A Source of Accurately Calculable Quasi-Static Magnetic Fields

A Dissertation Presented

by

Peter Traneus Anderson

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
Specializing in Electrical Engineering

October, 2001

Accepted by the Faculty of the Graduate College, The University of Vermont, in partial fulfillment of the requirements for the degree of Doctor of Philosophy, specializing in Electrical Engineering.

Dissertation Examination Committee:

_____ Advisor
Richard G. Absher, Ph.D.

_____
Stephen L. Titcomb, Ph.D.

_____
Walter J. Varhue, Ph.D.

_____ Chairperson
Martha D. Fitzgerald, Ed.D.

_____ Interim Dean,
Anne E. Huot, Ph.D. Graduate College

Date: August 24, 2001

ABSTRACT

Various areas of technology require the generation of accurately known quasi-static multi-axis dipole or quasi-dipole magnetic fields in a region of free space. Traditionally, this is accomplished using arrays of small coils of wire driven by known or accurately-measured current sources. The traditional small coils cannot be accurately constructed, so the generated fields must be laboriously characterized by measurements at various known points in the region of interest. Since the field-measuring instruments must themselves be calibrated, the measurement accuracy is problematic. A method of precisely constructing coil arrays would solve the problem by permitting the field to be calculated from knowledge of the coils' geometry and of the current flowing in the coils. Use of printed-circuit techniques, combined with the kinematic rigidity of a regular tetrahedron, provides an inexpensive means of constructing arrays of coils whose mechanical dimensions are accurately known. A tetrahedral arrangement of four coils is described. Each coil comprises a precisely located set of straight conductor segments. The magnetic field produced by each coil is accurately calculated from knowledge of the locations of the segments and of the current flowing through the coil. To test the accuracy of the generated fields, a second tetrahedron is placed in various positions and orientations in the vicinity of the first tetrahedron. The sixteen mutual inductances between two coils, one on each of the two tetrahedra, are measured, and also calculated using a closed-form mutual-inductance model for straight-line segments. The calculations and measurements are compared, and the results presented in terms of mutual-inductance error, position and orientation error, and goodness-of-fit of model to measurements.

ACKNOWLEDGMENTS

Thanks first to my advisor Richard Absher for long and patient support, and for his willingness to support a thesis topic not in his area of expertise.

Thanks to my dissertation committee members Stephen Titcomb, and Walter Varhue for their support.

Thanks to Martha Fitzgerald for being a longtime friend and supporter, for serving as the chairperson of my thesis committee, and for tutoring me in the ways of the graduate college.

Thanks to Chris Khamnei for tutoring me in the ways of the ECE department and for the use of his LaTeX style files, which provided the format structure for this work.

Thanks to James C. Krieg for decades of support and encouragement, and for initially suggesting I attempt this academic degree program.

Thanks to Adolph Gustof Anderson and Margaret Laird Anderson for supporting and encouraging my early interest in electronics and in higher education.

Thanks to G. Lee Beauregard for continual support and constructive discussions of the content of this work, for providing some of the functions used in some of the C programs, for data-collection software, and for initially suggesting the mutual-inductance approach.

Thanks to Chris Cherry for support and advice in electromagnetic theory and in numerical techniques.

Thanks to Yuan Cheng, Tina Kapur, Geza Lengyel, and Jonathan Schiff for various support.

Thanks to Polhemus, Inc. for financial support, for allowing me a flexible work schedule

built around my classes, and for the original inspiration for this area of research.

Thanks to Visualization Technology, Inc. for financial support, for the use of laboratory and computer facilities for this work, and for much encouragement and inspiration.

This work was formatted using LaTeX 2_ε.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Electromagnetic position and orientation tracking systems employ an array of accurately-calibrated transmitting coils to generate known fields, and use an accurately-calibrated field receiver at unknown position and orientation to measure the fields. The receiver measurements are then combined with a field model to calculate the receiver position and orientation with respect to the transmitter using a position-and-orientation (P&O) calculation algorithm.

To the system user, an electromagnetic tracking system comprises an electromagnetic transmitter, one or more electromagnetic receivers, and the system electronics unit. The transmitter ranges from a centimeter to a few meters in size, depending on the operating distance (range) needed. The receivers are usually a centimeter or less in size.

Electromagnetic tracking systems usually generate magnetic fields in the 100 Hz to 20 kHz frequency range. The distance between transmitter and receiver is much smaller than the wavelength, so these systems operate in the near-field region. In Appendix A, we show that the quasi-static approximation applies.

Electromagnetic tracking systems are fast enough to accurately track the motions of the objects to which the receivers are attached.

Electromagnetic tracking systems are manufactured by many companies in various parts of the world, including three companies in the Chittenden County, Vermont area: Polhemus Incorporated, Ascension Technology Incorporated, and Internav Incorporated.

1.1 Applications of electromagnetic tracking systems

In fighter aircraft, tracking the pilot's helmet permits the pilot to aim weapons by pointing her/his head at the target rather than waiting for the whole aircraft to point at the target.

In computer animation, tracking a human performer's body movements permits slaving a computer-animated character to the performer's movements. This permits the capture of realistic rapid movements which are difficult to calculate analytically. Also, "live" animated performances are possible.

In medicine, many imaging technologies exist which permit the visualization of structures and functions inside a patient. Unfortunately, almost all of these imaging technologies are too slow to produce real-time moving images, or require exposure of the patient to hazardous ionizing radiation.

Most of the benefits of real-time moving images can be gained by periodically capturing still images and then displaying the motion of interest superimposed on the still images. This is particularly true when the only motion is that of the surgeon's surgical instrument or that of a catheter being inserted into a patient.

In image-guided surgery, tracking a surgical instrument permits showing the surgeon where the instrument tip is in the patient's anatomy continuously as the surgery is performed. This permits the surgeon to avoid errors.

Similarly, tracking the tip of a catheter permits the surgeon to accurately place the tip of the catheter in the desired location.

Most of the applications of electromagnetic tracking systems can be performed also by optical tracking systems. Optical tracking systems have the disadvantage of requiring a clear line of sight from transmitter to receiver. Electromagnetic tracking systems have the disadvantage of being sensitive to conductors or ferromagnetic materials in the vicinity of the transmitter or receiver, which distort the electromagnetic fields.

1.2 Accuracy issues in electromagnetic tracking systems

Electromagnetic tracking systems require great accuracy of field generation and measurement. Accuracies of 1000 ppm (parts per million) are common, and accuracies of 100 ppm or better are needed for high-accuracy systems.

The coils in such tracking systems must be accurately calibrated against a precisely-known magnetic field.

The accurate generation or measurement of magnetic fields in this frequency range is difficult, as there are no standards or calibration services available from the National Institute of Science and Technology (NIST) for magnetic field measurements in this frequency range [1].

We describe a solution to the problem by the application of printed circuit board (PCB) techniques. PCB techniques permit the construction of mechanically precisely known coil arrays.

We demonstrate the accuracy of our solution by building two tetrahedra of PCBs, and creating a mathematical model of the mutual inductance between coils on the two tetrahedra. We verify the accuracy by comparing measured and calculated mutual inductances. Goodness-of-fit measurements verify mutual-inductance model accuracy without the need for precision mechanical fixtures. As a byproduct, we demonstrate tracking the position and orientation of one tetrahedron with respect to the other.

We derive a magnetic field model for the PCB coils. We verify the field model numerically against the mutual-inductance model by calculating the mutual inductance between each PCB coil and a small search coil using both models, and comparing results.

Chapter 2

The concept of mutual inductance

Electromagnetic systems are usually analyzed in terms of magnetic fields. Indeed, later in this paper we present a magnetic field model for our PCB coils.

However, for most of our analysis, the concept of mutual inductance is more useful.

Mutual inductance conveniently divides the system into two parts: the coils and the electronics.

Consider one transmitter coil and one receiver coil.

To the electronics, the coils are a four-terminal two-port network.

A varying current injected into one coil induces a voltage in the other coil. The induced voltage V is proportional to the rate of change of the applied current I:

$$V = L_m \frac{dI}{dt} \tag{2.1}$$

The constant of proportionality, L_m, is the mutual inductance.

The mutual inductance, L_m, between two closed circuits (in the absence of field distorters such as ferrites and conductors) is given by Neuman's formula [2] [3] [4]:

$$L_m = \frac{\mu_o}{4\pi} \oint_{circuit1} \oint_{circuit2} \frac{d\overline{s_1} \cdot d\overline{s_2}}{r_{12}} \tag{2.2}$$

where

$r_{12} = \mid \overline{s_1} - \overline{s_2} \mid$

and

$\mu_o = 4\pi 10^{-7} henries/meter$

L_m depends only on the geometry of the closed circuits (coils, in our case), and not on the elctronics which is used to measure the mutual inductance.

Strictly speaking, the closed circuit includes the coil, its connecting cable, and the electronics the cable is connected to. Proper design of the cable and electronics, permits replacement of cable and electronics with a direct short across the coil leads.

L_m is a ratio independent of applied current waveform or frequency, so is a well-defined property which can be measured precisely.

The myriad details of analyzing and ensuring that the measuring electronics are properly calibrated, are much simplified when we measure ratios rather than absolute quantities.

The electronics discussed in this paper measure the ratio of mutual inductance to a standard resistance using sinewaves of known frequencies.

2.1 Advantages of mutual-inductance perspective

The mutual inductance depends only on the physical design of the coils and the geometrical relationship between the coils, and not on the electronics used to measure the mutual inductance.

Note that the mutual inductance does not depend on which coil receives the applied current: We can interchange the current leads with the voltage leads, and will measure the same mutual inductance.

The magnetic field, on the other hand, depends on the applied current, and which coil the current is applied to, as well as on the geometry of the coils.

Most magnetic-field analysis assumes that the coil in which a voltage is induced is tiny in physical size.

Mutual inductance concepts inherently avoid this assumption. Indeed, the tracking system discussed herein uses transmitter and receiver coils which are the same size, and which are not tiny.

For such coil geometries, field-based analysis leads to endless approximations. Mutual-inductance-based analysis leads to a straightforward exact model.

Thus, most of the analysis in this paper is in terms of mutual inductance.

Chapter 3

Statement of the problem

Electromagnetic position and orientation tracking systems can be built with many transmitter coil architectures and with various types of receivers. Here, we consider only search coils as receivers, and ignore other types of receivers.

When the receiver is an array of coils, the field measurements are most conveniently recast as mutual-inductance meassurements.

The approximation that circuit 1 and circuit 2 are both tiny compared to R, is the approximation that both circuit 1 and circuit 2 are dipoles [5]. From appendix B, the mutual inductance between two dipoles is given by:

$$L_m = \frac{\mu_o A_1 A_2}{4\pi R^3} \left(3(\widehat{A_1} \cdot \widehat{R})(\widehat{R} \cdot \widehat{A_2}) - (\widehat{A_1} \cdot \widehat{A_2}) \right) \tag{3.1}$$

where

$\overline{A_1}$ = effective area of circuit 1

$\widehat{A_1}$ = unit vector in direction of $\overline{A_1}$

A_1 = magnitude of $\overline{A_1}$

$\overline{A_2}$ = effective area of circuit 2

$\widehat{A_2}$ = unit vector in direction of $\overline{A_2}$

A_2 = magnitude of $\overline{A_2}$

$\overline{R}$ = vector from circuit1 to circuit 2

$\widehat{R}$ = unit vector in direction of $\overline{R}$

R = magnitude of $\overline{R}$

3.1 Industry-standard coil architecture (ISCA)

One of the most well-known tracker coil architectures uses a three-axis dipole coil transmitter and a three-axis dipole coil receiver. We call this architecture the industry-standard coil architecture (ISCA).

Each three-axis transmitter or receiver is ideally built so the three coils exhibit the same effective area, are oriented orthogonally to one another, and are centered at the same point. Ideally, the coils are small enough, compared to the transmitter-receiver distance, that they exhibit dipole behavior.

With three transmitter coils and three receiver coils, we have nine mutual-inductance measurements. From these nine measurements, the P&O algorithm calculates three degrees of freedom of position plus three degrees of freedom of orientation. Thus, the problem is overdetermined.

3.2 Position-and-orientation (P&O) calculation algorithms

In most P&O algorithms, initial estimates, or seed values, of position and of orientation are obtained. Then the seed is used in an error-minimizing routine: The position and orientation values are substituted into a field model to calculate predicted mutual inductances. The error-minimizing routine determines the position and orientation values which minimize the error between measured and calculated mutual inductances.

In the classic ISCA algorithms, various assumptions are used to perform as much of the algorithm analytically as possible: The previous cycle's result is used as the seed, and the error-minimizing calculation is performed analytically [6] [7] [8].

More modern algorithms take advantage of modern high-speed embedded computers to perform more of the algorithm numerically, which permits relaxing the ISCA ideality assumptions [9].

The classic ISCA P&O calculations are straightforward enough that they were first performed using analog techniques only [10].

3.3 Inaccuracies of wound coils

At present, ISCA transmitters and receivers are usually arrays of three colocated orthogonal wound coils. Mechanical construction tolerances limit the accuracy of knowledge of the coils' shapes, so each coil must be empirically characterized.

Once characterized, the characterization data are used in the calculation algorithm to correct for the nonideality of each coil. The methods for this correction are beyond the scope of this paper.

The characterization is usually accomplished by measuring the mutual inductances between the coil being characterized and each of an array of accurately-known coils.

Of course, the accurately-known coils must themselves be known somehow.

One method, is to accurately build a set of Helmholtz coils, which provide relatively uniform fields at their center, and use an accurate mechanical fixture to place the coils under test at various orientations in the center of the Helmholtz coils [11].

Chapter 4

Statement of a solution to the problem

Use of printed-circuit-board (PCB) techniques permits the construction of coils with repeatable and accurately-known conductor patterns on boards whose edges are precisely milled. A set of four interlocking triangular boards are glued together to form one tetrahedral coil assembly.

The conductors are arranged in straight-line segments so a well-known analytical solution for the magnetic field generated by a straight current segment, can be used to calculate the field from each coil.

To test the accuracy of the field generated, two identical tetrahedra were built and used in a tracking system as shown in Figure 4.1. The two tetrahedra were placed in various relative positions and orientations, and the receiver tetrahedron was tracked with respect to the transmitter tetrahedron using a P&O tracking algorithm correct for this non-ISCA coil architecture.

The goodness-of-fit results of the P&O algorithm permit evaluation of the accuracy of the field generated by a tetrahedron, without the need for any precise mechanical fixturing.

The mutual inductance between each coil on the transmitter tetrahedron and each coil on the receiver tetrahedron was measured, for a total of sixteen mutual inductance measurements per position and orientation.

When modeling these mutual inductances, the use of straight-line segments simplifies the

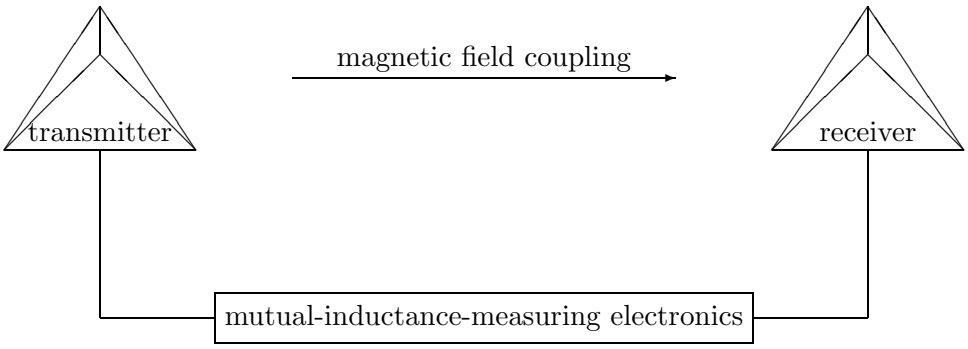

Figure 4.1: Tracking system block diagram

double integral in equation 2.2, as the first integral can be evaluated analytically. This leaves a well-behaved second integral to be evaluated numerically.

Chapter 5

Details of printed-circuit-board (PCB) coil

Each board contains one coil formed by copper tracks on the board, and the relationship among the four coils in one tetrahedron is controlled by the edges of the boards. The printed-circuit fabrication process puts tight tolerances on the track positions and on the relationship of the tracks to the edges. For boards 12.5 cM on a side as used here, normal board-fabrication tolerances of 0.01 cM give 800 ppm mechanical construction accuracy.

Figure 5.1 shows the component-side artwork, and figure 5.2 shows the solder-side artwork, of the two-sided ANT-003 board. Both artworks are viewed from the component side, as is traditional in the PCB industry.

The five round pads in the lower-left corner of the board, and the single round pad at the center of the board, each have a plated-through hole in the center of the pad.

The five pads in the lower-left corner are for mounting a miniature coaxial connector. The connector is mounted on the solder side of the board (the reverse of the usual practice) to permit the boards to be assembled into a tetrahedron with the component sides facing inward and the connectors on the outside.

The component side contains a spiral spiraling inward. The solder side contains a spiral similar to the component-side spiral, except spiraling outward. The two spirals are connected in series-aiding to the coaxial-cable connector at the lower-left corner of the board.

The edges of the board are castellated so that four boards will interlock to form a tetra-

Figure 5.1: ANT-003 component-side printed circuit board artwork

hedron.

The karet-shaped marks on the edges of the board facilitate measuring board-to-board alignment errors in the finished tetrahedron.

The square objects at the upper left and lower right are registration targets used in board fabrication.

Appendix C contains the C program used to generate the artwork files and current-segment model data for the ANT-003 board.

Appendix D contains the artwork files used to fabricate the ANT-003 boards.

Appendix E contains the current-segment model data file for the ANT-003 board.

Appendix F contains a test program used to check that the current-segment model data agrees with the artwork file data.

Figure 5.2: ANT-003 solder-side printed circuit board artwork

Chapter 6

Details of tetrahedron

Four ANT-003 boards are assembled component-side-inward into one tetrahedron. Figure 6.1 shows the tetrahedron used as a receiver. Board 0 is on the bottom, with its coax connector overhanging the front of the black mounting block. Board 3 is to the left and board 1 is to the right. Board 2 is in back and not visible. The coax connectors for boards 1, 2, 3 are grouped around the top vertex.

Figure 6.2 shows the unfolded tetrahedron. The four PCBs are shown without traces, but with pads where plated-through holes are located.

The coax-connector pads show the orientation of each board. When the tetrahedron is folded together, the coax connectors for boards 1, 2, 3 surround one vertex of the tetrahedron.

When folded, the boards are folded together component-sides-inward, and the connectors are soldered on the outside, which is the solder side.

Each tetrahedron has its own coordinate system. The tetrahedron coordinate system's X, Y, Z axes are parallel to, and point in the same direction as, the X, Y, Z axes of board 0 respectively.

Board 0 axes have their origin at the center of board 0 on the component side. The tetrahedron axes have their origin at the center of the tetrahedron. Thus the board 0 axes and the tetrahedron axes differ only by a translation along the Z axis. The X and Y axes are shown in Figure 6.2 in their proper relationship to board 0, which relationship is the same whether the tetrahedron is folded or unfolded.

Figure 6.1: Tetrahedron of four ANT-003 printed-circuit boards

The RS274X files in Appendix D use coordinates in integral numbers of mils, where 1 mil = 25.4 microns. To preserve this quantization accurately, all coordinates in this chapter are in mils.

For board 0, the board RS274X X and Y axes are parallel to, and point in the same direction as, the board 0 X and Y axes respectively. The Z axis is undefined in the RS274X files since the artwork layers are two-dimensional: A change in Z represents a jump from one artwork layer to another layer.

For board 0, the model data file `flattx3h.c` in Appendix E is in RS274X coordinates. $z = 0$ is the component side of the board, and $z = -63$ is the solder side of the board. Each line in the data file is the position of the vertex formed by the junction of two current segments.

For board 0, the board RS274X origin is offset from the board 0 origin to keep the RS274X coordinates positive all over the board:

center of board 0 component side = board 0 coord (0,0,z) = RS274X coord (3500,2443,0)

The board 0 coordinate system differs from the tetrahedron coordinate system by only a translation in Z:

center of tetrahedron = tetrahedron coord (0,0,0) = board 0 coord (0,0,$\frac{\sqrt{6}}{12}5000$)

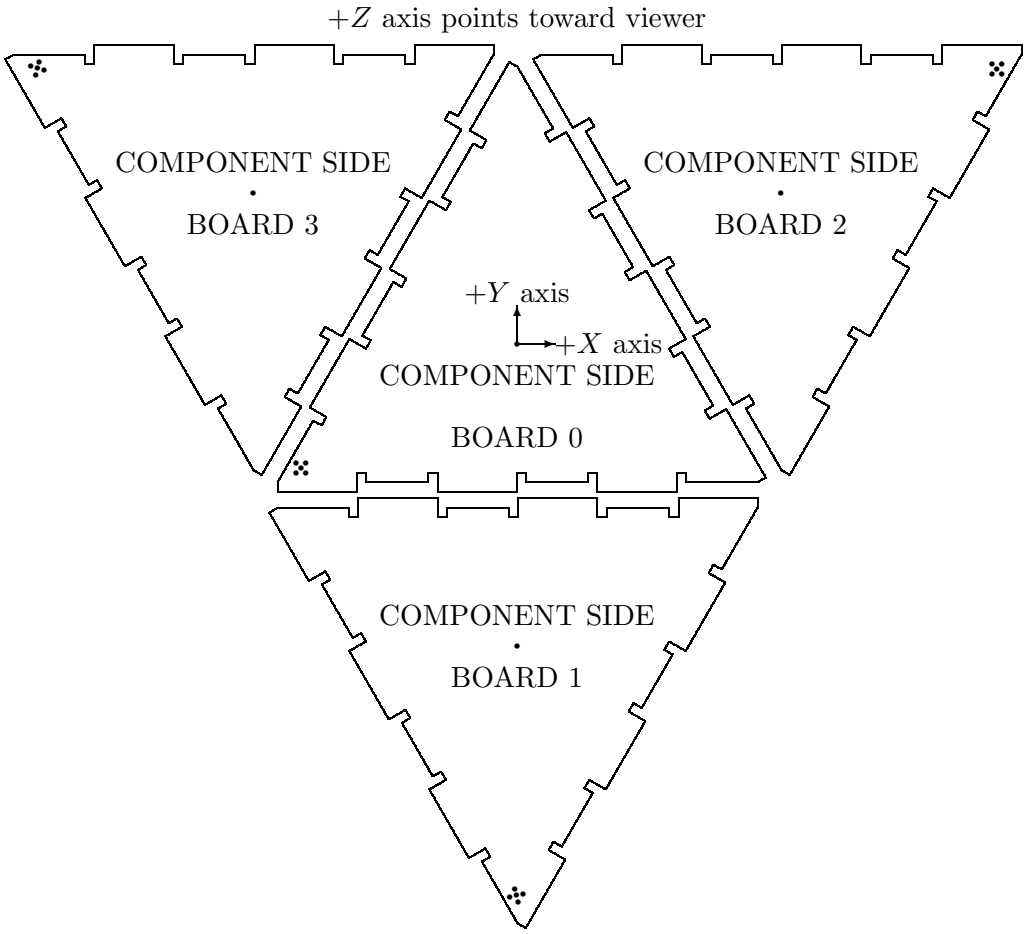

Figure 6.2: Unfolded tetrahedron looking at inside side

In the tetrahedron coordinate system, board 0 differs from each of the other boards by only a rotation around the origin of the tetrahedron coordinate system.

We represent a rotation using quaternions. Position vector $\overline{R} = R_x i + R_y j + R_z k$ is rotated into a vector $\overline{R_{rot}}$ by normalized quaternion $\mathcal{Q}$ as follows:

$\overline{R_{rot}} = \mathcal{Q}\overline{R}\mathcal{Q}'$

where $\mathcal{Q}'$ is the complement of $\mathcal{Q}$.

To rotate board 0 into board 0, $\mathcal{Q}_{00} = 1$

To rotate board 1 into board 0, $\mathcal{Q}_{10} = \frac{1}{2} + \frac{\sqrt{2}}{2}i - \frac{\sqrt{6}}{6}j - \frac{\sqrt{3}}{6}k$

To rotate board 2 into board 0, $\mathcal{Q}_{20} = \frac{\sqrt{2}}{2}i + \frac{\sqrt{6}}{6}j - \frac{\sqrt{3}}{3}k$

To rotate board 3 into board 0, $\mathcal{Q}_{30} = \frac{1}{2} + 0i - \frac{\sqrt{6}}{3}j + \frac{\sqrt{3}}{6}k$

Chapter 7

Magnetic field of circuit of straight-line segments

From Hayt [12], the magnetic field $\overline{B}$ at position $\overline{R}$ due to a current I flowing in a straight-line segment along the Z axis is:

$$\overline{B}(x_0, y_0, z_0) = \frac{\mu_o}{4\pi} \frac{I}{\rho}(sin(\alpha_2) - sin(\alpha_1))\widehat{\phi} \tag{7.1}$$

where

$\overline{R} = (x_0, y_0, z_0)$

$(0, 0, z_1)$ = cartesian coordinates of start of current segment

$(0, 0, z_2)$ = cartesian coordinates of end of current segment

$\rho = \sqrt{x_o^2 + y_o^2}$

$r_1 = \sqrt{x_o^2 + y_o^2 + (z_o - z_1)^2}$

$r_2 = \sqrt{x_o^2 + y_o^2 + (z_o - z_2)^2}$

$sin(\alpha_1) = \frac{z_1 - z_o}{r_1}$

$sin(\alpha_2) = \frac{z_2 - z_o}{r_2}$

$\widehat{\phi}$ = unit vector in the ϕ direction in cylindrical coordinates (ρ, ϕ, z)

From Hayt [13]:

$\widehat{\phi} = \widehat{z} \times \widehat{\rho}$

where

$\widehat{z}$ = unit vector in the z direction in cylindrical coordinates (ρ, ϕ, z)

$\widehat{\rho}$ = unit vector in the ρ direction

7.1 Coordinate-system-independent formulation

Recast equation 7.1 in a form which does not tie the coordinate system to the direction of the current segment A_k by calculating the parameters in equation 7.1 in coordinate-system-independent form using vector dot products and cross products:

$\overline{R}$ = position of field observer

(ρ_k, ϕ_k, s_k) is cylindrical coordinate system aligned with current segment k

$\overline{s_k^{start}}$ = vector from origin to start of segment A_k

$\overline{s_k^{finish}}$ = vector from origin to finish of segment A_k

$\overline{h_k} = \frac{1}{2}\left(\overline{s_k^{finish}} - \overline{s_k^{start}}\right)$

$\overline{m_k} = \frac{1}{2}\left(\overline{s_k^{finish}} + \overline{s_k^{start}}\right)$ = midpoint of segment A_k = origin of (ρ_k, ϕ_k, s_k) coordinates

$\overline{r_k} = \overline{R} - \overline{m_k}$ = position of field observer in (ρ_k, ϕ_k, s_k) coordinates

$\overline{\phi_k} = \overline{h_k} \times \overline{r_k}$

$\phi_k = \sqrt{\overline{\phi_k} \cdot \overline{\phi_k}}$

If $\phi_k < 10^{-100}$ meters, then $\overline{B_k} = 0$ and we are done for current segment k.

$\widehat{\phi_k} = \frac{\overline{\phi_k}}{\phi_k}$ = unit vector in ϕ_k direction

$h_k = \sqrt{\overline{h_k} \cdot \overline{h_k}}$ = half the length of segment A_k

$\widehat{h_k} = \frac{\overline{h_k}}{h_k}$ = unit vector in h_k direction

$\widehat{\rho_k} = \widehat{\phi_k} \times \widehat{h_k}$ = unit vector in ρ_k direction

$\rho_k = \overline{r_k} \cdot \widehat{\rho_k}$

$\overline{r_{1k}} = \overline{r_k} - \overline{h_k}$

$\overline{r_{2k}} = \overline{r_k} + \overline{h_k}$

$r_{1k} = \sqrt{\overline{r_{1k}} \cdot \overline{r_{1k}}}$

$r_{2k} = \sqrt{\overline{r_{2k}} \cdot \overline{r_{2k}}}$

$\widehat{r_{1k}} = \frac{\overline{r_{1k}}}{r_{1k}}$

$\widehat{r_{2k}} = \frac{\overline{r_{2k}}}{r_{2k}}$

$sin(\alpha_{1k}) = -\widehat{r_{1k}} \cdot \widehat{h_k}$

$sin(\alpha_{2k}) = -\widehat{r_{2k}} \cdot \widehat{h_k}$

Then the field due to segment k is:

$$\overline{B_k}(\overline{R}) = \frac{\mu_o}{4\pi} \frac{I}{\rho_k} \left(\widehat{r_{1k}} - \widehat{r_2 k}\right) \cdot \widehat{h_k}\, \widehat{\phi_k} \tag{7.2}$$

Circuit A is composed of straight-line segments $A_1 \cdots A_n$, so the field due to circuit A is:

$$\overline{B}(\overline{R}) = \sum_{k=1}^{n} \frac{\mu_o}{4\pi} \frac{I}{\rho_k} \left(\widehat{r_{1k}} - \widehat{r_2 k} \right) \cdot \widehat{h_k} \, \widehat{\phi_k} \tag{7.3}$$

7.2 Numerical considerations when ϕ_k approaches zero

When ϕ_k is zero, calculating $\widehat{\phi_k}$ fails due to division by zero. This is only a numerical problem, as the field continuously approaches zero when ϕ_k approaches zero, whether due to zero-length segment or due to the observation point being on the same line as the segment. This numerical problem can be eliminated by deriving a form which is numerically stable for small ϕ_k.

Double-precision floating-point calculations have much more resolution than we need, so we can, as stated above, simply skip calculating the field term when ϕ_k is small enough for the field error to be negligible. The field terms we do calculate will all have large enough ϕ_k so that the overall calculation is accurate.

Chapter 8

Mutual-inductance model of two PCB coils

We need to calculate the mutual inductance of two coils, each built of straight-line current segments, in arbitrary geometrical relationship to one another.

From Maxwell [3] [4] and Harnwell [2], the mutual inductance, L_{AB}, between two closed circuits is given by Neuman's formula:

$$L_{AB} = \frac{\mu_o}{4\pi} \oint_{circuitA} \oint_{circuitB} \frac{d\overline{s_A} \cdot d\overline{s_B}}{r_{AB}} \tag{8.1}$$

where

$r_{AB} = \mid \overline{s_A} - \overline{s_B} \mid$

$\mu_o = 4\pi 10^{-7}$ henries/meter

Strictly speaking, each closed circuit includes the coil, its connecting cable, and the electronics the cable is connected to.

In an ideal coaxial cable with perfectly-conducting shield, currents flowing down the center conductor and back through the shield emit no electromagnetic fields. Also, external electromagnetic fields can induce no normal-mode potential difference between center conductor and shield of an ideal coaxial cable.

External fields can induce common-mode potential differences between the two ends of the shield and simultaneously between the two ends of the center conductor. As long as there are no ground loops between the ends of the cable, there will be no normal-mode

potential difference inside the cable. In other words, an ideal coaxial cable with no ground loops exhibits infinite common-mode rejection ratio.

The use of coaxial cables (rather than twisted pairs) and well-shielded electronics, permits replacement of cable and electronics in the coil model with a direct short across the coil leads at the coil.

The printed-circuit coils are composed entirely of straight-line segments, so equation 8.1 becomes:

$$L_{AB} = \frac{\mu_o}{4\pi} \sum_{k=1}^{n} \sum_{l=1}^{m} \int_{B_l^{start}}^{B_l^{finish}} \int_{A_k^{start}}^{A_k^{finish}} \frac{d\overline{s_A} \cdot d\overline{s_B}}{r_{AB}} \tag{8.2}$$

where

Circuit A is composed of segments $A_1 \cdots A_n$. Circuit B is composed of segments $B_1 \cdots B_m$.

