0.652 & 0.687 & 0.735 & 0.795 & 0.757 & 0.729 & 0.709 & 0.696 & 0.689 \\ 0.603 & 0.622 & 0.655 & 0.700 & 0.757 & 0.824 & 0.793 & 0.771 & 0.757 & 0.749 \\ 0.582 & 0.600 & 0.632 & 0.675 & 0.729 & 0.793 & 0.869 & 0.844 & 0.828 & 0.820 \\ 0.567 & 0.584 & 0.615 & 0.657 & 0.709 & 0.771 & 0.844 & 0.929 & 0.911 & 0.901 \\ 0.557 & 0.574 & 0.604 & 0.645 & 0.696 & 0.757 & 0.828 & 0.911 & 1.006 & 0.994 \\ 0.552 & 0.569 & 0.599 & 0.639 & 0.689 & 0.749 & 0.820 & 0.901 & 0.994 & 1.104 \end{bmatrix} \tag{7.1}$$

$$I = \begin{bmatrix} 1.040E-05-j6.064E-04 \\ 2.738E-06-j5.571E-04 \\ 7.349E-06-j5.127E-04 \\ -1.051E-06-j4.430E-04 \\ 4.124E-06-j3.904E-04 \\ -2.826E-06-j3.238E-04 \\ -2.134E-06-j2.758E-04 \\ -2.455E-06-j2.411E-04 \\ -2.132E-06-j2.208E-04 \\ -1.028E-06-j2.138E-04 \end{bmatrix} = \begin{bmatrix} 6.065E-04\angle -89.017^\circ \\ 5.571E-04\angle -89.718^\circ \\ 5.127E-04\angle -89.179^\circ \\ 4.430E-04\angle -90.136^\circ \\ 3.905E-04\angle -89.395^\circ \\ 3.238E-04\angle -90.500^\circ \\ 2.758E-04\angle -90.443^\circ \\ 2.411E-04\angle -90.583^\circ \\ 2.208E-04\angle -90.553^\circ \\ 2.138E-04\angle -90.275^\circ \end{bmatrix} \tag{7.2}$$

$$I_T = 1.299E-05-j3.785E-03 = 3.785E-03\angle -89.803^\circ \tag{7.3}$$

$$Z_T = 9.065E-01+j2.642E+02 = 2.642E+02\angle 89.803^\circ \tag{7.4}$$

Looking at the coupling coefficient matrix, Eqn. 7.5, for the model reactor inductance matrix in Eqn. 7.6. We see there is high coupling between adjacent layers, with the coupling only dropping below 75% after the 4th element from the diagonal. This "tight coupling" between the reactor elements shows one of the reasons why reactor design before

$$K = \begin{bmatrix} 1.000 & 0.921 & 0.853 & 0.793 & 0.741 & 0.694 & 0.652 & 0.614 & 0.580 & 0.549 \\ 0.921 & 1.000 & 0.925 & 0.860 & 0.802 & 0.752 & 0.706 & 0.665 & 0.628 & 0.594 \\ 0.853 & 0.925 & 1.000 & 0.929 & 0.866 & 0.811 & 0.761 & 0.717 & 0.677 & 0.640 \\ 0.793 & 0.860 & 0.929 & 1.000 & 0.932 & 0.872 & 0.818 & 0.770 & 0.727 & 0.688 \\ 0.741 & 0.802 & 0.866 & 0.932 & 1.000 & 0.935 & 0.877 & 0.825 & 0.778 & 0.736 \\ 0.694 & 0.752 & 0.811 & 0.872 & 0.935 & 1.000 & 0.937 & 0.881 & 0.831 & 0.786 \\ 0.652 & 0.706 & 0.761 & 0.818 & 0.877 & 0.937 & 1.000 & 0.939 & 0.885 & 0.837 \\ 0.614 & 0.665 & 0.717 & 0.770 & 0.825 & 0.881 & 0.939 & 1.000 & 0.942 & 0.889 \\ 0.580 & 0.628 & 0.677 & 0.727 & 0.778 & 0.831 & 0.885 & 0.942 & 1.000 & 0.944 \\ 0.549 & 0.594 & 0.640 & 0.688 & 0.736 & 0.786 & 0.837 & 0.889 & 0.944 & 1.000 \end{bmatrix} \tag{7.5}$$

7.3.1 Single Faulted Turn

As discussed in the faults modeling section, the faulted reactor is modeled using a perturbed matrix to represent the loss of a turn due to the fault.

In the results below, Eqn. 7.7 and Eqn. 7.8 are the prefault current values, the reactor has a fault placed at the center of the $1st$ (innermost) package, Comparing them to the results in Eqn. 7.10 and

Eqn. 7.11, we can see a change in the real component of the 1st element of the current vector, but no impact on the other elements, and an imperceptible change in total current: prefault Eqn. 7.8 vs faulted Eqn. 7.11. Appendix C contains more iterations, where the fault is moved around each layer and effects are calculated.

Equation 7.6 is the prefault reactor inductance matrix, with the

$$L = \begin{bmatrix} 0.916 & 0.803 & 0.726 & 0.672 & 0.632 & 0.603 & 0.582 & 0.567 & 0.557 & 0.552 \\ 0.803 & 0.831 & 0.750 & 0.693 & 0.652 & 0.622 & 0.600 & 0.584 & 0.574 & 0.569 \\ 0.726 & 0.750 & 0.792 & 0.731 & 0.687 & 0.655 & 0.632 & 0.615 & 0.604 & 0.599 \\ 0.672 & 0.693 & 0.731 & 0.783 & 0.735 & 0.700 & 0.675 & 0.657 & 0.645 & 0.639 \\ 0.632 & 0.652 & 0.687 & 0.735 & 0.795 & 0.757 & 0.729 & 0.709 & 0.696 & 0.689 \\ 0.603 & 0.622 & 0.655 & 0.700 & 0.757 & 0.824 & 0.793 & 0.771 & 0.757 & 0.749 \\ 0.582 & 0.600 & 0.632 & 0.675 & 0.729 & 0.793 & 0.869 & 0.844 & 0.828 & 0.820 \\ 0.567 & 0.584 & 0.615 & 0.657 & 0.709 & 0.771 & 0.844 & 0.929 & 0.911 & 0.901 \\ 0.557 & 0.574 & 0.604 & 0.645 & 0.696 & 0.757 & 0.828 & 0.911 & 1.006 & 0.994 \\ 0.552 & 0.569 & 0.599 & 0.639 & 0.689 & 0.749 & 0.820 & 0.901 & 0.994 & 1.104 \end{bmatrix} \tag{7.6}$$

$$I = \begin{bmatrix} 3.615E+01\angle -89.733^\circ \\ 3.304E+01\angle -90.028^\circ \\ 3.064E+01\angle -89.926^\circ \\ 2.621E+01\angle -90.080^\circ \\ 2.337E+01\angle -89.940^\circ \\ 1.918E+01\angle -90.096^\circ \\ 1.642E+01\angle -90.054^\circ \\ 1.434E+01\angle -90.072^\circ \\ 1.313E+01\angle -90.067^\circ \\ 1.273E+01\angle -90.020^\circ \end{bmatrix} \tag{7.7}$$

$$I_T = 9.404E-02 - j2.252E+02 = 2.252E+02\angle -89.976^\circ \tag{7.8}$$

$$Z_T = 1.854E-06 + j4.440E-03 = 4.440E-03\angle 89.976^\circ \tag{7.9}$$

$$I_p = \begin{bmatrix} 3.615E+01\angle -89.732^\circ \\ 3.304E+01\angle -90.029^\circ \\ 3.064E+01\angle -89.926^\circ \\ 2.621E+01\angle -90.080^\circ \\ 2.337E+01\angle -89.940^\circ \\ 1.918E+01\angle -90.096^\circ \\ 1.642E+01\angle -90.054^\circ \\ 1.434E+01\angle -90.072^\circ \\ 1.313E+01\angle -90.067^\circ \\ 1.273E+01\angle -90.020^\circ \\ 4.346E+01\angle -179.961^\circ \end{bmatrix} \tag{7.10}$$

$$I_{pT} = 9.409E-02 - j2.252E+02 = 2.252E+02\angle -89.976^\circ \tag{7.11}$$

$$Z_{pT} = 1.855E-06 + j4.440E-03 = 4.440E-03\angle 89.976^\circ \tag{7.12}$$

With the fault occurring in the first layer of the reaactor, we see a minor change in magnitude the faulted current vector, Eqn. 7.10, and the prefault vector Eqn. 7.7. There is a significant current in the faulted turn, shown in the 11th element of Eqn. 7.10, the resistance of the faulted turn is $8.732\times10^{-4}\Omega$, so the power dissipated by the turn is only around $1.5\,[W]$ and as seen in the difference in the total current Eqn. 7.11, there is (practically) no noticeable change in either the magnitude or the phase angle.

7.3.2 Single Fault Position in a Model Reactor

Depending where the fault occurs in the reactor layer will have a different impact on the change in current. With 7.13 being the total current for prefault conditions, the greatest impact on the change in current is when the fault occurs in the center ($fz = 0$) of the layer, as seen in Eqn. 7.15. It can be seen that the effects of the fault being at the top Eqn. 7.16 or the bottom Eqn. 7.14 of the reactor ($fz = \pm0.5$) is the same. moving the fault from the innermost Eqn. 7.15 to the outermost Eqn. 7.17 layer, likewise has a greater impact, since the faulted loop will have a greater self-inductance, and will link more flux in it's enclosed surface area.

Equation 7.13 is the total current for prefault conditions:

$$I_{pre\,F} = 0.09404491 - j225.21319038 = 225.21321001\angle -89.976^\circ \tag{7.13}$$

Equation 7.14 is the total current for fault at $-0.5\,[m]$ of reactor layer height (bottom), in layer 0

(innermost):

$$I_{pT\ 0,fz=-0.5} = 1.86279697 - j229.97717007 = 229.98471420\angle -89.536° \tag{7.14}$$

Equation 7.15 is the total current for fault at $0\,[m]$ of reactor layer height (middle), in layer 0:

$$I_{pT\ 0,fz=0} = 6.90653399 - j242.15712227 = 242.25559246\angle -88.366° \tag{7.15}$$

Equation 7.16 is the total current for fault at $0.5\,[m]$ of reactor layer height (top), in layer 0:

$$I_{pT\ 0,fz=0.5} = 1.86279697 - j229.97717007 = 229.98471420\angle -89.536° \tag{7.16}$$

Equation 7.17 is total current for fault at $0\,[m]$ of reactor layer height, in layer 9 (outermost):

$$I_{pT\ 9,fz=0} = 6.15407727 - j249.13961819 = 249.21561352\angle -88.585° \tag{7.17}$$

The effect of fault locations for each layer of the model reactor can be found in appendix C.1.

7.3.3 Multiple faulted turns

Continuing with a selection of faults on the model reactor described in table 7.1, looking at the total current of the model reactor, with varying numbers of faulted turns, we can get a feel for how many faults it takes to see any noticeable change at the terminals: The results in Eqn. C.32 through Eqn. C.38 show the effects of sequential faults, i.e. one on-top of the other as though a more loops are getting shorted die to thermal breakdown on insulation. We can see there is a negligible difference going from 2 faults to 4 has a negligible impact on the total current, with the imaginary part of the total current not seeing a change until 6 turns are faulted.

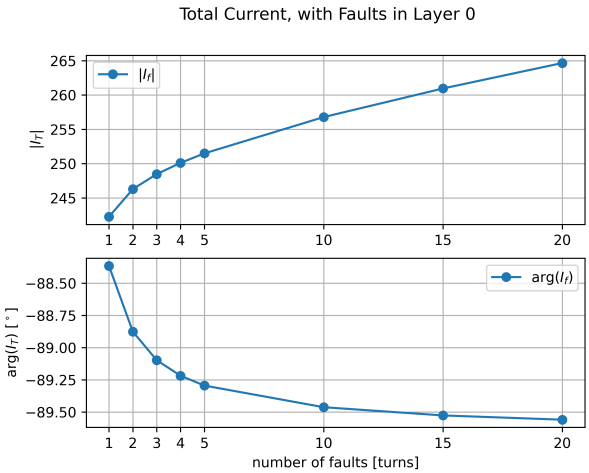


Figure 7.1

Equation C.32 is the total current for 2 faults in layer 0:

$$I_{pT\ 0,nf=2} = 4.82852335 - j246.24253931 = 246.28987556\angle -88.877° \tag{7.18}$$

Equation C.33 is the total current for 10 faults in layer 0:

$$I_{pT\ 0,nf=10} = 2.40648604 - j256.77246058 = 256.78373720\angle -89.463^\circ \tag{7.19}$$

Equation C.34 is the total current for 20 faults in layer 0:

$$I_{pT\ 0,nf=20} = 2.03295420 - j264.65137104 = 264.65917913\angle -89.560^\circ \tag{7.20}$$

Equation C.35 is the total current for 40 faults in layer 0:

$$I_{pT\ 0,nf=40} = 1.94022750 - j277.96406199 = 277.97083343\angle -89.600^\circ \tag{7.21}$$

Equation C.36 is the total current for 100 faults in layer 0:

$$I_{pT\ 0,nf=100} = 2.56847638 - j319.50374981 = 319.51407358\angle -89.539^\circ \tag{7.22}$$

Equation C.37 is the total current for 150 faults in layer 0:

$$I_{pT\ 0,nf=150} = 3.62243974 - j363.82962825 = 363.84766106\angle -89.430^\circ \tag{7.23}$$

Equation C.38 is the total current for 200 faults in layer 0:

$$I_{pT\ 0,nf=200} = 5.35972542 - j423.66196115 = 423.69586260\angle -89.275^\circ \tag{7.24}$$

See appendix C.2 for the fault progressions in each layer other than 0 (the innermost).

Chapter 8

Summary, Conclusions, and Future Work

This chapter presents the summary of the thesis, conclusions drawn from the research and results, and future work needed to produce useful insights into ACRs.

8.1 Summary

This thesis is the summary of the theory and methods used to model prefault and faulted reactors. Developed performance metrics, and defined a ”well behaved” reactor as one with unequal currents, but approximately in-phase, to minimize circulating currents. The method of describing faults is intuitive, with an analogy to explain the behavior of a fault. The method developed to evaluate the effects of faults in air-core reactors is an intuitive and computationally efficient method to evaluate the changes in the reactor die to the fault. Finally, the results presenting the effects of faults in a model similar to a physical air-core reactor show the effects of faults aren’t practically observable, and doesn’t present a method of reactor protection.

8.2 Conclusions

As a result of the work performed, and the experience gained through the course of the research, this section presents conclusions regarding the modeling and design of ACRs. With the small-scale reactor designs used to test the effectiveness of the reactor and fault modeling methods, there are questions regarding the influence of capacitance in models as layer turn counts increase to around 1000 turns or more.

Reactors are challenging to design, Improper turns in layers will cause circulating currents, causing large losses. The heat from large losses will degrade the insulation and encapsulation material. The optimization of the geometry, turn count, and package layer count to minimize circulating currents and unnecessary heating Because reactors are relatively simple devices to build, manufacturers keep details such as turn counts and layer radii to themselves in order to protect their investments. However, as demonstrated in the modeling of faults in reactor models similar to those deployed in industry, a reactor in a faulted state is difficult to detect until the fault has progressed to a significant number of turns.

The most likely fault to occur is between turns in series, that have a large ΔV compared to those turns in parallel layers. In the event of a turn-to-turn fault, the best case scenario is the connection opening up immediately due to the heat from the current in the closed loop The detection of a small number of faults before it progresses too far is practically impossible when the only inputs to the detection algorithm are measurements from the terminals of the reactor.

8.3 Future Work

The methods and results presented in this thesis are first-steps to thoroughly understanding the effects of faults in ACRs. More physical testing and comparisons to simulations are required to determine the effects of faults in larger models.

A shortcoming of the modeling methods used in this thesis is the purely inductive modeling method, where the turn-to-turn, layer-to-layer, and reactor-to-ground capacitance was ignored. Implementing a capacitance model that can be super-imposed on either the impedance matrix, or a separate analysis technique applied as a correction factor to the current vector.

The design of the reactor, i.e. the ”tuning”, is necessary to minimize losses from circulating currents. Determine more computationally efficient method of designing a reactor, or develop a minimally iterative approach.

Explore thermal optimization to minimize losses, and methods to balance the mass of the reactor with minimal losses to reduce material requirements. Thermal optimization methods are presented by Yuan et. al. [9]. But, the use of finite-element modeling software limits the use of the methods with the modeling methods presented in this thesis.

The work presented was performed with the goal of understanding the effects of turn-to-turn faults, cross-layer faults were ignored due to a high-level analysis performed that determined the effects of a cross-layer fault would be either negligible, or the result of a more serious turn-to-turn fault. Due to the difference in voltage across layers being relatively low, and based on the fault cases described by Mohammad et. al. in [13], the primary focus of this work was turn-to-turn faults. To properly model the effects of cross-layer faults would require a robust method of modeling the different network topology

possibilities, which would lend itself to the problem of a fault that propagates, or cascades through the package. Greater knowledge of the materials used in the construction of reactors, and the occurrence of insulation or encapsulation defects would be needed to adequately tackle those problems.

Bibliography

[1] K. Damron, "Practical considerations and experiences protecting 230kV shunt air-core reactors banks," in 43rd Annual Western Protective Relay Conference, Spokane, WA, pp. 18–20, 2016.

[2] A. H. Haziah, Transients in Reactors for Power Systems Compensation. Doctroal thesis, Cardiff University, 2012.

[3] C. Paul, Inductance: Loop and Partial. Hoboken, NJ, USA: Wiley, 2010.

[4] T. H. Fawzi and P. E. Burke, "The accurate computation of self and mutual inductances of circular coils," IEEE Transactions on Power Apparatus and Systems, vol. PAS-97, no. 2, pp. 464–468, 1978.

[5] A. A. Dahab, P. E. Burke, and T. H. Fawzi, "A complete model of a single layer air-cored reactor for impulse voltage distribution," IEEE Transactions on Power Delivery, vol. 3, no. 4, pp. 1745–1753, 1988.

[6] P. Burke and T. Fawzi, "Effect of eddy losses on the design and modelling of air-cored reactors," IEEE Transactions on Magnetics, vol. 27, no. 6, pp. 5001–5003, 1991.

[7] H. B. Zaninelli and E. C. Bortoni, "Optimized modeling process for air core reactors using finite element analysis," in 2021 IEEE Power & Energy Society General Meeting (PESGM), pp. 1–5, 2021.

[8] K. Nurminen, Thermal modeling and evaluation of harmonic effects on a dry-type air-core reactor. Doctoral thesis, Teknillinen korkeakoulu, 2008.

[9] F. Yuan, Z. Yuan, L. Chen, Y. Wang, J. Liu, J. He, and Y. Pan, "Thermal and electromagnetic combined optimization design of dry type air core reactor," Energies, vol. 10, no. 12, 2017.

[10] T. Fiorentin, L. Lopes, O. Silva, and A. Lenzi, "Vibroacoustic models of air-core reactors," International Journal of Acoustics and Vibrations, vol. 21, pp. 453–461, 12 2016.

[11] M. Faridi, V. Nabaei, S. A. Mousavi, and M. Mohammadi, "Modeling of continuously transposed cable in power transformer for fault analysis based on FEM," in 2009 International Conference on Electrical Machines and Systems, pp. 1–4, 2009.

[12] D. Geissler and T. Leibfried, "Mechanical breakdown of aged insulating paper around continuously transposed conductors for power transformers under the influence of short-circuit forces - analysis by numerical simulations," in 2015 IEEE Electrical Insulation Conference (EIC), pp. 401–406, 2015.

[13] A. I. Mohammad, T. Mort, J. Jeter, A. Hoth, J. England, B. K. Johnson, N. Fischer, and K. Damron, "Turn-to-turn fault protection for dry-type shunt reactors," in 2018 IEEE/PES Transmission and Distribution Conference and Exposition (T&D), pp. 1–5, 2018.

[14] F. K. Basha and M. Thompson, "Practical EHV reactor protection," in 2013 66th Annual Conference for Protective Relay Engineers, pp. 408–419, 2013.

[15] R. Chowdhury, N. Fischer, D. Taylor, D. Caverly, and A. B. Dehkordi, "A fresh look at practical shunt reactor protection," in 49th Annual Western Protective Relay Conference, 2022.

[16] C. Zhigang, Q. Guochao, H. Changjin, L. Yi, Y. Chao, and O. Yangyong, "Development and test of distributed current monitoring device for dry type air core reactor," in 2020 IEEE International Conference on High Voltage Engineering and Application (ICHVE), pp. 1–4, 2020.

[17] A. Guzman, Transformer Internal Fault Model for Protection Analysis. Masters thesis, The University of Idaho, College of Graduate Studies, 2002.

[18] G. Strang, Linear Algebra and its Applications, 3rd Edition. Toronto, On. Canada: Brooks/Cole, 1988.

[19] H. B. Dwight, Tables if Integrals and Other Mathematical Data, 4th Edition. New York, NY, USA: Macmillan, 1966.

[20] N. D. Tleis, Power Systems Modelling and Fault Analysis: Theory and Practice. New York, NY, USA: Newnes, 2008.

[21] J. Lammeraner and M. Stafl, Eddy currents. London, UK: Prague, publisher not identified T.L., 1966.