As detailed in Appendix G, the integral over A_k can be evaluated analytically, so equation 8.2 becomes:

$$L_{AB} = \frac{\mu_o}{4\pi} \sum_{k=1}^{n} \sum_{l=1}^{m} \frac{\overline{s_l} \cdot \overline{s_k}}{\sqrt{\overline{s_k} \cdot \overline{s_k}}} \int_{b=0}^{b=1} \log \frac{\sqrt{\frac{(\overline{s_k}+s_g\overline{s_b})\cdot(\overline{s_k}+s_g\overline{s_b})}{\overline{s_b}\cdot\overline{s_b}}} + \sqrt{\frac{\overline{s_k}\cdot\overline{s_k}}{\overline{s_b}\cdot\overline{s_b}}} + s_g \frac{\overline{s_b}\cdot\overline{s_k}}{\sqrt{(\overline{s_b}\cdot\overline{s_b})(\overline{s_k}\cdot\overline{s_k})}}}{1 + s_g \frac{\overline{s_b}\cdot\overline{s_k}}{\sqrt{(\overline{s_b}\cdot\overline{s_b})(\overline{s_k}\cdot\overline{s_k})}}} db \tag{8.3}$$

where

$\overline{s_k^{start}}$ = vector from origin to start of segment A_k
$\overline{s_k^{finish}}$ = vector from origin to finish of segment A_k
$\overline{s_l^{start}}$ = vector from origin to start of segment B_l
$\overline{s_l^{finish}}$ = vector from origin to finish of segment B_l
$\overline{s_k} = \overline{s_k^{finish}} - \overline{s_k^{start}}$
$\overline{s_l} = \overline{s_l^{finish}} - \overline{s_l^{start}}$
$s_g = +1$ if $((\overline{s_l^{start}} - \overline{s_k^{start}}) \cdot \overline{s_k} \geq 0)$, -1 if $((\overline{s_l^{start}} - \overline{s_k^{start}}) \cdot \overline{s_k} < 0)$
$\overline{s_e} = \overline{s_k^{finish}}$ if $(s_g = +1)$, $\overline{s_k^{start}}$ if $(s_g = -1)$
$\overline{s_b} = \overline{s_l^{start}} - \overline{s_e} + \overline{s_l} b$

We were unable to evaluate the second integral analytically, so we evaulated the second integral numerically.

We found that the seven-point integration rule derived in Appendix H accurately evaluated the integral using a minimum of points of evaluation: use of the rule once over the whole interval of integration gave the same accuracy as using the rule two or three times (each time over half or one third of the interval).

8.1 Effect of finite conductor cross-section

Our mutual-inductance model makes the physically-incorrect approximation that the current is all flowing along the centerline of the conductors. This approximation does not matter when we are calculating mutual inductances between two tetrahedra, for the separation between coils is large compared to the conductor width.

For all except the closest one of the measured tetrahedron-tetrahedron spacings, the closest points of the two tetrahedra are at least 0.125 meter apart.

For the ANT-003 PCB coil, the conductor width is 0.000254 meters. Thus, the ratio of the conductor width to r_{12} is (except for the closest tetrahedra spacing measured) always at least $.000254/.125 = 0.00203$. The relative mutual-inductance error is second-order in this ratio, so the error is less than $2 * 0.00203^2 = 8.3e - 6$.

Chapter 9

Tests of accuracy of calculation

We now check the accuracy of our mutual-inductance model by using this model to calculate the self-inductance of one PCB coil and to calculate the mutual inductance between two coils in one tetrahedron.

9.1 Self-inductance of one coil

Exactly calculating the self-inductance of one coil requires modeling the finite current density in the finite-cross-section conductors. Our mutual-inductance model does not do this.

The self-inductance of one coil can be calculated approximately as a special case of the mutual inductance between two identical coils which are almost, but not quite, superimposed. One coil is displaced from the other by a distance on the order of the conductor width.

A naive approach, is to superimpose two identical coils and calculate their mutual inductance. This gives an infinite result, due to the r_{12} term in the denominator of the integrand of equation 2.1.

To avoid the infinity, we do not superimpose the two coils. Instead, we keep the two coils parallel to each other, and displace one coil from the other by a small ϵ in X Y and Z position.

We use the same ϵ for X Y and Z position, so the displacement is diagonal. This ensures that the displaced coils' current paths do not intersect when ϵ is not zero.

Table 9.1: Self-inductance calculations in microhenries

Coil	$L_{\texttt{Step=0}}$	$L_{\texttt{Step=2}}$
0	409.3	409.8
1	410.0	410.5
2	408.5	408.9
3	410.3	410.8
measured	410	410

Numerical tests show that the signs of the displacements appear to not matter for the small displacements we are considering here.

Starting from a large displacement, as the displacement is reduced, the inductance is proportional to the inverse cube of the displacement until the displacement approaches the conductor spacing.

As we further reduce the displacement, the inductance grows more slowly.

For displacements smaller than the conductor spacing, the inductance is apparently equal to a constant plus a term proportional to the logarithm of the displacement. The inductance varies 1% for ϵ values between the conductor spacing and one sixteenth of the conductor spacing.

Choosing $\epsilon = 0.0635$ millimeters, which is one quarter of the conductor spacing (and also one quarter of the track width), gives the results in Table 9.1 from calculation results in Appendix K. The calculated self-inductance varies between 408.9 and 410.8 millihenries. Measurements of ANT-003 coils gave self-inductances between 407 to 412 millihenries, with the most common value being 410 microhenries.

9.2 Mutual inductance of two coils in one tetrahedron

By choosing $\epsilon = 0$, we superimpose one tetrahedron on itself. The calculation L_m matrices in Appendix K show the expected infinite diagonal elements as "nan", meaning "not a number".

The off-diagonal elements show variations of less than one part in 10000 as `Step` is varied

from its usual 0 to 2. Higher values of `Step` give slower and more accurate numerical integration. Thus, `Step` = 0 is adequate for our integrations.

9.3 Condition of frequency times transmitter current matrix

The mutual inductance of two coils in one tetrahedron is about 0.12 of the self-inductance of one coil from the calculations in Appendix K. Thus, we expect the off-diagonal elements of the frequency times transmitter current matrix `Iiw` to be smaller than the on-diagonal terms. The calculations in Appendix Q show this to be true.

Since the diagonal terms of `Iiw` are of similar magnitudes and are significantly larger in magnitude than the non-diagonal terms, we expect `Iiw` to be well-conditioned and accurately invertible.

The condition number [14] for `Iiw`, calculated with Octave, is between 3.7 and 3.8 in all eleven locations. A perfectly-conditioned matrix has a condition number of 1, and a singular matrix has a condition number of infinity, so `Iiw` is well-conditioned.

Chapter 10

Tracker P&O algorithm overview

Electromagnetic tracker P&O algorithms are usually based on first calculating an initial estimate of P&O, called the seed, then improving the P&O estimate by performing a least-squares best fit to the measured Lm matrix.

Raab [15] describes an algorithm for calculating the seed from a 3x3 orthogonal-dipole Lm matrix.

Raab,Blood,Steiner,Jones [6] describe a 3x3 orthogonal-dipole P&O algorithm which uses previous P&O results for a seed, and uses analytically-derived Lm model and partial derivatives in the least-squares fit.

We are starting with a non-dipole 4x4 Lm matrix, so we cannot use the orthogonal-dipole seeds.

Schneider [9] discusses the issues of seed and least-squares fitters in generalized P&O algorithms and the importance of a good seed, without disclosing the details of such a seed.

Here, we are using the tracking system as a means of verifying the generation of accurately-known fields. Thus, we can arrange to have at least a crude idea of the mechanical P&O to use as a seed. Alternatively, we can use the previous cycle's results.

Chapter 11

Goodness-of-fit (GOF)

The goodness-of-fit G_f is a dimensionless measure of the discrepancies between the modeled mutual inductances L_{modl} (which are functions of position estimate $\overline{R}_{modl}$ and orientation estimate $\mathcal{O}_{modl}$), and the measured mutual inductances L_{meas}:

$$G_f = \frac{\sum_{t=0}^{3}\sum_{r=0}^{3}(L_{modl} - L_{meas})_{t,r}^2}{\sum_{t=0}^{3}\sum_{r=0}^{3}(L_{meas})_{t,r}^2} \tag{11.1}$$

To estimate the receiver position and orientation, $\overline{R}_{modl}$ and $\mathcal{O}_{modl}$ are adjusted, using well-known error-minimization algorithms, to minimize G_f.

When G_f is small, we know that the remaining errors are small, and thus that the errors in $\overline{R}_{modl}$ and $\mathcal{O}_{modl}$ are small. By monitoring G_f after error-minimization, the accuracy of the tracking system can be continuously assessed during system operation, and warnings issued whenever the system becomes inaccurate.

The GOF is an excellent measure of the accuracy of the fit of L_{modl} to L_{meas}. A small GOF ensures that the field model is accurate, and that the electronics are working properly, without any need for precise mechanical positioning fixtures.

If anything is not working properly, the GOF suffers. Thus, while a tiny GOF is good assurance that the model and electronics are okay, a large GOF gives little information as to what is wrong.

Chapter 12

Mutual inductance measuring system hardware

The mutual-impedance-measuring system is designed and built specifically for the measurement of small mutual impedances at frequencies between 100 Hz and 40 kHz. See Figure 12.1.

Our measurements were performed between 11 and 15 kHz.

The system is designed to simultaneously measure the mutual inductances between all possible pairs of coils, where one coil in each measured pair is in coil array T, called the transmitter, and the other coil in each measured pair is in coil array R, called the receiver.

We use a 10-ohm, ±100 parts per million four-terminal resistor as a mutual-impedance reference, $Z_{ref} = 10$ ohms. Note that Z_{ref} is real and independent of frequency at the frequencies of interest.

Simultaneous measurements are achieved by driving the transmitter coils and the mutual-impedance reference with sinusoidal voltages at distinct frequencies, and by having a separate receive measurement channel for each receiver coil.

Periodically, the inputs to the receive measurement channels are switched from the receiver coils to the mutual-impedance reference.

A separate current-measurement channel is provided to sequentially measure the current in each transmitter coil, and in the mutual-impedance reference. A current-mode multiplexer (CMX) accomplishes the switching. Sequential measurements permits all the

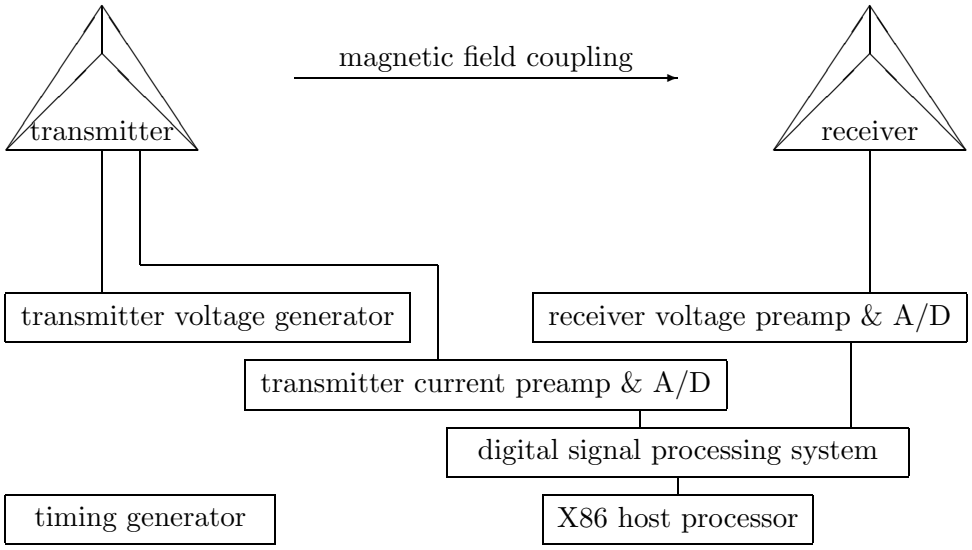

Figure 12.1: Mutual-inductance-measuring system block diagram

currents to be measured with one current-measurement channel. Thus, ratios of currents are independent of the gain of the current-measurement channel.

This channel is identical to a receiver channel, except for a virtual-ground current input replacing the receiver-channel differential voltage input.

The whole system is ratiometric, and mutual impedances are measured with respect to the mutual-impedance reference. Thus, there are no precision voltage or current references needed.

The frequency-independent factors of the complex gains of the receiver measuring channels are canceled in the process of determining the ratios of coil mutual impedances to Z_{ref}.

The frequency responses (varying-with-frequency factors of the complex gains) of the receiver measuring channels and of the current-measuring channel are the same, so the ratioing process cancels the effects of the frequency responses.

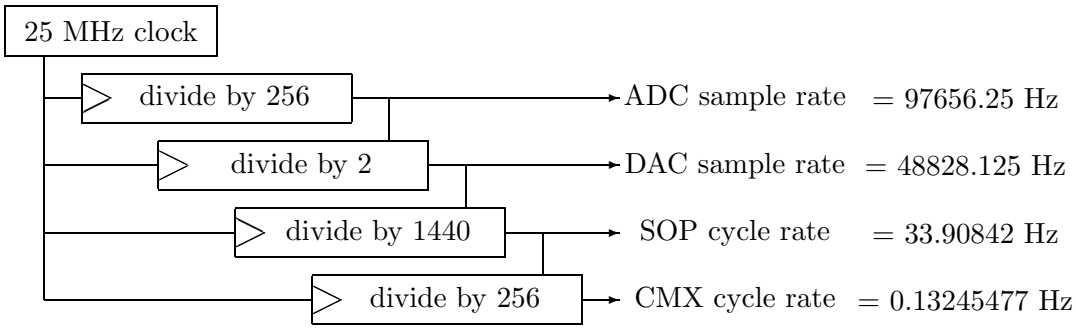

Figure 12.2: Timing generator block diagram

12.1 Timing generator

The entire system runs from a single clock. As shown in Figure 12.2, this clock is divided by a synchronous counter to generate slower rates. Since the counters are synchronous, they all run from the same clock. Each counter uses the output of the previous counter to determine on which clock pulse to change state. The rates shown in Figure 12.2 are for a clock which is exactly 25000000 Hz. The clock actually used, has a tolerance of ±100 parts per million.

12.2 Digital sinewave calculation

Sample n of the digital sinewave produced by driver d is given by:

$$D_{dn} = D_d \sin\left(\frac{\omega_d}{F_s} n\right) \tag{12.1}$$

where

D_d = digital amplitude of driver d sinewave

$\frac{\omega_d}{F_s}$ = sinewave frequency of driver d in radians per ADC sample

ω_d = sinewave frequency of driver d in radians per second

F_s = ADC sample rate in samples per second

Since the DAC sample rate is half the ADC sample rate, only the even-numbered samples are actually calculated and used.

Each driver frequency is an integral multiple of the SOP cycle rate:

$$\frac{\omega_d}{F_s} = \frac{2\pi}{2880} f_d \tag{12.2}$$

Table 12.1: Analog frequencies output by the transmitter drivers

d	f_d	F_d in hertz
0	335	11359.321
1	344	11664.497
2	353	11969.672
3	362	12274.848

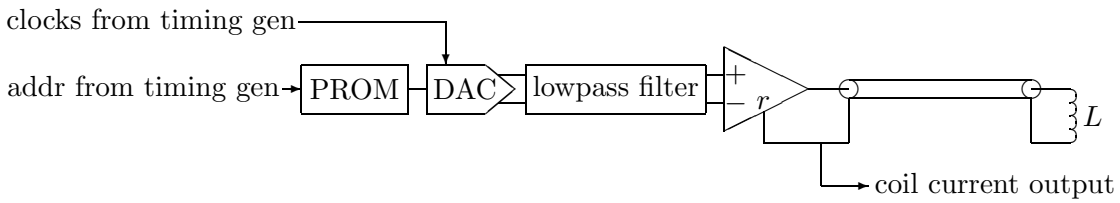

Figure 12.3: One transmitter coil driver block diagram

The analog frequencies output by the drivers are given by:

$$F_d = \frac{F_{clock}}{256 * 2880} f_d \tag{12.3}$$

where

f_d = integer between 1 and 1439

The approximate analog frequencies output by the transmitter drivers for $Fclock = 25000000$ hertz exactly, are given in Table 12.1.

12.3 Transmitter driver sinewave generator

The block diagram of one transmitter driver channel is shown in Figure 12.3. The clock drives a digital sinewave generator, which produces a digital continuous-wave (CW) sinewave of precise frequency and phase. The sinewave generator outputs only frequencies which are integral multiples of the SOP cycle rate. Thus, the sinewave repeats exactly every SOP cycle, with no discontinuity between SOP cycles.

The frequencies generated are fixed, so the sinewave generator is implemented by precalculating the samples of the desired sinewaves, and storing the samples in a programmable

read-only memory (PROM).

The PROM addresses are driven by signals from the timing generator counters. These signals are not shown in Figure 12.2. The PROM outputs the samples in the serial format required by the particular digital-to-analog converter (DAC) used here. Another PROM output (not shown in Figure 12.2) implements the divide-by-1440 control logic.

Each digital sinewave is converted to a differential analog voltage in a 24-bit deltasigma digital-to-analog converter (DAC). The DAC output goes through a lowpass filter to a differential amplifier.

12.4 Transmitter driver power amplifier

The differential amplifier has three inputs $+$,$-$,r, and one output. The amplifier has a fixed gain G, such that:

$$V_{output} = V_r + G(V_+ - V_-) \tag{12.4}$$

The amplifier drives the high end of the transmitter coil L through a coaxial cable. The amplifier r input is connected to the low end of L, which is also connected to virtual ground.

A coaxial cable is used because a coaxial cable with a perfectly-conducting shield emits no electromagnetic fields due to currents within the cable.

Twisted-pair cables are unsatisfactory because they do emit fields due to currents flowing within the cable.

The virtual ground is not exactly ground, and the r connection ensures that the voltage across L will not be affected by the voltage on the virtual ground. Thus, a stable sinusoidal voltage is applied across L.

Note that the impedance of the amplifier r input is high compared to the impedance of the virtual ground or real ground, so the current drawn through r is less than 10^{-4} times the current through L. This small error is easily estimated and corrected mathematically.

The coaxial cable capacitance shunts a small, but significant, current around L. This small error is easily estimated and corrected mathematically.

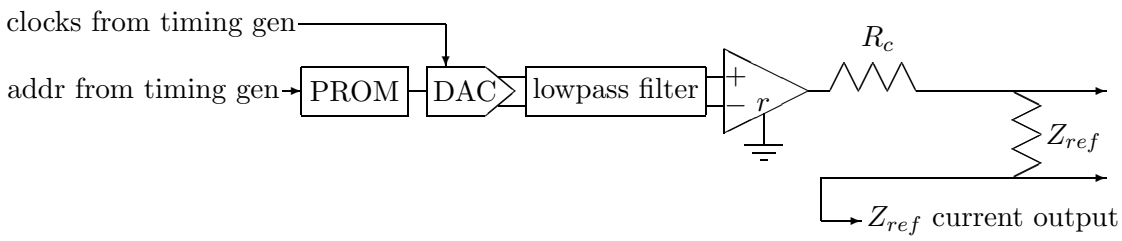

Figure 12.4: Mutual-impedance-reference Z_{ref} driver block diagram

12.5 Z_{ref} driver

The mutual impedance of Z_{ref} is much larger than the coil-coil mutual impedances, so the Z_{ref} driver provides a smaller current. As seen in Figure 12.4, a resistor, R_c, which is large compared to Z_{ref}, converts the voltage output of the difference amplifier to a small current in Z_{ref}.

The difference amplifier input r is grounded, rather than connected to the low end of $Zref$, for two reasons. First, the input r draws a small current which is significant compared to the current in Z_{ref}. Second, the receiver measurement channel responses to Z_{ref} are measured only when the current through Z_{ref} is being measured by the current-measurement channel.

There is no coaxial cable because $Zref$ plugs directly onto the connectors on the receiver electronics board.

12.6 Current-switching multiplexer (CMX)

The block diagram of the current-switching multiplexer (CMX) is shown in Figure 12.5. The virtual grounds for all the transmitter coils go through a current-mode switch matrix. The switch matrix connects none or one transmitter coil current to the single-turn primary of current transformer T_c, while connecting all the remaining transmitter coils to the high-current transmitter-coil common-ground point t. The selected current goes through the primary of T_c to t.

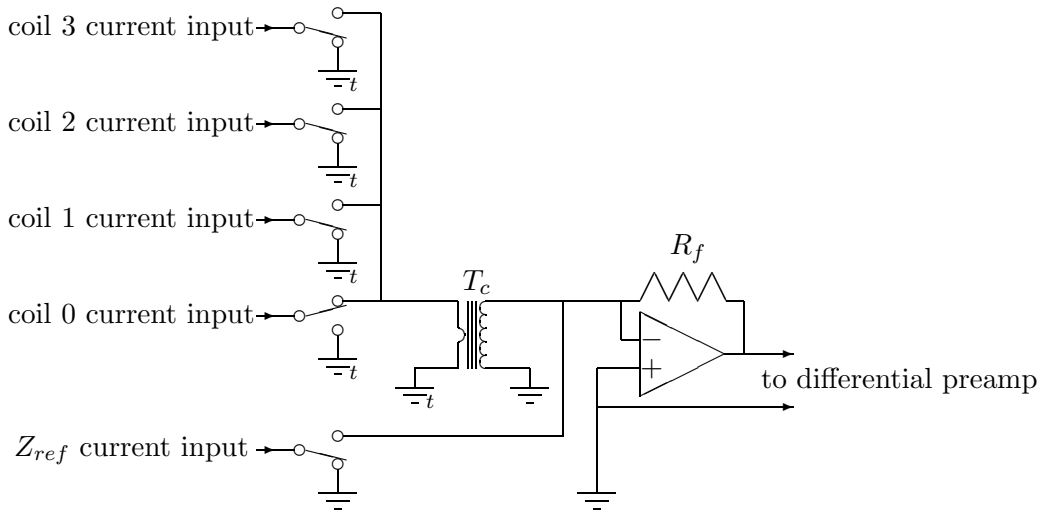

Figure 12.5: Current multiplexer and current preamp: switches are make-before-break

12.7 Current transformer

T_c has a primary-to-secondary turns ratio of 1:200. T_c isolates the large transmitter currents from the receiver electronics analog ground, accurately reduces the transmitter currents to values within the range of the virtual-ground preamp, and minimizes the reflected virtual-ground impedance seen by the transmitter coils.

The mutual-impedance reference is operated at a lower current than the transmitter coils are operated, so the mutual-impedance-reference current is switched in or out on the secondary side of T_c.

The secondary of T_c is always in the circuit, to cancel effects due to transformer magnetizing current: The magnetizing current is just another constant complex gain error in the current-measurement channel, and constant gain errors cancel out in the data-reduction calculations.

12.8 Virtual-ground transresistance amplifier

The operational amplifier and resistor R_f form a virtual-ground current-input voltage-output (transresistance) amplifier.

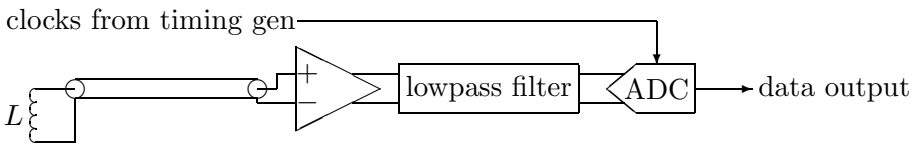

Figure 12.6: One receiver channel block diagram

12.9 Receiver preamp and ADC channel

The block diagram of one voltage-input preamp and ADC channel is shown in Figure 12.6. The voltage from inductor L is applied via the coaxial cable to the differential-voltage-input preamp. The preamp output is applied to the input of a 24-bit deltasigma (ADC) through a passive lowpass filter. The filter eliminates both alias responses and effects due to nonlinear ADC input currents.

A coaxial cable is used because external electromagnetic fields can induce no potentials between the center conductor and shield of a coaxial cable with a perfectly-conducting shield.

Twisted-pair cables are unsatisfactory. A twisted-pair cable cancels effects due to uniform applied electromagnetic fields, but does respond to field gradients. We have gradients in our fields, and would induce voltages in twisted-pair cables.

There is one voltage-input channel for each receiver coil and one channel for the current preamp.

For the current-preamp channel, the Figure 12.5 output and ground replace L and coaxial cable in Figure 12.6.

The ADC sampling is controlled by the timing generator, Figure 12.2, so the timing relationship between ADC samples and DAC samples is tightly controlled.

The ADC outputs, and timing reference information from the timing generator, go to a digital signal processing system.

Chapter 13

Digital signal processing system

The digital signal processing system (DSPS) performs the calculations needed to extract the desired real and imaginary frequency components from the ADC data, delivering a new result once every 2880 ADC samples, the SOP cycle rate. To show the function, consider the process of extracting one phase component of one frequency from the data from one ADC.

First, we describe the function we want to implement, then describe this function mathematically, then transform the equations into the form actually implemented in the digital signal processing system.

Remember that we are receiving continuous-wave (CW) signals because the transmitter coils are energized with CW drives, and the transmitter and receiver are fixed with respect to each other for any given measurement, giving constant L_m values.

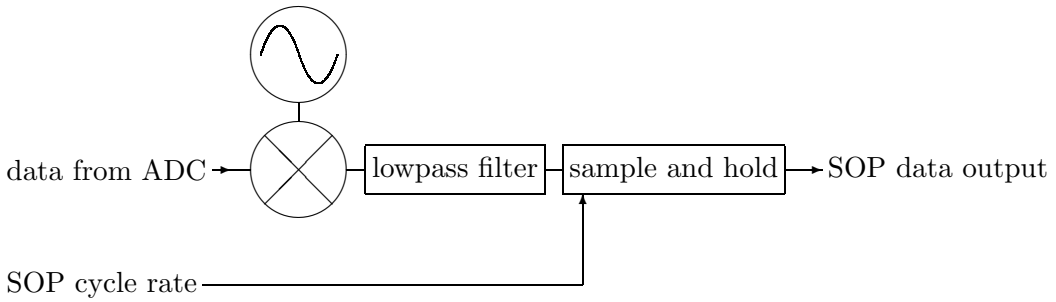

Figure 13.1: Heterodyne receiver block diagram

Since we are using multiple frequencies, we have multiple CW frequency components in the data from each ADC. Thus, we need a receiver with excellent selectivity.

13.1 Heterodyne radio receiver

Armstrong described existing heterodyne radio receivers as being excellent for receiving CW signals [16] with high selectivity.

Note that a heterodyne receiver is not a superheterodyne receiver, which Armstrong invented to handle spark-transmitter-generated damped-wave (DW) signals which the heterodyne receiver could not reproduce adequately [17].

Armstrong [16] implemented a heterodyne receiver in analog form using vacuum triodes. Here, we implement a real-time numerical model of a heterodyne receiver in a digital signal processing system.

The ADC data is applied to a heterodyne receiver as shown in Figure 13.1. The ADC data drives one input of an ideal double-balanced mixer. A local oscillator drives the other input of the mixer with an ideal sinewave of the desired frequency and phase. The mixer output is the product of the two mixer inputs.

In the general case, a CW signal has an amplitude and a phase. Equivalently, a CW signal has real (in-phase) and imaginary (quadrature) components. Thus, two heterodyne receivers are used for each frequency. The first receiver has a cosinewave for the local oscillator and outputs the real part of the signal. The second receiver has a sinewave for the local oscillator and outputs the imaginary part of the signal.

The mixer output goes to the input of a lowpass filter. This filter has $136\,dB$ attenuation for undesired frequencies. The filter output goes to a sample-and-hold circuit, which samples the filter output once every 2880 ADC samples. The output of the sample-and-hold is the amplitude of the real or imaginary part of the desired frequency component of the ADC data.

13.2 Lowpass finite-impulse-response (FIR) filter

The lowpass filter is implemented as a 2879-sample Dolph-Chebyshev [18] [19] [20] finite-impulse-response (FIR) filter:

$$S_n = \sum_{m=1}^{2879} C_m \, M_{n-m} \tag{13.1}$$

where

S_n = filter output sample number n

C_m = filter coefficient number m

M_r = mixer output sample number r

13.3 Mixer or multiplier

Since the mixer multiplies ADC data samples by local oscillator samples, we can write:

$$S_n = \sum_{m=1}^{2879} C_m \, \sin\left(\frac{\omega}{F_s}(n-m) + \phi\right) \, A_{n-m} \tag{13.2}$$

where

$\frac{\omega}{F_s}$ = sinewave frequency in radians per ADC sample

ω = sinewave frequency in radians per second

F_s = ADC sample rate in samples per second

ϕ = sinewave starting phase, $\frac{\pi}{2}$ for the real part and 0 for the imaginary part

A_r = ADC sample number r

13.4 Local oscillator

The local oscillator frequency is the same as a frequency generated by the sinewave generator in Figures 12.3 and 12.4, so the frequency is an integral multiple of the SOP cycle rate in Figure 12.2:

$$\frac{\omega}{F_s} = \frac{2\pi}{2880} f \tag{13.3}$$

where

f = integer between 1 and 1439

Then equation 13.2 becomes:

$$S_n = \sum_{m=1}^{2879} C_m \sin\left(\frac{2\pi}{2880}(n-m)+\phi\right) A_{n-m} \tag{13.4}$$

13.5 Sample-and-hold

The sample-and-hold keeps one filter output sample of every 2880, say sample number $2880n$:

$$S_{2880n} = \sum_{m=1}^{2879} C_m \sin\left(\frac{2\pi}{2880}(2880n-m)+\phi\right) A_{2880n-m} \tag{13.5}$$

Then:

$$S_{2880n} = \sum_{m=1}^{2879} C_m \sin\left(2\pi n - \frac{2\pi}{2880}m+\phi\right) A_{2880n-m} \tag{13.6}$$

By the periodicity of the sine function:

$$S_{2880n} = \sum_{m=1}^{2879} C_m \sin\left(-\frac{2\pi}{2880}m+\phi\right) A_{2880n-m} \tag{13.7}$$

Define a weight function W by:

$\mathtt{W}_{-m} = C_m \sin\left(-\frac{2\pi}{2880}m+\phi\right)$

Then we have:

$$S_{2880n} = \sum_{m=1}^{2879} \mathtt{W}_{-m} A_{2880n-m} \tag{13.8}$$

Define a new variable $\mathtt{M} = 2880 - m$. Then we have:

$$\mathtt{SOP} = S_{2880n} = \sum_{\mathtt{M}=2879}^{1} \mathtt{W}_{\mathtt{M}} A_{2880(n-1)+\mathtt{M}} \tag{13.9}$$

13.6 Pseudocode

Equation 13.9 is the sum-of-products (SOP) representation of the heterodyne radio receiver, and is easily implemented in software by the following pseudocode executed once as each ADC sample arrives in the processor:

```
if(ADC_sample_number/2880 is an integer){
   SOP = accumulator;
   accumulator = 0;
   M = 0;}
```

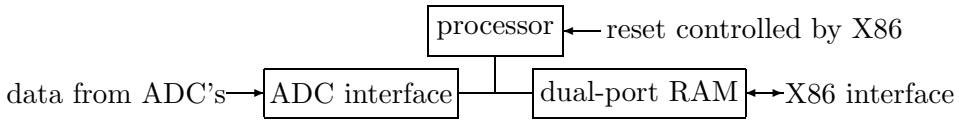

Figure 13.2: Digital signal processing system block diagram

```
else{
   M = M + 1;
   accumulator = accumulator + W(M) * ADC_data;}
```

The digital signal processing system hardware, Figure 13.2, executes this pseudocode on a 200-MHz, 64-bit MIPS-instruction-set processor with a 64x64 multiply instruction. A 64-bit integer accumulator is used to ensure that the roundoff errors in the sums will be insignificant. The weight function W is scaled so that the accumulator will not quite overflow for the largest possible signal, which is a square wave at frequency f and fullscale in the ADC data.

13.7 Assembly language advantages over compiler language

The actual code is written in assembly language for speed, and because the execution time could be reliably estimated from the pseudocode and the processor data sheet. The known execution time permits the design of the hardware with assurance that the code will run fast enough on the hardware.

Compilers do not permit reliable design for determinate execution speed. Additionally, most compilers cannot express multiplying two 64-bit integers to get a 128-bit product.

13.8 Host interface

The dual-port read/write memory (RAM) in Figure 13.2 has two independent ports accessing one array of memory bits. One port is connected to the processor in Figure 13.2, and the other port connects to the X86 host processor. The X86 loads executable code into the dual-port RAM, starts the Figure 13.2 processor executing the code, and reads SOP results from the dual-port RAM.

We have described the extraction of one SOP output (the real or imaginary part of one frequency component) from the data from one ADC.

The digital signal processing system in Figure 13.2 can calculate 120 SOP outputs at once, though we do not use so many here. Since all the digital signal processing system memory is writeable, the system is easily reprogrammed for different situations.

13.9 Discrete Fourier transform (DFT)

Note that equation 13.7 represents one component of a windowed discrete Fourier transform (DFT). Thus, the pseudocode above can be viewed as implementing one component of a windowed discrete Fourier transform.

The Dolph-Chebyshev window used here [18] [19] [20], has steep skirts, and sidelobes that can be reduced to levels far below those of more well-known window functions.

A fast Fourier transform (FFT) is not used, as it would be much slower than the pseudocode used here: The pseudocode results are available one ADC sample time after the last ADC sample occurs. An FFT requires all the ADC samples to arrive before any of the calculation can begin, so has a much longer time delay from last ADC sample to results available.

Chapter 14

Experimental fixture

The experimental fixture permits the location of the receiver tetrahedron in any of eleven positions with respect to the transmitter tetrahedron.

Figure 14.1 shows the transmitter and receiver tetrahedra on the fixture.

The fixture is a white rigid teflon plate with eleven sets of precision locating holes for the receiver mounting block. The edge of the fixture is difficult to see because the white fixture is resting on a white tabletop.

The transmitter sits directly on the fixture, with board 0 coax connector overhanging the edge of the fixture so board 0 lays flat on the fixture.

The receiver is on its black mounting block, at location 7 on the fixture, with board 0 coax connector overhanging the edge of the block so board 0 lays flat on the mounting block. A small notch (not visible) is cut in the block to clear the coax connector.

The locating holes for location 4 are visible between the transmitter and the receiver. The locating holes for location 11 are visible to the right of the receiver.

The fixture was built for another project, which defined the locations and their numbers. For consistency with the documentation of that project, we have retained the same location numbering.

The picture is printed sideways on the following page.

Figure 14.1: Experimental fixture showing receiver in position 7

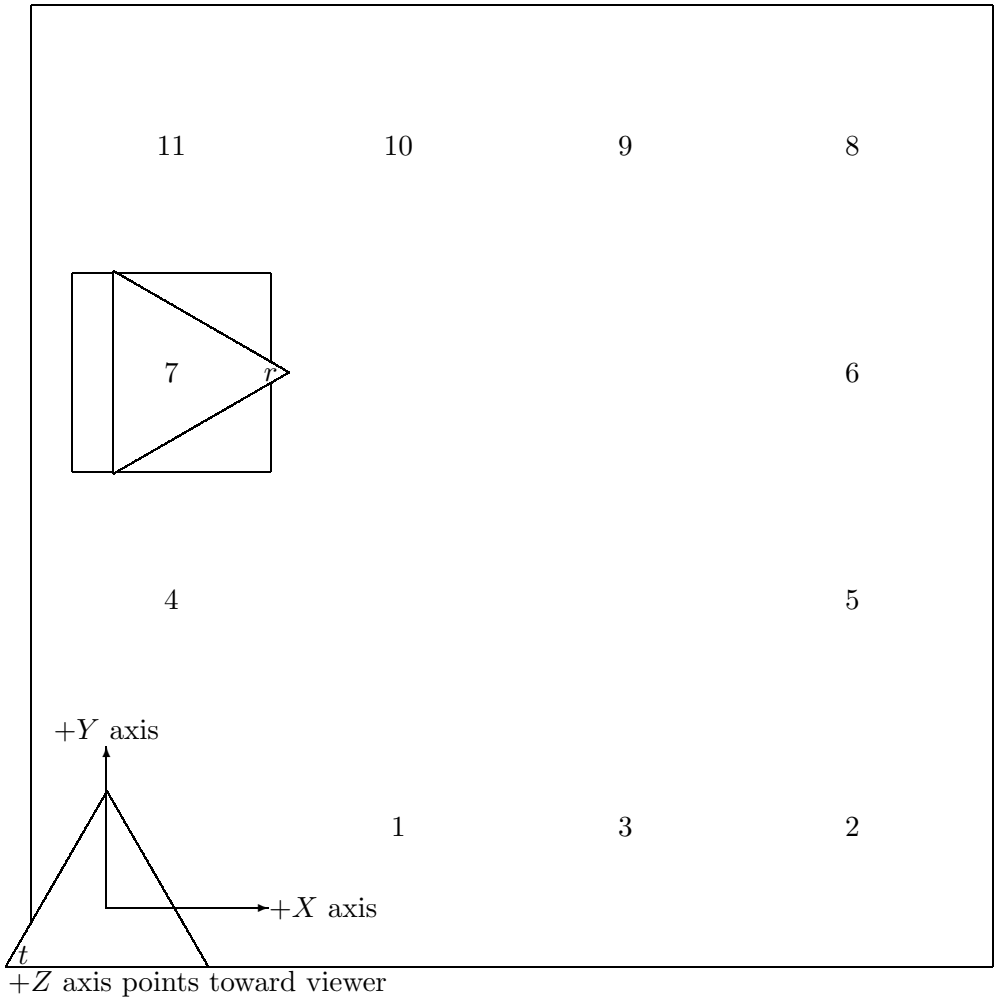

Figure 14.2: Arrangement of locations on fixture showing receiver in position 7

The positions have positional tolerances of ±0.0003 meter, excluding position errors due to the transmitter being rotated from its nominal placement.

The receiver position and orientation are determined in the nominal transmitter tetrahedron coordinate system.

14.1 Receiver locations

The receiver locations are shown in Figure 14.2. The numbers 1 through 11 are centered on the center of receiver board 0 at locations 1 through 11 respectively. The letter t shows the position of the board 0 coax connector on the transmitter. The letter r shows the

Table 14.1: Transmitter coordinates of receiver locations on fixture

location	receiver XYZ position in meters
1	(0.1830, 0.0513, 0.0675)
3	(0.3252, 0.0513, 0.0675)
2	(0.4675, 0.0513, 0.0675)
5	(0.4675, 0.1935, 0.0675)
6	(0.4675, 0.3358, 0.0675)
8	(0.4675, 0.4780, 0.0675)
9	(0.3252, 0.4780, 0.0675)
10	(0.1830, 0.4780, 0.0675)
11	(0.0408, 0.4780, 0.0675)
7	(0.0408, 0.3358, 0.0675)
4	(0.0408, 0.1935, 0.0675)

position of the board 0 coax connector on the receiver when the receiver is in location 7. The receiver locations in nominal transmitter coordinates are given in Table 14.1.

The receiver orientation is the same at all the locations. The receiver is rotated nominally +150 degrees = $\frac{5}{6}\pi$ radians about the +Z axis from the transmitter orientation, giving a nominal orientation quaternion:

$\mathcal{O}_{nom} = cos(\frac{5\pi}{12}) + k\ sin(\frac{5\pi}{12})$

Both transmitter and receiver have rotational tolerances of 0.04 radian about the Z axis and 0.002 radian about the X and Y axes.

Chapter 15

Mutual inductance measuring system data-reduction calculation

The driver output voltages are all constant-frequency sinewaves, so all desired signals in the system are sums of sinewaves of various frequencies and phases. Thus, we can most easily calculate the mutual inductances in terms of matrices of complex Fourier components.

The mutual inductances among the transmitter coils are nonzero, so the current flowing through each transmitter coil contains components at all four driver frequencies. Thus, the current I_t flowing through transmitter coil t as a function of time τ is:

$$I_t(\tau) = \Re\left(\sum_{d=0}^{3} I_{d,t} e^{-\jmath\omega_d \tau}\right) \tag{15.1}$$

where

$\Re()$ means "the real part of"

t = transmitter coil number

d = driver number

$I_{d,t}$ = complex current flowing in transmitter coil t at frequency ω_d

$\jmath = \sqrt{-1}$ used in complex numbers

ω_d = frequency of driver d in radians per second

The rate of change of I_t is:

$$\frac{dI_t}{dt}(\tau) = \Re\left(\sum_{d=0}^{3} -\jmath\omega_d I_{d,t} e^{-\jmath\omega_d \tau}\right) \tag{15.2}$$

The voltage V_r induced in receiver coil r as a function of time τ is:

$$V_r(\tau) = \Re\left(\sum_{d=0}^{3} V_{d,r} e^{-\jmath\omega_d \tau}\right) \tag{15.3}$$

The voltage V_r induced in receiver coil r can be expressed in terms of the (complex for generality's sake) mutual inductances $\mathcal{L}_{t,r}$ as:

$$V_r(\tau) = \Re\left(\sum_{t=0}^{3}\sum_{d=0}^{3} \mathcal{L}_{t,r}(-\jmath\omega_d) I_{d,t} e^{-\jmath\omega_d \tau}\right) \tag{15.4}$$

Equating our two expressions for $V_r(\tau)$, we have:

$$\Re\left(\sum_{d=0}^{3} V_{d,r} e^{-\jmath\omega_d \tau}\right) = \Re\left(\sum_{t=0}^{3}\sum_{d=0}^{3} \mathcal{L}_{t,r}(-\jmath\omega_d) I_{d,t} e^{-\jmath\omega_d \tau}\right) \tag{15.5}$$

Equating the imaginary parts as well as the real parts gives:

$$\sum_{d=0}^{3} V_{d,r} e^{-\jmath\omega_d \tau} = \sum_{t=0}^{3}\sum_{d=0}^{3} \mathcal{L}_{t,r}(-\jmath\omega_d) I_{d,t} e^{-\jmath\omega_d \tau} \tag{15.6}$$

Define a new variable:

$$\Xi_{d,t} = -\jmath\omega_d I_{d,t} \tag{15.7}$$

Then we have:

$$\sum_{d=0}^{3} V_{d,r} e^{-\jmath\omega_d \tau} = \sum_{t=0}^{3}\sum_{d=0}^{3} \mathcal{L}_{t,r} \Xi_{d,t} e^{-\jmath\omega_d \tau} \tag{15.8}$$

The terms containing different values of d are independent of one another. Thus 15.8 is really four equations, one for each value of d. Remove the common exponential factors from each of the four equations:

$$V_{d,r} = \sum_{t=0}^{3} \Xi_{d,t} \mathcal{L}_{t,r} \tag{15.9}$$

Treat $V_{d,r}$, $\Xi_{d,t}$, and $\mathcal{L}_{t,r}$ as four-by-four matrices, where the first subscript is the row index and the second subscript is the column index. Then 15.9 is a complex matrix multiply:

$$V_{d,r} = \Xi_{d,t} \mathcal{L}_{t,r} \tag{15.10}$$

Note that $\Xi_{d,t}$ is a square matrix and (unless it is singular) has an inverse, $\Xi^{-1}_{t,d}$, such that:

$$\Xi_{d,t}\, \Xi^{-1}_{t,d} = \Xi^{-1}_{t,d}\, \Xi_{d,t} = \mathcal{I} \tag{15.11}$$

where $\mathcal{I}$ is the four by four identity matrix.

Multiply 15.10 on the left by $\Xi_{t,d}^{-1}$ to get an equation for the mutual inductance matrix:

$$\mathcal{L}_{t,r} = \Xi_{t,d}^{-1} V_{d,r} \tag{15.12}$$

This is the basic calculation performed in the data-reduction code listed in Appendix N. Additional corrections are included in the code for the gains of the electronics and the effects of cable capacitances.

In 15.12, $\mathcal{L}_{t,r}$ is complex. The real part of $\mathcal{L}_{t,r}$ contains the mutual inductances plus variations due to noise and unmodeled phase shifts. The imaginary part of $\mathcal{L}_{t,r}$ contains zeros plus variations due to noise and unmodeled phase shifts.

The data-reduction code listed in Appendix N calculates the phase error in each element of $\mathcal{L}_{t,r}$ by calculating the arctangent of the ratio of the real and imaginary parts of that element:

$$\Phi_{t,r} = \arctan\left(\frac{\Im\left(\mathcal{L}_{t,r}\right)}{\Re\left(\mathcal{L}_{t,r}\right)}\right) \tag{15.13}$$

By taking the real part of $\mathcal{L}_{t,r}$, we get the real mutual-inductance matrix:

$$L_{t,r} = \Re\left(\mathcal{L}_{t,r}\right) \tag{15.14}$$

Chapter 16

Mutual inductance measuring system noise-index calculation

We need a measure of the noisiness of the measured mutual-inductance matrix, to compare to the goodness of fit defined in equation 11.1.

The mean of a group of n mutual-inductance measurements is:

$$\overline{L_{t,r}} = \frac{1}{n} \sum_{m=0}^{n-1} L_{t,r,m} \tag{16.1}$$

where

t = transmitter coil number 0,1,2, or 3

r = receiver coil number 0,1,2, or 3

m = measurement number 0 through $n-1$

The square of the mean of a group of n mutual-inductance measurements is then:

$$\overline{L_{t,r}}^2 = \left(\frac{1}{n} \sum_{m=0}^{n-1} L_{t,r,m} \right)^2 \tag{16.2}$$

The mean of the squares of a group of n mutual-inductance measurements is:

$$\overline{L_{t,r}^2} = \frac{1}{n} \sum_{m=0}^{n-1} L_{t,r,m}^2 \tag{16.3}$$

The variance, which is the square of the standard deviation, of the mutual inductance is defined to be [21]:

$$\mathcal{L}_{t,r}^2 = \frac{1}{n-1} \sum_{m=0}^{n-1} \left(L_{t,r,m} - \overline{L_{t,r}} \right)^2 \tag{16.4}$$

Some algebra shows that the variance can be expressed as:

$$\mathcal{L}_{t,r}^2 = \frac{n}{n-1}\left(\overline{L_{t,r}^2} - \overline{L_{t,r}}^2\right) \tag{16.5}$$

This formulation is useful for calculation, since the data array needs to be traversed only once.

We define the dimensionless noise index (NI) for a group of receiver mutual-inductance measurements as the normalized sum of the variances of all sixteen $L_{t,r}$ matrix elements:

$$N_i = \frac{n-1}{n}\,\frac{\sum_{t=0}^{3}\sum_{r=0}^{3}\mathcal{L}_{t,r}^2}{\sum_{t=0}^{3}\sum_{r=0}^{3}\overline{L_{t,r}^2}} \tag{16.6}$$

Expanding 16.6 for calculation gives:

$$N_i = \frac{\sum_{t=0}^{3}\sum_{r=0}^{3}\left(\overline{L_{t,r}^2} - \overline{L_{t,r}}^2\right)}{\sum_{t=0}^{3}\sum_{r=0}^{3}\overline{L_{t,r}^2}} \tag{16.7}$$

Expanding 16.6 for comparison to 11.1 gives:

$$N_i = \frac{\sum_{t=0}^{3}\sum_{r=0}^{3}\sum_{m=0}^{n-1}\left(L_{t,r,m} - \overline{L_{t,r}}\right)^2}{\sum_{t=0}^{3}\sum_{r=0}^{3}\overline{L_{t,r}^2}} \tag{16.8}$$

A similar noise index can be defined for the transmitter current measurements. The calculations in Appendix Q show that the transmitter-current noise index is much smaller than the mutual-inductance noise index, so we can ignore the transmitter-current noise index when calculating the mutual-inductance noise index.

Chapter 17

Measurement results and analysis

The detailed measurement results are given in Appendices O, Q, and S. The simplified results presented in Table 17.1 are based on the third-fit results in Appendix O. Note that there is little difference between second-fit and third-fit results, showing that three fits are enough.

We define the mechanical range R_{mech} to be the distance from the center of the transmitter tetrahedron to the center of the receiver tetrhedron based on the ideal mechanical position:

$$R_{mech} = \sqrt{x_{mech}^2 + y_{mech}^2 + z_{mech}^2} \tag{17.1}$$

Similarly, we define the fitted range R_{fit} to be:

$$R_{fit} = \sqrt{x_{fit}^2 + y_{fit}^2 + z_{fit}^2} \tag{17.2}$$

The orientation error quaternion $\mathcal{O}_{err}$ is the ratio of the fitted orientation quaternion $\mathcal{O}_{fit}$ to the nominal mechanical orientation quaternion $\mathcal{O}_{nom}$:

$$\mathcal{O}_{err} = \frac{\mathcal{O}_{fit}}{\mathcal{O}_{nom}} \tag{17.3}$$

The total orientation angle error in radians $\mathtt{A}_{err}$ is, assuming that the angle is small, twice the magnitude of the imaginary part of the quaternion $\mathcal{O}_{err}$:

$$\mathtt{A}_{err} = 2|\Im\left(\mathcal{O}_{err}\right)| \tag{17.4}$$

The location, goodness of fit G_f, mutual-inductance noise index N_i, ratio of N_i to G_f, fitted range R_{fit}, nominal mechanical range in the experimental fixture R_{mech}, and orien-

Table 17.1: Measurement results summary, locations sorted by range

location	G_f	N_i	$\frac{N_i}{G_f}$	R_{fit}, meters	R_{mech}, meters	$\mathtt{A}_{err}$, radians
1	3.3e-6	0.3e-6	0.08	0.2015	0.2017	0.013
4	2.8e-6	0.4e-6	0.15	0.2082	0.2090	0.010
3	4.1e-6	6.1e-6	1.5	0.3364	0.3361	0.013
7	2.9e-6	8.8e-6	3.0	0.3438	0.3449	0.010
2	1.1e-5	4.2e-5	3.6	0.4760	0.4751	0.023
11	4.8e-6	32.4e-6	6.8	0.4830	0.4845	0.008
5	1.8e-5	8.1e-5	4.6	0.5110	0.5104	0.026
10	1.4e-5	4.1e-5	3.0	0.5150	0.5163	0.009
6	3.3e-5	7.9e-5	2.4	0.5795	0.5795	0.023
9	5.6e-5	16.0e-5	2.9	0.5810	0.5821	0.013
8	9.3e-5	17.1e-5	1.8	0.6713	0.6720	0.019

tation error angle $\mathtt{A}_{err}$ of the eleven measured data points are listed in order of increasing range in Table 17.1.

The standard deviations of the measured mutual inductances reported in Appendix Q do not vary much among the eleven measurement locations. The mutual inductances shrink as the range increases, so the noise index increases as the range increases.

The R_{fit} values included in Table 17.1 are independent of orientation errors, so can be usefully compared to the R_{mech} values.

The $\mathtt{A}_{err}$ values included in Table 17.1 are independent of position.

The error in each measured mutual inductance L_m is on the order of the square root of G_f, so we see that the L_m error varies from 1700 ppm to 9300 ppm, so we did not achieve our goal of 1000 ppm.

Chapter 18

Future directions

Further improvements in the accuracy of the mutual inductances can be achieved by more careful mechanical assembly of the printed-circuit boards into tetrahedra, and by tighter control of the accuracy of the board fabrication. Multilayer boards can be used to get more turns and hence higher mutual inductances, which give better signal-to-noise ratios.

Using an array of planar coils on a single printed-circuit board would improve the accuracy by eliminating the interboard mechanical-assembly tolerances.

Chapter 19

Conclusions

We have described a coil built as straight-sided spiral tracks on a printed-circuit board. We have derived an analytic expression for the magnetic field produced by a current flowing in such a coil. This expression is easily evaluated numerically as needed.

We have described building four of these boards into a tetrahedron.

We have derived an analytic expression for the mutual inductance between two coils, one on one tetrahedron and the other on a second tetrahedron. We have reduced this expression to an integral easily evaluated numerically.

We have compared numerically-calculated mutual inductances with measured mutual inductances and found substantial agreement, though not as good as we had hoped to achieve.

Bibliography

[1] National Institute of Science and Technology. *http://www.nist.gov*, 2000.

[2] Gaylord P. Harnwell. *Principles of Electricity and Electromagnetism*, page 322. McGraw-Hill, second edition, 1949.

[3] James Clerk Maxwell. *A Treatise on Electricity and Magnetism*, volume 2, page 172. Clarendon Press, 1891. Reprinted by Dover Publications in 1954.

[4] James Clerk Maxwell. *A Treatise on Electricity and Magnetism*, volume 2, page 190. Clarendon Press, 1891. Reprinted by Dover Publications in 1954.

[5] William R. Smythe. *Static and Dynamic Electricity*, page 5. Hemisphere Publishing Corporation, third edition, 1989. Revised printing.

[6] Frederick H. Raab, Ernest B. Blood, Terry O. Steiner, and Herbert R. Jones. Magnetic position and orientation tracking system. *IEEE Transactions on Aerospace and Electronic Systems*, AES-15(5):709–718, September 1979.

[7] J. Kuipers. Object tracking and orientation determination means, system, and process. *U.S.Patent 3,868,565*, 1975.

[8] Herbert R. Jones. Method and apparatus for determining remote object orientation and position. *U.S. Patent 4,737,794*, 1988.

[9] Mark R. Schneider. Measuring position and orientation using magnetic fields. *US patent 6,073,043*, 2000.

[10] A. R. DeRuyck and J. Kuipers. An electromagnetic position sensor. Technical report, Polhemus Navigation Sciences, Inc., Burlington, VT, November 1973. AFAL-TR-73-281.

[11] Julius Adams Stratton. *Electromagnetic Theory*, page 263. McGraw-Hill, 1941.

[12] Jr. William H. Hayt. *Engineering Electromagnetics*, pages 244–245. McGraw-Hill, fourth edition, 1981.

[13] Jr. William H. Hayt. *Engineering Electromagnetics*, pages 16–20. McGraw-Hill, fourth edition, 1981.

[14] Erwin Kreyszig. *Advanced Engineering Mathematics*, pages 1021–1027. John Wiley and Sons,Inc, sixth edition, 1988.

[15] Frederick H. Raab. Quasi-static magnetic-field technique for determining position and orientation. *IEEE Transactions on Geoscience and Remote Sensing*, GE-19(4):235–243, October 1981.

[16] Edwin H. Armstrong. A study of heterodyne amplification by the electron relay. *Procceedings of the IRE*, 5:145–159, April 1917. The electron relay is the vacuum-tube triode.

[17] Edwin H. Armstrong. A new system of short wave amplification. *Procceedings of the IRE*, 9:3–27, February 1921. First article on the superheterodyne receiver.

[18] C. L. Dolph. A current distribution for broadside arrays which optimizes the relationship between beam width and sidelobe level. *Procceedings of the IRE*, 35:335–348, June 1946.

[19] H. D. Helms. Nonrecursive digital filters: Design methods for achieving specifications on frequency response. *IEEE Transactions on Audio and Electroacoustics*, AU-16(3):336–342, September 1968.

[20] ed. Digital Signal Processing Committee (C. J. Weinstein, Chair). *Programs for Digital Signal Processing*, pages 5.2–1 to 5.2–19. IEEE Press, 1979. FORTRAN program for designing FIR filters.

[21] Erwin Kreyszig. *Advanced Engineering Mathematics*, page 740. John Wiley and Sons,Inc, second edition, 1968.

[22] Julius Adams Stratton. *Electromagnetic Theory*, page 437. McGraw-Hill, 1941.

[23] G. Lee Beauregard. Mutual-inductance formulation. Personal communication.

[24] Julius Adams Stratton. *Electromagnetic Theory*, page 435. McGraw-Hill, 1941.

[25] Philip M. Morse and Herman Feshbach. *Methods of Theoretical Physics*, volume 1, pages 10–61. McGraw-Hill, 1953.

[26] Robert C. Weast; editor. *Handbook of Chemistry and Physics*, page A124. The Chemical Rubber Company, fifty-second edition, 1971.

[27] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes, The Art of Scientific Computing (FORTRAN version)*, pages 105–106. Cambridge University Press, 1989.

Appendix A

Accuracy of the quasi-static approximation

Throughout this paper, we make the quasi-static approximation: We assume that the fields are varying slowly enough that we can calculate as if the fields were static. Hence the name, quasi-static.

Equivalently, we are assuming that the dimensions of our apparatus, including the separation between transmitter and receiver, are tiny compared to the wavelengths of electromagnetic waves at the frequencies we are using.

Our transmitter driver frequencies are between $f_{min} = 10$ KHz and $f_{max} = 15$ KHz. The corresponding wavelengths are, where $c = 3 * 10^8$ meters/second is the speed of light:

$\lambda_{max} = \frac{c}{f_{min}} = 30,000$ meters

$\lambda_{min} = \frac{c}{f_{max}} = 20,000$ meters

The corresponding wavenumbers are:

$k_{min} = \frac{2\pi\, f_{min}}{c} = 0.000209$/meter

$k_{max} = \frac{2\pi\, f_{max}}{c} = 0.000314$/meter

The largest transmitter-receiver separation we use is less than $R_{max} = 0.5$ meter, so the maximum kR is:

$$(kR)_{max} = 0.000157 \tag{A.1}$$

Stratton [22] showed that the field generated by a dipole coil driven by a sinewave of frequency f contains three components: a magnetic-field radial component $\overline{H_r}$, a magnetic-

field tangential component $\overline{H_t}$, and an electric-field tangential component $\overline{E_t}$. Note that $\overline{H_t}$ and $\overline{E_t}$ are perpendicular to each other.

The current in the coil is a sinewave at frequency f, so we use the complex exponential formulation to describe the time variation. The current in the coil is:

$$I_t = Ie^{-\jmath\omega t} \tag{A.2}$$

where

I is the peak current in the coil

$\jmath = \sqrt{-1}$ used in complex numbers

$\omega = 2\pi\, f$

The magnitudes of the three field components are [22]:

$$H_r = e^{-\jmath\omega t}\frac{IAe}{2\pi}\,\frac{1}{R^3}\,(1-\jmath kR)\cos(\theta) \tag{A.3}$$

$$H_t = e^{-\jmath\omega t}\frac{IAe}{4\pi}\,\frac{1}{R^3}\left(1-\jmath kR-(kR)^2\right)\sin(\theta) \tag{A.4}$$

$$E_t = e^{-\jmath\omega t}\frac{IAe}{4\pi}\sqrt{\frac{\mu_0}{\epsilon_0}}\,\frac{1}{R^3}\left(\jmath kR+(kR)^2\right)\sin(\theta) \tag{A.5}$$

where

A_e = effective area of coil

R = distance from dipole coil to field observer

μ_0 = magnetic permeability of free space

ϵ_0 = electric permittivity of free space

θ = colatitude angle of field observer

In the limit of large R, we keep the most positive powers of R, and get the familiar radiation fields:

$$H_{r\,rad} = 0 \tag{A.6}$$

$$H_{t\,rad} = e^{-\jmath\omega t}\frac{IAe}{4\pi}\,\frac{-k^2}{R}\sin(\theta) \tag{A.7}$$

$$E_{t\,rad} = e^{-\jmath\omega t}\frac{IAe}{4\pi}\sqrt{\frac{\mu_0}{\epsilon_0}}\,\frac{k^2}{R}\sin(\theta) \tag{A.8}$$

In the limit of small R, the most negative powers of R dominate, and we rewrite equations A.3 A.4 A.5, keeping only the dominant term in the electric field:

$$H_r = e^{-\jmath\omega t}\frac{IAe}{4\pi}\,\frac{2}{R^3}\eta\cos(\theta) \tag{A.9}$$

$$H_t = e^{-\jmath\omega t}\frac{IAe}{4\pi}\,\frac{1}{R^3}\eta\zeta\sin(\theta) \tag{A.10}$$

$$E_t = e^{-\jmath\omega t}\frac{IAe}{4\pi}\,\sqrt{\frac{\mu_0}{\epsilon_0}}\,\frac{1}{R^3}\jmath kR\sin(\theta) \tag{A.11}$$

where η and ζ are corrections for the magnetic field from the exactly-static condition:

$$\eta = (1-\jmath kR) \tag{A.12}$$

$$\zeta = \frac{(1-\jmath kR-(kR)^2)}{(1-\jmath kR)} = 1-\frac{(kR)^2+\jmath(kR)^3}{1-(kR)^2} = 1-\frac{(kR)^2}{1-(kR)^2}\,(1+\jmath kR) \tag{A.13}$$

From equation A.1 above, we can calculate the farthest-from-unity value of ζ:

$$\zeta_{far} = 1-\frac{(0.000157)^2}{1-(0.000157)^2}\,(1+\jmath 0.000157) = 1-\left(2.47*10^{-8}(1+\jmath 0.000157)\right) \tag{A.14}$$

We are looking for accuracies to one part in 10,000, so ζ_{far} is indistinguishable from unity for our purposes.

Similarly, the farthest farthest-from-unity value of η is:

$$\eta_{far} = 1-\jmath 0.000157 \tag{A.15}$$

The magnitude of η_{far} is 0.999999988, indistingishable from unity for our purposes. The phase of η_{far} is -0.000157 radian $= 0.009$ degree, which is negligible compared to other phase errors in the electronics.

Thus, we can make the quasi-static approximation for the magnetic field without significant error:

$$H_r = e^{-\jmath\omega t}\frac{IAe}{4\pi}\,\frac{2}{R^3}\cos(\theta) \tag{A.16}$$

$$H_t = e^{-\jmath\omega t}\frac{IAe}{4\pi}\,\frac{1}{R^3}\sin(\theta) \tag{A.17}$$

To calculate the effect of the electric field, consider the ratio of E_t to H_t, which has the units of a reactance:

$$\frac{E_t}{H_t} = \sqrt{\frac{\mu_0}{\epsilon_0}} \jmath kR \tag{A.18}$$

From equation A.1 above, we can calculate the maximum value of $\frac{E_t}{H_t}$:

$$\left(\frac{E_t}{H_t}\right)_{max} = (376.6\, ohms)\, \jmath 0.000157 = \jmath 0.059\, ohms \tag{A.19}$$

This is a tiny reactance compared to the reactance of the transmitter coil.

Appendix B

Derivation of mutual inductance between two dipole coils

Here, we derive the mutual inductance between two magnetic dipole coils, A and B [23] (This result appears in Stratton [24] in field form rather than in mutual-inductance form):

$$L_m = \frac{\mu_o AB}{4\pi R^3}\left(3(\widehat{a}\cdot\widehat{r})(\widehat{r}\cdot\widehat{b}) - \widehat{a}\cdot\widehat{b}\right) \tag{B.1}$$

where

$\widehat{a}$ = unit vector in direction of dipole A

A = magnitude of effective area of dipole A

$\widehat{b}$ = unit vector in direction of dipole B

B = magnitude of effective area of dipole B

$\widehat{r}$ = unit vector in direction from center of dipole A to center of dipole B

R = magnitude of distance from center of dipole A to center of dipole B

Each dipole is replaced by a small but finite-sized single-turn square coil. Each coil is shrunk to an infinitesimal size (while increasing its number of turns to infinity) to obtain the dipole limit. In the dipole limit, the shape of the coil does not matter. The coil's properties are described by the coil's effective area, the product of the coil's geometrical area and number of turns.

The mutual inductance, L_m, between two closed circuits is given by Neuman's formula

[3] [4] [2]:

$$L_m = \frac{\mu_o}{4\pi} \oint_{circuit A} \oint_{circuit B} \frac{d\overline{s_A} \cdot d\overline{s_B}}{r_{AB}} \tag{B.2}$$

where

r_{AB} = distance between point on circuit A and point on circuit$B$

$\mu_o = 4\pi 10^{-7} henries/meter$

Here, we consider the case where each circuit is composed of four straight-line segments.

Define some quantities:

$\overline{A} = A\widehat{a}$ = effective area of dipole A

$\widehat{u}$ and $\widehat{v}$ are two unit vectors such that $\widehat{a} = \widehat{u} \times \widehat{v}$

$\overline{B} = B\widehat{b}$ = effective area of dipole B

$\widehat{p}$ and $\widehat{q}$ are two unit vectors such that $\widehat{b} = \widehat{p} \times \widehat{q}$

h = a small dimensionless scalar which goes to zero in the dipole limit

$\overline{R} = R\widehat{r}$ = vector from center of dipole A to center of dipole B

Use the $\widehat{u}$, $\widehat{v}$, $\widehat{a}$ cartesian coordinate system, where dipole A is at the origin and dipole B is at $\overline{R} = R\widehat{r} = u\widehat{u} + v\widehat{v} + a\widehat{a}$.