[22] S. V. Kulkarni and S. A. Khaparde, Transformer Engineering: Design and Practice. New York, NY, USA: Dekker, 2005.

[23] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane,

J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” Nature, vol. 585, pp. 357–362, Sept. 2020.

[24] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” Nature Methods, vol. 17, pp. 261–272, 2020.

[25] J. D. Hunter, “Matplotlib: A 2d graphics environment,” Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007.

Appendix A

Derivations

Flux internal to a cylindrical conductor

This appendix section is the development of equation 2.11. The result is an approximation of the internal wire (cylindrical conductor) inductance for a given loop of wire, where the radius of the wire is significantly larger than the wire: $r_{wire} << r_{loop}$.

Permeability of free space: $\mu_0 = 4\pi \times 10^{-7} \approx 1.257 \times 10^{-6}$

Ampere's Law describing the magnetic filed inside a conductor: $\mathbf{B}_{int}(r_x) = \dfrac{\mu_0 \, I}{2\pi r_{wire}^2}$

The fraction of the flux linked by a current: $\dfrac{\pi r_x^2}{\pi r_{wire}^2}$

$$da = (r_{loop} + r_x) d\theta \, dr_x \tag{A.1}$$

$$\psi_{internal} = \int_{r_x=0}^{r_{loop}-r_{wire}} \int_{\theta=0}^{2\pi} \left(\frac{\pi r_x^2}{\pi r_{wire}^2} \right) \mathbf{B}_{int}(r_x)(r_{loop} + r_x) \, d\theta \, dr_x \tag{A.2}$$

r_{loop} is the distance from the origin to the center of the conductor.

$$
\begin{aligned}
\psi_{internal} &= \int_{r_x=0}^{r_{loop}-r_{wire}} \int_{\theta=0}^{2\pi} \left(\frac{\pi r_x^2}{\pi r_{wire}^2}\right)\left(\frac{\mu_0\, I}{2\pi r_{wire}^2}\right)(r_{loop}+r_x)\, d\theta\, dr_x \\
&= \int_{r_x=0}^{r_{loop}-r_{wire}} \left(\frac{\pi r_x^2}{\pi r_{wire}^2}\right)\left(\frac{\mu_0\, I}{2\pi r_{wire}^2}\right)(r_{loop}+r_x)\, dr_x \int_{\theta=0}^{2\pi} 1\, d\theta \\
&= 2\pi \int_{r_x=0}^{r_{loop}-r_{wire}} \left(\frac{r_x^2}{r_{wire}^2}\right)\left(\frac{\mu_0\, I}{2\pi r_{wire}^2}\right)(r_{loop}+r_x)\, dr_x \\
&= \frac{2\pi\, \mu_0\, I}{2\pi\, r_{wire}^4} \int_{r_x=0}^{r_{wire}} r_x^3 \Big(r_{loop}+r_x\Big) dr_x \\
&= \frac{\mu_0\, I}{r_{wire}^4} \int_{r_x=0}^{r_{wire}} \Big(r_x^3\, r_{loop} + r_x^4\Big) dr_x \\
&= \frac{\mu_0\, I}{r_{wire}^4} \left(\frac{r_{wire}^4}{4} r_{loop} + \frac{r_{wire}^5}{5}\right) \\
&= \mu_0\, I \left(\frac{r_{loop}}{4} + \frac{r_{wire}}{5}\right)
\end{aligned}
\tag{A.3}
$$

Now, using the identity $L = \psi / I$;

$$
L_{internal} = \mu_0 \left(\frac{r_{loop}}{4} + \frac{r_{wire}}{5}\right) \tag{A.4}
$$

Appendix B

Testing Designs

The information in this appendix are the oscilloscope screen captures from the testing of the small-scale test reactors.

B.1 Testing Data

In this section are the Oscilloscope screen captures of the reactor modules and assembled reactors in normal and faulted condition. Channel 3, the blue curve, is the terminal excitation voltage V_{term}. Channel 4, the green curve is the measured voltage across the current sensing resistor R_i.

B.1.1 Faulted Turn in Isolation

Prefault and faulted innermost module measurements outside of assembled test reactor:

Figure B.1: 41 Turn, $50mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 1.06V$, $f = 10kHz$, $|I_{pk-pk}| = 78.0mA$, $\Phi_{V-I} = 27.29°$

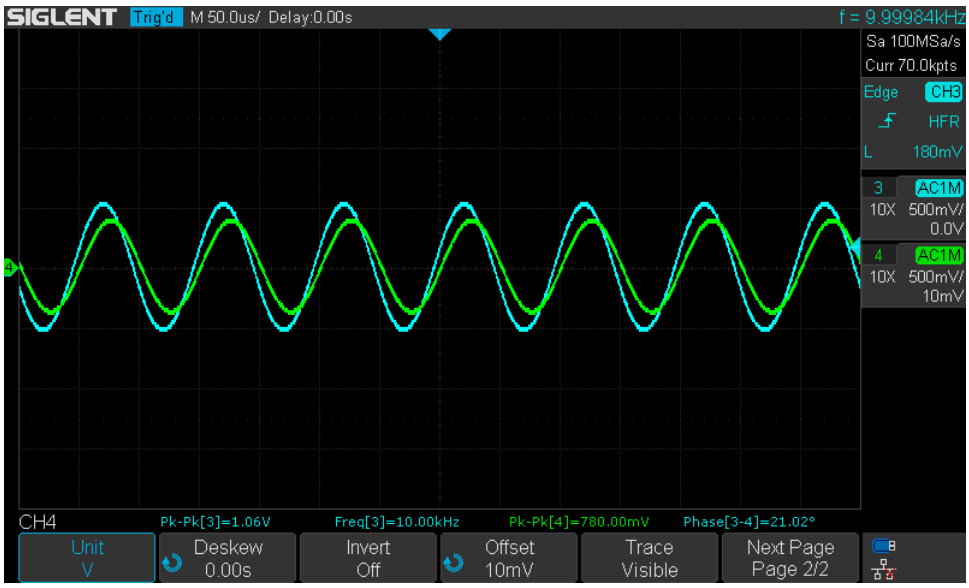


Figure B.2: 41 Turn, $50mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series, with Turn 21 shorted. $|V_{pk-pk}| = 1.06V$, $f = 10kHz$, $|I_{pk-pk}| = 78.0mA$, $\Phi_{V-I} = 21.02°$

B.1.2 Improperly Tuned Reactor Assembly

Prefault reactor assembly with all elements having 41 turns:

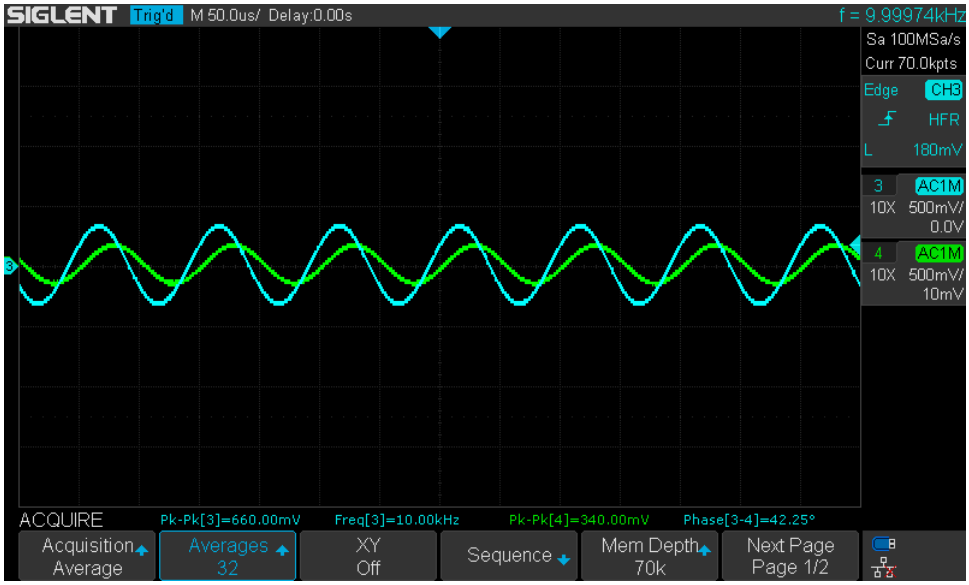


Figure B.3: 41 Turn, $50mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 660mV$, $f = 10kHz$, $|I_{pk-pk}| = 34.0mA$, $\Phi_{V-I} = 42.25°$

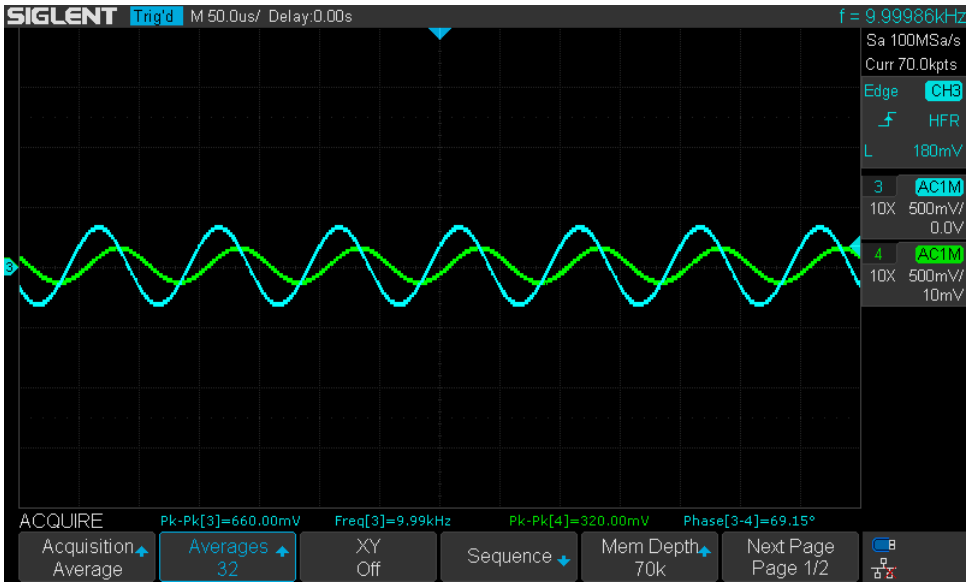


Figure B.4: 41 Turn, $54mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 660mV$, $f = 10kHz$, $|I_{pk-pk}| = 32.0mA$, $\Phi_{V-I} = 69.15°$

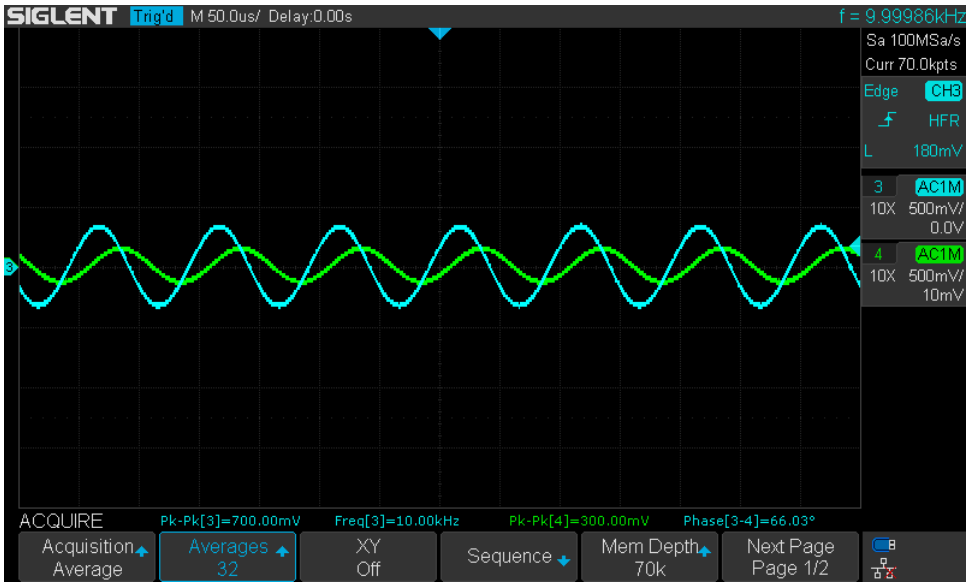


Figure B.5: 41 Turn, $58mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 700mV$, $f = 10kHz$, $|I_{pk-pk}| = 30.0mA$, $\Phi_{V-I} = 66.03°$

B.1.3 Improperly Tuned Reactor Assembly with a Fault

Faulted reactor assembly with all elements having 41 turns, and a fault in package 1 (innermost) on turn 21:

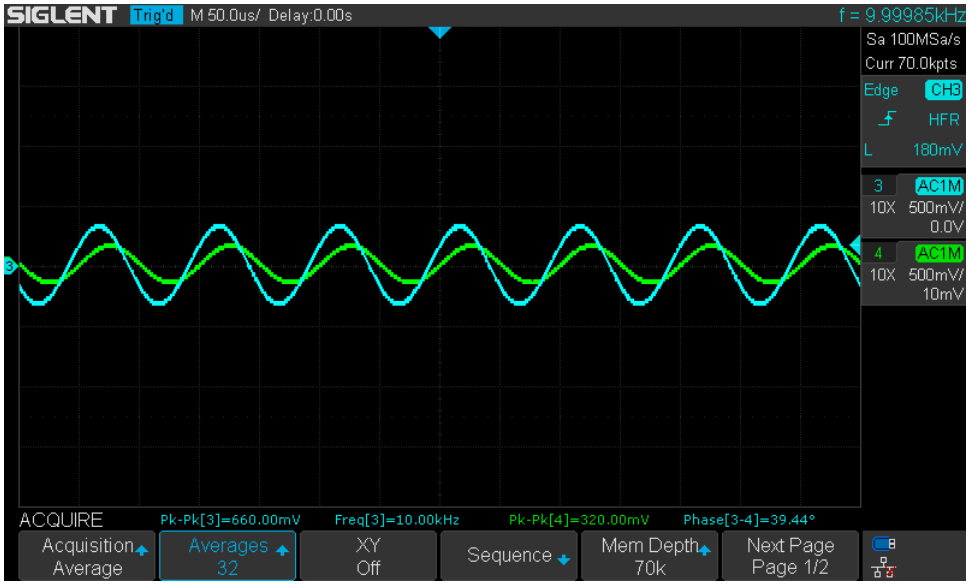


Figure B.6: 41 Turn, $50mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 660mV$, $f = 10kHz$, $|I_{pk-pk}| = 32.0mA$, $\Phi_{V-I} = 39.44°$

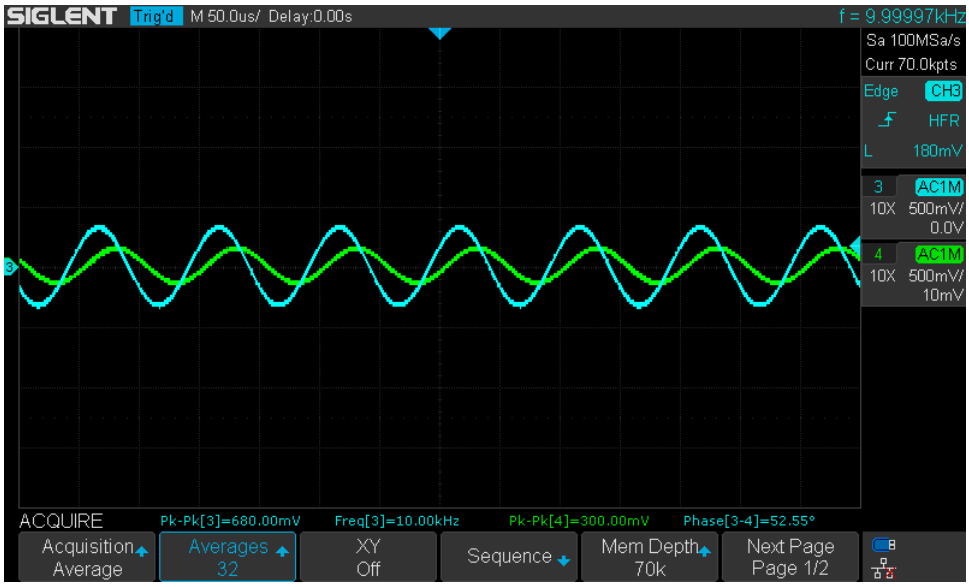


Figure B.7: 41 Turn, $54mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series.$|V_{pk-pk}| = 680mV$, $f = 10kHz$, $|I_{pk-pk}| = 30.0mA$, $\Phi_{V-I} = 52.55°$

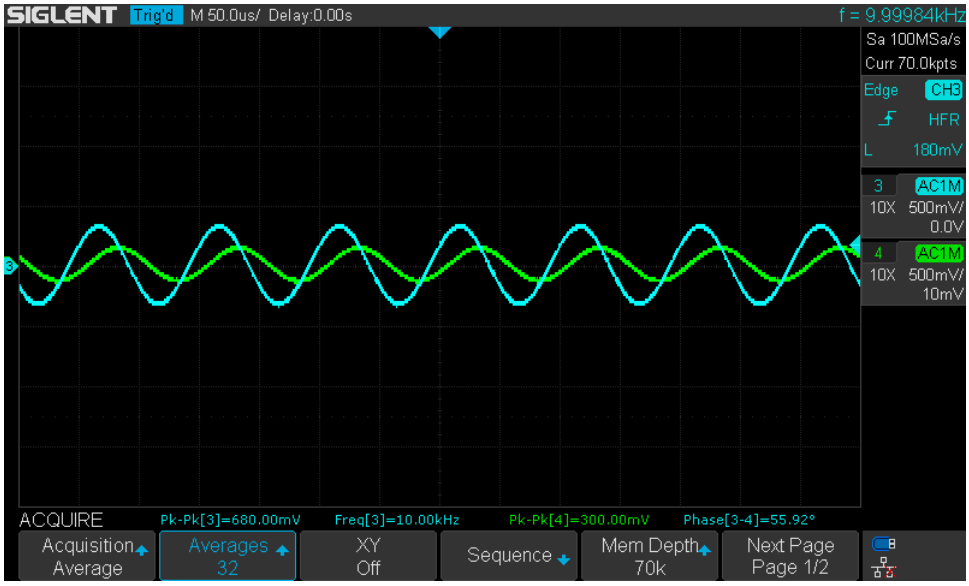


Figure B.8: 41 Turn, $58mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 680mV$, $f = 10kHz$, $|I_{pk-pk}| = 30.0mA$, $\Phi_{V-I} = 55.92°$

B.1.4 Tuned Reactor Assembly

Prefault "Tuned" reactor:

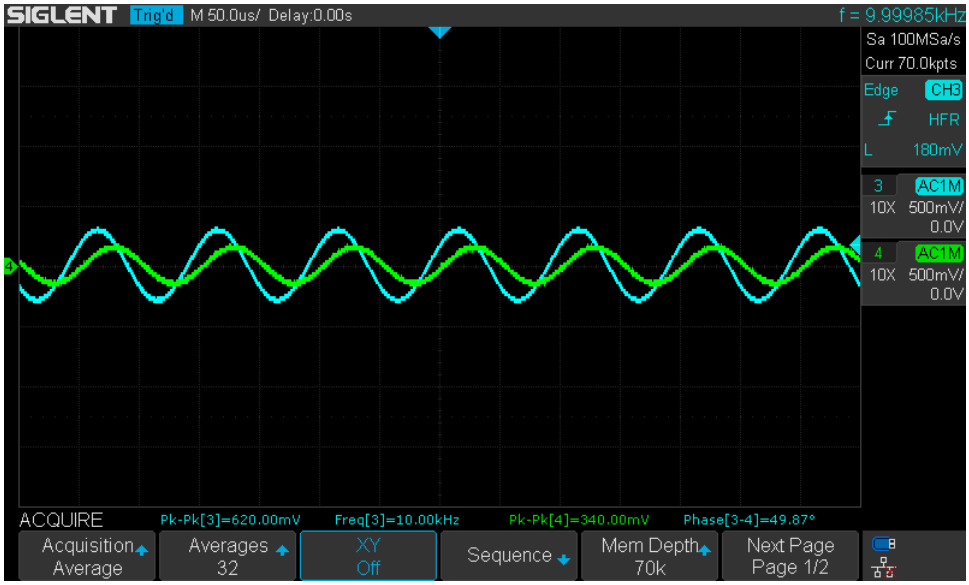


Figure B.9: 41 Turn, $50mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 620mV$, $f = 10kHz$, $|I_{pk-pk}| = 34.0mA$, $\Phi_{V-I} = 49.87°$