The four corners of coil A are:

$\overline{A_1} = \frac{h}{2}\widehat{u} + \frac{h}{2}\widehat{v}$

$\overline{A_2} = \frac{h}{2}\widehat{u} - \frac{h}{2}\widehat{v}$

$\overline{A_3} = -\frac{h}{2}\widehat{u} - \frac{h}{2}\widehat{v}$

$\overline{A_4} = -\frac{h}{2}\widehat{u} + \frac{h}{2}\widehat{v}$

Note that the ratio of the effective area of dipole A to the effective area of coil A is $\frac{A}{h^2}$.

A point on coil A is given by:

$(\mu, \nu) = \mu\,\widehat{u} + \nu\,\widehat{v}$

where μ has nothing to do with μ_o.

The four corners of coil B are:

$\overline{B_1} = \frac{h}{2}\widehat{p} + \frac{h}{2}\widehat{q} + \overline{R}$

$\overline{B_2} = \frac{h}{2}\widehat{p} - \frac{h}{2}\widehat{q} + \overline{R}$

$\overline{B_3} = -\frac{h}{2}\widehat{p} - \frac{h}{2}\widehat{q} + \overline{R}$

$\overline{B_4} = -\frac{h}{2}\widehat{p} + \frac{h}{2}\widehat{q} + \overline{R}$

Note that the ratio of the effective area of dipole B to the effective area of coil B is $\frac{B}{h^2}$.

A point on coil B is given by:

$(p,q) = p\,\widehat{p} + q\,\widehat{q} + \overline{R}$

The distance from a point on coil A to a point on coil B is:

$r_{AB} = |\ \overline{s_A} - p\,\widehat{p} - q\,\widehat{q}\ |$

where the vector from center of coil B to the point on coil A is:

$\overline{s_A} = \mu\,\widehat{u} + \nu\,\widehat{v} - \overline{R}$

Expand the integral around circuit B in equation B.2 by explicitly going around circuit B from point $\overline{B_4}$ through points $\overline{B_1}$, $\overline{B_2}$, $\overline{B_3}$, and back to point $\overline{B_4}$:

$$L_m = \frac{\mu_o B}{4\pi} \lim_{h\to 0} \oint_A \int_{-\frac{h}{2}}^{+\frac{h}{2}} \overline{I_3} \cdot d\overline{s_A} \tag{B.3}$$

where the integrand $\overline{I_3}$ in equation B.3 is:

$$\overline{I_3} = \frac{1}{h^2}\left(\frac{\widehat{p}\,db}{|\overline{s_A} - bh\widehat{p} - \frac{h}{2}\widehat{q}|} - \frac{\widehat{q}\,db}{|\overline{s_A} - \frac{h}{2}\widehat{p} + bh\widehat{q}|} - \frac{\widehat{p}\,db}{|\overline{s_A} + bh\widehat{p} + \frac{h}{2}\widehat{q}|} + \frac{\widehat{q}\,db}{|\overline{s_A} + \frac{h}{2}\widehat{p} - bh\widehat{q}|}\right) \tag{B.4}$$

Define g to be the inverse of the distance from a point on coil A to a point on coil B:

$$g = \frac{1}{r_{AB}} = \frac{1}{|\ \overline{s_A} - p\,\widehat{p} - q\,\widehat{q}\ |} \tag{B.5}$$

Calculate some directional partial derivatives along the $\widehat{p}$ and $\widehat{q}$ axes, remembering that $\overline{s_A}$ is independent of p and q:

$$\left[\frac{\partial g}{\partial p}\right]_{p=0}^{q=\frac{h}{2}} = \frac{\partial}{\partial p}\left[\frac{1}{|\overline{s_A} - p\widehat{p} - q\widehat{q}|}\right]_{p=0,q=\frac{h}{2}} = \lim_{\epsilon\to 0}\frac{1}{2b\epsilon}\left(\frac{1}{|\overline{s_A} - b\epsilon\widehat{p} - \frac{h}{2}\widehat{q}|} - \frac{1}{|\overline{s_A} + b\epsilon\widehat{p} - \frac{h}{2}\widehat{q}|}\right) \tag{B.6}$$

$$\left[\frac{\partial g}{\partial q}\right]_{p=-bh}^{q=0} = \frac{\partial}{\partial q}\left[\frac{1}{|\overline{s_A} - p\widehat{p} - q\widehat{q}|}\right]_{p=-bh,q=0} = \lim_{\epsilon\to 0}\frac{1}{\epsilon}\left(\frac{1}{|\overline{s_A} + bh\widehat{p} - \frac{\epsilon}{2}\widehat{q}|} - \frac{1}{|\overline{s_A} + bh\widehat{p} + \frac{\epsilon}{2}\widehat{q}|}\right) \tag{B.7}$$

$$\left[\frac{\partial g}{\partial p}\right]_{p=0}^{q=-bh} = \frac{\partial}{\partial p}\left[\frac{1}{|\overline{s_A} - p\widehat{p} - q\widehat{q}|}\right]_{p=0,q=-bh} = \lim_{\epsilon\to 0}\frac{1}{\epsilon}\left(\frac{1}{|\overline{s_A} - \frac{\epsilon}{2}\overline{p} + bh\widehat{q}|} - \frac{1}{|\overline{s_A} + \frac{\epsilon}{2}\overline{p} + bh\widehat{q}|}\right) \tag{B.8}$$

$$\left[\frac{\partial g}{\partial q}\right]_{p=-\frac{h}{2}}^{q=0} = \frac{\partial}{\partial q}\left[\frac{1}{|\overline{s_A} - p\widehat{p} - q\widehat{q}|}\right]_{p=-\frac{h}{2},q=0} = \lim_{\epsilon\to 0}\frac{1}{2b\epsilon}\left(\frac{1}{|\overline{s_A} + \frac{h}{2}\widehat{p} - b\epsilon\widehat{q}|} - \frac{1}{|\overline{s_A} + \frac{h}{2}\widehat{p} + b\epsilon\widehat{q}|}\right) \tag{B.9}$$

Set $\epsilon = h$, and substitute into equation B.4:

$$\overline{I_3} = \frac{1}{h}\left(2b\widehat{p}\left[\frac{\partial g}{\partial p}\right]_{p=0}^{q=\frac{h}{2}} + \widehat{p}\left[\frac{\partial g}{\partial q}\right]_{p=-bh}^{q=0} - \widehat{q}\left[\frac{\partial g}{\partial p}\right]_{p=0}^{q=-bh} + 2b\widehat{q}\left[\frac{\partial g}{\partial q}\right]_{p=-\frac{h}{2}}^{q=0}\right) \tag{B.10}$$

Substitute into equation B.3 and perform the integration over b:

$$L_m = \frac{\mu_o B}{4\pi} \lim_{h\to 0} \oint_A \left(\widehat{p} \left[\frac{\partial g}{\partial q}\right]_{p=-bh}^{q=0} - \widehat{q} \left[\frac{\partial g}{\partial p}\right]_{p=0}^{q=-bh} \right) \cdot d\overline{s_A} \tag{B.11}$$

Expand the integral around circuit A in equation B.11 by explicitly going around circuit A from point $\overline{A_4}$ through points $\overline{A_1}$, $\overline{A_2}$, $\overline{A_3}$, and back to point $\overline{A_4}$:

$$L_m = \frac{\mu_o B}{4\pi} \lim_{h\to 0} (I_1 + I_2 + I_3 + I_4) \tag{B.12}$$

where:

$$I_1 = \frac{A}{h^2} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left(\widehat{p} \left[\frac{\partial g}{\partial q}\right]_{p=q=0} - \widehat{q} \left[\frac{\partial g}{\partial p}\right]_{p=q=0} \right)_{\nu=\frac{h}{2}} \cdot \widehat{u}\, d\mu \tag{B.13}$$

$$I_2 = -\frac{A}{h^2} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left(\widehat{p} \left[\frac{\partial g}{\partial q}\right]_{p=q=0} - \widehat{q} \left[\frac{\partial g}{\partial p}\right]_{p=q=0} \right)_{\mu=\frac{h}{2}} \cdot \widehat{v}\, d\nu \tag{B.14}$$

$$I_3 = -\frac{A}{h^2} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left(\widehat{p} \left[\frac{\partial g}{\partial q}\right]_{p=q=0} - \widehat{q} \left[\frac{\partial g}{\partial p}\right]_{p=q=0} \right)_{\nu=-\frac{h}{2}} \cdot \widehat{u}\, d\mu \tag{B.15}$$

$$I_4 = \frac{A}{h^2} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left(\widehat{p} \left[\frac{\partial g}{\partial q}\right]_{p=q=0} - \widehat{q} \left[\frac{\partial g}{\partial p}\right]_{p=q=0} \right)_{\mu=-\frac{h}{2}} \cdot \widehat{v}\, d\nu \tag{B.16}$$

Rearrange to separate the four dotproducts:

$$L_m = \frac{\mu_o AB}{4\pi} \lim_{h\to 0} (I_{pu}\, \widehat{p} \cdot \widehat{u} - I_{qu}\, \widehat{q} \cdot \widehat{u} - I_{pv}\, \widehat{p} \cdot \widehat{v} + I_{qv}\, \widehat{q} \cdot \widehat{v}) \tag{B.17}$$

where:

$$I_{pu} = \frac{1}{h^2} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left(\left[\frac{\partial g}{\partial q}\right]_{p=q=0}^{\nu=\frac{h}{2}} - \left[\frac{\partial g}{\partial q}\right]_{p=q=0}^{\nu=-\frac{h}{2}} \right) d\mu \tag{B.18}$$

$$I_{qu} = \frac{1}{h^2} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left(\left[\frac{\partial g}{\partial p}\right]_{p=q=0}^{\nu=\frac{h}{2}} - \left[\frac{\partial g}{\partial p}\right]_{p=q=0}^{\nu=-\frac{h}{2}} \right) d\mu \tag{B.19}$$

$$I_{pv} = \frac{1}{h^2} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left(\left[\frac{\partial g}{\partial q}\right]_{p=q=0}^{\mu=\frac{h}{2}} - \left[\frac{\partial g}{\partial q}\right]_{p=q=0}^{\mu=-\frac{h}{2}} \right) d\nu \tag{B.20}$$

$$I_{qv} = \frac{1}{h^2} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left(\left[\frac{\partial g}{\partial p} \right]_{p=q=0}^{\mu=\frac{h}{2}} - \left[\frac{\partial g}{\partial p} \right]_{p=q=0}^{\mu=-\frac{h}{2}} \right) d\nu \tag{B.21}$$

Apply the definition of a partial derivative and the fact that h is small to equations B.18, B.19, B.20, and B.21:

$$I_{pu} = \frac{1}{h} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left[\frac{\partial^2 g}{\partial \nu \partial q} \right]_{p=q=0}^{\mu=\nu=0} d\mu = \left[\frac{\partial^2 g}{\partial \nu \partial q} \right]_{p=q=0}^{\mu=\nu=0} \tag{B.22}$$

$$I_{qu} = \frac{1}{h} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left[\frac{\partial^2 g}{\partial \nu \partial p} \right]_{p=q=0}^{\mu=\nu=0} d\mu = \left[\frac{\partial^2 g}{\partial \nu \partial p} \right]_{p=q=0}^{\mu=\nu=0} \tag{B.23}$$

$$I_{pv} = \frac{1}{h} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left[\frac{\partial^2 g}{\partial \mu \partial q} \right]_{p=q=0}^{\mu=\nu=0} d\nu = \left[\frac{\partial^2 g}{\partial \mu \partial q} \right]_{p=q=0}^{\mu=\nu=0} \tag{B.24}$$

$$I_{qv} = \frac{1}{h} \int_{-\frac{h}{2}}^{\frac{h}{2}} \left[\frac{\partial^2 g}{\partial \mu \partial p} \right]_{p=q=0}^{\mu=\nu=0} d\nu = \left[\frac{\partial^2 g}{\partial \mu \partial p} \right]_{p=q=0}^{\mu=\nu=0} \tag{B.25}$$

Recall that g is:

$$g = \frac{1}{r_{AB}} = \frac{1}{\mid \mu\, \widehat{u} + \nu\, \widehat{v} - \overline{R} - p\, \widehat{p} - q\, \widehat{q} \mid} = \frac{1}{\mid (\mu - u)\, \widehat{u} + (\nu - v)\, \widehat{v} - a\, \widehat{a} - p\, \widehat{p} - q\, \widehat{q} \mid} \tag{B.26}$$

Then:

$$I_{pu} = \left[\frac{\partial^2 g}{\partial \nu \partial q} \right]_{p=q=0}^{\mu=\nu=0} = - \left[\frac{\partial^2 g}{\partial v \partial q} \right]_{p=q=0}^{\mu=\nu=0} \tag{B.27}$$

$$I_{qu} = \left[\frac{\partial^2 g}{\partial \nu \partial p} \right]_{p=q=0}^{\mu=\nu=0} = - \left[\frac{\partial^2 g}{\partial v \partial p} \right]_{p=q=0}^{\mu=\nu=0} \tag{B.28}$$

$$I_{pv} = \left[\frac{\partial^2 g}{\partial \mu \partial q} \right]_{p=q=0}^{\mu=\nu=0} = - \left[\frac{\partial^2 g}{\partial u \partial q} \right]_{p=q=0}^{\mu=\nu=0} \tag{B.29}$$

$$I_{qv} = \left[\frac{\partial^2 g}{\partial \mu \partial p} \right]_{p=q=0}^{\mu=\nu=0} = - \left[\frac{\partial^2 g}{\partial u \partial p} \right]_{p=q=0}^{\mu=\nu=0} \tag{B.30}$$

Substitute into equation B.17:

$$L_m = \frac{\mu_o AB}{4\pi} \left(-\frac{\partial^2 g}{\partial v \partial q} \widehat{p} \cdot \widehat{u} + \frac{\partial^2 g}{\partial v \partial p} \widehat{q} \cdot \widehat{u} + \frac{\partial^2 g}{\partial u \partial q} \widehat{p} \cdot \widehat{v} - \frac{\partial^2 g}{\partial u \partial p} \widehat{q} \cdot \widehat{v} \right) \tag{B.31}$$

Calculate g in equation B.31 and its partial derivatives in the $\widehat{u}$, $\widehat{v}$, $\widehat{a}$ cartesian coordinate system, where dipole A is at the origin and dipole B is at $\overline{R} = R\widehat{r} = u\widehat{u} + v\widehat{v} + a\widehat{a}$:

$g = \frac{1}{R} = \frac{1}{\sqrt{u^2+v^2+a^2}}$

$\frac{\partial g}{\partial p} = \widehat{p}\cdot\widehat{u}\frac{\partial g}{\partial u} + \widehat{p}\cdot\widehat{v}\frac{\partial g}{\partial v} + \widehat{p}\cdot\widehat{a}\frac{\partial g}{\partial a} = -g^3\left(\widehat{p}\cdot\widehat{u}u + \widehat{p}\cdot\widehat{v}v + \widehat{p}\cdot\widehat{a}a\right)$

$\frac{\partial^2 g}{\partial u \partial p} = 3g^5\left(\widehat{p}\cdot\widehat{u}u + \widehat{p}\cdot\widehat{v}v + \widehat{p}\cdot\widehat{a}a\right)u - g^3\widehat{p}\cdot\widehat{u}$

$\frac{\partial^2 g}{\partial v \partial p} = 3g^5\left(\widehat{p}\cdot\widehat{u}u + \widehat{p}\cdot\widehat{v}v + \widehat{p}\cdot\widehat{a}a\right)v - g^3\widehat{p}\cdot\widehat{v}$

$\frac{\partial g}{\partial q} = \widehat{q}\cdot\widehat{u}\frac{\partial g}{\partial u} + \widehat{q}\cdot\widehat{v}\frac{\partial g}{\partial v} + \widehat{q}\cdot\widehat{a}\frac{\partial g}{\partial a} = -g^3\left(\widehat{q}\cdot\widehat{u}u + \widehat{q}\cdot\widehat{v}v + \widehat{q}\cdot\widehat{a}a\right)$

$\frac{\partial^2 g}{\partial u \partial q} = 3g^5\left(\widehat{q}\cdot\widehat{u}u + \widehat{q}\cdot\widehat{v}v + \widehat{q}\cdot\widehat{a}a\right)u - g^3\widehat{q}\cdot\widehat{u}$

$\frac{\partial^2 g}{\partial v \partial q} = 3g^5\left(\widehat{q}\cdot\widehat{u}u + \widehat{q}\cdot\widehat{v}v + \widehat{q}\cdot\widehat{a}a\right)v - g^3\widehat{q}\cdot\widehat{v}$

Substitute into equation B.31:

$$L_m = \frac{\mu_o AB}{4\pi\, R^3}\left(\frac{-3}{R^2}F - T\right) \tag{B.32}$$

where

$T = 2\left((\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u}) - (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})\right) = -2(\widehat{a}\cdot\widehat{b})$

and

$F = \left(\widehat{q}\cdot\widehat{u}u + \widehat{q}\cdot\widehat{v}v + \widehat{q}\cdot\widehat{a}a\right)\widehat{p}\cdot\widehat{u}v - \left(\widehat{p}\cdot\widehat{u}u + \widehat{p}\cdot\widehat{v}v + \widehat{p}\cdot\widehat{a}a\right)\widehat{q}\cdot\widehat{u}v - \left(\widehat{q}\cdot\widehat{u}u + \widehat{q}\cdot\widehat{v}v + \widehat{q}\cdot\widehat{a}a\right)\widehat{p}\cdot\widehat{v}\,u + \left(\widehat{p}\cdot\widehat{u}u + \widehat{p}\cdot\widehat{v}v + \widehat{p}\cdot\widehat{a}a\right)\widehat{q}\cdot\widehat{v}\,u$

Expand F:

$F = (\widehat{q}\cdot\widehat{u})(\widehat{p}\cdot\widehat{u})\,vu + (\widehat{q}\cdot\widehat{v})(\widehat{p}\cdot\widehat{u})\,v^2 + (\widehat{q}\cdot\widehat{a})(\widehat{p}\cdot\widehat{u})\,va - (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{u})\,vu - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})\,v^2 - (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{u})va - (\widehat{q}\cdot\widehat{u})(\widehat{p}\cdot\widehat{v})u^2 - (\widehat{q}\cdot\widehat{v})(\widehat{p}\cdot\widehat{v})uv - (\widehat{q}\cdot\widehat{a})(\widehat{p}\cdot\widehat{v})ua + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})u^2 + (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{v})uv + (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{v})ua$

Rearrange terms:

$F = +(\widehat{q}\cdot\widehat{a})(\widehat{p}\cdot\widehat{u})\,va - (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{u})\,va + (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{v})\,ua - (\widehat{q}\cdot\widehat{a})(\widehat{p}\cdot\widehat{v})\,ua - (\widehat{q}\cdot\widehat{u})(\widehat{p}\cdot\widehat{v})\,u^2 + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})u^2 + (\widehat{q}\cdot\widehat{v})(\widehat{p}\cdot\widehat{u})v^2 - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})v^2 + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{u})vu - (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{u})vu - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{v})uv + (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{v})uv$

Eliminate cancelling terms at end:

$F = +(\widehat{q}\cdot\widehat{a})(\widehat{p}\cdot\widehat{u})\,va - (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{u})\,va + (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{v})\,ua - (\widehat{q}\cdot\widehat{a})(\widehat{p}\cdot\widehat{v})\,ua - (\widehat{q}\cdot\widehat{u})(\widehat{p}\cdot\widehat{v})\,u^2 + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})\,u^2 + (\widehat{q}\cdot\widehat{v})(\widehat{p}\cdot\widehat{u})\,v^2 - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})\,v^2$

Insert new cancelling terms at end:

$F = +(\widehat{q}\cdot\widehat{a})(\widehat{p}\cdot\widehat{u})\,va - (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{u})\,va + (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{v})\,ua - (\widehat{q}\cdot\widehat{a})(\widehat{p}\cdot\widehat{v})\,ua - (\widehat{q}\cdot\widehat{u})(\widehat{p}\cdot\widehat{v})\,u^2 + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})u^2 + (\widehat{q}\cdot\widehat{v})(\widehat{p}\cdot\widehat{u})v^2 - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})v^2 + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})a^2 + (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})a^2 + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})a^2 - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})a^2$

Since $\widehat{a} = \widehat{u}\times\widehat{v}$ and $\widehat{b} = \widehat{p}\times\widehat{q}$, we have:

$-\widehat{a}\cdot\widehat{b} = (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u}) - (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})$

Apply this equality in the u^2 and v^2 terms in F:

$F = +(\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{a})\,va - (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{u})\,va - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{a})\,ua + (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{v})\,ua + (\widehat{a}\cdot\widehat{b})\,u^2 + (\widehat{a}\cdot\widehat{b})\,v^2 - (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})\,a^2 + (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})\,a^2 + (\widehat{a}\cdot\widehat{b})\,a^2$

Rearrange terms:

$F = -(\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{u})\,va + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{a})\,va - (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})\,a^2 + (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})\,a^2 - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{a})\,ua + (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{v})\,ua + (\widehat{a}\cdot\widehat{b})\,u^2 + (\widehat{a}\cdot\widehat{b})\,v^2 + (\widehat{a}\cdot\widehat{b})\,a^2$

Recall that $R^2 = u^2 + v^2 + a^2$, and substitute for last three terms:

$F = -(\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{u})\,va + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{a})\,va - (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})\,a^2 + (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})\,a^2 - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{a})\,ua + (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{v})\,ua + (\widehat{a}\cdot\widehat{b})\,R^2$

Since $\widehat{a} = \widehat{u}\times\widehat{v}$ and $\widehat{b} = \widehat{p}\times\widehat{q}$, we have:

$\widehat{a}\cdot\widehat{r} = \frac{a}{R}$

and

$(\widehat{a}\cdot\widehat{r})(\widehat{b}\cdot\widehat{r}) = \frac{(\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{u})\,va - (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{a})\,va + (\widehat{p}\cdot\widehat{u})(\widehat{q}\cdot\widehat{v})\,a^2 - (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{u})\,a^2 + (\widehat{p}\cdot\widehat{v})(\widehat{q}\cdot\widehat{a})\,ua - (\widehat{p}\cdot\widehat{a})(\widehat{q}\cdot\widehat{v})\,ua}{R^2}$

Then:

$F = -(\widehat{a}\cdot\widehat{r})(\widehat{b}\cdot\widehat{r})\,R^2 + (\widehat{a}\cdot\widehat{b})\,R^2$

Substitute F and T into equation B.32:

$$L_m = \frac{\mu_o AB}{4\pi\,R^3}\Big(3(\widehat{a}\cdot\widehat{r})(\widehat{b}\cdot\widehat{r}) - 3(\widehat{a}\cdot\widehat{b}) + 2(\widehat{a}\cdot\widehat{b})\Big) \tag{B.33}$$

Simplify to get result B.1:

$$L_m = \frac{\mu_o AB}{4\pi\,R^3}\Big(3(\widehat{a}\cdot\widehat{r})(\widehat{r}\cdot\widehat{b}) - (\widehat{a}\cdot\widehat{b})\Big) \tag{B.34}$$

We can write equation B.34 using a dyadic operator [25]:

$$L_m = \frac{\mu_o AB}{4\pi\,R^3}\,\widehat{a}\cdot\mathcal{D}\cdot\widehat{b} \tag{B.35}$$

If the vectors $\widehat{a}$, $\widehat{b}$, and $\widehat{r}$ are three-element row vectors, then the dyadic operator $\mathcal{D}$ can be written as a 3x3 matrix, and the dot products are matrix multiplies:

$$L_m = \frac{\mu_o AB}{4\pi\,R^3}\,\widehat{a}\,\mathcal{D}\,\widehat{b}^t \tag{B.36}$$

where $\mathcal{D}$ is given by ($\mathcal{I}$ is the 3x3 identity matrix):

$$\mathcal{D} = 3\widehat{r}^t\widehat{r} - \mathcal{I} \tag{B.37}$$

Appendix C

C code to generate PCB files

```
/*  flattx3h.c generates all files needed for ANT-003 board and
model

rev 1.0, 15sep1999pta

See trakr221.txt for design details.

Output files:

flattx3a.gbr  component side copper RS274X artwork file
flattx3b.gbr  solder side copper RS274X artwork file
flattx3.h     header file of current element coordinates for
                 field model, origin is Gerber-file origin
                 on component side
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

/* X component of vector rotated through 120 degrees around
center */
long Xrot120(long xpt, long ypt)
{
  long templ;
  double xomc;
  double yomc;
  double xrmc;
  xomc = xpt - 3500;
  yomc = ypt - 2443;
  xrmc =  ( -0.5 * xomc ) - ( 0.5 * sqrt(3.0) * yomc);
  templ = (long)xrmc + 3500;
  return(templ);
}

/* Y component of vector rotated through 120 degrees around
center */
long Yrot120(long xpt, long ypt)
```

```
{
  long templ;
  double xomc;
  double yomc;
  double yrmc;
  xomc = xpt - 3500;
  yomc = ypt - 2443;
  yrmc =  ( -0.5 * yomc ) + ( 0.5 * sqrt(3.0) * xomc);
  templ = (long)yrmc + 2443;
  return(templ);
}

/* X component of vector rotated through 240 degrees around
center */
long Xrot240(long xpt, long ypt)
{
  long templ;
  double xomc;
  double yomc;
  double xrmc;
  xomc = xpt - 3500;
  yomc = ypt - 2443;
  xrmc =  ( -0.5 * xomc ) - ( -0.5 * sqrt(3.0) * yomc);
  templ = (long)xrmc + 3500;
  return(templ);
}

/* Y component of vector rotated through 240 degrees around
center */
long Yrot240(long xpt, long ypt)
{
  long templ;
  double xomc;
  double yomc;
  double yrmc;
  xomc = xpt - 3500;
  yomc = ypt - 2443;
  yrmc =  ( -0.5 * yomc ) + ( -0.5 * sqrt(3.0) * xomc);
  templ = (long)yrmc + 2443;
  return(templ);
}

main(){
  long Xrot120(long xpt, long ypt);
  long Yrot120(long xpt, long ypt);
  long Xrot240(long xpt, long ypt);
  long Yrot240(long xpt, long ypt);
  FILE *comside;
  FILE *solside;
  FILE *header;
  char temp_string[602];
  int i;
  long templong1;
  long templong2;

  int edge_max = 22;
  long lower_edge[22][2] = { 1000, 1000,
```

```
                                1000,  900,
                                1823,  900,
                                1823, 1100,
                                1923, 1100,
                                1923, 1000,
                                2567, 1000,
                                2567, 1100,
                                2667, 1100,
                                2667,  900,
                                3490,  900,
                                3490, 1100,
                                3590, 1100,
                                3590, 1000,
                                4233, 1000,
                                4233, 1100,
                                4333, 1100,
                                4333,  900,
                                5157,  900,
                                5157, 1100,
                                5257, 1100,
                                5257, 1000 };

  int regis_max = 6;
  long regis_track[18][2] = { 1367, 1040,
                              1417,  990,
                              1467, 1040,
                              2200, 1070,
                              2250, 1020,
                              2300, 1070,
                              3033, 1040,
                              3083,  990,
                              3133, 1040,
                              3867, 1070,
                              3917, 1020,
                              3967, 1070,
                              4700, 1040,
                              4750,  990,
                              4800, 1040,
                              5533, 1070,
                              5583, 1020,
                              5633, 1070 };
  int turns_max = 41;
  long radius_pitch = 40;
  long spiral_outer_radius = 2487;
  long spiral_radius = 0;

  /* open the files */
  if(
       ((comside = fopen("flattx3a.gbr","w")) != NULL) &&
       ((solside = fopen("flattx3b.gbr","w")) != NULL) &&
       ((header = fopen("flattx3.h","w")) != NULL)
    )
{

  /* file preambles */
  strcpy( temp_string, "*\n");
  strcat( temp_string, "%FSAX23Y23*%\n");
```

```
strcat( temp_string, "%MOIN*%\n");
strcat( temp_string, "%AD*%\n");
strcat( temp_string, "%ADD10C, 0.060*%\n");
strcat( temp_string, "%ADD11C, 0.003*%\n");
strcat( temp_string, "%ADD12C, 0.010*%\n");
strcat( temp_string, "%ADD13R, 0.150X0.150*%\n");
fputs(temp_string,comside);
fputs(temp_string,solside);
fputs("        /* generated by flattx3h.c, rev 1.0 */\n",
    header);
fputs("#define coil_tx_num 253\n",header);
fputs("/* coax shield, L end of coil */\n",header);
fputs("long coil_tx_points[coil_tx_num][3] = {\n",header);

/* layer-registration targets */
strcpy( temp_string, "G54D13*\n");
strcat( temp_string, "G1X01425Y03925D03*\n");
strcat( temp_string, "X01575Y04075D03*\n");
strcat( temp_string, "X06425Y00925D03*\n");
strcat( temp_string, "X06575Y01075D03*\n");
strcat( temp_string, "G54D11*\n");
strcat( temp_string, "G1X01500Y03850D02*\n");
strcat( temp_string, "X01650Y03850D01*\n");
strcat( temp_string, "X01650Y04000D01*\n");
strcat( temp_string, "X01500Y04150D02*\n");
strcat( temp_string, "X01350Y04150D01*\n");
strcat( temp_string, "X01350Y04000D01*\n");
strcat( temp_string, "X06500Y00850D02*\n");
strcat( temp_string, "X06650Y00850D01*\n");
strcat( temp_string, "X06650Y01000D01*\n");
strcat( temp_string, "X06500Y01150D02*\n");
strcat( temp_string, "X06350Y01150D01*\n");
strcat( temp_string, "X06350Y01000D01*\n");
fputs(temp_string,comside);
fputs(temp_string,solside);

/* pads for plated-through holes, all 60 mils diameter */
strcpy( temp_string, "G54D10*\n");
  /* center hole */
strcat( temp_string, "G1X03500Y02443D03*\n");
  /* connector holes */
strcat( temp_string, "X01250Y01150D03*\n");
strcat( temp_string, "X01200Y01100D03*\n");
strcat( temp_string, "X01200Y01200D03*\n");
strcat( temp_string, "X01300Y01200D03*\n");
strcat( temp_string, "X01300Y01100D03*\n");
fputs(temp_string,comside);
fputs(temp_string,solside);

/* tracks, all 10 mils wide */
strcpy( temp_string, "G54D12*\n");
strcat( temp_string, "G1");
fputs(temp_string,comside);
fputs(temp_string,solside);

/* join four connector shield holes on component side */
strcpy( temp_string, "X01200Y01100D02*\n");
```

```
strcat( temp_string, "X01200Y01200D01*\n");
strcat( temp_string, "X01300Y01200D01*\n");
strcat( temp_string, "X01300Y01100D01*\n");
strcat( temp_string, "X01200Y01100D01*\n");
fputs(temp_string,comside);

/* board outline lower side */
sprintf(temp_string,"X%5.5dY%5.5dD02*\n",lower_edge[0][0],
  lower_edge[0][1]);
fputs(temp_string,comside);
fputs(temp_string,solside);
for(i = 1; i<edge_max; i++)
  {
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",lower_edge[i][0],
    lower_edge[i][1]);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  }

/* board outline upper right side
          (lower side rotated 120 degrees) */
for(i=0; i<edge_max; i++)
  {
  templong1 = Xrot120(lower_edge[i][0], lower_edge[i][1]);
  templong2 = Yrot120(lower_edge[i][0], lower_edge[i][1]);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  }

/* board outline upper left side
          (lower side rotated 240 degrees) */
for(i=0; i<edge_max; i++)
  {
  templong1 = Xrot240(lower_edge[i][0], lower_edge[i][1]);
  templong2 = Yrot240(lower_edge[i][0], lower_edge[i][1]);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  }
sprintf(temp_string,"X%5.5dY%5.5dD01*\n",lower_edge[0][0],
  lower_edge[0][1]);
fputs(temp_string,comside);
fputs(temp_string,solside);

/* inter-board registration marks */

/* registration marks lower side */
for(i=0; i<regis_max; i++)
  {
  templong1 = regis_track[3*i][0];
  templong2 = regis_track[3*i][1];
  sprintf(temp_string,"X%5.5dY%5.5dD02*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  templong1 = regis_track[3*i+1][0];
```

```
  templong2 = regis_track[3*i+1][1];
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  templong1 = regis_track[3*i+2][0];
  templong2 = regis_track[3*i+2][1];
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  }

/* registration marks upper right side
   (lower side rotated 120 degrees) */
for(i=0; i<regis_max; i++)
  {
  templong1 = Xrot120(regis_track[3*i][0],regis_track[3*i][1]);
  templong2 = Yrot120(regis_track[3*i][0],regis_track[3*i][1]);
  sprintf(temp_string,"X%5.5dY%5.5dD02*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  templong1 = Xrot120(regis_track[3*i+1][0],
    regis_track[3*i+1][1]);
  templong2 = Yrot120(regis_track[3*i+1][0],
    regis_track[3*i+1][1]);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  templong1 = Xrot120(regis_track[3*i+2][0],
    regis_track[3*i+2][1]);
  templong2 = Yrot120(regis_track[3*i+2][0],
    regis_track[3*i+2][1]);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  }

/* registration marks upper left side
   (lower side rotated 240 degrees) */
for(i=0; i<regis_max; i++)
  {
  templong1 = Xrot240(regis_track[3*i][0],regis_track[3*i][1]);
  templong2 = Yrot240(regis_track[3*i][0],regis_track[3*i][1]);
  sprintf(temp_string,"X%5.5dY%5.5dD02*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  templong1 = Xrot240(regis_track[3*i+1][0],
     regis_track[3*i+1][1]);
  templong2 = Yrot240(regis_track[3*i+1][0],
     regis_track[3*i+1][1]);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
```

```
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  templong1 = Xrot240(regis_track[3*i+2][0],
    regis_track[3*i+2][1]);
  templong2 = Yrot240(regis_track[3*i+2][0],
    regis_track[3*i+2][1]);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  }

/* coil current path */

/* component-side path from connector shield to spiral */
templong1 = 1300;
templong2 = 1200;
sprintf(temp_string,"X%5.5dY%5.5dD02*\n",templong1,templong2);
fputs(temp_string,comside);
sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,0);
fputs(temp_string,header);

/* component-side spiral, from connector to center */
spiral_radius = spiral_outer_radius;
for(i=0; i<turns_max; i++)
  {
  templong1 = Xrot120(3500, 2443+spiral_radius);
  templong2 = Yrot120(3500, 2443+spiral_radius);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,0);
  fputs(temp_string,header);
  templong1 = Xrot240(3500, 2443+spiral_radius);
  templong2 = Yrot240(3500, 2443+spiral_radius);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,0);
  fputs(temp_string,header);
  spiral_radius = spiral_radius - radius_pitch;
  templong1 = 3500;
  templong2 = 2443+spiral_radius;
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,comside);
  sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,0);
  fputs(temp_string,header);
  }

/* center via */
templong1 = 3500;
templong2 = 2443;
sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,templong2);
fputs(temp_string,comside);
sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,0);
fputs(temp_string,header);
```

```
sprintf(temp_string,"X%5.5dY%5.5dD02*\n",templong1,templong2);
fputs(temp_string,solside);
sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,-63);
fputs(temp_string,header);

/* solder-side spiral, from center to connector */
for(i=0; i<turns_max; i++)
  {
  templong1 = Xrot240(3500, 2443+spiral_radius);
  templong2 = Yrot240(3500, 2443+spiral_radius);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,solside);
  sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,-63);
  fputs(temp_string,header);
  spiral_radius = spiral_radius + radius_pitch;
  templong1 = 3500;
  templong2 = 2443+spiral_radius;
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,solside);
  sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,-63);
  fputs(temp_string,header);
  templong1 = Xrot120(3500, 2443+spiral_radius);
  templong2 = Yrot120(3500, 2443+spiral_radius);
  sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,
    templong2);
  fputs(temp_string,solside);
  sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,-63);
  fputs(temp_string,header);
  }

/* solder-side path from spiral to connector center */
templong1 = 1346;
templong2 = 1200;
sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,templong2);
fputs(temp_string,solside);
sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,-63);
fputs(temp_string,header);
templong1 = 1346;
templong2 = 1150;
sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,templong2);
fputs(temp_string,solside);
sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,-63);
fputs(temp_string,header);
templong1 = 1250;
templong2 = 1150;
sprintf(temp_string,"X%5.5dY%5.5dD01*\n",templong1,templong2);
fputs(temp_string,solside);
sprintf(temp_string,"%d, %d, %d,\n",templong1,templong2,-63);
fputs(temp_string,header);

/* path connector center to shield to close loop */
templong1 = 1300;
templong2 = 1200;
sprintf(temp_string,"%d, %d, %d\n",templong1,templong2,0);
fputs(temp_string,header);
```

```
  /* file postambles */
  strcpy( temp_string, "M2*\n");
  fputs(temp_string,comside);
  fputs(temp_string,solside);
  fputs("};\n",header);
  fputs("/* coax center conductor, H end of coil */\n",header);

  /* close files */
  fclose( comside );
  fclose( solside );
  fclose( header );
  }
  else
    printf("file open failed\n");
}
```

Appendix D

RS274X files for PCB

Here, we present the RS274X-format, also called Gerber-format, data files used by the printed-circuit board (PCB) fabricator to fabricate the boards used in the tetrahedra.

The boards were fabricated 0.0625-inch-thick glass-epoxy material. There are two layers of copper: the component side and the solder side.

The artwork files may be viewed and inspected for errors from using any of a number of freely-available RS274X file viewer programs. We used GCPREVUE, from Graphicode (web site www.graphicode.com).

The two copper-artwork files, flattx3a.gbr and flattx3b.gbr, were created by the program flattx3h.c. The remaining files were created by hand with a text editor.

flattx3e.txt, parameters for drills used:

```
Drill Sizes Report
==================
Tool   Size   Pltd   Feed   Speed   Qty
====   ====   ====   ====   =====   ===
   1     33    x      197    550      6
```

flattx3f.txt, listing of hole locations:

```
Drill Listing
=============
Drill: 33  Tool: 1  Feed: 197  Speed: 550
X 1250  Y 1150
X 1200  Y 1100
X 1200  Y 1200
X 1300  Y 1200
X 1300  Y 1100
X 3500  Y 2443

End of Listing
```

flattx3d.drl, numerically-controlled (NC) tool drill file:

```
%
T1F197S55
X01250Y01150
X01200Y01100
X01200Y01200
X01300Y01200
X01300Y01100
X03500Y02443
M30
```

flattx3c.gbr, soldermask (same soldermask on both sides of board) artwork RS274X data file:

```
*
%FSAX23Y23*%
%MOIN*%
%AD*%
%ADD10C, 0.060*%
G54D10*
G1X01250Y01150D03*
G1X01200Y01100D03*
G1X01200Y01200D03*
G1X01300Y01200D03*
G1X01300Y01100D03*
G1X03500Y02443D03*
M2*
```

flattx3a.gbr, component-side copper artwork RS274X data file:

```
*
%FSAX23Y23*%
%MOIN*%
%AD*%
%ADD10C, 0.060*%
%ADD11C, 0.003*%
%ADD12C, 0.010*%
%ADD13R, 0.150X0.150*%
G54D13*
G1X01425Y03925D03*
X01575Y04075D03*
X06425Y00925D03*
X06575Y01075D03*
G54D11*
G1X01500Y03850D02*
X01650Y03850D01*
X01650Y04000D01*
X01500Y04150D02*
X01350Y04150D01*
X01350Y04000D01*
X06500Y00850D02*
X06650Y00850D01*
X06650Y01000D01*
X06500Y01150D02*
X06350Y01150D01*
```

```
X06350Y01000D01*
G54D10*
G1X03500Y02443D03*
X01250Y01150D03*
X01200Y01100D03*
X01200Y01200D03*
X01300Y01200D03*
X01300Y01100D03*
G54D12*
G1X01200Y01100D02*
X01200Y01200D01*
X01300Y01200D01*
X01300Y01100D01*
X01200Y01100D01*
X01000Y01000D02*
X01000Y00900D01*
X01823Y00900D01*
X01823Y01100D01*
X01923Y01100D01*
X01923Y01000D01*
X02567Y01000D01*
X02567Y01100D01*
X02667Y01100D01*
X02667Y00900D01*
X03490Y00900D01*
X03490Y01100D01*
X03590Y01100D01*
X03590Y01000D01*
X04233Y01000D01*
X04233Y01100D01*
X04333Y01100D01*
X04333Y00900D01*
X05157Y00900D01*
X05157Y01100D01*
X05257Y01100D01*
X05257Y01000D01*
X05999Y01000D01*
X06086Y01050D01*
X05674Y01763D01*
X05501Y01663D01*
X05451Y01749D01*
X05538Y01799D01*
X05216Y02357D01*
X05129Y02307D01*
X05079Y02394D01*
X05252Y02493D01*
X04841Y03205D01*
X04668Y03105D01*
X04618Y03192D01*
X04704Y03242D01*
X04383Y03799D01*
X04296Y03749D01*
X04246Y03835D01*
X04419Y03935D01*
X04007Y04649D01*
X03834Y04549D01*
X03784Y04636D01*
```

```
X03871Y04686D01*
X03500Y05329D01*
X03414Y05379D01*
X03003Y04666D01*
X03176Y04566D01*
X03126Y04480D01*
X03039Y04530D01*
X02717Y03972D01*
X02804Y03922D01*
X02754Y03835D01*
X02581Y03935D01*
X02169Y03223D01*
X02342Y03123D01*
X02292Y03036D01*
X02206Y03086D01*
X01884Y02529D01*
X01971Y02479D01*
X01921Y02394D01*
X01748Y02493D01*
X01336Y01780D01*
X01509Y01680D01*
X01459Y01593D01*
X01372Y01643D01*
X01000Y01000D01*
X01367Y01040D02*
X01417Y00990D01*
X01467Y01040D01*
X02200Y01070D02*
X02250Y01020D01*
X02300Y01070D01*
X03033Y01040D02*
X03083Y00990D01*
X03133Y01040D01*
X03867Y01070D02*
X03917Y01020D01*
X03967Y01070D01*
X04700Y01040D02*
X04750Y00990D01*
X04800Y01040D01*
X05533Y01070D02*
X05583Y01020D01*
X05633Y01070D01*
X05781Y01298D02*
X05799Y01366D01*
X05731Y01384D01*
X05339Y02004D02*
X05357Y02072D01*
X05289Y02091D01*
X04948Y02740D02*
X04966Y02808D01*
X04898Y02826D01*
X04505Y03447D02*
X04523Y03515D01*
X04455Y03533D01*
X04115Y04183D02*
X04133Y04252D01*
X04065Y04270D01*
```

```
X03672Y04890D02*
X03690Y04958D01*
X03622Y04976D01*
X03352Y04991D02*
X03284Y04973D01*
X03302Y04905D01*
X02961Y04255D02*
X02893Y04237D01*
X02911Y04168D01*
X02519Y03548D02*
X02451Y03530D01*
X02469Y03462D01*
X02128Y02811D02*
X02060Y02793D01*
X02078Y02725D01*
X01685Y02106D02*
X01617Y02087D01*
X01635Y02019D01*
X01295Y01369D02*
X01227Y01351D01*
X01245Y01283D01*
X01300Y01200D02*
X01347Y01200D01*
X05653Y01200D01*
X03500Y04890D01*
X01381Y01220D01*
X05619Y01220D01*
X03500Y04850D01*
X01416Y01240D01*
X05584Y01240D01*
X03500Y04810D01*
X01451Y01260D01*
X05549Y01260D01*
X03500Y04770D01*
X01485Y01280D01*
X05515Y01280D01*
X03500Y04730D01*
X01520Y01300D01*
X05480Y01300D01*
X03500Y04690D01*
X01555Y01320D01*
X05445Y01320D01*
X03500Y04650D01*
X01589Y01340D01*
X05411Y01340D01*
X03500Y04610D01*
X01624Y01360D01*
X05376Y01360D01*
X03500Y04570D01*
X01658Y01380D01*
X05342Y01380D01*
X03500Y04530D01*
X01693Y01400D01*
X05307Y01400D01*
X03500Y04490D01*
X01728Y01420D01*
X05272Y01420D01*
```

```
X03500Y04450D01*
X01762Y01440D01*
X05238Y01440D01*
X03500Y04410D01*
X01797Y01460D01*
X05203Y01460D01*
X03500Y04370D01*
X01832Y01480D01*
X05168Y01480D01*
X03500Y04330D01*
X01866Y01500D01*
X05134Y01500D01*
X03500Y04290D01*
X01901Y01520D01*
X05099Y01520D01*
X03500Y04250D01*
X01936Y01540D01*
X05064Y01540D01*
X03500Y04210D01*
X01970Y01560D01*
X05030Y01560D01*
X03500Y04170D01*
X02005Y01580D01*
X04995Y01580D01*
X03500Y04130D01*
X02040Y01600D01*
X04960Y01600D01*
X03500Y04090D01*
X02074Y01620D01*
X04926Y01620D01*
X03500Y04050D01*
X02109Y01640D01*
X04891Y01640D01*
X03500Y04010D01*
X02143Y01660D01*
X04857Y01660D01*
X03500Y03970D01*
X02178Y01680D01*
X04822Y01680D01*
X03500Y03930D01*
X02213Y01700D01*
X04787Y01700D01*
X03500Y03890D01*
X02247Y01720D01*
X04753Y01720D01*
X03500Y03850D01*
X02282Y01740D01*
X04718Y01740D01*
X03500Y03810D01*
X02317Y01760D01*
X04683Y01760D01*
X03500Y03770D01*
X02351Y01780D01*
X04649Y01780D01*
X03500Y03730D01*
X02386Y01800D01*
X04614Y01800D01*
```

```
X03500Y03690D01*
X02421Y01820D01*
X04579Y01820D01*
X03500Y03650D01*
X02455Y01840D01*
X04545Y01840D01*
X03500Y03610D01*
X02490Y01860D01*
X04510Y01860D01*
X03500Y03570D01*
X02524Y01880D01*
X04476Y01880D01*
X03500Y03530D01*
X02559Y01900D01*
X04441Y01900D01*
X03500Y03490D01*
X02594Y01920D01*
X04406Y01920D01*
X03500Y03450D01*
X02628Y01940D01*
X04372Y01940D01*
X03500Y03410D01*
X02663Y01960D01*
X04337Y01960D01*
X03500Y03370D01*
X02698Y01980D01*
X04302Y01980D01*
X03500Y03330D01*
X02732Y02000D01*
X04268Y02000D01*
X03500Y03290D01*
X03500Y02443D01*
M2*
```

flattx3b.gbr, solder-side copper artwork RS274X data file:

```
*
%FSAX23Y23*%
%MOIN*%
%AD*%
%ADD10C, 0.060*%
%ADD11C, 0.003*%
%ADD12C, 0.010*%
%ADD13R, 0.150X0.150*%
G54D13*
G1X01425Y03925D03*
X01575Y04075D03*
X06425Y00925D03*
X06575Y01075D03*
G54D11*
G1X01500Y03850D02*
X01650Y03850D01*
X01650Y04000D01*
X01500Y04150D02*
X01350Y04150D01*
X01350Y04000D01*
X06500Y00850D02*
```

```
X06650Y00850D01*
X06650Y01000D01*
X06500Y01150D02*
X06350Y01150D01*
X06350Y01000D01*
G54D10*
G1X03500Y02443D03*
X01250Y01150D03*
X01200Y01100D03*
X01200Y01200D03*
X01300Y01200D03*
X01300Y01100D03*
G54D12*
G1X01000Y01000D02*
X01000Y00900D01*
X01823Y00900D01*
X01823Y01100D01*
X01923Y01100D01*
X01923Y01000D01*
X02567Y01000D01*
X02567Y01100D01*
X02667Y01100D01*
X02667Y00900D01*
X03490Y00900D01*
X03490Y01100D01*
X03590Y01100D01*
X03590Y01000D01*
X04233Y01000D01*
X04233Y01100D01*
X04333Y01100D01*
X04333Y00900D01*
X05157Y00900D01*
X05157Y01100D01*
X05257Y01100D01*
X05257Y01000D01*
X05999Y01000D01*
X06086Y01050D01*
X05674Y01763D01*
X05501Y01663D01*
X05451Y01749D01*
X05538Y01799D01*
X05216Y02357D01*
X05129Y02307D01*
X05079Y02394D01*
X05252Y02493D01*
X04841Y03205D01*
X04668Y03105D01*
X04618Y03192D01*
X04704Y03242D01*
X04383Y03799D01*
X04296Y03749D01*
X04246Y03835D01*
X04419Y03935D01*
X04007Y04649D01*
X03834Y04549D01*
X03784Y04636D01*
X03871Y04686D01*
```

```
X03500Y05329D01*
X03414Y05379D01*
X03003Y04666D01*
X03176Y04566D01*
X03126Y04480D01*
X03039Y04530D01*
X02717Y03972D01*
X02804Y03922D01*
X02754Y03835D01*
X02581Y03935D01*
X02169Y03223D01*
X02342Y03123D01*
X02292Y03036D01*
X02206Y03086D01*
X01884Y02529D01*
X01971Y02479D01*
X01921Y02394D01*
X01748Y02493D01*
X01336Y01780D01*
X01509Y01680D01*
X01459Y01593D01*
X01372Y01643D01*
X01000Y01000D01*
X01367Y01040D02*
X01417Y00990D01*
X01467Y01040D01*
X02200Y01070D02*
X02250Y01020D01*
X02300Y01070D01*
X03033Y01040D02*
X03083Y00990D01*
X03133Y01040D01*
X03867Y01070D02*
X03917Y01020D01*
X03967Y01070D01*
X04700Y01040D02*
X04750Y00990D01*
X04800Y01040D01*
X05533Y01070D02*
X05583Y01020D01*
X05633Y01070D01*
X05781Y01298D02*
X05799Y01366D01*
X05731Y01384D01*
X05339Y02004D02*
X05357Y02072D01*
X05289Y02091D01*
X04948Y02740D02*
X04966Y02808D01*
X04898Y02826D01*
X04505Y03447D02*
X04523Y03515D01*
X04455Y03533D01*
X04115Y04183D02*
X04133Y04252D01*
X04065Y04270D01*
X03672Y04890D02*
```

```
X03690Y04958D01*
X03622Y04976D01*
X03352Y04991D02*
X03284Y04973D01*
X03302Y04905D01*
X02961Y04255D02*
X02893Y04237D01*
X02911Y04168D01*
X02519Y03548D02*
X02451Y03530D01*
X02469Y03462D01*
X02128Y02811D02*
X02060Y02793D01*
X02078Y02725D01*
X01685Y02106D02*
X01617Y02087D01*
X01635Y02019D01*
X01295Y01369D02*
X01227Y01351D01*
X01245Y01283D01*
X03500Y02443D02*
X04233Y02020D01*
X03500Y03330D01*
X02732Y02000D01*
X04268Y02000D01*
X03500Y03370D01*
X02698Y01980D01*
X04302Y01980D01*
X03500Y03410D01*
X02663Y01960D01*
X04337Y01960D01*
X03500Y03450D01*
X02628Y01940D01*
X04372Y01940D01*
X03500Y03490D01*
X02594Y01920D01*
X04406Y01920D01*
X03500Y03530D01*
X02559Y01900D01*
X04441Y01900D01*
X03500Y03570D01*
X02524Y01880D01*
X04476Y01880D01*
X03500Y03610D01*
X02490Y01860D01*
X04510Y01860D01*
X03500Y03650D01*
X02455Y01840D01*
X04545Y01840D01*
X03500Y03690D01*
X02421Y01820D01*
X04579Y01820D01*
X03500Y03730D01*
X02386Y01800D01*
X04614Y01800D01*
X03500Y03770D01*
X02351Y01780D01*
```

```
X04649Y01780D01*
X03500Y03810D01*
X02317Y01760D01*
X04683Y01760D01*
X03500Y03850D01*
X02282Y01740D01*
X04718Y01740D01*
X03500Y03890D01*
X02247Y01720D01*
X04753Y01720D01*
X03500Y03930D01*
X02213Y01700D01*
X04787Y01700D01*
X03500Y03970D01*
X02178Y01680D01*
X04822Y01680D01*
X03500Y04010D01*
X02143Y01660D01*
X04857Y01660D01*
X03500Y04050D01*
X02109Y01640D01*
X04891Y01640D01*
X03500Y04090D01*
X02074Y01620D01*
X04926Y01620D01*
X03500Y04130D01*
X02040Y01600D01*
X04960Y01600D01*
X03500Y04170D01*
X02005Y01580D01*
X04995Y01580D01*
X03500Y04210D01*
X01970Y01560D01*
X05030Y01560D01*
X03500Y04250D01*
X01936Y01540D01*
X05064Y01540D01*
X03500Y04290D01*
X01901Y01520D01*
X05099Y01520D01*
X03500Y04330D01*
X01866Y01500D01*
X05134Y01500D01*
X03500Y04370D01*
X01832Y01480D01*
X05168Y01480D01*
X03500Y04410D01*
X01797Y01460D01*
X05203Y01460D01*
X03500Y04450D01*
X01762Y01440D01*
X05238Y01440D01*
X03500Y04490D01*
X01728Y01420D01*
X05272Y01420D01*
X03500Y04530D01*
X01693Y01400D01*
```

```
X05307Y01400D01*
X03500Y04570D01*
X01658Y01380D01*
X05342Y01380D01*
X03500Y04610D01*
X01624Y01360D01*
X05376Y01360D01*
X03500Y04650D01*
X01589Y01340D01*
X05411Y01340D01*
X03500Y04690D01*
X01555Y01320D01*
X05445Y01320D01*
X03500Y04730D01*
X01520Y01300D01*
X05480Y01300D01*
X03500Y04770D01*
X01485Y01280D01*
X05515Y01280D01*
X03500Y04810D01*
X01451Y01260D01*
X05549Y01260D01*
X03500Y04850D01*
X01416Y01240D01*
X05584Y01240D01*
X03500Y04890D01*
X01381Y01220D01*
X05619Y01220D01*
X03500Y04930D01*
X01347Y01200D01*
X01346Y01200D01*
X01346Y01150D01*
X01250Y01150D01*
M2*
```

Appendix E

PCB coil current-segment model

```
        /* generated by flattx3h.c, rev 1.0 */
#define coil_tx_num 253
/* coax shield, L end of coil */
long coil_tx_points[coil_tx_num][3] = {
1300, 1200, 0,
1347, 1200, 0,
5653, 1200, 0,
3500, 4890, 0,
1381, 1220, 0,
5619, 1220, 0,
3500, 4850, 0,
1416, 1240, 0,
5584, 1240, 0,
3500, 4810, 0,
1451, 1260, 0,
5549, 1260, 0,
3500, 4770, 0,
1485, 1280, 0,
5515, 1280, 0,
3500, 4730, 0,
1520, 1300, 0,
5480, 1300, 0,
3500, 4690, 0,
1555, 1320, 0,
5445, 1320, 0,
3500, 4650, 0,
1589, 1340, 0,
5411, 1340, 0,
3500, 4610, 0,
1624, 1360, 0,
5376, 1360, 0,
3500, 4570, 0,
1658, 1380, 0,
5342, 1380, 0,
3500, 4530, 0,
1693, 1400, 0,
5307, 1400, 0,
3500, 4490, 0,
1728, 1420, 0,
5272, 1420, 0,
3500, 4450, 0,
1762, 1440, 0,
```

```
5238, 1440, 0,
3500, 4410, 0,
1797, 1460, 0,
5203, 1460, 0,
3500, 4370, 0,
1832, 1480, 0,
5168, 1480, 0,
3500, 4330, 0,
1866, 1500, 0,
5134, 1500, 0,
3500, 4290, 0,
1901, 1520, 0,
5099, 1520, 0,
3500, 4250, 0,
1936, 1540, 0,
5064, 1540, 0,
3500, 4210, 0,
1970, 1560, 0,
5030, 1560, 0,
3500, 4170, 0,
2005, 1580, 0,
4995, 1580, 0,
3500, 4130, 0,
2040, 1600, 0,
4960, 1600, 0,
3500, 4090, 0,
2074, 1620, 0,
4926, 1620, 0,
3500, 4050, 0,
2109, 1640, 0,
4891, 1640, 0,
3500, 4010, 0,
2143, 1660, 0,
4857, 1660, 0,
3500, 3970, 0,
2178, 1680, 0,
4822, 1680, 0,
3500, 3930, 0,
2213, 1700, 0,
4787, 1700, 0,
3500, 3890, 0,
2247, 1720, 0,
4753, 1720, 0,
3500, 3850, 0,
2282, 1740, 0,
4718, 1740, 0,
3500, 3810, 0,
2317, 1760, 0,
4683, 1760, 0,
3500, 3770, 0,
2351, 1780, 0,
4649, 1780, 0,
3500, 3730, 0,
2386, 1800, 0,
4614, 1800, 0,
3500, 3690, 0,
2421, 1820, 0,
```

```
4579, 1820, 0,
3500, 3650, 0,
2455, 1840, 0,
4545, 1840, 0,
3500, 3610, 0,
2490, 1860, 0,
4510, 1860, 0,
3500, 3570, 0,
2524, 1880, 0,
4476, 1880, 0,
3500, 3530, 0,
2559, 1900, 0,
4441, 1900, 0,
3500, 3490, 0,
2594, 1920, 0,
4406, 1920, 0,
3500, 3450, 0,
2628, 1940, 0,
4372, 1940, 0,
3500, 3410, 0,
2663, 1960, 0,
4337, 1960, 0,
3500, 3370, 0,
2698, 1980, 0,
4302, 1980, 0,
3500, 3330, 0,
2732, 2000, 0,
4268, 2000, 0,
3500, 3290, 0,
3500, 2443, 0,
3500, 2443, -63,
4233, 2020, -63,
3500, 3330, -63,
2732, 2000, -63,
4268, 2000, -63,
3500, 3370, -63,
2698, 1980, -63,
4302, 1980, -63,
3500, 3410, -63,
2663, 1960, -63,
4337, 1960, -63,
3500, 3450, -63,
2628, 1940, -63,
4372, 1940, -63,
3500, 3490, -63,
2594, 1920, -63,
4406, 1920, -63,
3500, 3530, -63,
2559, 1900, -63,
4441, 1900, -63,
3500, 3570, -63,
2524, 1880, -63,
4476, 1880, -63,
3500, 3610, -63,
2490, 1860, -63,
4510, 1860, -63,
3500, 3650, -63,
```

```
2455, 1840, -63,
4545, 1840, -63,
3500, 3690, -63,
2421, 1820, -63,
4579, 1820, -63,
3500, 3730, -63,
2386, 1800, -63,
4614, 1800, -63,
3500, 3770, -63,
2351, 1780, -63,
4649, 1780, -63,
3500, 3810, -63,
2317, 1760, -63,
4683, 1760, -63,
3500, 3850, -63,
2282, 1740, -63,
4718, 1740, -63,
3500, 3890, -63,
2247, 1720, -63,
4753, 1720, -63,
3500, 3930, -63,
2213, 1700, -63,
4787, 1700, -63,
3500, 3970, -63,
2178, 1680, -63,
4822, 1680, -63,
3500, 4010, -63,
2143, 1660, -63,
4857, 1660, -63,
3500, 4050, -63,
2109, 1640, -63,
4891, 1640, -63,
3500, 4090, -63,
2074, 1620, -63,
4926, 1620, -63,
3500, 4130, -63,
2040, 1600, -63,
4960, 1600, -63,
3500, 4170, -63,
2005, 1580, -63,
4995, 1580, -63,
3500, 4210, -63,
1970, 1560, -63,
5030, 1560, -63,
3500, 4250, -63,
1936, 1540, -63,
5064, 1540, -63,
3500, 4290, -63,
1901, 1520, -63,
5099, 1520, -63,
3500, 4330, -63,
1866, 1500, -63,
5134, 1500, -63,
3500, 4370, -63,
1832, 1480, -63,
5168, 1480, -63,
3500, 4410, -63,
```

```
1797, 1460, -63,
5203, 1460, -63,
3500, 4450, -63,
1762, 1440, -63,
5238, 1440, -63,
3500, 4490, -63,
1728, 1420, -63,
5272, 1420, -63,
3500, 4530, -63,
1693, 1400, -63,
5307, 1400, -63,
3500, 4570, -63,
1658, 1380, -63,
5342, 1380, -63,
3500, 4610, -63,
1624, 1360, -63,
5376, 1360, -63,
3500, 4650, -63,
1589, 1340, -63,
5411, 1340, -63,
3500, 4690, -63,
1555, 1320, -63,
5445, 1320, -63,
3500, 4730, -63,
1520, 1300, -63,
5480, 1300, -63,
3500, 4770, -63,
1485, 1280, -63,
5515, 1280, -63,
3500, 4810, -63,
1451, 1260, -63,
5549, 1260, -63,
3500, 4850, -63,
1416, 1240, -63,
5584, 1240, -63,
3500, 4890, -63,
1381, 1220, -63,
5619, 1220, -63,
3500, 4930, -63,
1347, 1200, -63,
1346, 1200, -63,
1346, 1150, -63,
1250, 1150, -63,
1300, 1200, 0
};
/* coax center conductor, H end of coil */
```

Appendix F

C code to test current-segment header file against RS274X files

This program, flattx3g.c, generates a RS274X-format file from the current-segment file flattx3.h.

The current segments exist in three-dimensional (3D) space, XYZ, while the RS274X files work in two-dimensional (2D) space, XY. The current segments are almost all in one of two planes parallel to the XY plane. A few segments are in another plane, to model the current path through the coax connector mounted on the board.

An input parameter is used to select either the plane of the component side of the PCB, or the plane of the solder side of the PCB.

Thus, the output of flattx3g.c contains all the segments in one plane. This output is set to a narrower linewidth than used in generating the artwork files.

When we load the flattx3g.c output into a RS274X-file viewer layer on top of the layer containing one of the artwork files, wisual inspection quickly confirms that the two files agree the way they are supposed to.

```
/* flattx3g.c  This program converts coils in header file
flattx3.h to RS274X-format files
for comparison to board artwork with gcprevue

rev0, 15sep1999pta: based on flattx2g.c, rev0

Integer input on command line:
    Z_level -63, 0, 1:
-63 outputs solder-side points.
0 outputs component-side points.
1 (or any value >0) outputs all points for both -63 and 0.
Negative != -63 outputs no points.

stdout receives RS274X gerber data
for coil_number points at Z_level
```

```
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <flattx3.h> /* header file for ANT-003 transmitter */

main(int argc, char *argv[])
{
  /* origin of model XY coordinates, in RS274X coordinates */
  long x_offset = 0;
  long y_offset = 0;

  /*   */

  long ger_z_level; /* current Z level in RS274X output */
  long prev_z_level; /* Z level in previous point */
  long xf,yf,zf; /* coord of current point in header file */
  long point_no; /* current point */
  long point_no_max; /* maximum value of point_number */
  int RS274X_D; /* RS274X draw light-on/light-off flag */

  /* print preamble to output file */
  printf("* \n");
  printf("%%FSLAX23Y23*%%\n");
  printf("%%MOIN*%%\n");
  printf("%%AD*%%\n");
  printf("%%ADD22C, 0.002*%%\n");  /* define aperture */
  printf("G54D22*\nG1");          /* start plotting */

  /* initialize variables */
  point_no = 0;
  point_no_max = 0;
  ger_z_level = (long)atoi(argv[1]);
  prev_z_level = -10000;   /* invalid value,
                   since first point has no predecessor*/

  point_no_max = coil_tx_num;

  /* handle the points in the coil */

  while (point_no < point_no_max) {

    /* get current point */
  xf = coil_tx_points[point_no][0];
  yf = coil_tx_points[point_no][1];
  zf = coil_tx_points[point_no][2];

    /* if zf matches ger_z_level, then plot the point */
    if ((zf == ger_z_level)||(0 < ger_z_level))
      {
      printf("X%5.5dY%5.5dD0",xf+x_offset,yf+y_offset);
      RS274X_D = 2; /* indicates previous point wasn't plotted */
      if ((prev_z_level == ger_z_level)
                ||(point_no != 0)&&(0 < ger_z_level))
{
```

```
        RS274X_D = 1; /* indicates previous point was plotted */
        }
      printf("%1.1d*\n",RS274X_D);
      }
    prev_z_level = zf; /*previous-point Z level for next point*/
    point_no++;
  }
  /* print postamble to output file */
  printf("M2*\n");  /* end-of-file code */
}
```

Appendix G

Mutual inductance of circuits of straight-line segments

The mutual inductance, L_m, between two closed circuits is given by Neuman's formula [3] [4] [2]:

$$L_m = \frac{\mu_o}{4\pi} \oint_{circuitA} \oint_{circuitB} \frac{d\overline{s_A} \cdot d\overline{s_B}}{r_{AB}} \tag{G.1}$$

where

$r_{AB} = \mid \overline{s_B} - \overline{s_A} \mid$

and

$\mu_o = 4\pi 10^{-7} henries/meter$

Here, we consider the case where both circuits are composed of straight-line segments.