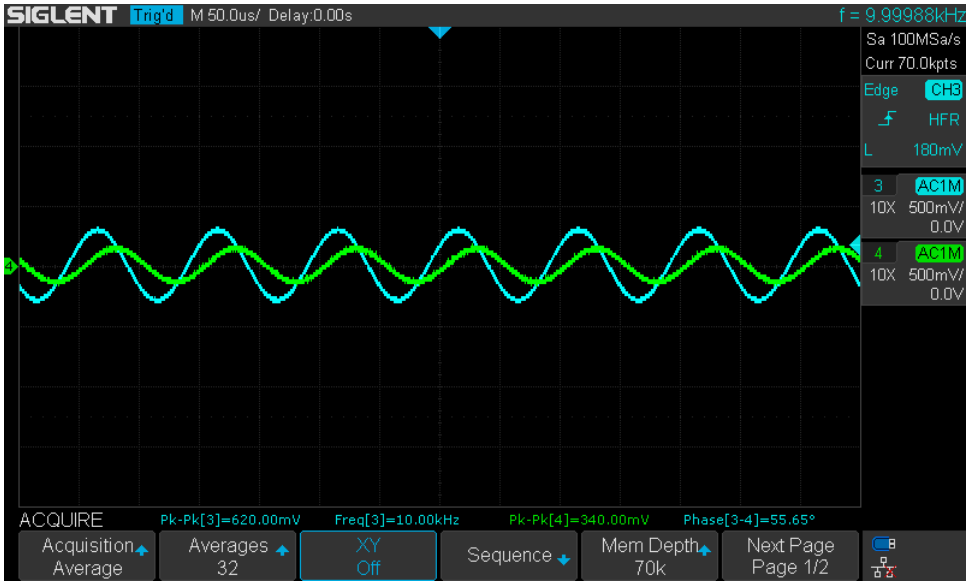


Figure B.10: 37 Turn, $54mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 620mV$, $f = 10kHz$, $|I_{pk-pk}| = 34.0mA$, $\Phi_{V-I} = 55.65°$

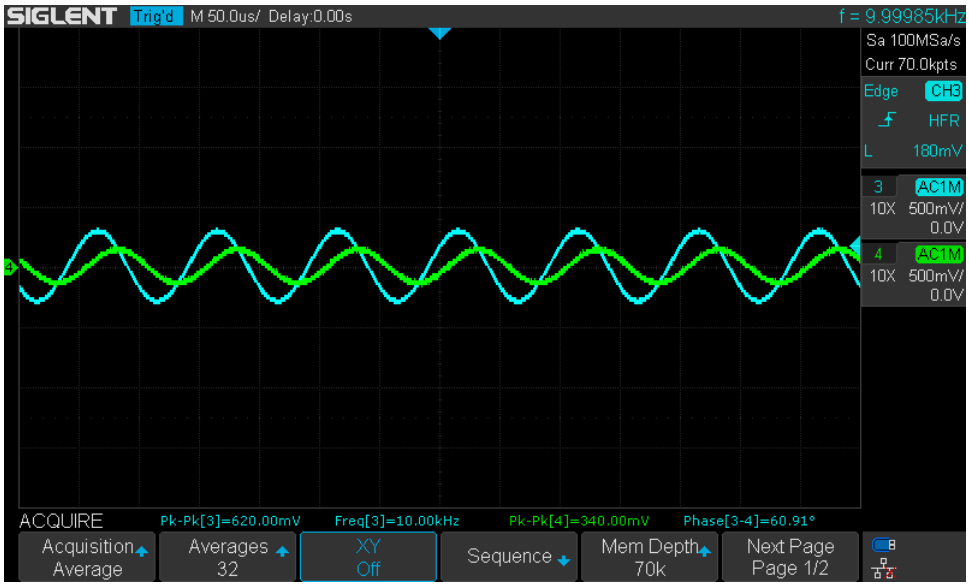


Figure B.11: 37 Turn, $58mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 620mV$, $f = 10kHz$, $|I_{pk-pk}| = 34.0mA$, $\Phi_{V-I} = 60.91°$

B.1.5 Tuned Reactor Assembly with a Fault

Faulted "Tuned" Reactor, with fault on turn 21 of the package 1 (innermost).

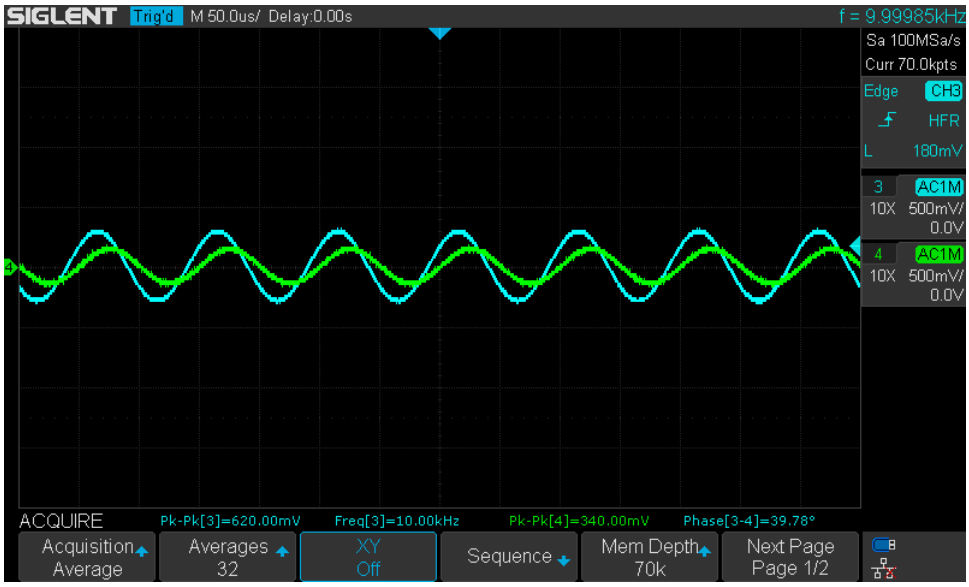


Figure B.12: 41 Turn, $50mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series, turn 21 shorted. $|V_{pk-pk}| = 620mV$, $f = 10kHz$, $|I_{pk-pk}| = 34.0mA$, $\Phi_{V-I} = 39.78°$

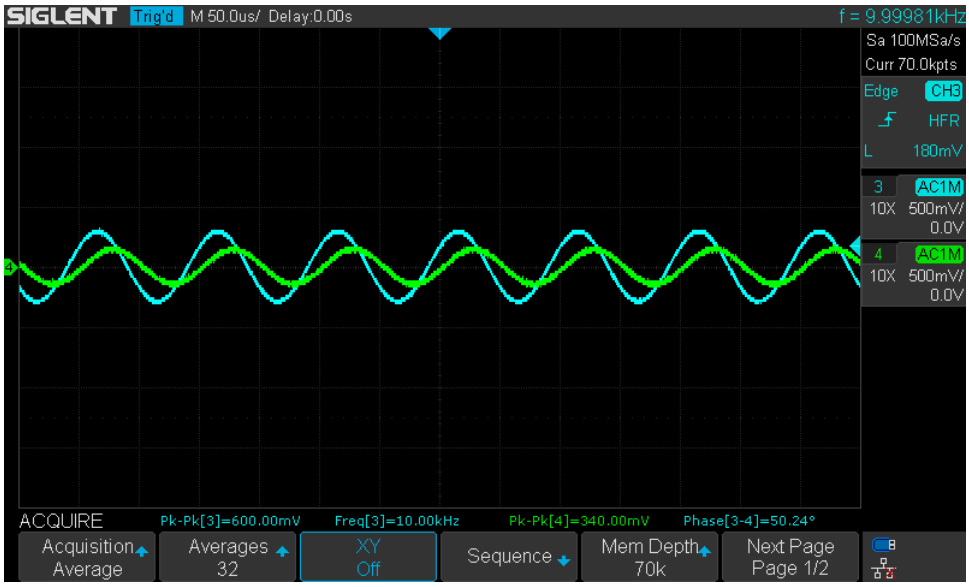


Figure B.13: 37 Turn, $54mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 600mV$, $f = 10kHz$, $|I_{pk-pk}| = 34.0mA$, $\Phi_{V-I} = 52.24°$

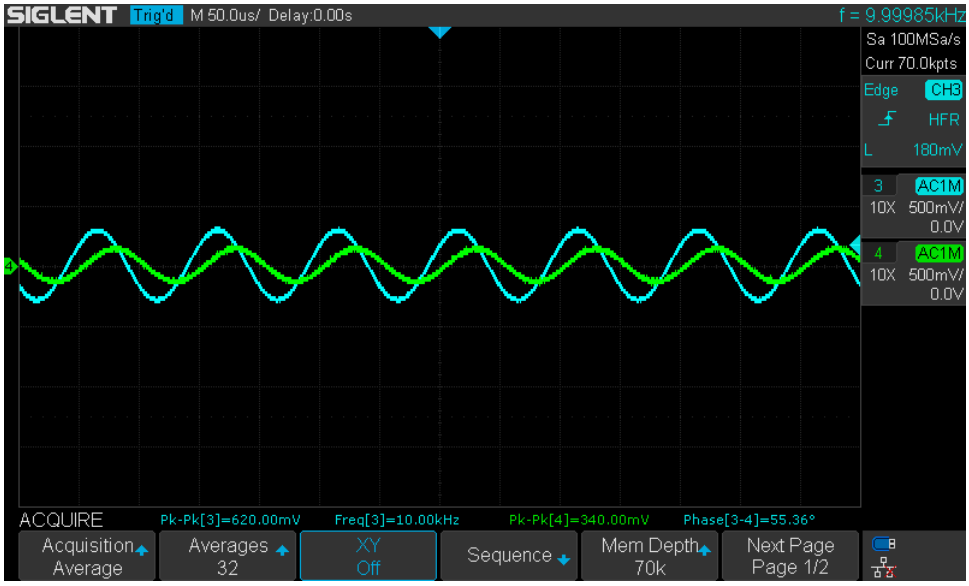


Figure B.14: 37 Turn, $58mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 620mV$, $f = 10kHz$, $|I_{pk-pk}| = 34.0mA$, $\Phi_{V-I} = 55.36°$

B.1.6 Second Test Core used in Tuned Reactor

Second test core used for the "Tuned" tests, this unit is wound counter-clockwise, where the non-tuned core is wound clockwise and is non-optimal.

Figure B.15: 41 Turn, $50mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series. $|V_{pk-pk}| = 1.06V$, $f = 10kHz$, $|I_{pk-pk}| = 78.0mA$, $\Phi_{V-I} = 27.67°$

Figure B.16: 41 Turn, $50mm$ diameter, Test Reactor Module with 10Ω Current-Sensing Resistor in-series, with turn 21 shorted. $|V_{pk-pk}| = 1.06V$, $f = 10kHz$, $|I_{pk-pk}| = 78.0mA$, $\Phi_{V-I} = 22.32°$

Appendix C

Extended Results

The sections following are extended results that can be referenced, looking at the the

C.1 Simple Behavior due to a Fault

This section presents extended results detailing the effects of a single turn fault, and how the fault effect will vary with the layer, and the position of the fault within that layer.

Current vectors here are described in the format: I_{pf}¡fault layer¿ ¡vertical position¿. With *fault layer* being the layer in which the fault occurs, and *vertical position* being the position on the layer, top or bottom are at 95% of the height. and center is at 50% of the layer height.

Equation C.1 total current for prefault conditions.

$$I_{pre\,F} = 0.09404491 - j225.21319038 = 225.21321001\angle -89.976^\circ \tag{C.1}$$

Equation C.2 total current for fault at -0.5 of reactor layer height, in layer 0.

$$I_{pT\ 0,fz=-0.5} = 1.86279697 - j229.97717007 = 229.98471420\angle -89.536^\circ \tag{C.2}$$

Equation C.3 total current for fault at 0 of reactor layer height, in layer 0.

$$I_{pT\ 0,fz=0} = 6.90653399 - j242.15712227 = 242.25559246\angle -88.366^\circ \tag{C.3}$$

Equation C.4 total current for fault at 0.5 of reactor layer height, in layer 0.

$$I_{pT\ 0,fz=0.5} = 1.86279697 - j229.97717007 = 229.98471420\angle -89.536^\circ \tag{C.4}$$

Equation C.5 total current for fault at -0.5 of reactor layer height, in layer 1.

$$I_{pT\ 1,fz=-0.5} = 2.07797265 - j230.88984468 = 230.89919521\angle -89.484^\circ \tag{C.5}$$

Equation C.6 total current for fault at 0 of reactor layer height, in layer 1.

$$I_{pT\ 1,fz=0} = 7.80868429 - j245.49622512 = 245.62038209\angle -88.178^{\circ} \tag{C.6}$$

Equation C.7 total current for fault at 0.5 of reactor layer height, in layer 1.

$$I_{pT\ 1,fz=0.5} = 2.07797265 - j230.88984468 = 230.89919521\angle -89.484^{\circ} \tag{C.7}$$

Equation C.8 total current for fault at -0.5 of reactor layer height, in layer 2.

$$I_{pT\ 2,fz=-0.5} = 2.20710496 - j231.62962077 = 231.64013584\angle -89.454^{\circ} \tag{C.8}$$

Equation C.9 total current for fault at 0 of reactor layer height, in layer 2.

$$I_{pT\ 2,fz=0} = 8.35884144 - j248.16679746 = 248.30753028\angle -88.071^{\circ} \tag{C.9}$$

Equation C.10 total current for fault at 0.5 of reactor layer height, in layer 2.

$$I_{pT\ 2,fz=0.5} = 2.20710496 - j231.62962077 = 231.64013584\angle -89.454^{\circ} \tag{C.10}$$

Equation C.11 total current for fault at -0.5 of reactor layer height, in layer 3.

$$I_{pT\ 3,fz=-0.5} = 2.25746609 - j232.17550506 = 232.18647959\angle -89.443^{\circ} \tag{C.11}$$

Equation C.12 total current for fault at 0 of reactor layer height, in layer 3.

$$I_{pT\ 3,fz=0} = 8.57978778 - j250.08596405 = 250.23309568\angle -88.035^{\circ} \tag{C.12}$$

Equation C.13 total current for fault at 0.5 of reactor layer height, in layer 3.

$$I_{pT\ 3,fz=0.5} = 2.25746609 - j232.17550506 = 232.18647959\angle -89.443^{\circ} \tag{C.13}$$

Equation C.14 total current for fault at -0.5 of reactor layer height, in layer 4.

$$I_{pT\ 4,fz=-0.5} = 2.24635895 - j232.54291650 = 232.55376614\angle -89.447^{\circ} \tag{C.14}$$

Equation C.15 total current for fault at 0 of reactor layer height, in layer 4.

$$I_{pT\ 4,fz=0} = 8.53881088 - j251.30357675 = 251.44860107\angle -88.054^{\circ} \tag{C.15}$$

Equation C.16 total current for fault at 0.5 of reactor layer height, in layer 4.

$$I_{pT\ 4,fz=0.5} = 2.24635895 - j232.54291650 = 232.55376614\angle -89.447^{\circ} \tag{C.16}$$

Equation C.17 total current for fault at -0.5 of reactor layer height, in layer 5.

$$I_{pT\ 5,fz=-0.5} = 2.18513549 - j232.73713338 = 232.74739112\angle -89.462^{\circ} \tag{C.17}$$

Equation C.18 total current for fault at 0 of reactor layer height, in layer 5.

$$I_{pT\ 5,fz=0} = 8.28227585 - j251.84696186 = 251.98311112\angle -88.116^{\circ} \tag{C.18}$$

Equation C.19 total current for fault at 0.5 of reactor layer height, in layer 5.

$$I_{pT\ 5,fz=0.5} = 2.18513549 - j232.73713338 = 232.74739112\angle -89.462^{\circ} \tag{C.19}$$

Equation C.20 total current for fault at -0.5 of reactor layer height, in layer 6.

$$I_{pT\ 6,fz=-0.5} = 2.08979724 - j232.78845689 = 232.79783701\angle -89.486^{\circ} \tag{C.20}$$

Equation C.21 total current for fault at 0 of reactor layer height, in layer 6.

$$I_{pT\ 6,fz=0} = 7.87705386 - j251.83033767 = 251.95350156\angle -88.208^{\circ} \tag{C.21}$$

Equation C.22 total current for fault at 0.5 of reactor layer height, in layer 6.

$$I_{pT\ 6,fz=0.5} = 2.08979724 - j232.78845689 = 232.79783701\angle -89.486^{\circ} \tag{C.22}$$

Equation C.23 total current for fault at -0.5 of reactor layer height, in layer 7.

$$I_{pT\ 7,fz=-0.5} = 1.97038942 - j232.71540918 = 232.72375062\angle -89.515^{\circ} \tag{C.23}$$

Equation C.24 total current for fault at 0 of reactor layer height, in layer 7.

$$I_{pT\ 7,fz=0} = 7.36671795 - j251.33168606 = 251.43962486\angle -88.321^{\circ} \tag{C.24}$$

Equation C.25 total current for fault at 0.5 of reactor layer height, in layer 7.

$$I_{pT\ 7,fz=0.5} = 1.97038942 - j232.71540918 = 232.72375062\angle -89.515^{\circ} \tag{C.25}$$

Equation C.26 total current for fault at -0.5 of reactor layer height, in layer 8.

$$I_{pT\ 8,fz=-0.5} = 1.83458886 - j232.53442418 = 232.54166109\angle -89.548^{\circ} \tag{C.26}$$

Equation C.27 total current for fault at 0 of reactor layer height, in layer 8.

$$I_{pT\ 8,fz=0} = 6.78498828 - j250.41950789 = 250.51140892\angle -88.448^{\circ} \tag{C.27}$$

Equation C.28 total current for fault at 0.5 of reactor layer height, in layer 8.

$$I_{pT\ 8,fz=0.5} = 1.83458886 - j232.53442418 = 232.54166109\angle -89.548^{\circ} \tag{C.28}$$

Equation C.29 total current for fault at -0.5 of reactor layer height, in layer 9.

$$I_{pT\ 9,fz=-0.5} = 1.68747116 - j232.25602145 = 232.26215159\angle -89.584^{\circ} \tag{C.29}$$

Equation C.30 total current for fault at 0 of reactor layer height, in layer 9.

$$I_{pT\ 9,fz=0} = 6.15407727 - j249.13961819 = 249.21561352\angle -88.585^{\circ} \tag{C.30}$$

Equation C.31 total current for fault at 0.5 of reactor layer height, in layer 9.

$$I_{pT\ 9,fz=0.5} = 1.68747116 - j232.25602145 = 232.26215159\angle -89.584^{\circ} \tag{C.31}$$

C.2 Multiple Faults in Each Layer

This section contains the total current values for each instance of $2, 10, 20, ..., 200$ faults in each layer of the sheet reactor model described in table 7.1, other than layer 0, which is in section 7.3.3 The Reactor has a terminal voltage of $13.7\,[kV]$, with the parameters given in table C.1, which is the same as table 7.1.