Circuit A is composed of segments $A_1 \cdots A_n$.

Circuit B is composed of segments $B_1 \cdots B_m$.

Then G.1 becomes:

$$L_m = \frac{\mu_o}{4\pi} \sum_{k=1}^{n} \sum_{l=1}^{m} \int_{B_l^{start}}^{B_l^{finish}} \int_{A_k^{start}}^{A_k^{finish}} \frac{d\overline{s_A} \cdot d\overline{s_B}}{r_{AB}} \tag{G.2}$$

The integral over A_k can be evaluated analytically:

$$L_m = \frac{\mu_o}{4\pi} \sum_{k=1}^{n} \sum_{l=1}^{m} \frac{\overline{s_l} \cdot \overline{s_k}}{\sqrt{s_k^2}} \int_{b=0}^{b=1} \log\left(\frac{\sqrt{\frac{(\overline{s_k}+s_g\overline{s_b})\cdot(\overline{s_k}+s_g\overline{s_b})}{s_b^2}} + \sqrt{\frac{s_k^2}{s_b^2}} + s_g c_t}{1 + s_g c_t}\right) db \tag{G.3}$$

where

$\overline{s_k^{start}}$ = vector from origin to start of segment A_k

$\overline{s_k^{finish}}$ = vector from origin to finish of segment A_k

$\overline{s_l^{start}}$ = vector from origin to start of segment B_l

$\overline{s_l^{finish}}$ = vector from origin to finish of segment B_l

$\overline{s_k} = \overline{s_k^{finish}} - \overline{s_k^{start}}$

$\overline{s_l} = \overline{s_l^{finish}} - \overline{s_l^{start}}$

$s_k^2 = \overline{s_k} \cdot \overline{s_k}$

$s_l^2 = \overline{s_l} \cdot \overline{s_l}$

$\overline{s_s} = (\overline{s_l^{start}} - \overline{s_k^{start}})$

$s_g = +1$ if $(\overline{s_s} \cdot \overline{s_k} \geq 0)$, -1 if $(\overline{s_s} \cdot \overline{s_k} < 0)$

$\overline{s_e} = \overline{s_k^{finish}}$ if $(s_g = +1)$, $\overline{s_k^{start}}$ if $(s_g = -1)$

$\overline{s_b} = \overline{s_l^{start}} - \overline{s_e} + \overline{s_l} b$

$s_b^2 = \overline{s_b} \cdot \overline{s_b}$

$c_t = \frac{\overline{s_b} \cdot \overline{s_k}}{\sqrt{s_b^2 s_k^2}}$ = cosine of angle between $\overline{s_b}$ and $\overline{s_k}$

Equation G.3 is easily evaluated numerically using the seven-point rule in equation H.1, or by other numerical integration methods. The logarithm in G.3 makes the integrand vary slowly and smoothly, so the integral is accurately calculated by applying H.1 once over the whole interval from $b = 0$ to $b = 1$.

Equation G.3 combines two forms, which integrate in opposite directions along segment A_k. The two forms are combined because each form has a region of potential numerical instability due to a zero denominator.

The first region of numerical instability occurs when $\overline{s_s}$ and $\overline{s_b}$ approach being parallel. In this region, $\overline{s_s} \cdot \overline{s_b} > 0$, and $1 - c_t$ approaches zero. When $\overline{s_s}$ and $\overline{s_b}$ become exactly parallel, $1 - c_t = 0$ exactly. In this region, we choose $s_g = +1$ to avoid a zero denominator in the integral in equation G.3.

The second region of numerical instability occurs when $\overline{s_s}$ and $\overline{s_b}$ approach being antiparallel. In this region, $\overline{s_s} \cdot \overline{s_b} < 0$, and $1 + c_t$ approaches zero. When $\overline{s_s}$ and $\overline{s_b}$ become exactly parallel, $1 + c_t = 0$ exactly. In this region, we choose $s_g = -1$ to avoid a zero denominator in the integral in equation G.3.

To derive equation G.3, start with the term in equation G.1 for one pair of current segments:

$$I_{kl} = \int_{B_l^{start}}^{B_l^{finish}} \int_{A_k^{start}}^{A_k^{finish}} \frac{d\overline{s_A} \cdot d\overline{s_B}}{r_{AB}} \tag{G.4}$$

Current segments A_k and B_l are both straight lines, so it is natural to describe them in

linear terms of dimensionless variables a and b respectively, where

$0 \le a \le 1$

$0 \le b \le 1$

We integrate over a symbolically first, so we will keep dependencies on a explicit. Dependencies on b are absorbed into quantities independent of a while we integrate over a.

A point on segment A_k is one of two linear functions of a, depending on s_g:

When $s_g = +1$, then $\overline{s_A(a)} = \overline{s_k^{finish}} - \overline{s_k}a$

When $s_g = -1$, then $\overline{s_A(a)} = \overline{s_k^{start}} + \overline{s_k}a$

A point on segment B_l is a linear function of b:

$\overline{s_B(b)} = \overline{s_l^{start}} + \overline{s_l}b$

Then:

$d\overline{s_A} = -s_g\overline{s_k}\,da$

$d\overline{s_B} = \overline{s_l}\,db$

$d\overline{s_A} \cdot d\overline{s_B} = -s_g\overline{s_k} \cdot \overline{s_l}\,da\,db$

$r_{AB} = \sqrt{(\overline{s_b} + s_g\overline{s_k}a) \cdot (\overline{s_b} + s_g\overline{s_k}a)}$

Now equation G.4 becomes:

$$I_{kl} = -s_g\overline{s_k} \cdot \overline{s_l} \int_0^1 \int_{(1+s_g)/2}^{(1-s_g)/2} \frac{da}{\sqrt{(\overline{s_b} + s_g\overline{s_k}a) \cdot (\overline{s_b} + s_g\overline{s_k}a)}}\,db \tag{G.5}$$

Expand the denominator in equation G.5, separate out the coefficient of a^2, and remove the s_g dependence from the limits of integration:

$$I_{kl} = \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \int_0^1 \frac{da}{\sqrt{\frac{s_b^2}{s_k^2} + \frac{2s_g\overline{s_b} \cdot \overline{s_k}}{s_k^2}a + a^2}}\,db \tag{G.6}$$

Define two quantities which do not depend on a:

$\alpha = \frac{s_g\overline{s_b} \cdot \overline{s_k}}{s_k^2} = \sqrt{\frac{s_b^2}{s_k^2}}s_g c_t$

$\beta = \frac{s_b^2}{s_k^2} - \alpha^2 = \frac{s_b^2}{s_k^2}(1 - c_t^2)$

Note that $0 \le \beta \le 2$.

Define a new variable x:

$x = a + \alpha$

Now equation G.6 becomes:

$$I_{kl} = \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \int_\alpha^{\alpha+1} \frac{dx}{\sqrt{x^2 + \beta}}\,db \tag{G.7}$$

From Weast [26], we have for $\beta \geq 0$:

$$\int \frac{dx}{\sqrt{x^2+\beta}} = \log(x + \sqrt{x^2+\beta}) \tag{G.8}$$

Substitute equation G.8 into equation G.7:

$$I_{kl} = \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \log(\frac{\alpha + 1 + \sqrt{(\alpha+1)^2+\beta}}{\alpha + \sqrt{\alpha^2+\beta}})\, db \tag{G.9}$$

Substitute the definitions of α and β into equation G.9:

$$I_{kl} = \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \log(\frac{\sqrt{\frac{s_b^2}{s_k^2}} s_g c_t + 1 + \sqrt{(\sqrt{\frac{s_b^2}{s_k^2}} s_g c_t + 1)^2 + \frac{s_b^2}{s_k^2}(1-c_t^2)}}{\sqrt{\frac{s_b^2}{s_k^2}} s_g c_t + \sqrt{(\sqrt{\frac{s_b^2}{s_k^2}} s_g c_t)^2 + \frac{s_b^2}{s_k^2}(1-c_t^2)}})\, db \tag{G.10}$$

Remove a common factor of $\sqrt{\frac{s_b^2}{s_k^2}}$ from numerator and denominator:

$$I_{kl} = \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \log(\frac{s_g c_t + \sqrt{\frac{s_k^2}{s_b^2}} + \sqrt{(s_g c_t + \sqrt{\frac{s_k^2}{s_b^2}})^2 + (1-c_t^2)}}{s_g c_t + \sqrt{(s_g c_t)^2 + (1-c_t^2)}})\, db \tag{G.11}$$

Simplify:

$$I_{kl} = \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \log(\frac{s_g c_t + \sqrt{\frac{s_k^2}{s_b^2}} + \sqrt{\frac{2 s_g c_t \sqrt{s_k^2 s_b^2} + s_k^2 + s_b^2}{s_b^2}}}{s_g c_t + 1})\, db \tag{G.12}$$

Substitute for c_t in the last radical in the numerator:

$$I_{kl} = \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \log(\frac{s_g c_t + \sqrt{\frac{s_k^2}{s_b^2}} + \sqrt{\frac{2 s_g \overline{s_b} \cdot \overline{s_k} + s_k^2 + s_b^2}{s_b^2}}}{s_g c_t + 1})\, db \tag{G.13}$$

Factor the numerator inside the last radical in the numerator:

$$I_{kl} = \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \log(\frac{s_g c_t + \sqrt{\frac{s_k^2}{s_b^2}} + \sqrt{\frac{(\overline{s_k} + s_g \overline{s_b}) \cdot (\overline{s_k} + s_g \overline{s_b})}{s_b^2}}}{s_g c_t + 1})\, db \tag{G.14}$$

Substitute equation G.14 into equation G.4, and that into equation G.2:

$$L_m = \frac{\mu_o}{4\pi} \sum_{k=1}^{n} \sum_{l=1}^{m} \frac{\overline{s_k} \cdot \overline{s_l}}{\sqrt{s_k^2}} \int_0^1 \log(\frac{s_g c_t + \sqrt{\frac{s_k^2}{s_b^2}} + \sqrt{\frac{(\overline{s_k} + s_g \overline{s_b}) \cdot (\overline{s_k} + s_g \overline{s_b})}{s_b^2}}}{s_g c_t + 1})\, db \tag{G.15}$$

This is our result, equation G.3.

Appendix H

Seven-point integration rule

Closed Newton-Cotes formulas, such as the seven-point integration rule described here, are inefficient or inaccurate for many integrations, and other methods, such as the Runge-Kutta method, are usually used [27].

The integrand in equation G.3 is a smoothly-varying and slowly-varying function of b, so the seven-point rule works well for this particular integral.

The seven-point rule is an extension of lower-order formulas given in [27]:

$$\int_0^8 f(x)\,dx = m\,f(0) + n\,f(1) + p\,f(3) + q\,f(4) + p\,f(5) + n\,f(7) + m\,f(8) \qquad \text{(H.1)}$$

where

$m = \frac{\frac{1867776}{7} - \frac{797568}{3}}{3360}$ = approximately 0.2884354

$n = \frac{-240m + \frac{2752}{15}}{72}$ = approximately 1.586697

$p = -16m - 9n + \frac{64}{3}$ = approximately 2.438095

$q = -2m - 2n - 2p + 8$ = approximately -0.6264550

Note that the seven-point rule is based on nine equally-spaced points, with zero coefficients for two of the nine points.

The seven-point rule is exact when $f(x)$ is a polynomial in x containing no powers of x higher than x^6. This can be verified by testing the rule using functions $f(x) = 1$, $f(x) = x$, $f(x) = x^2$, $f(x) = x^3$, $f(x) = x^4$, $f(x) = x^5$, and $f(x) = x^6$.

Note that the coefficients are all between 0.25 and 2.5, not too far from unity. This means that uncancelled higher powers of x in $f(x)$ will not be grossly amplified when the seven-point rule is applied.

The integral in equation G.3 is evaluated accurately using 7 points (b = 0, 1/8, 3/8, 1/2, 5/8, 7/8, 1) per segment when using the seven-point rule.

We apply the seven-point rule once over the whole interval $0 \leq b \leq 1$. Let E_r be the error due to using the rule:

$$\int_0^1 I_g \, db = \frac{1}{8}\left(mI_g(0) + nI_g(\frac{1}{8}) + pI_g(\frac{3}{8}) + qI_g(\frac{1}{2}) + pI_g(\frac{5}{8}) + nI_g(\frac{7}{8}) + mI_g(1)\right) + E_r \tag{H.2}$$

Perform a Taylor-series expansion around the center of the interval, $b = 1/2$. Thus, I_g and its derivatives are evaluated at $b = 1/2$:

$$\int_0^1 I_g \, db = \int_0^1 \left(I_g + \sum_{n=1}^{\infty} \frac{1}{n!} \frac{d^n I_g}{db^n} (b - \frac{1}{2})^n\right) db \tag{H.3}$$

Move the integral inside the sum and integrate, remembering that I_g and its derivatives are evaluated at $b = 1/2$ before integrating:

$$\int_0^1 I_g \, db = I_g + \sum_{n=1}^{\infty} \frac{1}{(n-1)!} \frac{d^n I_g}{db^n} \left((\frac{1}{2})^{n-1} - (-\frac{1}{2})^{n-1}\right) \tag{H.4}$$

Note that equation H.4 is exact if we keep all the terms in the infinite summation.

The terms in equation H.4 for $n = 1$ through $n = 6$ are included exactly in the seven-point rule in equation H.2.

By substituting $I_g = (b - 1/2)^7$ into equation H.2, we see that, by symmetry, the $n = 7$ term of the integrand does not contribute to the integral. The $n = 7$ term in H.4 is exactly zero by symmetry. Thus, the error E_r contains no terms with $n < 8$.

Approximate E_r by the $n = 8$ term in H.4, remembering that the derivative is evaluated at $b = 1/2$:

$$E_r \approx T_8 = \frac{1}{7!} \frac{d^8 I_g}{db^8} \left((\frac{1}{2})^7 - (-\frac{1}{2})^7\right) = \frac{1}{322560} \frac{d^8 I_g}{db^8} \tag{H.5}$$

Appendix I

C code for mutual-inductance model of two PCB coils

```
/* currel93.c  This program contains functions Lm and main,
and uses flattx3.h to calculate the mutual inductance of
two ANT-003 boards, one as transmitter and the other as receiver.

rev0, 28sep2000pta: based on currel63.c rev1
rev1, 27oct2000pta: fixed error in comment

mutual-inductance integral from trakr380.txt using seven-point
rule from trakr408.txt .
=====
main() is a simple C program to demonstrate currelm by printing
out the mutual inductance of two PCBs at the
position and orientation passed on the command line.

main() function:
Input on command line:
    Step, XYZ position in meters, ijk orientation.

Step is a non-negative integer.  Step is normally zero for speed.
If integral accuracy is suspect, increase the value of step.

Output is mutual-inductance in henries = webers/ampere

*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <flattx3.h> /* header file for ANT-003 transmitter */

/* multiply one quaternion by another quaternion */
quatmul(double leftquat[], double rightquat[], double prodquat[])
{
  prodquat[0] =leftquat[0]*rightquat[0]-leftquat[1]*rightquat[1]
              -leftquat[2]*rightquat[2]-leftquat[3]*rightquat[3];

  prodquat[1] =leftquat[0]*rightquat[1]+leftquat[1]*rightquat[0]
              +leftquat[2]*rightquat[3]-leftquat[3]*rightquat[2];

  prodquat[2] =leftquat[0]*rightquat[2]-leftquat[1]*rightquat[3]
```

```
              +leftquat[2]*rightquat[0]+leftquat[3]*rightquat[1];

  prodquat[3] =leftquat[0]*rightquat[3]+leftquat[1]*rightquat[2]
              -leftquat[2]*rightquat[1]+leftquat[3]*rightquat[0];
}

/* complement a quaternion */
/* this gives the inverse of a unit-norm quaternion */
quatcomp(double qoriginal[], double qcomplemented[])
{
  qcomplemented[0] = qoriginal[0];
  qcomplemented[1] = -qoriginal[1];
  qcomplemented[2] = -qoriginal[2];
  qcomplemented[3] = -qoriginal[3];
}

/* rotate a position quaternion by a rotation quaternion */
quatrot(double qrotation[], double qto_be_rotated[],
 double qrotated[]){
  double qrotation_inv[4];
  double qtemporary[4];
  quatcomp(qrotation,qrotation_inv);
  quatmul(qto_be_rotated, qrotation_inv, qtemporary);
  quatmul(qrotation, qtemporary, qrotated);
}

/* get quaternion to rotate board 0 into the desired board */
board_tet_qrot(int board_no, double qrotation[]){

  double qface[4][4];

  /* quaternion to rotate board 0 into board0 */
  qface[0][0] = 1.0;
  qface[0][1] = 0.0;
  qface[0][2] = 0.0;
  qface[0][3] = 0.0;

  /* quaternion to rotate board 1 into board 0 */
  qface[1][0] =  0.5;
  qface[1][1] =  sqrt(2.0)/2.0;
  qface[1][2] = -sqrt(6.0)/6.0;
  qface[1][3] = -sqrt(3.0)/6.0;

  /* quaternion to rotate board 2 into board 0 */
  qface[2][0] =  0.0;
  qface[2][1] =  sqrt(2.0)/2.0;
  qface[2][2] = +sqrt(6.0)/6.0;
  qface[2][3] = -sqrt(3.0)/3.0;

  /* quaternion to rotate board 3 into board 0 */
  qface[3][0] =  0.5;
  qface[3][1] =  0.0;
  qface[3][2] = -sqrt(6.0)/3.0;
  qface[3][3] =  sqrt(3.0)/6.0;

  /* quaternion to rotate board 0 into the desired board */
  qrotation[0] =  qface[board_no][0];
```

```
  qrotation[1] = -qface[board_no][1];
  qrotation[2] = -qface[board_no][2];
  qrotation[3] = -qface[board_no][3];
}

get_tx_seg_end(int board_no,long point_no,
double point_position[3]){

  double meters_per_mil = 2.54e-5;
  double xf,yf,zf;
  double qpoint_in_board[4],qpoint_in_tetrahedron[4];
  double qrotation[4];
  double Mpmil = 1;

  Mpmil = 2.54e-5; /* meters per mil */

  /* model the bottom board */

  /* get point in board 0 in flattx3.h coordinates */
  qpoint_in_board[0] = 0.0;
  qpoint_in_board[1] = coil_tx_points[point_no][0];
  qpoint_in_board[2] = coil_tx_points[point_no][1];
  qpoint_in_board[3] = coil_tx_points[point_no][2];

  /*
  center of bottom board comp side in tetrahedron coord
     = (0,0,-1020.62) mils
  center of bottom board comp side in flattx3.h coord
     = (3500,2443,0) mils
  center of tetrahedron in tetrahedron coord
     = (0,0,0) mils
  center of tetrahedron in flattx3.h coord
     = (3500,2443,1020.62) mils
  */

  /* translate to tetrahedron coordinates */
  qpoint_in_board[1] = qpoint_in_board[1] - 3500.0;
  qpoint_in_board[2] = qpoint_in_board[2] - 2443.0;
  qpoint_in_board[3] = qpoint_in_board[3] - 1020.62;

  /* convert from mils to meters */
  qpoint_in_board[1] = qpoint_in_board[1] * Mpmil;
  qpoint_in_board[2] = qpoint_in_board[2] * Mpmil;
  qpoint_in_board[3] = qpoint_in_board[3] * Mpmil;

  /* get quaternion to rotate about the center of the tetrahedron
  to rotate board 0 into the desired board */

  board_tet_qrot(board_no, qrotation);

  /* rotate the point's position */
  quatrot(qrotation,qpoint_in_board,qpoint_in_tetrahedron);

  /* convert point-position quaternion to vector */
  point_position[0] = qpoint_in_tetrahedron[1];
  point_position[1] = qpoint_in_tetrahedron[2];
  point_position[2] = qpoint_in_tetrahedron[3];
```

```
}

get_rx_seg_end(int rx_coil_no,long rx_seg_num,double qpos[4],
double qori[4],double rx_seg_end[3]){
  double model_seg_end[3];
  double qmodel_seg_end[4];
  double qrx_seg_end[4];

  /* get segment endpoint for untranslated, unrotated receiver */
  get_tx_seg_end(rx_coil_no, rx_seg_num, model_seg_end);

  qmodel_seg_end[0] = 0;
  qmodel_seg_end[1] = model_seg_end[0];
  qmodel_seg_end[2] = model_seg_end[1];
  qmodel_seg_end[3] = model_seg_end[2];

  /* rotate the receiver */
  quatrot(qori, qmodel_seg_end, qrx_seg_end);

  /* translate the receiver */
  rx_seg_end[0] = qrx_seg_end[1] + qpos[1];
  rx_seg_end[1] = qrx_seg_end[2] + qpos[2];
  rx_seg_end[2] = qrx_seg_end[3] + qpos[3];
}

double Lm(int Step,int tx_coil_no,int rx_coil_no,double qpos[4],
double qori[4]){

  /* mutual-inductance integral from trakr380.txt */
  /* seven-point rule from trakr408.txt */

  long tx_seg_num_max = coil_tx_num;
  long rx_seg_num_max = coil_tx_num;
  long step_in_b_max_increment = 8; /* for seven-point rule */
  long step_in_b_max = 8;
  double Weight = 1; /* rule step weight */
  double step_in_b_max_inv;
  long step_in_b;
  double b;
  long tx_seg_num;
  long rx_seg_num;
  double sum,II,b_integral;
  double Uo_over_4pi = 1e-7;
  double seg_end[3];
  double s1start[3]; /* s1 = transmitter */
  double s2start[3]; /* s2 = receiver */
  double s1end[3];
  double s2end[3];
  double s1len[3];
  double s2len[3];
  double s1se[3];
  double sbmb[3];
  double sb[3];
  double s1len2,s2len2,sb2; /* squares of s1len, s2len, sb */
  double sbDOTs1len;
  double s1lenDOTs2len;
  double ssb2;
```

```
double sgnarg,signe,sign_cos_theta;
double logargnum,logargden;
double b_integrand;
int skip_flag = 0;

/* define the coefficients for seven-point formula */

double spfn,spfp,spfq;
double spfm = ( 1867776.0/7.0 - 797568.0/3.0 )/3360.0;
spfn = (-240.0*spfm +2752.0/15.0)/72.0;
spfp = -16.0*spfm -9.0*spfn +64.0/3.0;
spfq = -2.0*spfm -2.0*spfn -2.0*spfp +8.0;

/* set integration steps: */

if (Step<0){
  Step = 0;
}
step_in_b_max = step_in_b_max_increment * (Step + 1);

step_in_b_max_inv = 1.0 / (double)step_in_b_max;
sum=0;
get_tx_seg_end(tx_coil_no,0,s1end);
for(tx_seg_num=1;tx_seg_num<tx_seg_num_max;tx_seg_num++){
  s1start[0] = s1end[0];
  s1start[1] = s1end[1];
  s1start[2] = s1end[2];
  get_tx_seg_end(tx_coil_no,tx_seg_num,s1end);
  get_rx_seg_end(rx_coil_no,0,qpos,qori,s2end);
  for(rx_seg_num=1;rx_seg_num<rx_seg_num_max;rx_seg_num++){
    s2start[0] = s2end[0];
    s2start[1] = s2end[1];
    s2start[2] = s2end[2];
    get_rx_seg_end(rx_coil_no,rx_seg_num,qpos,qori,s2end);
    s1len[0] = s1end[0] - s1start[0];
    s1len[1] = s1end[1] - s1start[1];
    s1len[2] = s1end[2] - s1start[2];
    s2len[0] = s2end[0] - s2start[0];
    s2len[1] = s2end[1] - s2start[1];
    s2len[2] = s2end[2] - s2start[2];
    s1lenDOTs2len = s1len[0]*s2len[0]+s1len[1]*s2len[1]
        +s1len[2]*s2len[2];
    if(fabs(s1lenDOTs2len)>1e-200){
      s1len2 = s1len[0]*s1len[0]+s1len[1]*s1len[1]
        +s1len[2]*s1len[2];
      s2len2 = s2len[0]*s2len[0]+s2len[1]*s2len[1]
        +s2len[2]*s2len[2];
      sgnarg = (s2start[0]-s1start[0])*s1len[0]
       +(s2start[1]-s1start[1])*s1len[1]
       +(s2start[2]-s1start[2])*s1len[2];
      signe = 1.0;
      s1se[0] = s1end[0];
      s1se[1] = s1end[1];
      s1se[2] = s1end[2];
      if(sgnarg<0.0){/*sgnarg*/
        signe = -1.0;
        s1se[0] = s1start[0];
```

```
        s1se[1] = s1start[1];
        s1se[2] = s1start[2];
}
      sbmb[0] = s2start[0] - s1se[0];
      sbmb[1] = s2start[1] - s1se[1];
      sbmb[2] = s2start[2] - s1se[2];
      b_integral = 0;
      for(step_in_b=0;step_in_b<step_in_b_max+1;step_in_b++){
        switch (step_in_b&7) {
          case 0:
            Weight = 2.0*spfm;
            if(step_in_b==0){
              Weight = spfm;
            }
            if(step_in_b==step_in_b_max){
              Weight = spfm;
            }
            skip_flag = 0;
            break;
          case 1:
            Weight = spfn;
            skip_flag = 0;
            break;
          case 2:
            Weight = 0.0;
            skip_flag = 1;
            break;
          case 3:
            Weight = spfp;
            skip_flag = 0;
            break;
          case 4:
            Weight = spfq;
            skip_flag = 0;
            break;
          case 5:
            Weight = spfp;
            skip_flag = 0;
            break;
          case 6:
            Weight = 0.0;
            skip_flag = 1;
            break;
          case 7:
            Weight = spfn;
            skip_flag = 0;
            break;
          default:
            Weight = 0.0;
            skip_flag = 1;
            break;
        }
        if(skip_flag == 0){
          b = (double)step_in_b * step_in_b_max_inv;
          sb[0] = sbmb[0] + s2len[0] * b;
          sb[1] = sbmb[1] + s2len[1] * b;
          sb[2] = sbmb[2] + s2len[2] * b;
```

```
            sb2 = sb[0]*sb[0]+sb[1]*sb[1]+sb[2]*sb[2];
            sbDOTs1len = sb[0]*s1len[0]+sb[1]*s1len[1]
             +sb[2]*s1len[2];
            sign_cos_theta = signe*sbDOTs1len/sqrt(sb2*s1len2);
            ssb2 = (s1len[0]+signe*sb[0])*(s1len[0]+signe*sb[0])
                 + (s1len[1]+signe*sb[1])*(s1len[1]+signe*sb[1])
                 + (s1len[2]+signe*sb[2])*(s1len[2]+signe*sb[2]);
            logargnum = sqrt(ssb2/sb2)+sqrt(s1len2/sb2)
                 +sign_cos_theta;
            logargden = 1.0 + sign_cos_theta;
            b_integrand = Weight * log(logargnum/logargden);
            b_integral = b_integral + b_integrand;
  }
        }
        II = b_integral*step_in_b_max_inv
                * s1lenDOTs2len / sqrt(s1len2);
        sum = sum + II;
      }
    }
  }
  return Uo_over_4pi * sum;
}

main(int argc, char *argv[])
{
  double B[3];
  double Lmq[3];
  double X,Y,Z;
  double qpos[4];
  double qori[4];
  int Step;
  printf("currel93.c rev 1, Lm from trakr380.txt,
inputs Step X Y Z i j k\n");
  printf("transmitter is ANT-003 board, receiver is
ANT-003 board\n");
  printf("Each board is treated as board 0 in trakr376
numbering\n");
  Step = atoi(argv[1]);
  X = atof(argv[2]);
  Y = atof(argv[3]);
  Z = atof(argv[4]);
  qpos[0] = 0;
  qpos[1] = X;
  qpos[2] = Y;
  qpos[3] = Z;
  qori[1] = atof(argv[5]);
  qori[2] = atof(argv[6]);
  qori[3] = atof(argv[7]);
  qori[0] = sqrt(1.0-qori[1]*qori[1]-qori[2]
    *qori[2]-qori[3]*qori[3]);
  printf("   position %15e %15e %15e\n",qpos[1],qpos[2],qpos[3]);
  printf("orientation %15e %15e %15e %15e\n",qori[0],qori[1],
    qori[2],qori[3]);
  printf("Lm =%15e  henries\n",Lm(Step,0,0,qpos,qori));
}
```