Package	1	2	3	4	5	6	7	8	9	10
package turns	1327	1187	1093	1029	985	956	938	929	928	935
average radius [m]	0.7	0.75	0.8	0.85	0.9	0.95	1.0	1.05	1.1	1.15
package height [m]	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1

Table C.1: Cylindrical Modeled Reactor Parameters

Layer 0 (Innermost)

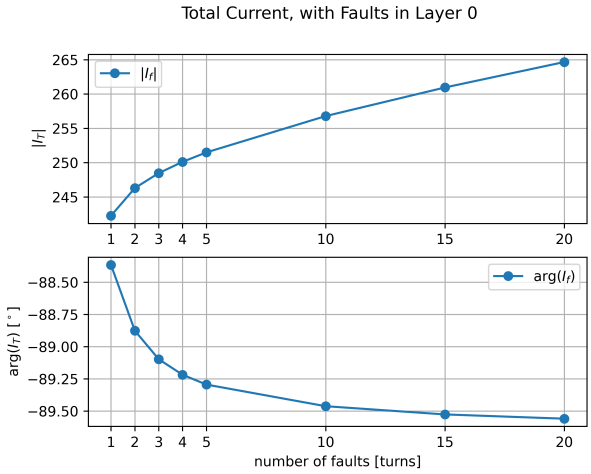


Figure C.1

Equation C.32 is the total current for 2 faults in layer 0:

$$I_{pT\ 0,nf=2} = 4.82852335 - j246.24253931 = 246.28987556\angle -88.877^\circ \tag{C.32}$$

Equation C.33 is the total current for 10 faults in layer 0:

$$I_{pT\ 0,nf=10} = 2.40648604 - j256.77246058 = 256.78373720\angle -89.463^\circ \tag{C.33}$$

Equation C.34 is the total current for 20 faults in layer 0:

$$I_{pT\ 0,nf=20} = 2.03295420 - j264.65137104 = 264.65917913\angle -89.560^\circ \tag{C.34}$$

Equation C.35 is the total current for 40 faults in layer 0:

$$I_{pT\ 0,nf=40} = 1.94022750 - j277.96406199 = 277.97083343\angle -89.600^\circ \tag{C.35}$$

Equation C.36 is the total current for 100 faults in layer 0:

$$I_{pT\ 0,nf=100} = 2.56847638 - j319.50374981 = 319.51407358\angle -89.539^\circ \tag{C.36}$$

Equation C.37 is the total current for 150 faults in layer 0:

$$I_{pT\ 0,nf=150} = 3.62243974 - j363.82962825 = 363.84766106\angle -89.430^\circ \tag{C.37}$$

Equation C.38 is the total current for 200 faults in layer 0:

$$I_{pT\ 0,nf=200} = 5.35972542 - j423.66196115 = 423.69586260\angle -89.275^\circ \tag{C.38}$$

Layer 1

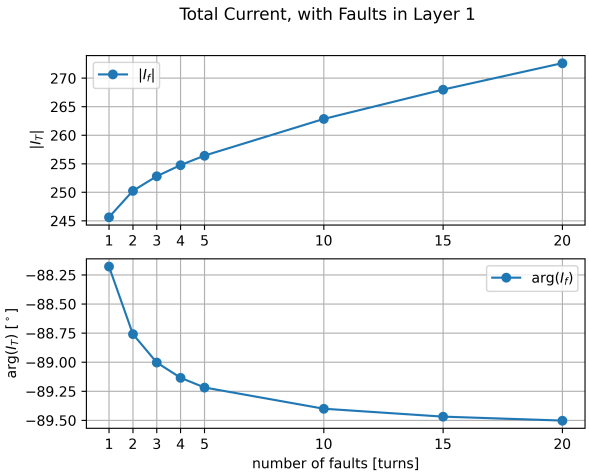


Figure C.2

Equation C.39 total current for 2 faults in layer 1

$$I_{pT\ 1,nf=2} = 5.42079829 - j250.18155687 = 250.24027744\angle -88.759^\circ \tag{C.39}$$

Equation C.40 total current for 10 faults in layer 1

$$I_{pT\ 1,nf=10} = 2.75018424 - j262.82679113 = 262.84117952\angle -89.400^\circ \tag{C.40}$$

Equation C.41 total current for 20 faults in layer 1

$$I_{pT\ 1,nf=20} = 2.36786155 - j272.60508224 = 272.61536572\angle -89.502^\circ \tag{C.41}$$

Equation C.42 total current for 40 faults in layer 1

$$I_{pT\ 1,nf=40} = 2.34982174 - j289.65515963 = 289.66469091\angle -89.535° \tag{C.42}$$

Equation C.43 total current for 100 faults in layer 1

$$I_{pT\ 1,nf=100} = 3.39550212 - j345.52246743 = 345.53915109\angle -89.437° \tag{C.43}$$

Equation C.44 total current for 150 faults in layer 1

$$I_{pT\ 1,nf=150} = 5.06771857 - j408.22734923 = 408.25880325\angle -89.289° \tag{C.44}$$

Equation C.45 total current for 200 faults in layer 1

$$I_{pT\ 1,nf=200} = 8.03209571 - j498.06723447 = 498.13199517\angle -89.076° \tag{C.45}$$

Layer 2

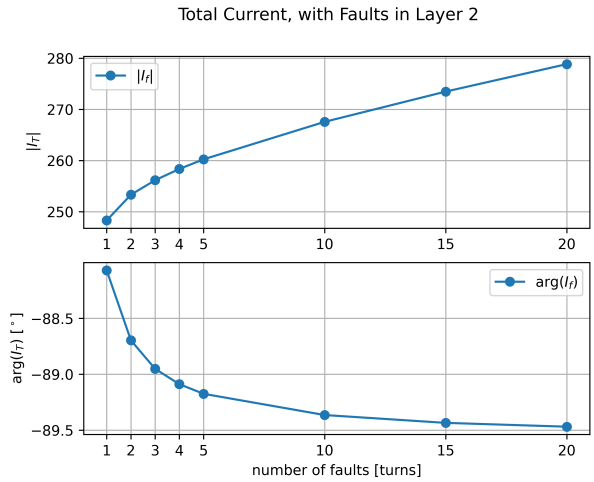


Figure C.3

Equation C.46 total current for 2 faults in layer 2

$$I_{pT\ 2,nf=2} = 5.76501140 - j253.25574484 = 253.32135254\angle -88.696° \tag{C.46}$$

Equation C.47 total current for 10 faults in layer 2

$$I_{pT\ 2,nf=10} = 2.96673282 - j267.54288817 = 267.55933644\angle -89.365° \tag{C.47}$$

Equation C.48 total current for 20 faults in layer 2

$$I_{pT\ 2,nf=20} = 2.58421523 - j278.82240312 = 278.83437853\angle -89.469° \tag{C.48}$$

Equation C.49 total current for 40 faults in layer 2

$$I_{pT\ 2,nf=40} = 2.61854137 - j298.83769840 = 298.84917056\angle -89.498^\circ \tag{C.49}$$

Equation C.50 total current for 100 faults in layer 2

$$I_{pT\ 2,nf=100} = 3.98423533 - j366.74348017 = 366.76512154\angle -89.378^\circ \tag{C.50}$$

Equation C.51 total current for 150 faults in layer 2

$$I_{pT\ 2,nf=150} = 6.24657554 - j446.78831169 = 446.83197644\angle -89.199^\circ \tag{C.51}$$

Equation C.52 total current for 200 faults in layer 2

$$I_{pT\ 2,nf=200} = 10.64439298 - j568.95530791 = 569.05487038\angle -88.928^\circ \tag{C.52}$$

Layer 3

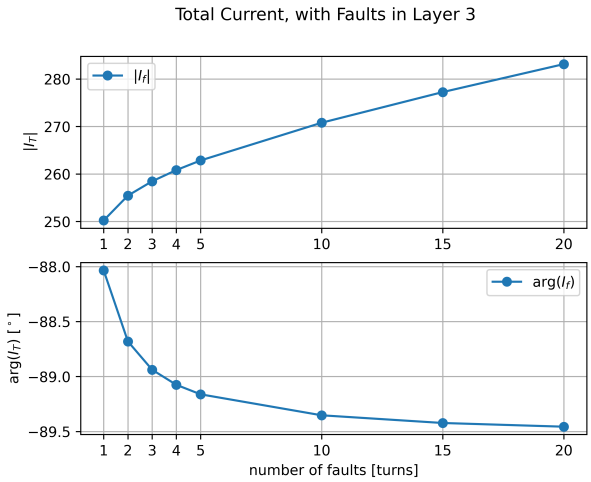


Figure C.4

Equation C.53 total current for 2 faults in layer 3

$$I_{pT\ 3,nf=2} = 5.88049490 - j255.38557002 = 255.45326303\angle -88.681^\circ \tag{C.53}$$

Equation C.54 total current for 10 faults in layer 3

$$I_{pT\ 3,nf=10} = 3.05884609 - j270.77714786 = 270.79442451\angle -89.353^\circ \tag{C.54}$$

Equation C.55 total current for 20 faults in layer 3

$$I_{pT\ 3,nf=20} = 2.68602337 - j283.12116774 = 283.13390886\angle -89.456^\circ \tag{C.55}$$

Equation C.56 total current for 40 faults in layer 3

$$I_{pT\ 3,nf=40} = 2.76192255 - j305.32543958 = 305.33793127\angle -89.482^\circ \tag{C.56}$$

Equation C.57 total current for 100 faults in layer 3

$$I_{pT\ 3,nf=100} = 4.36949668 - j382.82187243 = 382.84680815\angle -89.346^\circ \tag{C.57}$$

Equation C.58 total current for 150 faults in layer 3

$$I_{pT\ 3,nf=150} = 7.14832223 - j478.13447616 = 478.18790847\angle -89.143^\circ \tag{C.58}$$

Equation C.59 total current for 200 faults in layer 3

$$I_{pT\ 3,nf=200} = 13.04843995 - j632.40475704 = 632.53935729\angle -88.818^\circ \tag{C.59}$$

Layer 4

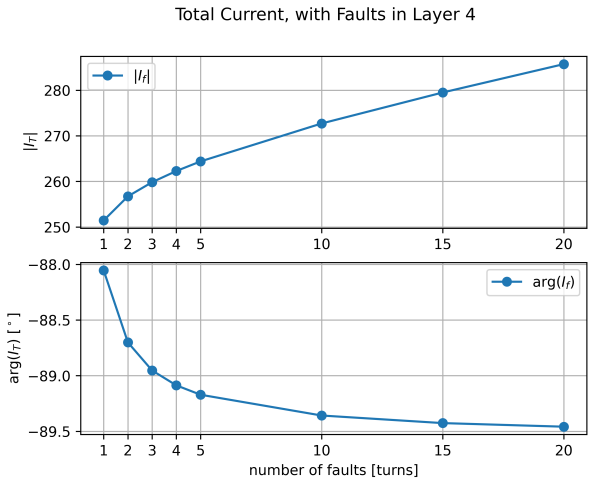


Figure C.5

Equation C.60 total current for 2 faults in layer 4

$$I_{pT\ 4,nf=2} = 5.81943354 - j256.65995039 = 256.72591599\angle -88.701^\circ \tag{C.60}$$

Equation C.61 total current for 10 faults in layer 4

$$I_{pT\ 4,nf=10} = 3.05630955 - j272.70010804 = 272.71723443\angle -89.358^\circ \tag{C.61}$$

Equation C.62 total current for 20 faults in layer 4

$$I_{pT\ 4,nf=20} = 2.70144101 - j285.72393846 = 285.73670887\angle -89.458^\circ \tag{C.62}$$

Equation C.63 total current for 40 faults in layer 4

$$I_{pT\ 4,nf=40} = 2.80831519 - j309.38394273 = 309.39668818\angle -89.480^\circ \tag{C.63}$$

Equation C.64 total current for 100 faults in layer 4

$$I_{pT\ 4,nf=100} = 4.57099012 - j393.75497758 = 393.78150836\angle -89.335^\circ \tag{C.64}$$

Equation C.65 total current for 150 faults in layer 4

$$I_{pT\ 4,nf=150} = 7.74470452 - j501.25143481 = 501.31126194\angle -89.115^\circ \tag{C.65}$$

Equation C.66 total current for 200 faults in layer 4

$$I_{pT\ 4,nf=200} = 15.04765451 - j684.66001787 = 684.82535874\angle -88.741^\circ \tag{C.66}$$

Layer 5

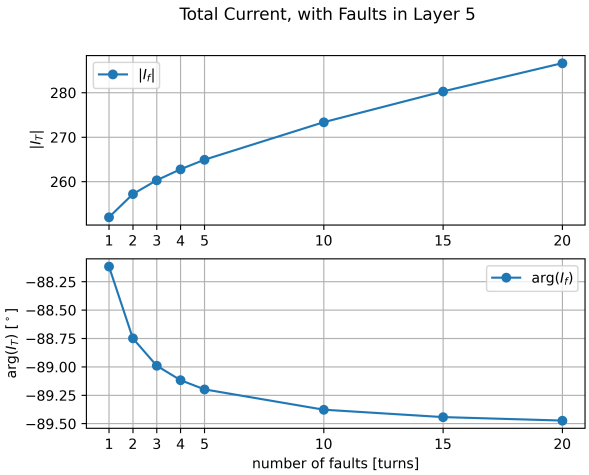


Figure C.6

Equation C.67 total current for 2 faults in layer 5

$$I_{pT\ 5,nf=2} = 5.61432050 - j257.12348392 = 257.18477128\angle -88.749^\circ \tag{C.67}$$

Equation C.68 total current for 10 faults in layer 5

$$I_{pT\ 5,nf=10} = 2.96864002 - j273.34254164 = 273.35866164\angle -89.378^\circ \tag{C.68}$$

Equation C.69 total current for 20 faults in layer 5

$$I_{pT\ 5,nf=20} = 2.63377749 - j286.62631410 = 286.63841459\angle -89.474^\circ \tag{C.69}$$

Equation C.70 total current for 40 faults in layer 5

$$I_{pT\ 5,nf=40} = 2.75581391 - j310.93174039 = 310.94395265\angle -89.492^\circ \tag{C.70}$$

Equation C.71 total current for 100 faults in layer 5

$$I_{pT\ 5,nf=100} = 4.57443259 - j399.04572776 = 399.07194624\angle -89.343^\circ \tag{C.71}$$

Equation C.72 total current for 150 faults in layer 5

$$I_{pT\ 5,nf=150} = 7.97101970 - j514.54968556 = 514.61142240\angle -89.112^\circ \tag{C.72}$$

Equation C.73 total current for 200 faults in layer 5

$$I_{pT\ 5,nf=200} = 16.34371028 - j720.63417543 = 720.81948619\angle -88.701^\circ \tag{C.73}$$

Layer 6

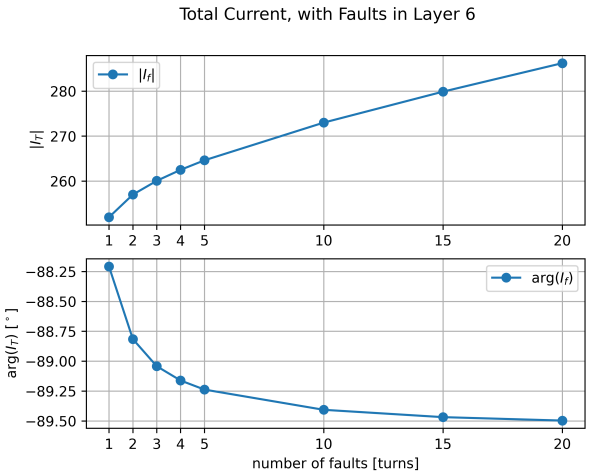


Figure C.7

Equation C.74 total current for 2 faults in layer 6

$$I_{pT\ 6,nf=2} = 5.31374231 - j256.93106701 = 256.98600944\angle -88.815^\circ \tag{C.74}$$

Equation C.75 total current for 10 faults in layer 6

$$I_{pT\ 6,nf=10} = 2.82590335 - j272.98049552 = 272.99512205\angle -89.407^\circ \tag{C.75}$$

Equation C.76 total current for 20 faults in layer 6

$$I_{pT\ 6,nf=20} = 2.51308402 - j286.20584622 = 286.21687931\angle -89.497^\circ \tag{C.76}$$

Equation C.77 total current for 40 faults in layer 6

$$I_{pT\ 6,nf=40} = 2.63870318 - j310.51062673 = 310.52183831\angle -89.513^\circ \tag{C.77}$$

Equation C.78 total current for 100 faults in layer 6

$$I_{pT\ 6,nf=100} = 4.42992616 - j399.60227726 = 399.62683123\angle -89.365^\circ \tag{C.78}$$

Equation C.79 total current for 150 faults in layer 6

$$I_{pT\ 6,nf=150} = 7.87728450 - j518.90078357 = 518.96057153\angle -89.130^\circ \tag{C.79}$$

Equation C.80 total current for 200 faults in layer 6

$$I_{pT\ 6,nf=200} = 16.86020196 - j739.49693988 = 739.68911747\angle -88.694^\circ \tag{C.80}$$

Layer 7

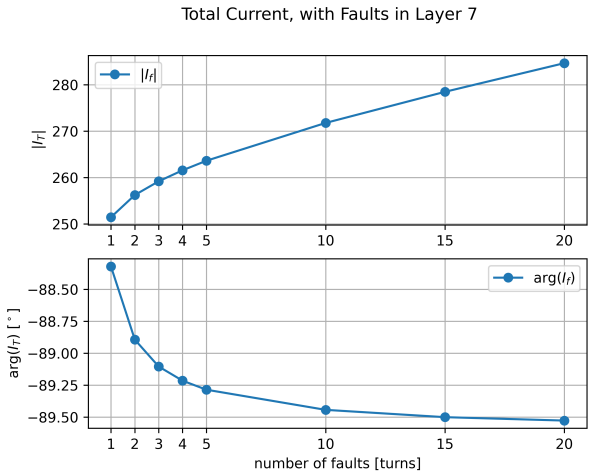


Figure C.8

Equation C.81 total current for 2 faults in layer 7

$$I_{pT\ 7,nf=2} = 4.94733496 - j256.18251147 = 256.23027789\angle -88.894^\circ \tag{C.81}$$

Equation C.82 total current for 10 faults in layer 7

$$I_{pT\ 7,nf=10} = 2.64228034 - j271.76222257 = 271.77506741\angle -89.443^\circ \tag{C.82}$$

Equation C.83 total current for 20 faults in layer 7

$$I_{pT\ 7,nf=20} = 2.35128058 - j284.64786627 = 284.65757726\angle -89.527^\circ \tag{C.83}$$

Equation C.84 total current for 40 faults in layer 7

$$I_{pT\ 7,nf=40} = 2.46974381 - j308.37646654 = 308.38635629\angle -89.541^\circ \tag{C.84}$$

Equation C.85 total current for 100 faults in layer 7

$$I_{pT\ 7,nf=100} = 4.16246422 - j395.90600417 = 395.92788516\angle -89.398^\circ \tag{C.85}$$

Equation C.86 total current for 150 faults in layer 7

$$I_{pT\ 7,nf=150} = 7.49610493 - j514.79404914 = 514.84862301\angle -89.166^\circ \tag{C.86}$$

Equation C.87 total current for 200 faults in layer 7

$$I_{pT\ 7,nf=200} = 16.53946494 - j740.32084768 = 740.50557824\angle -88.720^\circ \tag{C.87}$$

Layer 8

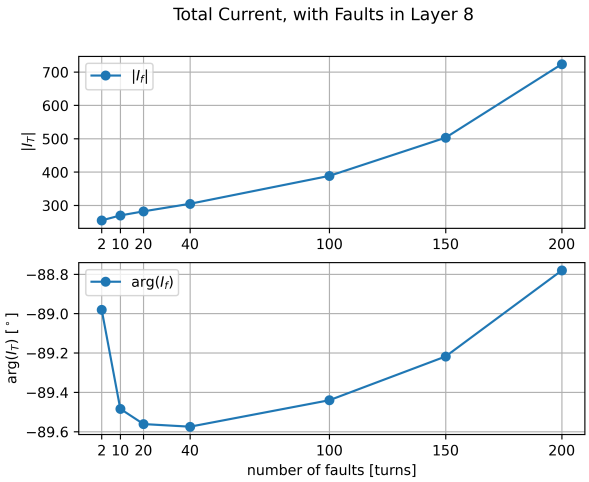


Figure C.9

Equation C.88 total current for 2 faults in layer 8

$$I_{pT\ 8,nf=2} = 4.53804719 - j254.96455434 = 255.00493690\angle -88.980^\circ \tag{C.88}$$

Equation C.89 total current for 10 faults in layer 8

$$I_{pT\ 8,nf=10} = 2.43078186 - j269.82791631 = 269.83886510\angle -89.484^\circ \tag{C.89}$$

Equation C.90 total current for 20 faults in layer 8

$$I_{pT\ 8,nf=20} = 2.16073005 - j282.13930242 = 282.14757614\angle -89.561^\circ \tag{C.90}$$

Equation C.91 total current for 40 faults in layer 8

$$I_{pT\ 8,nf=40} = 2.26305254 - j304.80471496 = 304.81311598\angle -89.575^\circ \tag{C.91}$$

Equation C.92 total current for 100 faults in layer 8

$$I_{pT\ 8,nf=100} = 3.79770766 - j388.54106838 = 388.55962786\angle -89.440^\circ \tag{C.92}$$

Equation C.93 total current for 150 faults in layer 8

$$I_{pT\ 8,nf=150} = 6.86867698 - j503.06735438 = 503.11424325\angle -89.218^\circ \tag{C.93}$$

Equation C.94 total current for 200 faults in layer 8

$$I_{pT\ 8,nf=200} = 15.39857942 - j723.42748173 = 723.59134708\angle -88.781^\circ \tag{C.94}$$

Layer 9 (outermost)

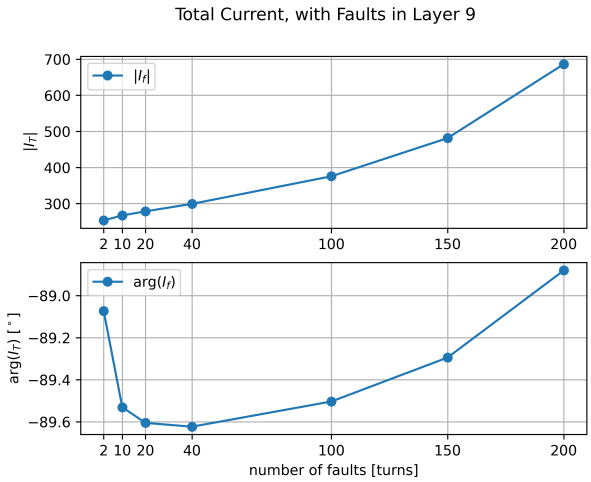


Figure C.10

Equation C.95 total current for 2 faults in layer 9

$$I_{pT\ 9,nf=2} = 4.10005148 - j253.33281865 = 253.36599501\angle -89.073^\circ \tag{C.95}$$

Equation C.96 total current for 10 faults in layer 9

$$I_{pT\ 9,nf=10} = 2.18775153 - j267.18634912 = 267.19530574\angle -89.531^\circ \tag{C.96}$$

Equation C.97 total current for 20 faults in layer 9

$$I_{pT\ 9,nf=20} = 1.91901782 - j278.50966805 = 278.51627928\angle -89.605^\circ \tag{C.97}$$

Equation C.98 total current for 40 faults in layer 9

$$I_{pT\ 9,nf=40} = 1.96681935 - j299.14417369 = 299.15063936\angle -89.623^{\circ} \quad \text{(C.98)}$$

Equation C.99 total current for 100 faults in layer 9

$$I_{pT\ 9,nf=100} = 3.25617265 - j375.65993547 = 375.67404725\angle -89.503^{\circ} \quad \text{(C.99)}$$

Equation C.100 total current for 150 faults in layer 9

$$I_{pT\ 9,nf=150} = 5.92939175 - j481.41557257 = 481.45208609\angle -89.294^{\circ} \quad \text{(C.100)}$$

Equation C.101 total current for 200 faults in layer 9

$$I_{pT\ 9,nf=200} = 13.41697770 - j686.31683825 = 686.44797163\angle -88.880^{\circ} \quad \text{(C.101)}$$

Appendix D

Programs

D.1 Python Dependencies

These modules were written in Python v3.6+, there is no garentee that these will work in any version less that v3.6 The NumPy and SciPy processing Libraries, and the MatPlotLib graphing Library are needed for these programs to function