Appendix J

C code for mutual-inductance model of two tetrahedra

```
/* currel63.c  This program contains functions Lm and main,
and uses flattx3.h to model two tetrahedra of four ANT-003
boards, one as transmitter and the other as receiver.

rev0, 14july2000pta: based on currel62.c rev1 and on
currel54.c rev0
rev1, 30july2000pta: errors fixed in comments

=====
main() is a simple C program to demonstrate currelm by printing
out the sixteen mutual inductances at the position and
orientation passed on the command line.

main() function:
Input on command line:
    Step, XYZ position in meters, ijk orientation.

Step is a non-negative integer.  Step is normally zero for speed.
If integral accuracy is suspect, increase the value of step.

Output is 4x4 transmitter-receiver mutual-inductance matrix

   Lm[tx_coil][rx_coil]  in henries = webers/ampere

where coils are numbered as in trakr376.txt .

*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <flattx3.h> /* header file for ANT-003 transmitter */

/* multiply one quaternion by another quaternion */
quatmul(double leftquat[], double rightquat[], double prodquat[])
{
  prodquat[0] = leftquat[0]*rightquat[0]-leftquat[1]*rightquat[1]
              - leftquat[2]*rightquat[2]-leftquat[3]*rightquat[3];

  prodquat[1] = leftquat[0]*rightquat[1]+leftquat[1]*rightquat[0]
```

```
              + leftquat[2]*rightquat[3]-leftquat[3]*rightquat[2];

  prodquat[2] = leftquat[0]*rightquat[2]-leftquat[1]*rightquat[3]
              + leftquat[2]*rightquat[0]+leftquat[3]*rightquat[1];

  prodquat[3] = leftquat[0]*rightquat[3]+leftquat[1]*rightquat[2]
              - leftquat[2]*rightquat[1]+leftquat[3]*rightquat[0];
}

/* complement a quaternion */
/* this gives the inverse of a unit-norm quaternion */
quatcomp(double qoriginal[], double qcomplemented[])
{
  qcomplemented[0] = qoriginal[0];
  qcomplemented[1] = -qoriginal[1];
  qcomplemented[2] = -qoriginal[2];
  qcomplemented[3] = -qoriginal[3];
}

/* rotate a position quaternion by a rotation quaternion */
quatrot(double qrotation[], double qto_be_rotated[],
double qrotated[]){
  double qrotation_inv[4];
  double qtemporary[4];
  quatcomp(qrotation,qrotation_inv);
  quatmul(qto_be_rotated, qrotation_inv, qtemporary);
  quatmul(qrotation, qtemporary, qrotated);
}

/* get quaternion to rotate board 0 into the desired board */
board_tet_qrot(int board_no, double qrotation[]){

  double qface[4][4];

  /* quaternion to rotate board 0 into board0 */
  qface[0][0] = 1.0;
  qface[0][1] = 0.0;
  qface[0][2] = 0.0;
  qface[0][3] = 0.0;

  /* quaternion to rotate board 1 into board 0 */
  qface[1][0] =  0.5;
  qface[1][1] =  sqrt(2.0)/2.0;
  qface[1][2] = -sqrt(6.0)/6.0;
  qface[1][3] = -sqrt(3.0)/6.0;

  /* quaternion to rotate board 2 into board 0 */
  qface[2][0] =  0.0;
  qface[2][1] =  sqrt(2.0)/2.0;
  qface[2][2] = +sqrt(6.0)/6.0;
  qface[2][3] = -sqrt(3.0)/3.0;

  /* quaternion to rotate board 3 into board 0 */
  qface[3][0] =  0.5;
  qface[3][1] =  0.0;
  qface[3][2] = -sqrt(6.0)/3.0;
  qface[3][3] =  sqrt(3.0)/6.0;
```

```
  /* quaternion to rotate board 0 into the desired board */
  qrotation[0] =  qface[board_no][0];
  qrotation[1] = -qface[board_no][1];
  qrotation[2] = -qface[board_no][2];
  qrotation[3] = -qface[board_no][3];
}

get_tx_seg_end(int board_no,long point_no,
double point_position[3]){

  double meters_per_mil = 2.54e-5;
  double xf,yf,zf;
  double qpoint_in_board[4],qpoint_in_tetrahedron[4];
  double qrotation[4];
  double Mpmil = 1;

  Mpmil = 2.54e-5; /* meters per mil */

  /* model the bottom board */

  /* get point in board 0 in flattx3.h coordinates */
  qpoint_in_board[0] = 0.0;
  qpoint_in_board[1] = coil_tx_points[point_no][0];
  qpoint_in_board[2] = coil_tx_points[point_no][1];
  qpoint_in_board[3] = coil_tx_points[point_no][2];

  /*
  center of bottom board comp side in tetrahedron coord
     = (0,0,-1020.62) mils
  center of bottom board comp side in flattx3.h coord
     = (3500,2443,0) mils
  center of tetrahedron in tetrahedron coord
     = (0,0,0) mils
  center of tetrahedron in flattx3.h coord
     = (3500,2443,1020.62) mils
  */

  /* translate to tetrahedron coordinates */
  qpoint_in_board[1] = qpoint_in_board[1] - 3500.0;
  qpoint_in_board[2] = qpoint_in_board[2] - 2443.0;
  qpoint_in_board[3] = qpoint_in_board[3] - 1020.62;

  /* convert from mils to meters */
  qpoint_in_board[1] = qpoint_in_board[1] * Mpmil;
  qpoint_in_board[2] = qpoint_in_board[2] * Mpmil;
  qpoint_in_board[3] = qpoint_in_board[3] * Mpmil;

  /* get quaternion to rotate about the center of the tetrahedron
  to rotate board 0 into the desired board */

  board_tet_qrot(board_no, qrotation);

  /* rotate the point's position */
  quatrot(qrotation,qpoint_in_board,qpoint_in_tetrahedron);

  /* convert point-position quaternion to vector */
```

```
  point_position[0] = qpoint_in_tetrahedron[1];
  point_position[1] = qpoint_in_tetrahedron[2];
  point_position[2] = qpoint_in_tetrahedron[3];
}

get_rx_seg_end(int rx_coil_no,long rx_seg_num,double qpos[4],
double qori[4],double rx_seg_end[3]){
  double model_seg_end[3];
  double qmodel_seg_end[4];
  double qrx_seg_end[4];

  /* get segment endpoint for untranslated, unrotated receiver */
  get_tx_seg_end(rx_coil_no, rx_seg_num, model_seg_end);

  qmodel_seg_end[0] = 0;
  qmodel_seg_end[1] = model_seg_end[0];
  qmodel_seg_end[2] = model_seg_end[1];
  qmodel_seg_end[3] = model_seg_end[2];

  /* rotate the receiver */
  quatrot(qori, qmodel_seg_end, qrx_seg_end);

  /* translate the receiver */
  rx_seg_end[0] = qrx_seg_end[1] + qpos[1];
  rx_seg_end[1] = qrx_seg_end[2] + qpos[2];
  rx_seg_end[2] = qrx_seg_end[3] + qpos[3];
}

double Lm(int Step,int tx_coil_no,int rx_coil_no,double qpos[4],
double qori[4]){

  long tx_seg_num_max = coil_tx_num;
  long rx_seg_num_max = coil_tx_num;
  long step_in_b_max_increment = 8; /* for seven-point rule */
  long step_in_b_max = 8;
  double Weight = 1; /* rule step weight */
  double step_in_b_max_inv;
  long step_in_b;
  double b;
  long tx_seg_num;
  long rx_seg_num;
  double sum,II,b_integral;
  double Uo_over_4pi = 1e-7;
  double seg_end[3];
  double s1start[3]; /* s1 = transmitter */
  double s2start[3]; /* s2 = receiver */
  double s1end[3];
  double s2end[3];
  double s1len[3];
  double s2len[3];
  double s1se[3];
  double sbmb[3];
  double sb[3];
  double s1len2,s2len2,sb2; /* squares of s1len, s2len, sb */
  double sbDOTs1len;
  double s1lenDOTs2len;
  double ssb2;
```

```
double sgnarg,signe,sign_cos_theta;
double logargnum,logargden;
double b_integrand;
int skip_flag = 0;

/* define the coefficients for seven-point formula */

double spfn,spfp,spfq;
double spfm = ( 1867776.0/7.0 - 797568.0/3.0 )/3360.0;
spfn = (-240.0*spfm +2752.0/15.0)/72.0;
spfp = -16.0*spfm -9.0*spfn +64.0/3.0;
spfq = -2.0*spfm -2.0*spfn -2.0*spfp +8.0;

/* set integration steps: */

if (Step<0){
  Step = 0;
}
step_in_b_max = step_in_b_max_increment * (Step + 1);

step_in_b_max_inv = 1.0 / (double)step_in_b_max;
sum=0;
get_tx_seg_end(tx_coil_no,0,s1end);
for(tx_seg_num=1;tx_seg_num<tx_seg_num_max;tx_seg_num++){
  s1start[0] = s1end[0];
  s1start[1] = s1end[1];
  s1start[2] = s1end[2];
  get_tx_seg_end(tx_coil_no,tx_seg_num,s1end);
  get_rx_seg_end(rx_coil_no,0,qpos,qori,s2end);
  for(rx_seg_num=1;rx_seg_num<rx_seg_num_max;rx_seg_num++){
    s2start[0] = s2end[0];
    s2start[1] = s2end[1];
    s2start[2] = s2end[2];
    get_rx_seg_end(rx_coil_no,rx_seg_num,qpos,qori,s2end);
    s1len[0] = s1end[0] - s1start[0];
    s1len[1] = s1end[1] - s1start[1];
    s1len[2] = s1end[2] - s1start[2];
    s2len[0] = s2end[0] - s2start[0];
    s2len[1] = s2end[1] - s2start[1];
    s2len[2] = s2end[2] - s2start[2];
    s1lenDOTs2len = s1len[0]*s2len[0]+s1len[1]*s2len[1]
         +s1len[2]*s2len[2];
    if(fabs(s1lenDOTs2len)>1e-200){
      s1len2 = s1len[0]*s1len[0]+s1len[1]*s1len[1]
         +s1len[2]*s1len[2];
      s2len2 = s2len[0]*s2len[0]+s2len[1]*s2len[1]
         +s2len[2]*s2len[2];
      sgnarg = (s2start[0]-s1start[0])*s1len[0]
       +(s2start[1]-s1start[1])*s1len[1]+(s2start[2]
         -s1start[2])*s1len[2];
      signe = 1.0;
      s1se[0] = s1end[0];
      s1se[1] = s1end[1];
      s1se[2] = s1end[2];
      if(sgnarg<0.0){/*sgnarg*/
        signe = -1.0;
        s1se[0] = s1start[0];
```

```
        s1se[1] = s1start[1];
        s1se[2] = s1start[2];
}
      sbmb[0] = s2start[0] - s1se[0];
      sbmb[1] = s2start[1] - s1se[1];
      sbmb[2] = s2start[2] - s1se[2];
      b_integral = 0;
      for(step_in_b=0;step_in_b<step_in_b_max+1;step_in_b++){
        switch (step_in_b&7) {
          case 0:
            Weight = 2.0*spfm;
            if(step_in_b==0){
              Weight = spfm;
            }
            if(step_in_b==step_in_b_max){
              Weight = spfm;
            }
            skip_flag = 0;
            break;
          case 1:
            Weight = spfn;
            skip_flag = 0;
            break;
          case 2:
            Weight = 0.0;
            skip_flag = 1;
            break;
          case 3:
            Weight = spfp;
            skip_flag = 0;
            break;
          case 4:
            Weight = spfq;
            skip_flag = 0;
            break;
          case 5:
            Weight = spfp;
            skip_flag = 0;
            break;
          case 6:
            Weight = 0.0;
            skip_flag = 1;
            break;
          case 7:
            Weight = spfn;
            skip_flag = 0;
            break;
          default:
            Weight = 0.0;
            skip_flag = 1;
            break;
        }
        if(skip_flag == 0){
          b = (double)step_in_b * step_in_b_max_inv;
          sb[0] = sbmb[0] + s2len[0] * b;
          sb[1] = sbmb[1] + s2len[1] * b;
          sb[2] = sbmb[2] + s2len[2] * b;
```

```
            sb2 = sb[0]*sb[0]+sb[1]*sb[1]+sb[2]*sb[2];
            sbDOTs1len = sb[0]*s1len[0]+sb[1]*s1len[1]
                +sb[2]*s1len[2];
            sign_cos_theta = signe*sbDOTs1len/sqrt(sb2*s1len2);
            ssb2 = (s1len[0]+signe*sb[0])*(s1len[0]+signe*sb[0])
                 + (s1len[1]+signe*sb[1])*(s1len[1]+signe*sb[1])
                 + (s1len[2]+signe*sb[2])*(s1len[2]+signe*sb[2]);
            logargnum = sqrt(ssb2/sb2)+sqrt(s1len2/sb2)
                 +sign_cos_theta;
            logargden = 1.0 + sign_cos_theta;
            b_integrand = Weight * log(logargnum/logargden);
            b_integral = b_integral + b_integrand;
  }
        }
        II = b_integral*step_in_b_max_inv
                   * s1lenDOTs2len / sqrt(s1len2);
        sum = sum + II;
      }
    }
  }
  return Uo_over_4pi * sum;
}

main(int argc, char *argv[])
{
  double B[3];
  double Lmq[3];
  double X,Y,Z;
  double qpos[4];
  double qori[4];
  int Step;
  printf("currel63.c rev 1, Lm from trakr380.txt, inputs
Step X Y Z i j k\n");
  printf("transmitter is tetrahedron of ANT-003 boards\n");
  printf("receiver is tetrahedron of ANT-003 boards\n");
  Step = atoi(argv[1]);
  X = atof(argv[2]);
  Y = atof(argv[3]);
  Z = atof(argv[4]);
  qpos[0] = 0;
  qpos[1] = X;
  qpos[2] = Y;
  qpos[3] = Z;
  qori[1] = atof(argv[5]);
  qori[2] = atof(argv[6]);
  qori[3] = atof(argv[7]);
  qori[0] = sqrt(1.0-qori[1]*qori[1]-qori[2]*qori[2]
    -qori[3]*qori[3]);
  printf("   position %15e %15e %15e\n",qpos[1],qpos[2],qpos[3]);
  printf("orientation %15e %15e %15e %15e\n",qori[0],qori[1],
   qori[2],qori[3]);
  printf("Lm[tx-coil][rx-coil], see trakr376.txt for coil
numbering:\n");
  printf("%15e  %15e  %15e  %15e\n",Lm(Step,0,0,qpos,qori),
                                    Lm(Step,0,1,qpos,qori),
                                    Lm(Step,0,2,qpos,qori),
                                    Lm(Step,0,3,qpos,qori));
```

```
  printf("%15e  %15e  %15e  %15e\n",Lm(Step,1,0,qpos,qori),
                                   Lm(Step,1,1,qpos,qori),
                                   Lm(Step,1,2,qpos,qori),
                                   Lm(Step,1,3,qpos,qori));
  printf("%15e  %15e  %15e  %15e\n",Lm(Step,2,0,qpos,qori),
                                   Lm(Step,2,1,qpos,qori),
                                   Lm(Step,2,2,qpos,qori),
                                   Lm(Step,2,3,qpos,qori));
  printf("%15e  %15e  %15e  %15e\n",Lm(Step,3,0,qpos,qori),
                                   Lm(Step,3,1,qpos,qori),
                                   Lm(Step,3,2,qpos,qori),
                                   Lm(Step,3,3,qpos,qori));
}
```

Appendix K

Self- and mutual-inductance calculation results

The results of running the code in the preceding appendix for the cases of two almost-superimposed tetrahedra and of two superimposed tetrahedra are below. These results approximate the calculation of self- and mutual inductances of one tetrahedron.

For two almost-superimposed tetrahedra, the diagonal terms of the mutual-inductance `Lm` matrix are approximate values for the identical self-inductances of the four coils of one tetrahedron. The variations between `Step` = `0` and `Step` = `2` are smaller than the variations between diagonal terms, which are due to our use of translational displacement to approximate the cross-section of the conductor.

```
currel63.c rev 1, Lm from trakr380.txt, inputs Step X Y Z i j k
transmitter is tetrahedron of ANT-003 boards
receiver is tetrahedron of ANT-003 boards
   position    6.350000e-05    6.350000e-05    6.350000e-05
orientation    1.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
Lm[tx-coil][rx-coil], see trakr376.txt for coil numbering:
   4.093351e-04    -4.933638e-05    -4.909026e-05    -4.920916e-05
  -4.942200e-05     4.100148e-04    -4.907677e-05    -4.925501e-05
  -4.966513e-05    -4.956566e-05     4.084597e-04    -4.949881e-05
  -4.942502e-05    -4.938567e-05    -4.914188e-05     4.103232e-04
```

The results above are for `Step` = `0`. The results below are for `Step` = `2`.

```
currel63.c rev 1, Lm from trakr380.txt, inputs Step X Y Z i j k
transmitter is tetrahedron of ANT-003 boards
receiver is tetrahedron of ANT-003 boards
   position    6.350000e-05    6.350000e-05    6.350000e-05
orientation    1.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
Lm[tx-coil][rx-coil], see trakr376.txt for coil numbering:
   4.097835e-04    -4.933346e-05    -4.908749e-05    -4.920619e-05
  -4.941899e-05     4.104875e-04    -4.907399e-05    -4.925189e-05
  -4.966206e-05    -4.956234e-05     4.088778e-04    -4.949580e-05
  -4.942218e-05    -4.938277e-05    -4.913878e-05     4.108071e-04
```

For two superimposed tetrahedra, the diagonal terms of the mutual-inductance `Lm` matrix are "nan", meaning not-a-number, because the self-inductance of a coil of infinitely-thin conductors is infinite.

The off-diagonal terms are the mutual inductances between pairs of coils of one tetrahedron. The matrix should be symmetrical since mutual inductance is unchanged when two coils are interchanged.

Each coil is slightly asymmetrical from perfect three-fold rotational symmetry. This leads to small variations in the mutual inductance between coil 0 and the other three coils. Coils 1, 2, and 3 are mounted so the asymmetries are the same for any pair of these three coils, so their mutual inductances should be identical, though slightly different from mutual inductances to coil 0.

The ratio of mutual inductance below to self-inductance above, which is about 0.12, determines how large the off-diagonal terms in the transmitter current matrix will be compared to the diagonal terms.

The superimposed-tetrahera results are on the next page.

The results below are for `Step` = 0.

```
currel63.c rev 1, Lm from trakr380.txt, inputs Step X Y Z i j k
transmitter is tetrahedron of ANT-003 boards
receiver is tetrahedron of ANT-003 boards
   position    0.000000e+00    0.000000e+00    0.000000e+00
orientation    1.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
Lm[tx-coil][rx-coil], see trakr376.txt for coil numbering:
            nan    -4.938005e-05    -4.937655e-05    -4.931752e-05
  -4.938009e-05              nan    -4.932048e-05    -4.932075e-05
  -4.937655e-05    -4.932075e-05              nan    -4.932048e-05
  -4.931728e-05    -4.932048e-05    -4.932075e-05              nan
```

The results below are for `Step` = 1.

```
currel63.c rev 1, Lm from trakr380.txt, inputs Step X Y Z i j k
transmitter is tetrahedron of ANT-003 boards
receiver is tetrahedron of ANT-003 boards
   position    0.000000e+00    0.000000e+00    0.000000e+00
orientation    1.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
Lm[tx-coil][rx-coil], see trakr376.txt for coil numbering:
            nan    -4.937696e-05    -4.937352e-05    -4.931435e-05
  -4.937695e-05              nan    -4.931749e-05    -4.931747e-05
  -4.937352e-05    -4.931747e-05              nan    -4.931749e-05
  -4.931435e-05    -4.931749e-05    -4.931747e-05              nan
```

The results below are for `Step` = 2.

```
currel63.c rev 1, Lm from trakr380.txt, inputs Step X Y Z i j k
transmitter is tetrahedron of ANT-003 boards
receiver is tetrahedron of ANT-003 boards
   position    0.000000e+00    0.000000e+00    0.000000e+00
orientation    1.000000e+00    0.000000e+00    0.000000e+00    0.000000e+00
Lm[tx-coil][rx-coil], see trakr376.txt for coil numbering:
            nan    -4.937707e-05    -4.937364e-05    -4.931447e-05
  -4.937707e-05              nan    -4.931760e-05    -4.931759e-05
  -4.937364e-05    -4.931759e-05              nan    -4.931760e-05
  -4.931447e-05    -4.931760e-05    -4.931759e-05              nan
```

The results above demonstrate that `Step` = 0 is sufficient to accurately numerically integrate all the cases in this work. Here the coils are closer together than they are for any of the mechanical positions discussed in this work, and thus should exhibit larger errors in the numerical integration than do the mechanical positions. Note that each off-diagonal term above varies by less than one part in 10000 over the three runs, which is less than other sources of error in the experimental data points.

Note also that each matrix is symmetrical to one part in 100000 of the terms being compared. The asymmetry is due to numerical errors in the calculation, so the numerical errors are small enough to be neglected.

Appendix L

Complex data class for C++ programs

```
//ptacompl.h (was Complex.h)
#ifndef __ComplexH
#define __ComplexH

#include <math.h>
//#include "globals.h"
#include <assert.h>


class Complex
{

public:
double r;
double i;
//=====================================
// Constructors
//=====================================

Complex()
{
r = 0; i = 0;
}

Complex(double r0, double i0)
{
r = r0; i = i0;
}
// Copy Constructor
Complex(Complex &M) {
r = M.r;
i = M.i;
}
//=====================================
// Data Accessors
//=====================================
// None
//=====================================
// Operators
//=====================================
```

```
// None
//=====================================
// Methods
//=====================================
void Print();
double Magn();
};


Complex inline
operator+(Complex c1, Complex c2)
{
Complex result;
result.r = c1.r + c2.r;
result.i = c1.i + c2.i;

return result;
}

Complex inline
operator-(Complex c1, Complex c2)
{
Complex result;
result.r = c1.r - c2.r;
result.i = c1.i - c2.i;

return result;
}

Complex inline
operator*(Complex c1, Complex c2)
{
Complex result;
result.r = c1.r*c2.r - c1.i*c2.i;
result.i = c1.r*c2.i + c1.i*c2.r;

return result;
}

double inline
operator|(double f1, Complex c2)
{
double result;
result = f1*sqrt(c2.r*c2.r+c2.i*c2.i);

return result;
}

Complex inline
operator*(double f1, Complex c2)
{
Complex result;
result.r = f1*c2.r;
result.i = f1*c2.i;

return result;
}
```

```
Complex inline
operator*(Complex c1, double f2)
{
Complex result;
result.r = f2*c1.r;
result.i = f2*c1.i;

return result;
}

Complex inline
operator/(Complex c1, Complex c2)
{
Complex result;
result.r = (c1.r*c2.r + c1.i*c2.i)/(c2.r*c2.r+c2.i*c2.i);
result.i = (c1.i*c2.r - c1.r*c2.i)/(c2.r*c2.r+c2.i*c2.i);

return result;
}

Complex inline
operator/(double f1, Complex c2)
{
Complex result;
result.r = (f1*c2.r)/(c2.r*c2.r+c2.i*c2.i);
result.i = (-1.0*f1*c2.i)/(c2.r*c2.r+c2.i*c2.i);

return result;
}


#endif
```

Appendix M

Quaternion data class for C++ programs

```
//ptaquats.h (based on ptacompl.h)
#ifndef __QuaternionsH
#define __QuaternionsH

#include <math.h>
//#include "globals.h"
#include <assert.h>


class Quaternion
{

public:
double r;
double i;
double j;
double k;
//=====================================
// Constructors
//=====================================

Quaternion()
{
r = 0; i = 0; j = 0; k = 0;
}

Quaternion(double r0, double i0, double j0, double k0)
{
r = r0; i = i0; j = j0; k = k0;
}
// Copy Constructor
Quaternion(Quaternion &M) {
r = M.r;
i = M.i;
j = M.j;
k = M.k;
}
//=====================================
// Data Accessors
//=====================================
```

```
// None
//=====================================
// Operators
//=====================================
// None
//=====================================
// Methods
//=====================================
};



Quaternion inline
operator+(Quaternion c1, Quaternion c2)
{
Quaternion result;
result.r = c1.r + c2.r;
result.i = c1.i + c2.i;
result.j = c1.j + c2.j;
result.k = c1.k + c2.k;

return result;
}

Quaternion inline
operator-(Quaternion c1, Quaternion c2)
{
Quaternion result;
result.r = c1.r - c2.r;
result.i = c1.i - c2.i;
result.j = c1.j - c2.j;
result.k = c1.k - c2.k;

return result;
}

Quaternion inline
operator*(Quaternion c1, Quaternion c2)
{
Quaternion result;
result.r = c1.r*c2.r - c1.i*c2.i - c1.j*c2.j - c1.k*c2.k;
result.i = c1.r*c2.i + c1.i*c2.r + c1.j*c2.k - c1.k*c2.j;
result.j = c1.r*c2.j + c1.j*c2.r + c1.k*c2.i - c1.i*c2.k;
result.k = c1.r*c2.k + c1.k*c2.r + c1.i*c2.j - c1.j*c2.i;

return result;
}

Quaternion inline
operator*(double f1, Quaternion c2)
{
Quaternion result;
result.r = f1*c2.r;
result.i = f1*c2.i;
result.j = f1*c2.j;
result.k = f1*c2.k;
```

```
return result;
}

Quaternion inline
operator*(Quaternion c1, double f2)
{
Quaternion result;
result.r = f2*c1.r;
result.i = f2*c1.i;
result.j = f2*c1.j;
result.k = f2*c1.k;

return result;
}

double inline
operator|(int mode, Quaternion c2)
{
double result;
result = c2.i*c2.i+c2.j*c2.j+c2.k*c2.k;
if(mode==0) result = sqrt(1.0-result);
if(mode==1) result = sqrt(result+c2.r*c2.r);
if(mode==2) result = result+c2.r*c2.r;

return result;
}

double inline
operator|(double f1, Quaternion c2)
{
double result;
result = f1*sqrt(c2.r*c2.r+c2.i*c2.i+c2.j*c2.j+c2.k*c2.k);

return result;
}

Quaternion inline
operator&(double f1, Quaternion c2)
{
Quaternion result;
result.r = (f1*c2.r);
result.i = (-1.0*f1*c2.i);
result.j = (-1.0*f1*c2.j);
result.k = (-1.0*f1*c2.k);

return result;
}

Quaternion inline
operator/(double f1, Quaternion c2)
{
Quaternion result;
        double magsqrd;
        magsqrd = c2.r*c2.r+c2.i*c2.i+c2.j*c2.j+c2.k*c2.k;
result.r = (f1*c2.r)/magsqrd;
result.i = (-1.0*f1*c2.i)/magsqrd;
```

```
result.j = (-1.0*f1*c2.j)/magsqrd;
result.k = (-1.0*f1*c2.k)/magsqrd;

return result;
}

Quaternion inline
operator/(Quaternion c1, Quaternion c2)
{
Quaternion result;
result = c1*(1.0/c2);

return result;
}

Quaternion inline
operator&(Quaternion c1, Quaternion c2)
{
Quaternion result;
result = c1*(c2*(1.0&c1));

return result;
}


#endif
```