D.2 Reactor Python Library

D.2.1 Biot-Savart Methods

Turn-to-turn calculations based the Biot-Savart law and the Neumann Integral, Implementation based on Paul [3].

```
#!/bin/python3
"""
‘biot_savart_methods.py‘
methods to implement turn to turn inductance calculations.
18 May 2021
R. Sanford
"""
import numpy as np
from numpy import sin, cos, tan, exp, arcsin, arccos, arctan, sqrt, pi
import scipy.integrate as integrate

mu0 = 4*pi*10**(-7) # [H/m] permeability of the void
def numint(fx, x0, x1, nn):
  """
  Numeric Integration using trapezoid-rule (18 May 2021)
  fx : function handle
```

```
  x0 : starting point
  x1 : end point
  nn : the number of stepsto take
  fx_args : parameters to pass into fx
  """
  res = 0 # results: area under a curve.
  dx = (x1-x0)/nn # step

  for n in range(nn):
    res = res + dx * .5 * ( fx(n*dx) + fx((n+1)*dx) )

  return res

def numint_w_args(fx, x0, x1, nn, a, b, d):
  """
  Numeric Integration using trapezoid-rule
  fx : function handle
  x0 : starting point
  x1 : end point
  nn : the number of stepsto take
  fx_args : parameters to pass into fx
  """
  res = np.zeros(nn) # results: area under a curve.
  dx = (x1-x0)/nn # step

  for n in range(nn):
    res[n] = dx * .5 * ( fx(n*dx, a, b, d) + fx((n+1)*dx, a, b, d) )

  return np.sum(res)

def quadrature(fx, x0, x1, a, b, d):
    # wrapper for scipy quadrature.
    # from scipy quadrature integration; this is faster than anything I could write
    quickly
    #res = integrate.quad(lambda x: fx(x, a, b, d), x0, x1, epsabs=1e-13, limit=1000)
    #res = integrate.quad(fx, x0, x1, args=(a, b, d), epsabs=1.5e-8, limit=10)
    res = integrate.quad(fx, x0, x1, args=(a, b, d))
    #print(res[1])
    return res[0]

def neumann(phi,a,b,d):
    """The part of the Neumann integral within the integrad"""
    return cos(phi)/sqrt(a**2 + b**2 + d**2 - 2*a*b*cos(phi))

def bs_mutual(a,b,d=0):
```

```
    """
    Biot-Savart mutual calcuation for concentric (circular) loops, because
    we'll use this a good bit.

    parameters:
    a : loop radius (to center of wire) : "R_wire" [m]
    b : radius of the area enclosed by the wire loop : "R_loop-r_wire" [m]
    d : distance seperating the two loops (can be 0) [m]
    """
    mut = mu0 * a * b * quadrature(neumann, 0, pi, a, b, d)

    return mut

def internal(R, r):
    """
    Inductance internal to the wire loop
    R : loop radius (to center of wire) [m]
    r : wire radius (diam/2) [m]
    """
    return mu0*(R/4 + r/5)
```

D.2.2 Thin-Sheet Methods

Computationally efficient methods originally presented by Fawzi and Burke [4].

```
#!/bin/python3
"""
‘fawzi_and_burke_methods.py‘
Fawzi and Burke Method Impelementation
Computation Method From:
    The Accurate Computation of Self and Mutual Inductances of Circular Coils, 1978
        https://doi.org/10.1109/TPAS.1978.354506
12 December 2021
R. Sanford
"""

from numpy import pi, sqrt, sin, cos
import numpy as np
import scipy.integrate as integrate
import matplotlib.pyplot as plt
mu0 = 4*pi*10**(-7)

def Ci_1(R1,R2,z):
    """
    This is the integral (equation 3) as presented in the referenced paper.
```

```
21
22      Parameters:
23          'R1','R2' : Coil radius [m]
24          'z' : Vertical position [m]
25      Returns:
26          'Ci' : Constant used in calculating inductance
27      """
28      fn = lambda p: ( (sqrt((R1**2) + (R2**2) + (z**2) - (2*R1*R2*cos(p)))\
29                      * (sin(p)**2)) / ((R1**2) + (R2**2) - (2*R1*R2*cos(p))) )
30      res = integrate.quad(fn, 0, pi)
31      C = (sqrt(R1*R2)/(2*pi))*res[0]
32
33      return C
34
35
36  def fb_mutual(N1,R1,h1,N2,R2,h2,s):
37      """
38      Compute a layer-to-layer mutual using equation 2 from the reference paper.
39
40      Parameters:
41          'N1','N2' : Turn count (not turn density as in paper)
42          'R1','R2' : Radii of the layers [m]
43          'h1','h2' : Height of the coil [m]
44          's'       : concentric seperation (z) [m]
45
46      Turn denstiy is calculated as: N/h [turns/m]
47
48      Returns:
49          'M' : Mutual inductance
50      """
51
52      l1 = 0.5*h1
53      l2 = 0.5*h2
54      z1 = l1 + l2 + s
55      z2 = l1 - l2 + s
56      z3 = -l1 - l2 + s
57      z4 = -l1 + l2 + s
58
59      M = 2*pi*mu0*( (R1*R2)**(3/2) ) * (N1/h1)*(N2/h2)\
60                  *( (Ci_1(R1,R2,z1) - Ci_1(R1,R2,z2))\
61                  +  (Ci_1(R1,R2,z3) - Ci_1(R1,R2,z4)) )
62      return M
63
64
65  def fb_mutual_thick(N1,R1,h1,t1,N2,R2,h2,t2,s):
```

```
    """
    Compute a layer-to-layer mutual using equation 18 from the reference paper,
    which takes into account the thickness of a coil set (as for non-finite
    thickness coils).

    Parameters:
        'N1','N2' : Turn count (not turn density as in paper) *
        'R1','R2' : Radii of the layers [m]
        'h1','h2' : Height of the coil [m]
        't1','t2' : Thickness of the coils [m]
        's'       : concentric seperation (z) [m]

    * Turn denstiy is calculated within the function as: n = N/h [turns/m]

    Returns:
        'M' : Mutual inductance
    """
    l1 = 0.5*h1
    l2 = 0.5*h2
    z1 = l1 + l2 + s
    z2 = l1 - l2 + s
    z3 = -l1 - l2 + s
    z4 = -l1 + l2 + s

    t12 = (t1/2)
    t22 = (t2/2)

    M = lambda r2,r1: fb_mutual(N1,r1,h1,N2,r2,h2,s=s)

    res = integrate.dblquad(M, R1-t12, R1+t12, R2-t22, R2+t22, epsabs=5e-6)

    Mut = ((N1/h1)*(N2/h2)) * res[0] # need to scale by the turn density as per eqn
    18

    return Mut

def fb_self_ind(N,R,h): # good (1/5/2022)
    """
    Self-inductance method presented in the paper (equation 16)

    Parameters:
        'N' : Number of turns (not turn density as in paper)
        'R' : Radius of the layer [m]
        'h' : Layer height [m]

```

```
    Turn denstiy is calculated as: N/h [turns/m]

    Returns:
        'L' : self-inductance of the layer coil
    """
    L = fb_mutual(N1=N,R1=R,h1=h,
                        N2=N,R2=R,h2=h,s=0)
    return L
```

D.3 Utility Scripts

D.3.1 util.py

This resource is used by the library to handle things like making the matrix look pretty in the terminal.

```
"""
‘util.py‘
Utilities to make life easier.
R. Sanford
"""
import numpy as np
from os import get_terminal_size
import os

def imaginary_formatter(m):

    m_r = m.real
    m_i = m.imag

    string_mat = ''
    if m_r == 0:
        string_mat += '          0'
    elif m_r > 0:
        string_mat += '  {:1.3e}'.format(np.abs(m_r))
    else:
        string_mat += ' -{:1.3e}'.format(np.abs(m_r))

    if m_i == 0:
        string_mat += '             '.format(np.abs(m_i))
    elif m_i > 0:
        string_mat += '+{:1.3e}j '.format(np.abs(m_i))
    else:
```

```
28          string_mat += '-{:1.3e}j '.format(np.abs(m_i))
29
30      return string_mat
31
32  def vector_formatter(m):
33      # angle = '\u2220'
34      string_mat = ''
35
36
37      if m == 0:
38          string_mat += ' 0.'+' '*19
39      else:
40          # ang = np.arctan(m.imag/m.real)*180/np.pi
41          ang = np.angle(m,deg=True)
42          # print("ang =",ang)
43          # if m.real < 0:
44          #     ang = 90-abs(ang)
45          mag = np.abs(m)
46
47          if abs(ang) < 10:
48              string_mat += '{}{:1.3e}\u2220 {}{:0.3f}\u00B0   '.format(('' if mag < 0
    else ' '),mag,('' if ang < 0 else ' '),ang)
49          elif abs(ang) < 100:
50              string_mat += '{}{:1.3e}\u2220 {}{:0.3f}\u00B0  '.format(('' if mag < 0
    else ' '),mag,('' if ang < 0 else ' '),ang)
51          else:
52              string_mat += '{}{:1.3e}\u2220 {}{:0.3f}\u00B0 '.format(('' if mag < 0
    else ' '),mag,('' if ang < 0 else ' '),ang)
53
54      return string_mat
55
56  def str_matrix(mat,phasor=False,indicators=[None],**kwargs):
57      """
58      Stringify matrix or 2D array, and python list, that doens't have a preferred wrap
         length.
59      If the provided mat is a 1d list, it will assume you meant for it to be an "N by
        1" vector.
60
61      Arguments:
62      `phasor` determines if you are want to represent complex values as R+i or value
        at an angle (phasor)
63      `inticators` is a list of indexes that get an '*' at the end of the row (doens't
        accept reverse indexing -1) ex: `indicators=[0,3]`
64      `left_offset` provides the number of spaces to offset the matrix from the left
```

```
    ‘row_labels‘ and ‘col_labels‘ can be specified for provide Row (applied to the left) and Column (applied above) labels for the matrix
    ‘label‘ matrix label, have an inline label assigned: ex: G = [matrix]. the label will be centered.
    for compatability: ‘vector_value‘ can be used in place of the ‘phasor‘ argument

    Example:
    ‘‘‘python
    import nympy as np
    from util import str_matrix

    mat = np.array([[1,2],[3,4]])

    matrix_string = str_matrix(mat)
    print(matrix_string) # this will print the matrix in 2 rows
    print(str_matrix([[1,2],[3,4]])) # this will print the same as the line above

    c_mat = np.array([[1+1j,2],[3j,4+2j]]) # complex matrix
    print(str_matrix(c_mat)) # this will print the matrix so each element is a R+jI
    print(str_matrix(c_mat,phasor=True)) # this will print the matrix so each element is a magnitude at some angle

    print(str_matrix([k for k in range(8)])) # this will print the values [0,1,...,7] as an Nx1 vector
    ‘‘‘
    """

    # unicode parts:
    ULC = '\u23A1' # left ceiling
    URC = '\u23A4' # right ciling
    ULF = '\u23A3' # left floor
    URF = '\u23A6' # right floor
    UVR = '\u23A5' # right vertical
    UVL = '\u23A2' # left vertical
    # example use of the brakets
    # print(  ULC+' 1 2 3 '+URC+'\n'+\
    #         UVL+' 4 5 6 '+UVR+'\n'+\
    #         ULF+" 7 8 9 "+URF)
    left_offset = kwargs['left_offset'] if 'left_offset' in kwargs else 0
    row_labels = kwargs['row_labels']+[] if 'row_labels' in kwargs else []
    col_labels = kwargs['col_labels']+[] if 'col_labels' in kwargs else []
    matrix_label = kwargs['label'] if 'label' in kwargs else ""
    as_decimal = kwargs['as_decimal'] if 'as_decimal' in kwargs else False

    # phasor cpmpatability with 'vector_value':
```

```
    phasor = phasor or (kwargs['vector_value'] if 'vector_value' in kwargs else False
    )

    # print(mat)
    # print(type(mat))

    if type(mat) == list: # if it's a list, convert it to a numpy array
        # print("List")
        if isinstance(mat[0],list): # if a 2d list is provided instead of a numpy
    array
            mat = np.array(mat)
        else: # otherwise it is a 1d list, and needs fixed, and is assumed to be a
    vector
            mat = np.array([mat]).T

    try:
        N,M = mat.shape # N=no. rows, M=no. cols
    except ValueError:
        mat = np.reshape(mat,(-1,1))
        N,M = mat.shape

    if len(matrix_label) > 0:
        mli = int(N/2)
        if len(row_labels) > 0:
            row_labels[mli] = matrix_label + ' ' + row_labels[mli]
        else:
            RLM = ['']*(mli-1)+[matrix_label]+['']*(N-mli) # row-label modifications
            row_labels += RLM


    str_rows = []
    if 'complex' in str(mat.dtype):
        str_rows = __complex_matrix(mat,as_phasor=phasor)
        base_width = len('3.285e+01+2.103e+02j')
    elif 'int' in str(mat.dtype):
        str_rows, base_width = __int_matrix(mat)
    else: # default: just use float
        str_rows = __real_float_matrix(mat,as_decimal)
        base_width = len('3.285e+01')

    nr = len(str_rows)
    for r in range(nr):
        if r == 0:
            str_rows[r] = ULC + str_rows[r] + URC
        elif r==(nr-1):
```

```
148             str_rows[r] = ULF + str_rows[r] + URF
149         else:
150             str_rows[r] = UVL + str_rows[r] + UVR
151 
152         if r in indicators: # add the indicator
153             str_rows[r] += ' *'
154 
155     # Add Decorations:
156 
157     row_label_width = 0
158     if len(row_labels) > 0:
159         if not len(row_labels) == nr:
160             raise ValueError("Number of row labels doesn't match number of rows")
161         f_row_labels,row_label_width = __format_row_labels(row_labels)
162         for r in range(nr):
163             str_rows[r] = f_row_labels[r] + str_rows[r]
164 
165     if len(col_labels) > 0:
166         # print('row_label_width =',row_label_width)
167         # basically form a new row for str_rows
168         FCL = __format_col_labels(labels=col_labels,base_w=base_width,row_offset=
    row_label_width)
169 
170         str_rows = [FCL] + str_rows
171 
172     # str_mat = __fit_to_terminal(str_rows,left_offset)
173     # str_mat = __fit_to_terminal(str_rows,left_offset,len(matrix_label))
174     str_mat = '\n'
175     for row in str_rows:
176         # print('str_matrix_v2: row =',row)
177         str_mat += ' '*left_offset + row + '\n'
178 
179     return str_mat
180 
181 def __complex_matrix(mat,as_phasor):
182 
183     N,M = mat.shape # N=no. rows, M=no. cols
184     square_mat = (N==M) # square matrix flag for diagomal bolding
185     str_rows = ['']*N
186     for n in range(N): # rows
187         for m in range(M): # columns
188             if square_mat and n==m:
189                 str_rows[n] +='\033[1m' # special formatting bold begin unicode
190 
191             if as_phasor: # format as magnitude at angle
```

```
192                 str_rows[n] += vector_formatter(mat[n,m])
193             else: # format as re+j*im
194                 str_rows[n] +=  imaginary_formatter(mat[n,m])
195 
196             if square_mat and n==m:
197                 str_rows[n] +='\033[0m' # special formatting (bold) termination
    unicode
198 
199     # for row in str_rows:
200     #     print('__complex_matrix: row = ',row)
201     return str_rows
202 
203 def __real_float_matrix(mat, as_dec):
204     N,M = mat.shape # N=no. rows, M=no. cols
205     square_mat = (N==M) # square matrix flag for diagomal bolding
206     str_rows = ['']*N
207     for n in range(N): # rows
208         for m in range(M): # columns
209             if square_mat and n==m:
210                 str_rows[n] +='\033[1m' # special formatting bold begin unicode
211             if mat[n,m] == 0:
212                 if as_dec:
213                     str_rows[n] += ' '*1 + '0.' + ' '*4
214                 else:
215                     str_rows[n] += ' '*1 + '0.' + ' '*8
216             else:
217                 if as_dec:
218                     if mat[n,m] > 0:
219                         # str_rows[n] +=  ' {:1.3f} '.format(mat[n,m])
220                         if mat[n,m] < 10:
221                             str_rows[n] +=  ' {:1.3f} '.format(mat[n,m])
222                         elif mat[n,m] > 100:
223                             str_rows[n] +=  ' {:1.3f} '.format(mat[n,m])
224                         else:
225                             str_rows[n] +=  ' {:1.2f} '.format(mat[n,m])
226                     else: # leave space the the negative sign
227                         # str_rows[n] +=  '{:1.3f} '.format(mat[n,m])
228                         if mat[n,m] > -10:
229                             str_rows[n] +=  '{:1.3f} '.format(mat[n,m])
230                         elif mat[n,m] < -100:
231                             str_rows[n] +=  '{:1.1f} '.format(mat[n,m])
232                         else: # -10 to -99
233                             str_rows[n] +=  '{:1.2f} '.format(mat[n,m])
234                 else:
235                     if mat[n,m] > 0:
```

```
236                         str_rows[n] +=  ' {:1.3e} '.format(mat[n,m])
237                     else: # leave space the the negative sign
238                         str_rows[n] +=  '{:1.3e} '.format(mat[n,m])
239 
240             if square_mat and n==m:
241                 str_rows[n] +='\033[0m' # special formatting (bold) termination
    unicode
242 
243     # for row in str_rows:
244     #     print('__real_float_matrix: row =',row)
245     return str_rows
246 
247 def __int_matrix(mat):
248     N,M = mat.shape # N=no. rows, M=no. cols
249     square_mat = (N==M) # square matrix flag for diagomal bolding
250     # mat[0,0] = -1*mat[0,0]
251     base_length = len(str(np.max(mat))) # longest element in the matrix
252     # print('__int_matrix: base_length =',base_length)
253 
254     str_rows = ['']*N
255     for n in range(N): # rows
256         for m in range(M): # columns
257             if square_mat and n==m:
258                 str_rows[n] +='\033[1m' # special formatting bold begin unicode
259 
260             val = str(mat[n,m])
261             str_rows[n] +=  ' '*(base_length-len(val)) + (' ' if mat[n,m]>=0 else '')
     + val + ' '
262 
263             if square_mat and n==m:
264                 str_rows[n] +='\033[0m' # special formatting (bold) termination
    unicode
265 
266     # for row in str_rows:
267     #     print('__real_float_matrix: row =',row)
268     return str_rows, base_length # return base length for column labels if needed
269 
270 def __format_row_labels(labels):
271     # print(row_labels)
272     formatted_row_labels = [] # make row labels a uniform width
273     row_label_width=0
274 
275     row_label_width=max([len(ss) for ss in labels])
276     for k in range(len(labels)):
```

```
277         # formatted_row_labels.append(labels[k] + ' '*(row_label_width-len(labels[k])
    ) +' ')
278         formatted_row_labels.append(' '*(row_label_width-len(labels[k])) + labels[k]
    + ' ')
279 
280     return formatted_row_labels,row_label_width
281 
282 def __format_col_labels(labels,base_w=12,row_offset=0):
283     string_mat = ''
284     # print(col_labels)
285     fcl = ' '*(row_offset + 2)# make row labels a uniform width
286     # print('column base width =',base_w)
287     # print('row offset =',row_offset)
288     # base_w = 12
289     col_label_width = max([len(ss) for ss in labels])
290     for k in range(len(labels)):
291         cl = ' ' + labels[k] + ' '
292         # print(cl,len(cl))
293         if len(cl) > base_w+2:
294             # print(f'len({cl}) > {base_w+2}')
295             cl = cl[0:base_w+1] + ' '
296         elif len(cl) < base_w+2:
297             # print(f'len({cl}) < {base_w+2}')
298             # cl += ' '*((base_w+2)-len(cl)) # right justify
299             cl = ' '*((base_w+2)-len(cl)) + cl # left justify
300         # print(cl,len(cl))
301 
302         fcl += cl
303     # print('str_matrix_v2: row =',fcl)
304     return fcl
305 
306 def __fit_to_terminal(str_rows, left_off=0,label_width=0):
307     """
308     Fit to the terminal by breaking lines as needed.
309     """
310     term_size = get_terminal_size()
311     t_cols = term_size.columns
312     t_lines = term_size.lines
313     print('terminal size:',t_cols,'x',t_lines)
314     break_len = (t_cols-4)-left_off
315     print('line break at character:',break_len)
316 
317     str_mat = '\n'
318     if len(str_rows[1])+left_off > break_len:
319         str_rows_extended = {}
```

```
320
321         length_offset = 0
322         print('str_rows[-1]',str_rows[-1])
323         if not str_rows[-1][break_len] == ' ': # check that we are breaking on
    whitespace
324             for k in range(break_len):
325                 if str_rows[-1][break_len-k] == ' ':
326                     length_offset = k+1
327                     break
328         print('length_offset =',length_offset)
329         break_len += -1*length_offset
330
331         print('break_len =',break_len)
332
333         # figure out how many breaks it will need:
334         char_ratio= (len(str_rows[-1])+left_off)/(break_len)
335         num_breaks = int(char_ratio) if char_ratio - int(char_ratio) < 0.8 else int(
    char_ratio)+1
336         print('number of line breaks:',num_breaks)
337
338         for k in range(num_breaks+1):
339             str_rows_extended[k] = []
340
341         # print(str_rows_extended)
342         for ri in range(len(str_rows)):
343             broken_line = break_string_to_segments(str_rows[ri],break_len,num_breaks)
344             for line in range(len(broken_line)):
345                 str_rows_extended[line] += broken_line[line]
346
347         # print(str_rows_extended)
348         for k in str_rows_extended:
349             for row in str_rows_extended[k]:
350                 str_mat += ' '*(left_off+( label_width+1 if k > 0 else 0 )) + row + '
    \n'
351             str_mat += '\n'
352
353     else:
354         for row in str_rows:
355             # print('str_matrix_v2: row =',row)
356             str_mat += ' '*left_off + row + '\n'
357
358     return str_mat
359
360 def break_string_to_segments(s,length,n_breaks):
361     ss = [] # string segments
```

```
362     # if not s[length] == ' ': # check that we are breaking on whitespace
363     #     for k in range(length):
364     #         if s[length-k] == ' ':
365     #             length_offset = k
366     #             break
367     print('len(s) =',len(s),'extra:',len(s)-length)
368     break_char = s[length]
369
370     bold_start = '\033[1m'
371     bold_end = '\033[0m'
372
373     special_check_before = lambda c, i: s.index(bold_start) <= i and s.index(bold_end
        ) <= i
374     special_check_between = lambda c, i: s.index(bold_start) <= i <= s.index(bold_end
        )
375     print('bold_start index =',s.index(bold_start),
376           'bold_end index =',s.index(bold_end),
377           'length is between bold?',special_check_between(s,length),
378           'length is after bold?',special_check_before(s,length))
379
380     offset = 0
381     if special_check_before(s,length):
382         offset = len(bold_start) + len(bold_end) +0
383     elif special_check_between(s,length):
384         if length < s.index(bold_start) + 0.45*(s.index(bold_end)-s.index(bold_start)
        ): # closer to start
385             offset = s.index(bold_start) - length + 0
386         else: # closer to the end
387             offset = (length - s.index(bold_start))# + len(bold_start)
388             # s = s.replace(' '+bold_end,bold_end + ' ')
389             # pass
390         # offset = len(bold_end)
391     # else:
392     #     offset = 0
393
394     # s = s.replace('\033[0m','\b}')
395     # s = s.replace('\033[1m','\b{')
396     # s = s.replace('\033[0m','\033[0m'+'}')
397     # s = s.replace('\033[1m','\033[1m'+'{')
398
399     for k in range(n_breaks+1):
400         i0 = k*(length + offset)
401         i1 = (k+1)*(length + offset) # subreact k to deal with white-space
402         # ss_s = (' ' if k > 0 else '') + s[i0:i1] + '|'# + f' k={k}'
403         ss_s = s[i0:i1]
```

```
404         # print(f'k={k} >>>',ss_s)
405
406         # print(ss_s if k > 0 else ss_s)
407         ss.append([ss_s])
408
409     return ss
410
411
412 def increment_name(path,ext='.png'):
413     """
414     Check if the file already exists, if it does, ingrement the filepath
415     """
416     if os.path.isfile(path):
417         path_inc=path.replace(ext,'')
418
419         k=0
420         while k < 1e8:
421             k+=1
422             if not os.path.isfile(path_inc+'_{}{}'.format(k,ext)):
423                 return path_inc+'_{}{}'.format(k,ext)
424         print(f"file \"{path}\" exists, incrementing name to \"{path_inc}\"")
425     else:
426         return path
427
428 def check_for_path(path, create_if_dne=False):
429     """
430     Check that a path exists, create if it doens't exist (depending on `create_if_dne
    `)
431     returns the path (string) or `None` id DNE and Didn't create.
432     """
433
434     path_mod = None
435
436     # if the path DNE, and it's not a filename
437     if ( not os.path.isdir(path) ):
438         if create_if_dne: # if you want to make one
439             os.makedirs(path) # use the recursive functionality.
440             path_mod = path
441         else:
442             print(f"Path \"{path}\" DNE, not making one")
443         return path_mod
444     else: # it is a path
445         return path
446
447 def cofactor(M):
```

```
448     """
449     calculate the cofactor of a matrix, M should be square.
450     """
451     adj_M = np.zeros(M.shape)
452 
453     M = M.T
454 
455     for i in range(M.shape[0]):
456         for j in range(M.shape[1]):
457             a = M[:i,:j] # square up to i,j
458             b = M[:i,j+1:] # rows down to i
459             c = M[i+1:,:j] # columns up to j
460             d = M[i+1:,j+1:] # square from i+1,j+1
461             '''
462             arrange as:
463             cof = [[a, b],[c,d]]
464             '''
465             cof_1 = np.concatenate((a,b),axis=1)
466             cof_2 = np.concatenate((c,d),axis=1)
467             cof = np.concatenate((cof_1,cof_2),axis=0) # stack
468             # print(str_matrix(cof,label=f'cof({i},{j}) ='))
469             # print(f'|cof({i},{j})| =',np.linalg.det(cof))
470 
471             adj_M[i,j] = np.linalg.det(cof)
472 
473     return adj_M
474 
475 def coupling_coef(L):
476     K = np.zeros(L.shape)
477     for i in range(L.shape[0]):
478         for j in range(L.shape[1]):
479             if i==j:
480                 K[i,j] = 1
481             else:
482                 K[i,j] = L[i,j] / np.sqrt(L[i,i] * L[j,j])
483     return K
484 
485 def layer_ring_plot(plt, radii, layers):
486     """
487     Plot the rings of the reactor.
488     radii is an array of layer radii
489     layers indicate how many layers are in each "package" ex: [1,3,2,2] would
        indicate 1 layer in 0th position, positions 1-3 would be another package,
490     """
491     th = np.linspace(0,2*np.pi,64+1)
```

```
    # r,linespec,label=None,fn=64
    color_i = 0
    k = 0
    plt.figure("Reactor Layer Ring Plot")
    for layer in layers:
        # print(layer)
        for ln in range(layer):
            # r = self.layers[layer]['r_loop']
            # print(k)
            x = radii[k]*np.cos(th)
            y = radii[k]*np.sin(th)
            plt.plot(x,y,'C{}'.format(color_i))
            k += 1
        color_i += 1
    plt.axis('equal')

if __name__ == '__main__':

    # str_matrix testing
    v1 = np.array([[k*(1+0.5j) for k in range(15)]])
    # v1 = np.array([[k*(1) for k in range(35)]])
    mat = np.matmul(v1.T,v1)
    print(str_matrix(mat))

    # break_string_to_segments(s='['+' 0123456789ABC '*24+']' ,length=106,n_breaks=3)
```

D.3.2 tex_util.py

```
"""
‘tex_util.py‘
Utilities for generating reports in LaTeX.
R. Sanford
20 January 2023
"""
import numpy as np
import os,random
from util import str_matrix, vector_formatter, increment_name,
                 cofactor, coupling_coef, check_for_path


n_tab = 4 # number of spaces to be a tab

def tex_report(models, desc, report_file, sigfig):
    """
    make a report, models is a dict with the keys corresponding
```

```
    to the matrix variable label.

    Parameters:
    - models : Library of objects, the library label will be the variable label.
    - desc   : Object descriptions, will be printed as a comment under the equation
    - report_file : path to file (full filename included)
    - reference_desc : print the refernece before the description text so it ...
        becommes: \ref{eqn_label} {description}
    """
    report_file = increment_name(report_file,'.tex')

    for k in models:
        # print(type(models[k]))
        if 'array' in str(type(models[k])):
            save_matrix_as_tex(models[k],mat_label=k,savepath=report_file,
                               description=desc[k],write_mode='a',sigfig=sigfig)
        else:
            save_as_tex(models[k],label=k,savepath=report_file,
                        description=desc[k],write_mode='a',sigfig=sigfig)


def tex_vector_formatter(a,sigfig=3):
    format_string = '{:0.'+f'{sigfig}'+'E} \\angle {:0.3f}^\\circ'
    return format_string.format(np.abs(a),np.angle(a,deg=True))


def save_matrix_as_tex(M,mat_label,savepath,description,write_mode='w',
                       sigfig=3,as_decimal=False):
    """
    convert a numpy matrix into a LaTeX matrix
    """
    randid = random.randint(1e2,1e8)
    item_label = f'eqn:{labelify(mat_label)}_{randid}'
    is_vector = (M.shape[0] > 1) and (M.shape[1] == 1)
    print(mat_label, 'is vector?',is_vector)

    tex_str = '\n\\begin{equation}\label{'+item_label+'}\n'+' '*n_tab\
                +mat_label+'=\n'+' '*n_tab+'\\begin{bmatrix}\n'
    dtype = str(M.dtype)
    # print(M.dtype)
    for ii in range(M.shape[0]):
        tex_str += ' '*2*n_tab
        for jj in range(M.shape[1]):
            if 'int' in dtype :
                tex_str += '{:d}'.format(M[ii,jj])\
```

```
                        + ( ' & ' if jj<(M.shape[1]-1) else '' )

            elif 'float' in dtype:
                if M[ii,jj] == 0:
                    tex_str += '0' + ( ' & ' if jj<(M.shape[1]-1) else '' )
                else:
                    if M[ii,jj] >= 1e-3 or as_decimal:
                        tex_str += '{:0.3f}'.format(M[ii,jj])\
                                + ( ' & ' if jj<(M.shape[1]-1) else '' )
                    else:
                        tex_str += '{:0.3E}'.format(M[ii,jj])\
                                + ( ' & ' if jj<(M.shape[1]-1) else '' )

            elif 'complex' in dtype :
                m_r = M[ii,jj].real
                m_i = abs(M[ii,jj].imag)
                m_is = '+' if M[ii,jj].imag > 0 else '-'

                if m_r > 0:
                    tex_str += '{:0.3E}{}j{:0.3E}'.format(m_r,m_is,m_i)\
                            + ( ' & ' if jj<(M.shape[1]-1) else '' )
                else:
                    tex_str += '0{}j{:0.3E}'.format(m_is,m_i)\
                            + ( ' & ' if jj<(M.shape[1]-1) else '' )

        tex_str += ' \\\\\n'
    if is_vector:
        tex_str += ' '*n_tab+'\\end{bmatrix}\n\\end{equation}'\
                if (not 'complex' in dtype) else ' '*n_tab\
                +'\\end{bmatrix}\n'+' '*n_tab+'=\n'+' '*n_tab+'\\begin{bmatrix}\n'
        if 'complex' in dtype:
            for ii in range(M.shape[0]):
                tex_str += ' '*2*n_tab
                for jj in range(M.shape[1]):
                    m_r = M[ii,jj].real
                    m_i = abs(M[ii,jj].imag)
                    m_is = '+' if M[ii,jj].imag > 0 else '-'
                    tex_str += tex_vector_formatter(M[ii,jj])\
                            + ( ' & ' if jj<(M.shape[1]-1) else '' )
                tex_str += ' \\\\\n'
            tex_str += ' '*n_tab+'\\end{bmatrix}\n\\end{equation}'
    else:
        tex_str += ' '*n_tab+'\\end{bmatrix}\n\\end{equation}'

    desc = description.split('\n')
```

```
    tex_str += ('\n% ' + '\\ref{'+item_label+'} '\
        + desc[0].strip()) if len(desc[0]) > 0 else '\n %\\ref{'+item_label+'} '
    if len(desc) > 1:
        for dl in desc[1:]:
            sdl = dl.strip()
            tex_str += ('\n% ' + sdl) if len(sdl) > 0 else ''
    tex_str += '\n'
    # print(tex_str)

    with open(savepath,write_mode) as texfile:
        texfile.write(tex_str)
        texfile.close()


def save_as_tex(M,label,savepath,description,write_mode='w',sigfig=3):
    """
    Save a value to a tex equation in a file in the same style as a matrix
    """
    randid = random.randint(1e2,1e8)

    item_label = f'eqn:{labelify(label)}_{randid}'
    tex_str = '\n\\begin{equation}\label{'+item_label+'}\n'+' '*n_tab+label+'='
    dtype = str(M.dtype)
    # print(M.dtype)
    if 'int' in dtype :
        tex_str += '{:d}'.format(M)

    elif 'float' in dtype :
        tex_str += '{:0.3f}'.format(M)

    elif 'complex' in dtype :
        m_r = M.real
        m_i = abs(M.imag)
        m_is = '+' if M.imag > 0 else '-'
        complex_format_str = '{:0.'+f'{sigfig}'+'E}{}j{:0.'+f'{sigfig}'+'E} = '
        tex_str += complex_format_str.format(m_r,m_is,m_i)
        tex_str += tex_vector_formatter(M,sigfig)


    tex_str += '\n\\end{equation}'

    desc = description.split('\n')

    tex_str += ('\n' + '\\ref{'+item_label+'} '\
            + desc[0].strip()) if len(desc[0]) > 0 else '\n% \\ref{'+item_label+'} '
```

```
    if len(desc) > 1:
        for dl in desc[1:]:
            sdl = dl.strip()
            tex_str += ('\n% ' + sdl) if len(sdl) > 0 else ''

    tex_str += '\n'
    # print(tex_str)

    with open(savepath,write_mode) as texfile:
        texfile.write(tex_str)
        texfile.close()


def labelify(ml):
    if '\;' in ml:
        ml=ml.replace('\;','_')
    if '{' in ml or '}' in ml:
        ml=ml.replace('{','')
        ml=ml.replace('}','')
    return ml
```

D.3.3 wires.py

```
"""

Wire Parameter Values, and functions for useful stuff therein

"""
from numpy import pi

AWG_diam = { # AWG diameters in millimeters
    '4/0':11.684, '3/0':10.4049, '2/0':9.2658, '1/0':8.2515,
     1:7.3481,  2:6.5437,  3:5.8273,  4:5.1894,  5:4.6213,  6:4.1154,
     7:3.6649,  8:3.2636,  9:2.9064, 10:2.5882, 11:2.3048, 12:2.0525,
    13:1.8278, 14:1.6277, 15:1.4495, 16:1.2908, 17:1.1495, 18:1.0237,
    19:0.9116, 20:0.8118, 21:0.7229, 22:0.6438, 23:0.5733, 24:0.5106,
    26:0.4049, 27:0.3606, 28:0.3211, 29:0.2859, 30:0.2546, 31:0.2268,
    32:0.2019, 33:0.1798, 34:0.1601, 35:0.1426, 36:0.1270, 37:0.1131,
    38:0.1007, 39:0.0897, 40:0.0799,
}

resistivity = { # Ohm-m^2/m, use by dividing by the area of the wire
    # these are the 20C values.
    'Al':2.65*10**(-8),
    'Cu':1.68*10**(-8),
}
```

```
24
25 T_coeffs = { # temperatue coefficients [1/K]
26     'Al':0.00390,
27     'Cu':0.00380,
28 }
29
30 permeability = {
31     'mu0':4*pi*10**(-7),
32     'Al':1.256665*10**(-6),
33     'Cu':1.256629*10**(-6)
34
35 }
```

D.3.4 display_models.py

This script is a graphing utility for visualizing the reactor wiht either a turns-in-profile view or a 3D view of the reactor as a set of parallel sheets.

```
1 """
2
3 Wire Parameter Values, and functions for useful stuff therein
4
5 """
6 from numpy import pi
7
8 AWG_diam = { # AWG diameters in millimeters
9     '4/0':11.684, '3/0':10.4049, '2/0':9.2658, '1/0':8.2515,
10      1:7.3481,  2:6.5437,  3:5.8273,  4:5.1894,  5:4.6213,  6:4.1154,
11      7:3.6649,  8:3.2636,  9:2.9064, 10:2.5882, 11:2.3048, 12:2.0525,
12     13:1.8278, 14:1.6277, 15:1.4495, 16:1.2908, 17:1.1495, 18:1.0237,
13     19:0.9116, 20:0.8118, 21:0.7229, 22:0.6438, 23:0.5733, 24:0.5106,
14     26:0.4049, 27:0.3606, 28:0.3211, 29:0.2859, 30:0.2546, 31:0.2268,
15     32:0.2019, 33:0.1798, 34:0.1601, 35:0.1426, 36:0.1270, 37:0.1131,
16     38:0.1007, 39:0.0897, 40:0.0799,
17 }
18
19 resistivity = { # Ohm-m^2/m, use by dividing by the area of the wire
20     # these are the 20C values.
21     'Al':2.65*10**(-8),
22     'Cu':1.68*10**(-8),
23 }
24
25 T_coeffs = { # temperatue coefficients [1/K]
26     'Al':0.00390,
27     'Cu':0.00380,
28 }
```

```
29
30 permeability = {
31     'mu0':4*pi*10**(-7),
32     'Al':1.256665*10**(-6),
33     'Cu':1.256629*10**(-6)
34
35 }
```

D.4 Illustration Reactor

This python program produces the data used in the 4-turn 2-layer "2-element" reactor in section 3.8.1. The program also gives a comparison between turn-to-turn calculation method and the methods presented by Fawzi and Burke.

```
1 #!/bin/python3
2 """ illustration_reactor.py """
3 from biot_savart_methods import bs_mutual, internal
4 from fawzi_and_burke_methods import fb_mutual, fb_mutual_thick, fb_self_ind
5 from util import vector_formatter, str_matrix, coupling_coef
6 from tex_util import tex_report, save_matrix_as_tex
7 import numpy as np
8 from display_models import turn_model
9
10 rho_cu = 1.68*10**(-8) # resistivity of copper [Ohm * m^2/m]
11 mm2m = 10**(-3) # millimeters to meters
12
13 testing_freq = 60_000 # Hz
14
15 # r_cond = [0.5106*mm2m*0.5]+[0.3211*mm2m*0.5]*2 # conductor radius ~24awg,28 [meters
       ]
16 r_cond = 0.25*mm2m # conductor radius ~24awg[meters]
17 c_ins = 0.01*mm2m # insulation on enameled wire
18
19 d = 2*(r_cond + c_ins)
20
21 R_loop1 = 0.1
22 R_loop2 = 0.103
23
24 sp = check_for_path('./results/illustration_reactor.tex',True) # savepath
25
26 def t2t_illustration():
27     l11 = bs_mutual(a=R_loop1,b=(R_loop1-r_cond),d=0)
28     l22 = l11
29     l33 = bs_mutual(a=R_loop2,b=(R_loop2-r_cond),d=0)
```

```
    l44 = l33

    m12 = bs_mutual(a=R_loop1,b=(R_loop1-r_cond),d=d)
    m13 = bs_mutual(a=R_loop1,b=(R_loop2-r_cond),d=0)
    m14 = bs_mutual(a=R_loop1,b=(R_loop2-r_cond),d=d)

    m23 = bs_mutual(a=R_loop1,b=(R_loop2-r_cond),d=-d)
    m24 = bs_mutual(a=R_loop1,b=(R_loop2-r_cond),d=0)

    m34 = bs_mutual(a=R_loop2,b=(R_loop2-r_cond),d=d)

    L_4x4 = np.array([  [l11,m12,m13,m14],
                        [m12,l22,m23,m24],
                        [m13,m23,l33,m34],
                        [m14,m24,m34,l44]  ])

    K_4x4 = coupling_coef(L_4x4)

    print(str_matrix(L_4x4,label='L_4x4 ='))
    print(str_matrix(K_4x4,label='K_4x4 =', as_decimal=True))

    # Simplify to a 2x2 inductance matrix:

    L_1 = l11 + l22 + 2*m12
    L_2 = l33 + l44 + 2*m34
    M12 = m13 + m14 + m23 + m24

    L_2x2 = np.array([  [L_1, M12],
                        [M12, L_2]])

    K_2x2 = coupling_coef(L_2x2)

    print(str_matrix(L_2x2,label='L_2x2 ='))
    print(str_matrix(K_2x2,label='K_2x2 =', as_decimal=True))

    save_matrix_as_tex(M=L_4x4,mat_label='L_4x4',savepath=sp,description='4x4 L
    matrix using t2t',write_mode='w',sigfig=3,as_decimal=False)
    save_matrix_as_tex(M=K_4x4,mat_label='K_4x4',savepath=sp,description='4x4 K
    matrix using t2t',write_mode='a',sigfig=3,as_decimal=True)
    save_matrix_as_tex(M=L_2x2,mat_label='L_2x2',savepath=sp,description='2x2 L
    matrix using t2t',write_mode='a',sigfig=3,as_decimal=False)
    save_matrix_as_tex(M=K_2x2,mat_label='K_2x2',savepath=sp,description='2x2 K
    matrix using t2t',write_mode='a',sigfig=3,as_decimal=True)

    return L_2x2
```

```
71
72 def sheet_equiv():
73     h = d + 2*r_cond
74     L_1 = fb_mutual(N1=2,R1=R_loop1,h1=h,
75                     N2=2,R2=R_loop1-r_cond,h2=h,s=0)
76     L_2 = fb_mutual(N1=2,R1=R_loop2,h1=h,
77                     N2=2,R2=R_loop2-r_cond,h2=h,s=0)
78     M12 = fb_mutual(N1=2,R1=R_loop1,h1=h,
79                     N2=2,R2=R_loop2-r_cond,h2=h,s=0)
80
81     L_sheet = np.array([[L_1, M12],
82                         [M12, L_2]])
83
84     K_sheet = coupling_coef(L_sheet)
85
86     print(str_matrix(L_sheet,label='L_sheet ='))
87     print(str_matrix(K_sheet,label='K_sheet =', as_decimal=True))
88
89     save_matrix_as_tex(M=L_sheet,mat_label='L_sheet',savepath=sp,description='2x2 L
       matrix using F&B sheets',write_mode='a',sigfig=3,as_decimal=False)
90     save_matrix_as_tex(M=K_sheet,mat_label='K_sheet',savepath=sp,description='2x2 K
       matrix using F&B sheets',write_mode='a',sigfig=3,as_decimal=True)
91
92     return L_sheet
93
94 if __name__ == '__main__':
95     L_t2t = t2t_illustration()
96     L_sheet = sheet_equiv()
97
98     print("Compare t2t with sheet method: L_t2t - L_sheet:")
99     print(str_matrix(L_t2t - L_sheet,label='L_t2t - L_s =', as_decimal=True))
100    print(str_matrix(L_t2t - L_sheet,label='L_t2t - L_s ='))
101    L_diff = 100*(L_t2t - L_sheet)/L_t2t
102    print(str_matrix(L_diff,label='L_t2t - L_s (% diff) =', as_decimal=True))
103
104    save_matrix_as_tex(M=L_t2t - L_sheet,mat_label='L_diff',savepath=sp,
105                       description='difference between t2t and F&B sheets',
       write_mode='a',sigfig=3,as_decimal=False)
106    save_matrix_as_tex(M=L_diff,mat_label='L_diff',savepath=sp,description='
       difference between t2t and F&B sheets',write_mode='a',sigfig=3,as_decimal=True)
```

D.5 41 Turn Example

This is a script to demonstrate the use and computation accuracy of the varying inductance calculation methods.

```
#!/bin/python3
""" compare_41turn.py """
import numpy as np
from fawzi_and_burke_methods import fb_self_ind, fb_mutual, fb_mutual_thick
from util import vector_formatter, cofactor
import matplotlib.pyplot as plt

from biot_savart_methods import bs_mutual, internal
from display_models import sheet_model, turn_model

rho_cu = 1.68*10**(-8) # resistivity of copper [Ohm * m^2/m]
mm2m = 10**(-3) # millimeters to meters

turns = 41
r_loop = 50*mm2m * 0.5 # loop radius [meters]
r_cond = 0.5106*mm2m * 0.5 # conductor radius ~24awg [meters]
c_ins = 0.025187*mm2m # insulation on enameled wire
height = 23*mm2m#turns*(r_cond+c_ins)*2-2*c_ins*0 # take the top and bottom insualtion off the height...
                                        # b/c it doesn't have an effect on the magnetic properties
print(f'turns={turns}, r_loop={r_loop}, r_cond={r_cond}, height={height:.5f}')
def t2t_reactor():

    z_inc = height/(turns-1)# 2*(r_cond*c_ins) # z increment (conductor diameter)

    Lm = [] # inductance values for increasing turns distance. 0th index being the nearest adjacent turn
    for k in range(0,turns):
        m = bs_mutual(a=(r_loop),b=(r_loop-r_cond),d=k*z_inc)
        # print(f'k={k} d={k*z_inc}: m={m}')
        Lm.append(m)
    # print(len(Lm))

    L_int = internal(R=r_loop,r=r_cond) # flux internal to the conductor

    # account for the self and internal inductanve of each turn:
    L = turns*(Lm[0] + L_int)

    # apply superposition:
    for k in range(1,turns):
```

```
        # print(len(Lm[1:turns-k+1]))
        L += 2*np.sum(Lm[1:turns-k+1]) # double the quantity because of symmetry

    return L

def sheet_reactor():

    # L = fb_self_ind(N=turns,R=r_loop,h=(turns)*r_cond*2)
    # L = fb_self_ind(N=turns,R=r_loop,h=height)
    L = fb_mutual(N1=turns,R1=r_loop,h1=height,
                  N2=turns,R2=r_loop-r_cond,h2=height,s=0) # concentric
    return L

def shell_reactor():

    N = turns
    R = r_loop
    h = height#(turns)*r_cond*2
    t = 2*r_cond
    L = fb_mutual_thick(N1=N,R1=R,h1=h,t1=t,
                        N2=N,R2=R,h2=h,t2=t,s=0) # concentric

    return L

if __name__ == '__main__':
    L_bs = t2t_reactor()
    L_fb = sheet_reactor()
    L_fbt = shell_reactor()

    print(f'L_bs = {L_bs*1e6:.4f} [nH]')
    print(f'L_fb = {L_fb*1e6:.4f} [nH]')
    print(f'L_fbt = {L_fbt*1e6:.4f} [nH]')
    print(f'L_bs/L_fb = {L_bs/L_fb}')
    print(f'L_bs/L_fbt = {L_bs/L_fbt}')

    # to check the results with a measurable value
    f = 10000
    w_test = f*2*np.pi; # test angular frequency
    V_term = 1.06
    R_i = 10 # current sensing resistor for measuring phase angle:
    print(f'current sensing resistor: R_i = {R_i}')
    R_L = turns*(2*np.pi*r_loop)*( rho_cu/(np.pi*r_cond**2) ) # calculate the
    resistance of the wire used to construct the reactor

    Z_bs = R_L+R_i + L_bs*w_test*1j
```

```
    Z_fb = R_L+R_i + L_fb*w_test*1j
    Z_fbt = R_L+R_i + L_fbt*w_test*1j

    print(f'freq = {f}[Hz]')
    print(f'Z_bs  = {Z_bs:.4f} [\u03a9]')
    print(f'Z_fb  = {Z_fb:.4f} [\u03a9]')
    print(f'Z_fbt = {Z_fbt:.4f} [\u03a9]')

    I_bs = (V_term/(Z_bs))
    I_fb = (V_term/(Z_fb))
    I_fbt = (V_term/(Z_fbt))

    print('I_bs =',vector_formatter(I_bs))
    print('I_fb =',vector_formatter(I_fb))
    print('I_fbt =',vector_formatter(I_fbt))

    # turn_model(turns=[turns],radii=[r_loop],heights=[height],r_conds=[r_cond],s
    =[0], c=['#000'],lw=0.5)
    # sheet_model(radii=[r_loop],heights=[turns*r_cond*2],s=[0],c=['#000'],a=[.2])
    # plt.savefig('./figs/turnModel_41t_r50mm.png',dpi=600,bbox_inches='tight')
    # plt.show()
```

D.6 41 Turn Faulted Example

This is a script to demonstrate the use and comparison between faults calculated using the two inductance calculation methods.

```
#!/bin/python3
""" compare_41turn.py """
import numpy as np
from biot_savart_methods import bs_mutual, internal
from fawzi_and_burke_methods import fb_self_ind, fb_mutual, geometry_mat, turns_mat,
    NG_scalar
from util import vector_formatter, cofactor, str_matrix, check_for_path
from tex_util import save_matrix_as_tex
import matplotlib.pyplot as plt

from display_models import sheet_model, turn_model

rho_cu = 1.68*10**(-8) # resistivity of copper [Ohm * m^2/m]
mm2m = 10**(-3) # millimeters to meters

turns = 41
r_loop = 50*mm2m * 0.5 # loop radius [meters]
```

```
r_cond = 0.5106*mm2m * 0.5 # conductor radius ~24awg [meters]
c_ins = 0.025187*mm2m # insulation on enameled wire
height = 23*mm2m#turns*(r_cond+c_ins)*2-2*c_ins*0 # take the top and bottom insualtion off the height...
                                        # b/c it doesn't have an effect on the magnetic properties
n_fault = 21 # fault index

print(r_cond)

sp = './results/single_layer_test_fault.tex'

print(f'turns={turns}, r_loop={r_loop}, r_cond={r_cond}, height={height:.5f}, faulted turn:{n_fault}')

def t2t_reactor():
    L = np.zeros((2,2))
    z_inc = height/(turns-1)# 2*(r_cond*c_ins) # z increment (conductor diameter)

    Lm = [] # inductance values for increasing turns distance. 0th index being the nearest adjacent turn
    for k in range(0,turns):
        m = bs_mutual(a=(r_loop),b=(r_loop-r_cond),d=k*z_inc)
        # print(f'k={k} d={k*z_inc}: m={m}')
        Lm.append(m)
    # print(len(Lm))

    L_int = internal(R=r_loop,r=r_cond) # flux internal to the conductor

    # account for the self and internal inductanve of each turn:
    L[0,0] = turns*(Lm[0] + L_int)

    # apply superposition:
    for k in range(1,turns):
        # print(len(Lm[1:turns-k+1]))
        L[0,0] += 2*np.sum(Lm[1:turns-k+1]) # double the quantity because of symmetry

    m_1f = 2*np.sum(Lm[1:n_fault]) # get the mutuals from the center turn to the rest
    L[0,1] = m_1f # these are the off-diagonals
    L[1,0] = m_1f # also off-diagonal
    L[1,1] = Lm[0] + L_int # the self inductance is the same for a fault turn or a layer turn
    L[0,0] = L[0,0]-(Lm[0] + L_int + 2*m_1f) # remove the mutual and self of the fault from the layer
    return L
```

```
56
57
58 def sheet_reactor():
59     N = turns_mat([turns,1])
60     G = geometry_mat(radii=[r_loop]*2,h=[height,(height/turns)],h_frac=[1,1],z=0)
61     print(str_matrix(N,label='N ='))
62     print(str_matrix(G,label='G ='))
63
64     L = NG_scalar * N * G
65
66     print(str_matrix(L,label='L ='))
67     L[0,0] = L[0,0] - (L[1,1] + 2*L[0,1])
68     print(str_matrix(L,label='L ='))
69
70     return L
71
72
73 if __name__ == '__main__':
74     L_bs = t2t_reactor()
75     L_fb = sheet_reactor()
76
77     print(str_matrix(L_bs*1e6, label='L_bs [nH] =', as_decimal=False))
78     print(str_matrix(L_fb*1e6, label='L_fb [nH] =', as_decimal=False))
79
80     # to check the results with a measurable value
81     f = 10000
82     w_test = f*2*np.pi; # test angular frequency
83     V_term = 1.06
84     R_i = 10 # current sensing resistor for measuring phase angle:
85     print(f'current sensing resistor: R_i = {R_i}')
86     R_L = turns*(2*np.pi*r_loop)*( rho_cu/(np.pi*r_cond**2) ) # calculate the
       resistance of the wire used to construct the reactor
87     R_mat = np.array([[R_L*((turns-1)/turns)+R_i,0],[0,R_L*(1/turns)]])
88     Z_bs = R_mat + L_bs*w_test*1j
89     Z_fb = R_mat + L_fb*w_test*1j
90
91     print(f'freq = {f}[Hz]')
92     print(str_matrix(Z_bs, label='Z_bs [\u03a9] ='))
93     print(str_matrix(Z_fb, label='Z_fb [\u03a9] ='))
94
95     V = np.array([[V_term],[0]])
96
97     I_bs = np.matmul(np.linalg.inv(Z_bs),V)
98     I_fb = np.matmul(np.linalg.inv(Z_fb),V)
99
```

```
    print(str_matrix(I_bs, label='I_bs [A] =', phasor=True))
    print(str_matrix(I_fb, label='I_fb [A] =', phasor=True))

    turn_model(turns=[turns,1],radii=[r_loop]*2,heights=[height,0],r_conds=[r_cond
    ]*2,s=[0]*2, c=['#000','#800'],lw=[0.5,1])
    sheet_model(radii=[r_loop]*2,heights=[turns*r_cond*2, 2*r_cond],s=[0,0],c=['#000'
    ,'#f00'],a=[.2,.8])
    plt.show()
```

D.7 3-Layer Example

This script is used in the simulaiton of the physical 3-layer test reactors with the tuned and untuned variants.

```
#!/bin/python3
from biot_savart_methods import bs_mutual, internal
from fawzi_and_burke_methods import fb_mutual, fb_mutual_thick, fb_self_ind,
    geometry_mat, turns_mat, NG_scalar
from util import vector_formatter, str_matrix
from tex_util import save_matrix_as_tex, save_as_tex
import numpy as np
from display_models import turn_model, sheet_model
import matplotlib.pyplot as plt

rho_cu = 1.68*10**(-8) # resistivity of copper [Ohm * m^2/m]
mm2m = 10**(-3) # millimeters to meters

testing_freq = 10_000 # Hz

# r_cond = [0.5106*mm2m*0.5]+[0.3211*mm2m*0.5]*2 # conductor radius ~24awg,28 [meters
    ]
r_cond = [0.5106*mm2m*0.5]*4 # conductor radius ~24awg[meters]
c_ins = 0.025187*mm2m # insulation on enameled wire

def t2t_reactor(turns,radii,height,R_add):
    NN = len(turns)
    L = np.zeros((NN,NN))
    for k in range(NN):
        z_inc = height[k]/(turns[k]-1)# 2*(r_cond*c_ins) # z increment (conductor
    diameter)
        Lm = [] # inductance values for increasing turns distance. 0th index being
    the nearest adjacent turn
        for l in range(0,turns[k]):
            m = bs_mutual(a=(radii[k]),b=(radii[k]-r_cond[k]),d=l*z_inc)
```

```
            # print(f'k={k} d={k*z_inc}: m={m}')
            Lm.append(m)
        # print(len(Lm))

        L_int = internal(R=radii[k],r=r_cond[k]) # flux internal to the conductor

        # account for the self and internal inductanve of each turn:
        L_mut = turns[k]*(Lm[0] + L_int)

        # apply superposition:
        for l in range(1,turns[k]):
            # print(len(Lm[1:turns-k+1]))
            L_mut += 2*np.sum(Lm[1:turns[k]-l+1]) # double the quantity because of
symmetry

        L[k,k] = L_mut # insert diagonal elements

    # calculate the off-diagonal elements (mutuals)
    for k in range(0,NN):
        ta = turns[k]
        rca = r_cond[k]
        z_inc_a = height[k]/(ta-1)# 2*(r_cond*c_ins) # z increment (conductor
diameter)
        for j in range(k+1,NN):
            print(f'({k},{j})')
            tb = turns[j]
            rcb = r_cond[j]
            z_inc_b = height[j]/(tb-1)# 2*(r_cond*c_ins) # z increment (conductor
diameter)
            Lm = 0 # inductance values for increasing turns distance. 0th index being
 the nearest adjacent turn
            ct = 0 # counter for debugging, should go to (ta*tb)-1
            for l in range(0,ta):
                for i in range(0,tb):
                    m = bs_mutual(a=(radii[k]),b=(radii[j]-rcb),d=abs(i*z_inc_b-l*
z_inc_a))
                    # print(f'{ct} [{l},{i}], d={i*z_inc_b-l*z_inc_a}: m={m}')
                    Lm += m
                    ct += 1

            L[k,j] = Lm
            L[j,k] = Lm


    X = 2j*np.pi*testing_freq*L
```

```
67
68      R = calc_R(turns,radii,R_add)
69
70      Z = np.diag(R) + X
71
72      return L,Z
73
74
75  def sheet_reactor(turns,radii,height,R_add):
76      """  """
77      # L = fb_self_ind(N=turns,R=r_loop,h=(turns)*r_cond*2)
78      L1 = fb_self_ind(N=turns[0],R=radii[0],h=height[0])
79      L2 = fb_self_ind(N=turns[1],R=radii[1],h=height[1])
80      L3 = fb_self_ind(N=turns[2],R=radii[2],h=height[2])
81
82      L12 = fb_mutual(N1=turns[0],R1=radii[0],h1=height[0],
83                      N2=turns[1],R2=radii[1],h2=height[1],s=0)
84      L13 = fb_mutual(N1=turns[0],R1=radii[0],h1=height[0],
85                      N2=turns[2],R2=radii[2],h2=height[2],s=0)
86      L23 = fb_mutual(N1=turns[1],R1=radii[1],h1=height[1],
87                      N2=turns[2],R2=radii[2],h2=height[2],s=0)
88
89      L = np.array([[L1,L12,L13],[L12,L2,L23],[L13,L23,L3]])
90
91      X = 2j*np.pi*testing_freq*L
92
93      R = calc_R(turns,radii,R_add)
94
95      Z = np.diag(R) + X
96
97      return L,Z
98
99  def sheet_reactor_faulted(turns,radii,height,R_add):
100     """  """
101     # L = fb_self_ind(N=turns,R=r_loop,h=(turns)*r_cond*2)
102     # L1 = fb_self_ind(N=turns[0],R=radii[0],h=height[0])
103     # L2 = fb_self_ind(N=turns[1],R=radii[1],h=height[1])
104     # L3 = fb_self_ind(N=turns[2],R=radii[2],h=height[2])
105     #
106     # L12 = fb_mutual(N1=turns[0],R1=radii[0],h1=height[0],
107     #                 N2=turns[1],R2=radii[1],h2=height[1],s=0)
108     # L13 = fb_mutual(N1=turns[0],R1=radii[0],h1=height[0],
109     #                 N2=turns[2],R2=radii[2],h2=height[2],s=0)
110     # L23 = fb_mutual(N1=turns[1],R1=radii[1],h1=height[1],
111     #                 N2=turns[2],R2=radii[2],h2=height[2],s=0)
```

```
    #
    # L = np.array([[L1,L12,L13],[L12,L2,L23],[L13,L23,L3]])
    N = turns_mat(turns+[1])
    G = geometry_mat(radii=radii+[radii[0]],h=height+[height[0]])
    L = NG_scalar * N * G

    X = 2j*np.pi*testing_freq*L

    R = calc_R(turns+[1],radii+[radii[0]],R_add+[0])

    Z = np.diag(R) + X

    return L,Z

def calc_R(turns,radii,R_add):
    R = [0]*len(turns)
    print(turns)
    print(radii)
    print(R_add)
    for k,t in enumerate(turns):
        ll = (2*np.pi*radii[k])
        ca = rho_cu/(np.pi*r_cond[k]**2)
        r = t*( ca )*ll + R_add[k]
        print(f'k={k},R_k={r}')
        R[k] = r

    return R

sp = './results/multilayer_faults_combined.tex'

def analysis(Z,V_term,R_i,nF=0,wm='a',ss='a'):
    """ perform analysis on the matrix """
    cZ = np.linalg.cond(Z)
    print(str_matrix(Z,label=f'Z_{ss} ='))
    print('condition of Z: cond(d) =',cZ)

    Y = np.linalg.inv(Z)
    print(str_matrix(Y,label=f'Y_{ss} ='))
    V = V_term * np.ones((Z.shape[0],1))
    if nF > 0:
        V[-nF:] = 0
    print(str_matrix(V,label=f'V_{ss} ='))
    I = np.matmul(Y,V)
    V_Ri = np.array([R_i]).T * I
    print(str_matrix(I,label=f'I_{ss} =',phasor=True))
```

```
157     print(str_matrix(V_Ri,label='V_Ri =',phasor=True))
158 
159     It = np.sum(I[:-nF]) if nF > 0 else np.sum(I)
160     # It = np.sum(I)
161     Zt = V_term / It
162     print(f'I_{ss}t =', It,'=', vector_formatter(It))
163     print(f'Z_{ss}t =', Zt,'=', vector_formatter(Zt))
164 
165     # save_matrix_as_tex(M=L,mat_label=f'L_{ss}',savepath=sp,description='',
    write_mode=wm,sigfig=4,as_decimal=False)
166     save_matrix_as_tex(M=Z,mat_label=f'Z_{ss}',savepath=sp,description='impedance
    matrix',write_mode=wm,sigfig=4,as_decimal=False)
167     save_matrix_as_tex(M=I,mat_label=f'I_{ss}',savepath=sp,description='current
    vector',write_mode='a',sigfig=4,as_decimal=False)
168     save_as_tex(M=It,label=f'I_total{ss}',savepath=sp,description='total current',
    write_mode='a',sigfig=4)
169     save_as_tex(M=Zt,label=f'Z_total{ss}',savepath=sp,description='total impedance',
    write_mode='a',sigfig=4)
170     # save_matrix_as_tex(M=Z_bs,mat_label=f'Z_{ss}',savepath=sp,description='',
    write_mode='a',sigfig=4,as_decimal=False)
171 
172 if __name__ == '__main__':
173     untuned_turns = [41]*3
174     # tuned_turns = [41,37,37] # all 24awg, for 1 ohm R_sense
175     tuned_turns = [41,37,37] # all 24awg, for 10 ohm R_sense
176     # tuned_turns = [41,35+0,35-1] # 24,28,28 awg
177     radii = [mm2m*d*.5 for d in [50,54,58]]
178     Ri = [10]*3 # current sensing resistors
179 
180     untuned_heights = [t*(r_cond[0]+c_ins)*2-2*c_ins for i,t in enumerate(
    untuned_turns)]
181     tuned_heights = [t*(r_cond[i]+c_ins)*2-2*c_ins for i,t in enumerate(tuned_turns)]
182 
183     # L_untuned,Z_untuned = sheet_reactor(untuned_turns,radii,untuned_heights,R_add=
    Ri)
184     # L_tuned,Z_tuned = sheet_reactor(tuned_turns,radii,tuned_heights,R_add=Ri)
185     # L_untuned_f,Z_untuned_f = sheet_reactor_faulted(untuned_turns,radii,
    untuned_heights,R_add=Ri)
186     # L_tuned_f,Z_tuned_f = sheet_reactor_faulted(tuned_turns,radii,tuned_heights,
    R_add=Ri)
187     #
188     # print(str_matrix(L_untuned,label='L_untuned ='))
189     # analysis(Z=Z_untuned,V_term=0.66,R_i=Ri,wm='w',ss='ut')
190     #
191     # print(str_matrix(L_untuned_f,label='L_untuned_f ='))
```

```
    # analysis(Z=Z_untuned_f,V_term=0.66,R_i=Ri+[0],nF=1,ss='utf')
    #
    # print(str_matrix(L_tuned,label='L_tuned ='))
    # # print(str_matrix(Z_tuned,label='Z_tuned ='))
    # analysis(Z=Z_tuned,V_term=0.62,R_i=Ri,ss='t')
    #
    # print(str_matrix(L_untuned_f,label='L_untuned_f ='))
    # analysis(Z=Z_untuned_f,V_term=0.66,R_i=Ri+[0],nF=1,ss='tf')

    # plt = turn_model( turns=tuned_turns,
    #             radii=radii,
    #             heights=tuned_heights,
    #             r_conds=r_cond,
    #             s=[0]*3,
    #             c=['#000']*3,
    #             lw=0.3)
    # plt.show()

    # L_t2t_ut, Z_t2t_ut = t2t_reactor(untuned_turns,radii,untuned_heights,R_add=Ri)
    # print(str_matrix(L_t2t_ut,label='L_t2t_untuned ='))
    # print(str_matrix(Z_t2t_ut,label='Z_t2t_untuned ='))
    # analysis(Z=Z_t2t_ut,V_term=0.66,R_i=Ri)
    #
    # L_t2t_t, Z_t2t_t = t2t_reactor(tuned_turns,radii,tuned_heights,R_add=Ri)
    # print(str_matrix(L_t2t_t,label='L_t2t_tuned ='))
    # print(str_matrix(Z_t2t_t,label='Z_t2t_tuned ='))
    # analysis(Z=Z_t2t_t,V_term=0.62,R_i=Ri)

    # prefault plots
    turn_model(turns=untuned_turns,radii=radii,
                heights=untuned_heights,r_conds=r_cond,
                s=0, c=['#000']*3,lw=[0.5]*3)
    plt.savefig('./figs/untuned_mutlilayer_pref.png',dpi=600,bbox_inches='tight')

    turn_model(turns=tuned_turns,radii=radii,
                heights=tuned_heights,r_conds=r_cond,
                s=0, c=['#000']*3,lw=[0.5]*3)
    plt.savefig('./figs/tuned_mutlilayer_pref.png',dpi=600,bbox_inches='tight')


    # fAULTED PLOTs
    # turn_model(turns=untuned_turns+[1],radii=radii+[radii[0]],
    #             heights=untuned_heights+[0],r_conds=r_cond,
    #             s=0, c=['#000']*3+['#a00'],lw=[0.5]*3+[1])
    # plt.savefig('./figs/untuned_mutlilayer_f21.png',dpi=600,bbox_inches='tight')
```

```
    #
    # turn_model(turns=tuned_turns+[1],radii=radii+[radii[0]],
    #             heights=tuned_heights+[0],r_conds=r_cond,
    #             s=0, c=['#000']*3+['#a00'],lw=[0.5]*3+[1])
    # plt.savefig('./figs/tuned_mutlilayer_f21.png',dpi=600,bbox_inches='tight')
    #
    # sheet_model(radii=radii+[radii[0]],
    #             heights=[t*r_cond[0]*2 for t in untuned_turns]+[2*r_cond[0]],
    #             s=[0]*4,c=['#000']*3+['#f00'],a=[.2]*3+[.8])
    # plt.savefig('./figs/untunedMultilayer_sheetModel_f21.png',dpi=600,bbox_inches='tight')
    #
    # sheet_model(radii=radii+[radii[0]],
    #             heights=[t*r_cond[0]*2 for t in tuned_turns]+[2*r_cond[0]],
    #             s=[0]*4,c=['#000']*3+['#f00'],a=[.2]*3+[.8])
    # plt.savefig('./figs/tunedMultilayer_sheetModel_f21.png',dpi=600,bbox_inches='tight')

    plt.show()
```

D.8 Model Reactor and Faults

This script is the program used to generate the results for this thesis, and extended results data given in appendix C

```
#!/bin/python3

import numpy as np
from util import str_matrix, vector_formatter, increment_name, cofactor,
    coupling_coef
from fawzi_and_burke_methods import Ci_1#, Ci_2, Ci_same_R, Cb
from fawzi_and_burke_methods import geometry_mat, turns_mat
# from reactor_behavior.component_matrix_modeling import geometry_mat, turns_mat
import matplotlib.pyplot as plt

# from simple_behavior import resistances, base_definition

import sys

from tex_util import tex_report, save_matrix_as_tex, save_as_tex

V_term = 238_000 / 4
print(f'V-term = {V_term}')

```

```
## parameters from tuning in simple_behavior.py
turns = [1327, 1187, 1093, 1029, 985, 956, 938, 929, 928, 935] # good enough from
    tuning after fixing coupling.
radii = np.linspace(0.7,1.15,10) # radii of the reactor layers
RH = 3.1 # reactor height
r_cond = 0.001184 # conductor radii (for some resistance uses)
# R_dc_eq=8 # target rquivelant resistance for the reactor
R_dc_eq=0.9 # target rquivelant resistance for the reactor
G = geometry_mat(radii,RH)
mu0 = 4*np.pi*10**(-7)

def geometry_mat(radii, h, h_frac=1, z=0):
    """
    calculate Ci (compitationally intense process) for each radii, using

    radii: 1d list of radius values
    h : height of the layers, single value or list (for turns density)
    h_frac : fraction of the height (used for actual fault height, otherwise 1)
    z : concentric seperation, z=0 is at h/2 (centered vertically)
    """
    nr = len(radii)
    # print('geometry_mat : radii =',radii,len(radii))
    Cr = np.zeros((nr,nr))

    if not type(h)==list:
        h = [h] * len(radii)

    if not type(h_frac)==list:
        h_frac = [h_frac] * len(radii)

    # print(f'package heights = {h} [m]')

    if not type(z)==list:
        z = [z]*len(radii)


    for i in range(nr):
        for j in range(i,nr):
            s = abs(z[i] - z[j])

            l1 = 0.5*h[i] * h_frac[i]
            l2 = 0.5*h[j] * h_frac[j]
            z1 = (l1 + l2 + s)
            z2 = (l1 - l2 + s)
            z3 = (-l1 - l2 + s)
```

```
            z4 = (-l1 + l2 + s)
            R1 = radii[i]
            R2 = radii[j]

            C_z1 = Ci_1(R1,R2,z1)
            C_z2 = Ci_1(R1,R2,z2)
            C_z3 = Ci_1(R1,R2,z3)
            C_z4 = Ci_1(R1,R2,z4)

            c = ( 1/(h[i]*h[j]) ) * ( (R1*R2)**(3/2) ) * ( (C_z1 - C_z2) + (C_z3 - C_z4) )
            Cr[i,j] = c
            Cr[j,i] = c

    return Cr

def fault_test(fault_layer=[0], fault_z=[0]):
    """
    Parameters:
    - fault_layer : list of layers to place a fault in
    - fault_z     : list of seperation from z=0 (center of reactor, aka RH/2) of the fault

in what way does this work?
1) Lump parameters around the fault to minimize computation and condition:
[pack 1] ... [fault package] ... [pack n]

2) generate a geometry and turns matrix, check coupleing coeff and condition

3) calcualte resistance by defining the resistance at the terminals and scaling it by (turns)/(avg. turns)

4) generate Z matrix: R + 120j*pi*L

5) perturb the minimal matrix
    Note: the mutual between the short and the fault will appear larger than
          that between the fault and the pakcage it occurs in because the parameters are lumped.
    """
    # fault_layer = 0
    mod_radii = list(radii)
    mod_turns = turns
    nl = len(mod_radii)

    avg_turns = np.sum(mod_turns)/len(mod_turns)
```

```
104     R = [ R_dc_eq * t/avg_turns for t in mod_turns] # resistace array, based on DC
        equiv.
105     # print(R)
106
107     fault_turns = []
108     fault_radii = []
109     fault_z_off = []
110     fault_h_frac = []
111     fault_R_values = []
112     faulted_R = R+[]
113     for k,fl in enumerate(fault_layer):
114         fault_turns.append(1)#turns[fl])
115         fault_radii.append(radii[fl])
116         fault_z_off.append(fault_z[k])
117         fault_h_frac.append(1/turns[fl])
118
119         # Resistance values:
120         r_turn = R_dc_eq/avg_turns # resistance of a single turn (approx resistance
        of a fault)
121         fault_R_values.append(r_turn)
122         faulted_R[fl] += r_turn
123
124
125     # print('fault_radii =',fault_radii)
126     # print('fault_turns =',fault_turns)
127     # print('fault_z_off =',fault_z_off)
128     # print('fault_z_frac =',fault_h_frac)
129     #
130     # print('fault_R_values =',fault_R_values)
131     # print('faulted_R =', faulted_R)
132
133     nf = len(fault_turns)
134     Gp = geometry_mat(radii = mod_radii + fault_radii,
135                       h = ([RH]*nl) + ([RH*fh for fh in fault_h_frac]),#([RH]*nf),
136                       h_frac = ([1]*nl) + ([1]*nf),#fault_h_frac,
137                       z = ([0]*nl)+fault_z_off)
138
139     # print(str_matrix(Gp,label='Gp ='))
140
141     # sys.exit() # Test break ----------------------------------------------------
142
143     N = turns_mat(mod_turns)
144     L = (2*np.pi*mu0)*N*G
145     # L = base_definition(turns)
146     # print(str_matrix(G,label='G ='))
```

```
147     # print(str_matrix(N,label='N =', as_decimal=True))
148     # print(str_matrix(L,label='L =', as_decimal=True))
149     # print(str_matrix(coupling_coef(L),label='K =', as_decimal=True))
150
151
152     # 3 resistance: (is futile)
153     # R_dc_eq = 0.9
154
155     # 4) Z matrix:
156     Z = np.diag(R) + 120j * np.pi * L
157     Y = np.linalg.inv(Z)
158     # I = np.sum(Y,axis=1)
159     # V = np.array([[1]*10]).T
160     V = np.array([[V_term]*10]).T
161     I = np.matmul( Y, V )
162     I_T = np.sum(I)
163     # print(str_matrix(Z,label='Z ='))
164     # print(str_matrix(Y,label='Y =',phasor=True))
165
166
167     # 5) perturbation:
168     # print('fault radii =',fault_radii)
169     # mod_radii.append(fault_radii)
170     # input(f'{mod_radii}  {len(mod_radii)}')
171     # Gp = geometry_mat(mod_radii + [fault_radii*1.000],[RH]*len(mod_radii) + [RH/
    fault_turns])
172     # Gp = geometry_mat(mod_radii,[RH]*len(mod_radii) + [RH/fault_turns])
173     Np = turns_mat(mod_turns + [1]*nf)
174     Lp = (2*np.pi*mu0)*Np*Gp
175     mat_label = [f'{k}' for k in range(nl+nf)]
176     # print(str_matrix(coupling_coef(Lp),label='Kp =', as_decimal=True))
177     # print(str_matrix(Gp,label='Gp =', row_labels=mat_label, col_labels=mat_label ))
178     # print(str_matrix(Np,label='Np =', as_decimal=True, row_labels=mat_label,
    col_labels=mat_label ))
179     # print(str_matrix(Lp,label='Lp =', as_decimal=False, row_labels=mat_label,
    col_labels=mat_label ))
180
181     # print(mod_radii + fault_radii)
182
183     Rp = faulted_R + fault_R_values
184     # print('Rp =',Rp)
185     # form the perturbed Z:
186     Zp = np.diag(Rp) + 120j * np.pi * Lp
187     # print(str_matrix(Zp,label='Zp =', as_decimal=False, row_labels=mat_label,
    col_labels=mat_label ))
```

```
188
189     # fix the self and mutual elements between the faults and the layers to reflect
        the losses in turns:
190     z_shape = Z.shape
191     print(f'Z prefault shape: {z_shape}')
192     n_f = len(fault_layer)
193     dZ = np.zeros(z_shape,dtype='complex')
194     for k in range(nf):
195         dzM = Zp[0:z_shape[0],z_shape[1]+k] # changes to be applied to the reactor
        mutuals
196         # print('dzM',dzM)
197         # print('dzM',Zp[z_shape[1]+k,0:z_shape[0]])
198         # print('insert into:', z_shape[0],fault_layer[k])
199         dZ[0:z_shape[0],fault_layer[k]] += dzM
200         if fault_layer[k] > 0:
201             dZ[fault_layer[k],0:fault_layer[k]] += dzM[0:fault_layer[k]]
202             dZ[fault_layer[k],fault_layer[k]+1:z_shape[1]] += dzM[fault_layer[k]+1:]
203         elif fault_layer[k] >= z_shape[0]-1: # if it's the last row
204             dZ[fault_layer[k],0:z_shape[1]-1] += dzM[:-1]
205         else: # it's the first row
206             dZ[fault_layer[k],1:z_shape[1]] += dzM[1:] # works
207
208         dZ[fault_layer[k],fault_layer[k]] += Zp[z_shape[0]+k,z_shape[1]+k]
209
210         # print(str_matrix(dZ,label='dZ =')) # check that all the changes are in the
        right places
211
212     Zp[0:z_shape[0],0:z_shape[1]] += -1*dZ # add the change to the faulted matrix
213         # input('hold')
214
215     print(str_matrix(Z-Zp[0:z_shape[0],0:z_shape[1]],label='Z-Zp =')) # double check
        the changes
216
217     Yp = np.linalg.inv(Zp)
218     # Ip = np.sum(Yp[:,:-1],axis=1)
219     # Vp = np.array([[1]*nl+[0]*nf]).T
220     Vp = np.array([[V_term]*nl+[0]*nf]).T
221     Ip = np.matmul( Yp, Vp )
222     Ip_T = np.sum(Ip[:nl])
223     I_labels = [f'{j}' for j in range(len(I))]
224     # print(str_matrix(Zp,label='Zp ='))
225     # print(str_matrix(Yp,label='Yp =',phasor=False))
226     # print(str_matrix(Yp,label='Yp =',phasor=True))
227     # print(str_matrix(Vp,label='Vp ='))
```

```
228     # print(str_matrix(I[0:z_shape[0]],label='I =',phasor=True, row_labels=I_labels
        +[]))
229     # print(str_matrix(I[0:z_shape[0]],label='I =',phasor=False, row_labels=I_labels
        +[]))
230     print('I_T =',vector_formatter(I_T))
231     print('Z_T =',vector_formatter(V_term/I_T),'=',V_term/I_T)
232 
233     print(str_matrix(Ip[0:z_shape[0]],label='Ip =',phasor=True,
234                      indicators=[l for l in fault_layer]))#,
235                      # row_labels=I_labels+[f'f{k}' for k in range(nf)]))
236     print(str_matrix(Ip[0:z_shape[0]],label='Ip =',phasor=False,indicators=[len(Ip)
        -1]))#,row_labels=I_labels+[f'f{k}' for k in range(nf)]))
237     print('Ip_T =',vector_formatter(Ip_T))
238     print('Ip_T =',vector_formatter(V_term/Ip_T),'=',V_term/Ip_T)
239 
240     print('average no. turns:', np.average(turns))
241 
242     return I, I_T, Ip, Ip_T, L, Z, Lp, Zp
243 
244 # ------------------------------------------------------------------------------
245 #
246 
247 def report_multifault():
248 
249     NF = 22 # number of fautls
250     FS = 2 # fault step
251     fault_z_range = 0.4 #
252     # for l in range(2):
253     I = []
254     # for l in [0,1,4,7,9]:#range(len(turns)):
255     # for l in [0,4]:#range(len(turns)):
256     for l in range(len(turns)):
257         models = {}
258         m_desc = {}
259         I_total = []
260         Ip_total = []
261         nF = []
262         for nf in [2,10,20,40,100,150,200]:#range(2,NF,FS): # number of faults
263             # I, I_T, Ip, Ip_T, L, Z, Lp, Zp = fault_test(fault_layer=[l]*nf, fault_z
        =[0])
264             I, I_T, Ip, Ip_T, L, Z, Lp, Zp = fault_test(fault_layer=[l]*nf,
265                                                         fault_z=[(RH/turns[l])*(i-.5*nf) for
        i in range(nf)])
266             print(I)
267             I_total.append(I_T)
```

```
            Ip_total.append(Ip_T)
            nF.append(nf)
            Z_T = 1/I_T
            Zp_T = 1/Ip_T
            # models[f'I_ {{p\;{l},nf={nf}}}'] = Ip
            models[f'I_{{pT\;{l},nf={nf}}}'] = Ip_T
            # models[f'Z_{{pT\;{l},nf={nf}}}'] = Zp_T
            # m_desc[f'I_{{p\;{l},nf={nf}}}'] = f'current vector for {nf} faults in layer {l}'
            m_desc[f'I_{{pT\;{l},nf={nf}}}'] = f'total current for {nf} faults in layer {l}'
            # m_desc[f'Z_{{pT\;{l},nf={nf}}}'] = f'total impednace for {nf} faults in layer {l}'

        tex_report(models,m_desc,f'./results/MultiFault/tuned_reactor_simple_multifault_layer{l}.tex',sigfig=8,as_decimal=True)
        plt.figure()
        plt.suptitle(f'Total Current, with Faults in Layer {l}')
        plt.subplot(211)
        # plt.plot(nF,np.abs(I_total),'--',label=r'$|I_{pre\,f}|$')
        plt.plot(nF,np.abs(Ip_total),'o-',label=r'$|I_{f}|$')
        plt.xticks(nF)
        plt.ylabel(r'$|I_T|$')
        plt.legend()

        plt.subplot(212)
        # plt.plot(nF,np.angle(I_total,deg=True),'--',label=r'arg($I_{pre\,f}$)')
        plt.plot(nF,np.angle(Ip_total,deg=True),'o-',label=r'arg($I_{f}$)')
        plt.xticks(nF)
        plt.xlabel('number of faults [turns]')
        plt.ylabel(r'arg($I_T$) [$^\circ$]')
        plt.legend()
        plt.savefig(f'./results/MultiFault/figs_2to250/I_vs_nF_layer{l}.png',dpi=600,bbox_inches='tight')

    plt.show()

def report_singlefault():

    NF = 22 # number of fautls
    FS = 2 # fault step
    fault_z_range = 0.4 #
    # for l in range(2):
    I = []
    # for l in [0,1,4,7,9]:#range(len(turns)):
```

```
    # for l in [0,4]:#range(len(turns)):
    models = {}
    m_desc = {}
    for l in range(len(turns)):
        I_total = []
        Ip_total = []
        Fz = []
        for fz in [-0.5,0,0.5]:#range(2,NF,FS): # number of faults
            # I, I_T, Ip, Ip_T, L, Z, Lp, Zp = fault_test(fault_layer=[l]*nf, fault_z=[0])
            I, I_T, Ip, Ip_T, L, Z, Lp, Zp = fault_test(fault_layer=[l],
                                                        fault_z=[RH*fz])
            print(I)
            I_total.append(I_T)
            Ip_total.append(Ip_T)
            Fz.append(fz)
            Z_T = 1/I_T
            Zp_T = 1/Ip_T
            # models[f'I_ {{p\;{l},nf={nf}}}'] = Ip
            models['I_{{pre\,F}}'] = I_T
            m_desc['I_{{pre\,F}}'] = f'total current for prefault conditions.'

            models[f'I_{{pT\;{l},fz={fz}}}'] = Ip_T
            # models[f'Z_{{pT\;{l},nf={nf}}}'] = Zp_T
            # m_desc[f'I_{{p\;{l},nf={nf}}}'] = f'current vector for {nf} faults in layer {l}'
            m_desc[f'I_{{pT\;{l},fz={fz}}}'] = f'total current for  fault at {fz} of reactor layer height, in layer {l}.'
            # m_desc[f'Z_{{pT\;{l},nf={nf}}}'] = f'total impednace for {nf} faults in layer {l}'


        plt.figure()
        plt.suptitle(f'Total Current, with Faults in Layer {l}')
        plt.subplot(211)
        # plt.plot(nF,np.abs(I_total),'--',label=r'$|I_{pre\,f}|$')
        plt.plot(Fz,np.abs(Ip_total),'o-',label=r'$|I_{f}|$')
        plt.xticks(Fz)
        plt.ylabel(r'$|I_T|$')
        plt.legend()

        plt.subplot(212)
        # plt.plot(nF,np.angle(I_total,deg=True),'--',label=r'arg($I_{pre\,f}$)')
        plt.plot(Fz,np.angle(Ip_total,deg=True),'o-',label=r'arg($I_{f}$)')
        plt.xticks(Fz)
```

```
349         plt.xlabel('vertical position of fault')
350         plt.ylabel(r'arg($I_T$) [$^\circ$]')
351         plt.legend()
352         plt.savefig(f'./results/SingleFault/figs/I_vs_Fz_layer{l}.png',dpi=600,
    bbox_inches='tight')
353 
354     tex_report(models,m_desc,f'./results/SingleFault/tuned_reactor_moving_singlefault
    .tex',sigfig=8,as_decimal=True)
355     # plt.show()
356 
357 # ------------------------------------------------------------------------------
358 if __name__ == "__main__":
359     # report_multifault()
360     report_singlefault()
```