Appendix N

C++ code for data reduction to Lm matrix, P&O, and GOF

```
/* curre135.cpp data reduction for thesis data point

   rev14, 8july2001pta: based on curre129.cpp rev14
   rev15, 8july2001pta: fixed 14/15 error
   rev16, 29july2001pta: frequency 11644.497 corr to 11664.497

input from stdin:

  data point number 1 through 11, was data filename

output to stdout:

  mutual inductance matrix and fitted P&O

   */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <ptacompl.h>
#include <ptaquats.h>
#include <flattx3.h> /* header file for ANT-003 transmitter */

/* function to print a quaternion */
Quatprint(Quaternion R){
  printf("%+11.4e ",  R.r);
  printf("%+11.4e ",  R.i);
  printf("%+11.4e ",  R.j);
  printf("%+11.4e\n", R.k);
}

/* function to print a real 4x4 matrix stored as 16-element
vector */
Rmatprint16(double Rmatrix[16]){
  printf("    %+11.4e ",  Rmatrix[0]);
  printf("    %+11.4e ",  Rmatrix[1]);
  printf("    %+11.4e ",  Rmatrix[2]);
  printf("    %+11.4e\n", Rmatrix[3]);
  printf("    %+11.4e ",  Rmatrix[4]);
```

```
  printf("    %+11.4e ",  Rmatrix[5]);
  printf("    %+11.4e ",  Rmatrix[6]);
  printf("    %+11.4e\n", Rmatrix[7]);
  printf("    %+11.4e ",  Rmatrix[8]);
  printf("    %+11.4e ",  Rmatrix[9]);
  printf("    %+11.4e ",  Rmatrix[10]);
  printf("    %+11.4e\n", Rmatrix[11]);
  printf("    %+11.4e ",  Rmatrix[12]);
  printf("    %+11.4e ",  Rmatrix[13]);
  printf("    %+11.4e ",  Rmatrix[14]);
  printf("    %+11.4e\n", Rmatrix[15]);
}

/* function to print a real 4x4 matrix */
Rmatprint4x4(double Rmatrix[4][4]){
  printf("%+11.4e ",  Rmatrix[0][0]);
  printf("%+11.4e ",  Rmatrix[0][1]);
  printf("%+11.4e ",  Rmatrix[0][2]);
  printf("%+11.4e\n", Rmatrix[0][3]);
  printf("%+11.4e ",  Rmatrix[1][0]);
  printf("%+11.4e ",  Rmatrix[1][1]);
  printf("%+11.4e ",  Rmatrix[1][2]);
  printf("%+11.4e\n", Rmatrix[1][3]);
  printf("%+11.4e ",  Rmatrix[2][0]);
  printf("%+11.4e ",  Rmatrix[2][1]);
  printf("%+11.4e ",  Rmatrix[2][2]);
  printf("%+11.4e\n", Rmatrix[2][3]);
  printf("%+11.4e ",  Rmatrix[3][0]);
  printf("%+11.4e ",  Rmatrix[3][1]);
  printf("%+11.4e ",  Rmatrix[3][2]);
  printf("%+11.4e\n", Rmatrix[3][3]);
}

/* function to print a complex 4x4 matrix */
Cmatprint4x4(Complex Cmatrix[4][4]){
  printf("%+11.4e%+11.4ei ",  Cmatrix[0][0].r,Cmatrix[0][0].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[0][1].r,Cmatrix[0][1].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[0][2].r,Cmatrix[0][2].i);
  printf("%+11.4e%+11.4ei\n", Cmatrix[0][3].r,Cmatrix[0][3].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[1][0].r,Cmatrix[1][0].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[1][1].r,Cmatrix[1][1].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[1][2].r,Cmatrix[1][2].i);
  printf("%+11.4e%+11.4ei\n", Cmatrix[1][3].r,Cmatrix[1][3].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[2][0].r,Cmatrix[2][0].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[2][1].r,Cmatrix[2][1].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[2][2].r,Cmatrix[2][2].i);
  printf("%+11.4e%+11.4ei\n", Cmatrix[2][3].r,Cmatrix[2][3].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[3][0].r,Cmatrix[3][0].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[3][1].r,Cmatrix[3][1].i);
  printf("%+11.4e%+11.4ei ",  Cmatrix[3][2].r,Cmatrix[3][2].i);
  printf("%+11.4e%+11.4ei\n", Cmatrix[3][3].r,Cmatrix[3][3].i);
}

/* start of functions for 4x4 complex matrix inverse */

Cmatmul4x4(Complex l[4][4],Complex r[4][4],Complex prod[4][4]){
```

```
  prod[0][0] = l[0][0]*r[0][0]+l[0][1]*r[1][0]+l[0][2]*r[2][0]
    +l[0][3]*r[3][0];
  prod[0][1] = l[0][0]*r[0][1]+l[0][1]*r[1][1]+l[0][2]*r[2][1]
    +l[0][3]*r[3][1];
  prod[0][2] = l[0][0]*r[0][2]+l[0][1]*r[1][2]+l[0][2]*r[2][2]
    +l[0][3]*r[3][2];
  prod[0][3] = l[0][0]*r[0][3]+l[0][1]*r[1][3]+l[0][2]*r[2][3]
    +l[0][3]*r[3][3];

  prod[1][0] = l[1][0]*r[0][0]+l[1][1]*r[1][0]+l[1][2]*r[2][0]
    +l[1][3]*r[3][0];
  prod[1][1] = l[1][0]*r[0][1]+l[1][1]*r[1][1]+l[1][2]*r[2][1]
    +l[1][3]*r[3][1];
  prod[1][2] = l[1][0]*r[0][2]+l[1][1]*r[1][2]+l[1][2]*r[2][2]
    +l[1][3]*r[3][2];
  prod[1][3] = l[1][0]*r[0][3]+l[1][1]*r[1][3]+l[1][2]*r[2][3]
    +l[1][3]*r[3][3];

  prod[2][0] = l[2][0]*r[0][0]+l[2][1]*r[1][0]+l[2][2]*r[2][0]
    +l[2][3]*r[3][0];
  prod[2][1] = l[2][0]*r[0][1]+l[2][1]*r[1][1]+l[2][2]*r[2][1]
    +l[2][3]*r[3][1];
  prod[2][2] = l[2][0]*r[0][2]+l[2][1]*r[1][2]+l[2][2]*r[2][2]
    +l[2][3]*r[3][2];
  prod[2][3] = l[2][0]*r[0][3]+l[2][1]*r[1][3]+l[2][2]*r[2][3]
    +l[2][3]*r[3][3];

  prod[3][0] = l[3][0]*r[0][0]+l[3][1]*r[1][0]+l[3][2]*r[2][0]
    +l[3][3]*r[3][0];
  prod[3][1] = l[3][0]*r[0][1]+l[3][1]*r[1][1]+l[3][2]*r[2][1]
    +l[3][3]*r[3][1];
  prod[3][2] = l[3][0]*r[0][2]+l[3][1]*r[1][2]+l[3][2]*r[2][2]
    +l[3][3]*r[3][2];
  prod[3][3] = l[3][0]*r[0][3]+l[3][1]*r[1][3]+l[3][2]*r[2][3]
    +l[3][3]*r[3][3];
}

/* 3x3 matrix inverter and supporting routines from
"Advanced Engineering Mathematics", Seventh Edition,
by Erwin Kreyszig, John Wiley & Sons, Inc., 1993,
pp. 367-383. */

/* extension to 4x4 by PTA based on data from
"Linear Algebra and its Applications", Third Edition,
by Gilbert Strang, Harcourt Brace Jovanovich, Publishers, 1988,
pp. 231-232. */

/* determinant of 3x3 matrix */
Cmatdet3x3(Complex a[3][3], Complex determinant[1]){

  determinant[0] = ( a[0][0]*(a[1][1]*a[2][2]-a[2][1]*a[1][2])
                    -a[1][0]*(a[0][1]*a[2][2]-a[2][1]*a[0][2])
                    +a[2][0]*(a[0][1]*a[1][2]-a[1][1]*a[0][2]) );
}

/* determinant of 4x4 matrix */
Cmatdet4x4(Complex a[4][4], Complex determinant[1]){
```

```
Complex det0,det1,det2,det3;

  det0 = ( a[1][0]*(a[2][1]*a[3][2]-a[3][1]*a[2][2])
          -a[2][0]*(a[1][1]*a[3][2]-a[3][1]*a[1][2])
          +a[3][0]*(a[1][1]*a[2][2]-a[2][1]*a[1][2]) )*a[0][3];

  det1 = ( a[0][0]*(a[2][1]*a[3][2]-a[3][1]*a[2][2])
          -a[2][0]*(a[0][1]*a[3][2]-a[3][1]*a[0][2])
          +a[3][0]*(a[0][1]*a[2][2]-a[2][1]*a[0][2]) )*a[1][3];

  det2 = ( a[0][0]*(a[1][1]*a[3][2]-a[3][1]*a[1][2])
          -a[1][0]*(a[0][1]*a[3][2]-a[3][1]*a[0][2])
          +a[3][0]*(a[0][1]*a[1][2]-a[1][1]*a[0][2]) )*a[2][3];

  det3 = ( a[0][0]*(a[1][1]*a[2][2]-a[2][1]*a[1][2])
          -a[1][0]*(a[0][1]*a[2][2]-a[2][1]*a[0][2])
          +a[2][0]*(a[0][1]*a[1][2]-a[1][1]*a[0][2]) )*a[3][3];

  determinant[0] = det3 - det2 + det1 - det0;
}

Cmatinv4x4(Complex a[4][4], Complex ainv[4][4]){
  int error;
  Complex adet[1],adetinv,minor[3][3],minordet[1],minors[4][4];

  /* calculate minors of column 0 */

  /* minor of a[0][0] */
  minor[0][0] = a[1][1];
  minor[0][1] = a[1][2];
  minor[0][2] = a[1][3];
  minor[1][0] = a[2][1];
  minor[1][1] = a[2][2];
  minor[1][2] = a[2][3];
  minor[2][0] = a[3][1];
  minor[2][1] = a[3][2];
  minor[2][2] = a[3][3];
  Cmatdet3x3(minor,minordet);
  minors[0][0] = minordet[0];

  /* minor of a[1][0] */
  minor[0][0] = a[0][1];
  minor[0][1] = a[0][2];
  minor[0][2] = a[0][3];
  Cmatdet3x3(minor,minordet);
  minors[1][0] = minordet[0];

  /* minor of a[2][0] */
  minor[1][0] = a[1][1];
  minor[1][1] = a[1][2];
  minor[1][2] = a[1][3];
  Cmatdet3x3(minor,minordet);
  minors[2][0] = minordet[0];

  /* minor of a[3][0] */
  minor[2][0] = a[2][1];
  minor[2][1] = a[2][2];
```

```
minor[2][2] = a[2][3];
Cmatdet3x3(minor,minordet);
minors[3][0] = minordet[0];

/* calculate minors of column 1 */

/* minor of a[0][1] */
minor[0][0] = a[1][0];
minor[0][1] = a[1][2];
minor[0][2] = a[1][3];
minor[1][0] = a[2][0];
minor[1][1] = a[2][2];
minor[1][2] = a[2][3];
minor[2][0] = a[3][0];
minor[2][1] = a[3][2];
minor[2][2] = a[3][3];
Cmatdet3x3(minor,minordet);
minors[0][1] = minordet[0];

/* minor of a[1][1] */
minor[0][0] = a[0][0];
minor[0][1] = a[0][2];
minor[0][2] = a[0][3];
Cmatdet3x3(minor,minordet);
minors[1][1] = minordet[0];

/* minor of a[2][1] */
minor[1][0] = a[1][0];
minor[1][1] = a[1][2];
minor[1][2] = a[1][3];
Cmatdet3x3(minor,minordet);
minors[2][1] = minordet[0];

/* minor of a[3][1] */
minor[2][0] = a[2][0];
minor[2][1] = a[2][2];
minor[2][2] = a[2][3];
Cmatdet3x3(minor,minordet);
minors[3][1] = minordet[0];

/* calculate minors of column 2 */

/* minor of a[0][2] */
minor[0][0] = a[1][0];
minor[0][1] = a[1][1];
minor[0][2] = a[1][3];
minor[1][0] = a[2][0];
minor[1][1] = a[2][1];
minor[1][2] = a[2][3];
minor[2][0] = a[3][0];
minor[2][1] = a[3][1];
minor[2][2] = a[3][3];
Cmatdet3x3(minor,minordet);
minors[0][2] = minordet[0];

/* minor of a[1][2] */
minor[0][0] = a[0][0];
```

```
minor[0][1] = a[0][1];
minor[0][2] = a[0][3];
Cmatdet3x3(minor,minordet);
minors[1][2] = minordet[0];

/* minor of a[2][2] */
minor[1][0] = a[1][0];
minor[1][1] = a[1][1];
minor[1][2] = a[1][3];
Cmatdet3x3(minor,minordet);
minors[2][2] = minordet[0];

/* minor of a[3][2] */
minor[2][0] = a[2][0];
minor[2][1] = a[2][1];
minor[2][2] = a[2][3];
Cmatdet3x3(minor,minordet);
minors[3][2] = minordet[0];

/* calculate minors of column 3 */

/* minor of a[0][3] */
minor[0][0] = a[1][0];
minor[0][1] = a[1][1];
minor[0][2] = a[1][2];
minor[1][0] = a[2][0];
minor[1][1] = a[2][1];
minor[1][2] = a[2][2];
minor[2][0] = a[3][0];
minor[2][1] = a[3][1];
minor[2][2] = a[3][2];
Cmatdet3x3(minor,minordet);
minors[0][3] = minordet[0];

/* minor of a[1][3] */
minor[0][0] = a[0][0];
minor[0][1] = a[0][1];
minor[0][2] = a[0][2];
Cmatdet3x3(minor,minordet);
minors[1][3] = minordet[0];

/* minor of a[2][3] */
minor[1][0] = a[1][0];
minor[1][1] = a[1][1];
minor[1][2] = a[1][2];
Cmatdet3x3(minor,minordet);
minors[2][3] = minordet[0];

/* minor of a[3][3] */
minor[2][0] = a[2][0];
minor[2][1] = a[2][1];
minor[2][2] = a[2][2];
Cmatdet3x3(minor,minordet);
minors[3][3] = minordet[0];

/* if not singular, convert minors to ainv */
Cmatdet4x4(a,adet);
```

```
  error=1; /* singular matrix if |adet| = 0 */
  if((1.0|adet[0]) !=0){
    error = 0;
    adetinv = 1.0/adet[0];
    /* transpose, calc cofactor signs, divide by determinant */
    ainv[0][0] =       adetinv*minors[0][0];
    ainv[0][1] = -1.0*(adetinv*minors[1][0]);
    ainv[0][2] =       adetinv*minors[2][0];
    ainv[0][3] = -1.0*(adetinv*minors[3][0]);

    ainv[1][0] = -1.0*(adetinv*minors[0][1]);
    ainv[1][1] =       adetinv*minors[1][1];
    ainv[1][2] = -1.0*(adetinv*minors[2][1]);
    ainv[1][3] =       adetinv*minors[3][1];

    ainv[2][0] =       adetinv*minors[0][2];
    ainv[2][1] = -1.0*(adetinv*minors[1][2]);
    ainv[2][2] =       adetinv*minors[2][2];
    ainv[2][3] = -1.0*(adetinv*minors[3][2]);

    ainv[3][0] = -1.0*(adetinv*minors[0][3]);
    ainv[3][1] =       adetinv*minors[1][3];
    ainv[3][2] = -1.0*(adetinv*minors[2][3]);
    ainv[3][3] =       adetinv*minors[3][3];
  }
  return(error);
}

/* end of functions for 4x4 complex matrix inverse */

/* start of functions needed to calculate
   mutual-inductance double integral */

/* get quaternion to rotate board 0 into the desired board */
board_tet_qrot(int board_no, Quaternion qrotation[1]){

  Quaternion qface[4];

  /* quaternion to rotate board 0 into board0 */
  qface[0].r = 1.0;
  qface[0].i = 0.0;
  qface[0].j = 0.0;
  qface[0].k = 0.0;

  /* quaternion to rotate board 1 into board 0 */
  qface[1].r =  0.5;
  qface[1].i =  sqrt(2.0)/2.0;
  qface[1].j = -sqrt(6.0)/6.0;
  qface[1].k = -sqrt(3.0)/6.0;

  /* quaternion to rotate board 2 into board 0 */
  qface[2].r =  0.0;
  qface[2].i =  sqrt(2.0)/2.0;
  qface[2].j = +sqrt(6.0)/6.0;
  qface[2].k = -sqrt(3.0)/3.0;

  /* quaternion to rotate board 3 into board 0 */
```

```
  qface[3].r =  0.5;
  qface[3].i =  0.0;
  qface[3].j = -sqrt(6.0)/3.0;
  qface[3].k =  sqrt(3.0)/6.0;

  /* quaternion to rotate board 0 into the desired board */
  qrotation[0] =  1.0/qface[board_no];

}

get_tx_seg_end(int board_no,long point_no,
Quaternion qtx_seg_end[1]){

  double meters_per_mil = 2.54e-5;
  double xf,yf,zf;
  Quaternion qpoint_in_board,qrotation[1],qtetrminusboard;
  double Mpmil = 1;

  Mpmil = 2.54e-5; /* meters per mil */

  /* model the bottom board */

  /* get point in board 0 in flattx3.h coordinates */
  qpoint_in_board.r = 0.0;
  qpoint_in_board.i = coil_tx_points[point_no][0];
  qpoint_in_board.j = coil_tx_points[point_no][1];
  qpoint_in_board.k = coil_tx_points[point_no][2];

  /*
  center of bottom board comp side in tetrahedron coord
     = (0,0,-1020.62) mils
  center of bottom board comp side in flattx3.h coord
     = (3500,2443,0) mils
  center of tetrahedron in tetrahedron coord
     = (0,0,0) mils
  center of tetrahedron in flattx3.h coord
     = (3500,2443,1020.62) mils
  */

  /* difference in mils between tetrahedron and board coord */
  qtetrminusboard.r = 0.0;
  qtetrminusboard.i = -3500.0;
  qtetrminusboard.j = -2443.0;
  qtetrminusboard.k = -1020.62;

  /* translate to tetrahedron coordinates */
  qpoint_in_board = qpoint_in_board + qtetrminusboard;

  /* convert from mils to meters */
  qpoint_in_board = Mpmil * qpoint_in_board;

  /* get quaternion to rotate about the center of the tetrahedron
  to rotate board 0 into the desired board */

  board_tet_qrot(board_no, qrotation);

  /* rotate the point's position */
```

```
  qtx_seg_end[0] = qrotation[0] & qpoint_in_board;

}

get_rx_seg_end(int rx_coil_no,long rx_seg_num,
Quaternion qpos,Quaternion qori,Quaternion qrx_seg_end[1]){

  Quaternion qmodel_seg_end[1];

  /* get segment endpoint for untranslated, unrotated receiver */
  get_tx_seg_end(rx_coil_no, rx_seg_num, qmodel_seg_end);

  /* rotate the receiver */
  qrx_seg_end[0] = qori & qmodel_seg_end[0];

  /* translate the receiver */
  qrx_seg_end[0] = qrx_seg_end[0] + qpos;

}

double Lm(int Step,int tx_coil_no,int rx_coil_no,
Quaternion qpos,Quaternion qori){

  long tx_seg_num_max = coil_tx_num;
  long rx_seg_num_max = coil_tx_num;
  long step_in_b_max_increment = 8; /* for seven-point rule */
  long step_in_b_max = 8;
  double Weight = 1; /* rule step weight */
  double step_in_b_max_inv;
  long step_in_b;
  double b;
  long tx_seg_num;
  long rx_seg_num;
  double sum,II,b_integral;
  double Uo_over_4pi = 1e-7;
  double seg_end[3];
  double s1start[3]; /* s1 = transmitter */
  double s2start[3]; /* s2 = receiver */
  Quaternion qtemp[1];
  double s1end[3];
  double s2end[3];
  double s1len[3];
  double s2len[3];
  double s1se[3];
  double sbmb[3];
  double sb[3];
  double s1len2,s2len2,sb2; /* squares of s1len, s2len, sb */
  double sbDOTs1len;
  double s1lenDOTs2len;
  double ssb2;
  double sgnarg,signe,sign_cos_theta;
  double logargnum,logargden;
  double b_integrand;
  int skip_flag = 0;

  /* define the coefficients for seven-point formula */
```

```
double spfn,spfp,spfq;
double spfm = ( 1867776.0/7.0 - 797568.0/3.0 )/3360.0;
spfn = (-240.0*spfm +2752.0/15.0)/72.0;
spfp = -16.0*spfm -9.0*spfn +64.0/3.0;
spfq = -2.0*spfm -2.0*spfn -2.0*spfp +8.0;

/* set integration steps: */

if (Step<0){
  Step = 0;
}
step_in_b_max = step_in_b_max_increment * (Step + 1);

step_in_b_max_inv = 1.0 / (double)step_in_b_max;
sum=0;
get_tx_seg_end(tx_coil_no,0,qtemp);
s1end[0] = qtemp[0].i;
s1end[1] = qtemp[0].j;
s1end[2] = qtemp[0].k;
for(tx_seg_num=1;tx_seg_num<tx_seg_num_max;tx_seg_num++){
  s1start[0] = s1end[0];
  s1start[1] = s1end[1];
  s1start[2] = s1end[2];
  get_tx_seg_end(tx_coil_no,tx_seg_num,qtemp);
  s1end[0] = qtemp[0].i;
  s1end[1] = qtemp[0].j;
  s1end[2] = qtemp[0].k;
  get_rx_seg_end(rx_coil_no,0,qpos,qori,qtemp);
  s2end[0] = qtemp[0].i;
  s2end[1] = qtemp[0].j;
  s2end[2] = qtemp[0].k;
  for(rx_seg_num=1;rx_seg_num<rx_seg_num_max;rx_seg_num++){
    s2start[0] = s2end[0];
    s2start[1] = s2end[1];
    s2start[2] = s2end[2];
    get_rx_seg_end(rx_coil_no,rx_seg_num,qpos,qori,qtemp);
    s2end[0] = qtemp[0].i;
    s2end[1] = qtemp[0].j;
    s2end[2] = qtemp[0].k;
    s1len[0] = s1end[0] - s1start[0];
    s1len[1] = s1end[1] - s1start[1];
    s1len[2] = s1end[2] - s1start[2];
    s2len[0] = s2end[0] - s2start[0];
    s2len[1] = s2end[1] - s2start[1];
    s2len[2] = s2end[2] - s2start[2];
    s1lenDOTs2len = s1len[0]*s2len[0]+s1len[1]*s2len[1]
     +s1len[2]*s2len[2];
    if(fabs(s1lenDOTs2len)>1e-200){
      s1len2 = s1len[0]*s1len[0]+s1len[1]*s1len[1]
       +s1len[2]*s1len[2];
      s2len2 = s2len[0]*s2len[0]+s2len[1]*s2len[1]
       +s2len[2]*s2len[2];
      sgnarg = (s2start[0]-s1start[0])*s1len[0]
       +(s2start[1]-s1start[1])*s1len[1]
       +(s2start[2]-s1start[2])*s1len[2];
      signe = 1.0;
```

```
        s1se[0] = s1end[0];
        s1se[1] = s1end[1];
        s1se[2] = s1end[2];
        if(sgnarg<0.0){/*sgnarg*/
          signe = -1.0;
          s1se[0] = s1start[0];
          s1se[1] = s1start[1];
          s1se[2] = s1start[2];
}
        sbmb[0] = s2start[0] - s1se[0];
        sbmb[1] = s2start[1] - s1se[1];
        sbmb[2] = s2start[2] - s1se[2];
        b_integral = 0;
        for(step_in_b=0;step_in_b<step_in_b_max+1;step_in_b++){
          switch (step_in_b&7) {
            case 0:
              Weight = 2.0*spfm;
              if(step_in_b==0){
                Weight = spfm;
              }
              if(step_in_b==step_in_b_max){
                Weight = spfm;
              }
              skip_flag = 0;
              break;
            case 1:
              Weight = spfn;
              skip_flag = 0;
              break;
            case 2:
              Weight = 0.0;
              skip_flag = 1;
              break;
            case 3:
              Weight = spfp;
              skip_flag = 0;
              break;
            case 4:
              Weight = spfq;
              skip_flag = 0;
              break;
            case 5:
              Weight = spfp;
              skip_flag = 0;
              break;
            case 6:
              Weight = 0.0;
              skip_flag = 1;
              break;
            case 7:
              Weight = spfn;
              skip_flag = 0;
              break;
            default:
              Weight = 0.0;
              skip_flag = 1;
              break;
```

```
          }
          if(skip_flag == 0){
            b = (double)step_in_b * step_in_b_max_inv;
            sb[0] = sbmb[0] + s2len[0] * b;
            sb[1] = sbmb[1] + s2len[1] * b;
            sb[2] = sbmb[2] + s2len[2] * b;
            sb2 = sb[0]*sb[0]+sb[1]*sb[1]+sb[2]*sb[2];
            sbDOTs1len = sb[0]*s1len[0]+sb[1]*s1len[1]
             +sb[2]*s1len[2];
            sign_cos_theta = signe*sbDOTs1len/sqrt(sb2*s1len2);
            ssb2 = (s1len[0]+signe*sb[0])*(s1len[0]+signe*sb[0])
                 + (s1len[1]+signe*sb[1])*(s1len[1]+signe*sb[1])
                 + (s1len[2]+signe*sb[2])*(s1len[2]+signe*sb[2]);
            logargnum = sqrt(ssb2/sb2)+sqrt(s1len2/sb2)
             +sign_cos_theta;
            logargden = 1.0 + sign_cos_theta;
            b_integrand = Weight * log(logargnum/logargden);
            b_integral = b_integral + b_integrand;
  }
        }
        II = b_integral*step_in_b_max_inv
                 * s1lenDOTs2len / sqrt(s1len2);
        sum = sum + II;
      }
    }
  }
  return Uo_over_4pi * sum;
}

/* end of functions needed to calculate
   mutual-inductance double integral */

/* start of functions needed to calculate
   least-squares fit and goodness of fit */

/* function to calculate tetrahedral 4x4 mutual-inductance matrix
   Lm[tx_coil][rx_coil] = Lm44[tx_coil*4+rx_coil] */
Lm44calc(int Step,Quaternion qpos,Quaternion qori,
double Lm44[16]){
  Lm44[0]  = Lm(Step,0,0,qpos,qori);
  Lm44[1]  = Lm(Step,0,1,qpos,qori);
  Lm44[2]  = Lm(Step,0,2,qpos,qori);
  Lm44[3]  = Lm(Step,0,3,qpos,qori);
  Lm44[4]  = Lm(Step,1,0,qpos,qori);
  Lm44[5]  = Lm(Step,1,1,qpos,qori);
  Lm44[6]  = Lm(Step,1,2,qpos,qori);
  Lm44[7]  = Lm(Step,1,3,qpos,qori);
  Lm44[8]  = Lm(Step,2,0,qpos,qori);
  Lm44[9]  = Lm(Step,2,1,qpos,qori);
  Lm44[10] = Lm(Step,2,2,qpos,qori);
  Lm44[11] = Lm(Step,2,3,qpos,qori);
  Lm44[12] = Lm(Step,3,0,qpos,qori);
  Lm44[13] = Lm(Step,3,1,qpos,qori);
  Lm44[14] = Lm(Step,3,2,qpos,qori);
  Lm44[15] = Lm(Step,3,3,qpos,qori);
}
```

```
atamul(double ainp[16][6],double ataout[6][6]){
  int atarow = 0;
  int atacol = 0;
  int ataterm = 0;

  for (atarow = 0; atarow < 6; atarow++)
    {
    for (atacol = 0; atacol < 6; atacol++)
      {
        ataout[atarow][atacol] = 0.0;
        for (ataterm = 0; ataterm < 16; ataterm++)
  {
            ataout[atarow][atacol] +=
              ainp[ataterm][atarow] * ainp[ataterm][atacol];
  }
      }
    }
}

matmul6161(double lefm[6][16],double rigv[16], double prod61[6]){
  int prodraw = 0;
  int prodtern = 0;

    for (prodraw = 0; prodraw < 6; prodraw++)
      {
        prod61[prodraw] = 0.0;
        for (prodtern = 0; prodtern < 16; prodtern++)
          {
            prod61[prodraw] +=
              lefm[prodraw][prodtern] * rigv[prodtern];
          }
      }
}

matmul6616T(double lefmm[6][6], double rigmT[16][6],
double prod616[6][16]){
  int prodrowm = 0;
  int prodcolm = 0;
  int prodtermm = 0;

    for (prodrowm = 0; prodrowm < 6; prodrowm++)
      {
      for (prodcolm = 0; prodcolm < 16; prodcolm++)
{
          prod616[prodrowm][prodcolm] = 0.0;
          for (prodtermm = 0; prodtermm < 6; prodtermm++)
    {
              prod616[prodrowm][prodcolm] +=
                lefmm[prodrowm][prodtermm]
                * rigmT[prodcolm][prodtermm];
    }
}
      }
}

matmul66(double lef[6][6],double rig[6][6], double prod66[6][6]){
  int prodrow = 0;
```

```
  int prodcol = 0;
  int prodterm = 0;

    for (prodrow = 0; prodrow < 6; prodrow++)
      {
      for (prodcol = 0; prodcol < 6; prodcol++)
{
          prod66[prodrow][prodcol] = 0.0;
          for (prodterm = 0; prodterm < 6; prodterm++)
    {
              prod66[prodrow][prodcol] +=
                lef[prodrow][prodterm] * rig[prodterm][prodcol];
    }
}
      }
}

/* 6x6 real matrix inverter adapted by G. Lee Beauregard from
   ‘‘Numerical Recipes for C’’ */

int mInversion6x6(double source[6][6], double inverse[6][6]){
int ludcmp(double a[6][6], int n, int indx[6], double *d);
int lubksb(double a[6][6], int n, int indx[6], double b[6]);
  /* 6x6 matrix inverse: alters source matrix! */
  int N=6;
  int error_code = 0; /* 0 means no errors */
  double d;
  int i,j,indx[6];
  double colc[6];

  error_code = ludcmp(source, N, indx, &d);
  if ( error_code == 0){
    for(j=0; j<N; j++){
      for(i=0; i<N; i++) colc[i]=0.0;
      colc[j] = 1.0;
      lubksb(source,N,indx,colc);
      for(i=0; i<N; i++) {
        inverse[i][j]=colc[i];
      }
    }
    return( error_code );
  }
  else {
    printf("Singular 6x6 matrix, cannot be inverted!\n");
    return(1); /* 1 means singular matrix */
  }
}

int ludcmp(double a[6][6], int n, int indx[6], double *d){
  double TINY = 1e-200;
  int i, imax, j,k;
  double big, dum, sum, temp;
  double vv[6];

  *d=1.0;

  for(i=0;i<n;i++){
```

```
    big=0.0;
    for(j=0;j<n;j++)
      if ((temp=fabs(a[i][j])) > big) big =temp;
    if (big == 0.0) {
      printf("Singular matrix!\n");
      return(2); /* 2 means singular matrix */
    }
    vv[i]=1.0/big;
    /*    printf("vv[%d]:%lf\n", i, vv[i]);
    */
  }

  for(j=0;j<n;j++){
    for(i=0;i<j;i++){
      sum=a[i][j];
      for(k=0;k<i;k++) sum -= a[i][k]*a[k][j];
      a[i][j]=sum;
    }
    big=0.0;
    for(i=j;i<n;i++){
      sum=a[i][j];
      for(k=0;k<j;k++) sum -= a[i][k]*a[k][j];/*problems here ?*/
      a[i][j]=sum;
      if ((dum=vv[i]*fabs(sum)) >= big){
        big=dum;
        imax=i;
      }
    }
    if (j != imax) {
      for(k=0;k<n;k++) {
        dum=a[imax][k];
        a[imax][k]=a[j][k];
        a[j][k]=dum;
      }
      *d=-(*d);
      vv[imax]=vv[j];
    }
    indx[j]=imax;
    /*    printf("indx[%d]=%d\n", j, imax);
    */
    if (a[j][j] == 0.0) a[j][j]=TINY;
    if( j != n-1 ) {
      dum = 1.0/(a[j][j]);
      for (i=j+1; i<n; i++) a[i][j] *= dum;
    }
  }
  return(0); /* 0 means no errors */

}

int lubksb(double a[6][6], int n, int indx[6], double b[6]){
  int i, ii=0, ip, j;
  double sum;

  for (i=0; i<n; i++){
    ip=indx[i];
    sum=b[ip];
```

```
    b[ip]=b[i];
    if (ii)
      for (j=ii-1; j<=i-1; j++) sum-=a[i][j]*b[j];
    else if (sum) ii=i+1;
    b[i]=sum;
  }
  for (i=n-1; i>=0; i--){
    sum=b[i];
    for( j=i+1; j<n; j++) sum -=a[i][j]*b[j];  //i+1
    b[i]=sum/a[i][i];
  }
}
/* end of 6x6 real matrix inverter */

ratiodiff(double recip,double A[16],double B[16],
double AminusB[16]){
  AminusB[0] = recip*(A[0] - B[0]);
  AminusB[1] = recip*(A[1] - B[1]);
  AminusB[2] = recip*(A[2] - B[2]);
  AminusB[3] = recip*(A[3] - B[3]);
  AminusB[4] = recip*(A[4] - B[4]);
  AminusB[5] = recip*(A[5] - B[5]);
  AminusB[6] = recip*(A[6] - B[6]);
  AminusB[7] = recip*(A[7] - B[7]);
  AminusB[8] = recip*(A[8] - B[8]);
  AminusB[9] = recip*(A[9] - B[9]);
  AminusB[10] = recip*(A[10] - B[10]);
  AminusB[11] = recip*(A[11] - B[11]);
  AminusB[12] = recip*(A[12] - B[12]);
  AminusB[13] = recip*(A[13] - B[13]);
  AminusB[14] = recip*(A[14] - B[14]);
  AminusB[15] = recip*(A[15] - B[15]);
}

least_squares_fit(int Step,Quaternion Rseed, Quaternion Qseed,
double Lmmeas[4][4],Quaternion Rfitted[1],
Quaternion Qfitted[1]){
  double range;
  /* loop indices */
  int i,j;
  int loop1 = 0;
  /* delta position in meters for partial derivatives */
  double delta = 0.001;
  double halfdelta;
  double deltarecip;
  /* delta orientation for partial derivatives */
  double odelta = 0.001;
  double halfodelta;
  double odeltarecip;
  /* temporary delta-ed position quaternion: */
  Quaternion qtempos;
  /* temporary delta-ed orientation quaternion: */
  Quaternion qto;
  /* Lmmeas converted from matrix to vector */
  double Lm44_meas[16];
  /* elements of partial-derivative numerators */
  double Lm44_nom[16];
```

```
double Lm44_tdx[16];
double Lm44_tdy[16];
double Lm44_tdz[16];
double Lm44_bdx[16];
double Lm44_bdy[16];
double Lm44_bdz[16];
double Lm44_tdi[16];
double Lm44_tdj[16];
double Lm44_tdk[16];
double Lm44_bdi[16];
double Lm44_bdj[16];
double Lm44_bdk[16];
/* partial derivatives */
double dLm44_dx[16];
double dLm44_dy[16];
double dLm44_dz[16];
double dLm44_di[16];
double dLm44_dj[16];
double dLm44_dk[16];
/* least-squares fit vectors, quaternions, and matrices */
double y[16];
double x[6];
double A[16][6];
double ATA[6][6];
double ATAinv[6][6];
double temp66[6][6];
double ATAinvAT[6][16];
Quaternion qori_delta;
Quaternion vpos_delta;
Quaternion iay,jay,kay;

iay.r = 0.0;
iay.i = 1.0;
iay.j = 0.0;
iay.k = 0.0;
jay.r = 0.0;
jay.i = 0.0;
jay.j = 1.0;
jay.k = 0.0;
kay.r = 0.0;
kay.i = 0.0;
kay.j = 0.0;
kay.k = 1.0;

halfdelta = delta / 2.0;
deltarecip = 1.0 / delta;
halfodelta = odelta / 2.0;
odeltarecip = 1.0 / odelta;

/* calculate range */
Rseed.r = 0.0; /* real part of position quat must be zero */
range = 1.0 | Rseed; /* calculate magnitude of quaternion */

/* convert measured data from 4x4 matrix to 16-element vector*/
for(i=0;i<4;i++){
  for(j=0;j<4;j++){
    Lm44_meas[j+(4*i)] = Lmmeas[i][j];
```

```
  }
}

/* get nominal point */
qtempos = Rseed;
Lm44calc(Step, qtempos, Qseed, Lm44_nom);

/* get points for position partial derivatives */

qtempos = Rseed + (iay * halfdelta);
Lm44calc(Step, qtempos, Qseed, Lm44_tdx);

qtempos = Rseed + (jay * halfdelta);
Lm44calc(Step, qtempos, Qseed, Lm44_tdy);

qtempos = Rseed + (kay * halfdelta);
Lm44calc(Step, qtempos, Qseed, Lm44_tdz);

qtempos = Rseed - (iay * halfdelta);
Lm44calc(Step, qtempos, Qseed, Lm44_bdx);

qtempos = Rseed - (jay * halfdelta);
Lm44calc(Step, qtempos, Qseed, Lm44_bdy);

qtempos = Rseed - (kay * halfdelta);
Lm44calc(Step, qtempos, Qseed, Lm44_bdz);

/* get points for orientation partial derivatives */

qori_delta = iay * halfodelta;
qori_delta.r = 0 | qori_delta; /* normalize */
qto = qori_delta * Qseed;
Lm44calc(Step, Rseed, qto, Lm44_tdi);

qori_delta = jay * halfodelta;
qori_delta.r = 0 | qori_delta;
qto = qori_delta * Qseed;
Lm44calc(Step, Rseed, qto, Lm44_tdj);

qori_delta = kay * halfodelta;
qori_delta.r = 0 | qori_delta;
qto = qori_delta * Qseed;
Lm44calc(Step, Rseed, qto, Lm44_tdk);

qori_delta = iay * (-1.0 * halfodelta);
qori_delta.r = 0 | qori_delta;
qto = qori_delta * Qseed;
Lm44calc(Step, Rseed, qto, Lm44_bdi);

qori_delta = jay * (-1.0 * halfodelta);
qori_delta.r = 0 | qori_delta;
qto = qori_delta * Qseed;
Lm44calc(Step, Rseed, qto, Lm44_bdj);

qori_delta = kay * (-1.0 * halfodelta);
qori_delta.r = 0 | qori_delta;
qto = qori_delta * Qseed;
```

```
  Lm44calc(Step, Rseed, qto, Lm44_bdk);

  /* calculate partial derivatives */
  ratiodiff(deltarecip,Lm44_tdx,Lm44_bdx,dLm44_dx);
  ratiodiff(deltarecip,Lm44_tdy,Lm44_bdy,dLm44_dy);
  ratiodiff(deltarecip,Lm44_tdz,Lm44_bdz,dLm44_dz);
  ratiodiff(odeltarecip,Lm44_tdi,Lm44_bdi,dLm44_di);
  ratiodiff(odeltarecip,Lm44_tdj,Lm44_bdj,dLm44_dj);
  ratiodiff(odeltarecip,Lm44_tdk,Lm44_bdk,dLm44_dk);

  /* load y[16] vector */
  ratiodiff(1.0,Lm44_meas,Lm44_nom,y);

  /* load A[16][6] matrix */
  for(loop1=0;loop1<16;loop1++)A[loop1][0]=dLm44_dx[loop1]*range;
  for(loop1=0;loop1<16;loop1++)A[loop1][1]=dLm44_dy[loop1]*range;
  for(loop1=0;loop1<16;loop1++)A[loop1][2]=dLm44_dz[loop1]*range;
  for(loop1=0;loop1<16;loop1++) A[loop1][3] = dLm44_di[loop1];
  for(loop1=0;loop1<16;loop1++) A[loop1][4] = dLm44_dj[loop1];
  for(loop1=0;loop1<16;loop1++) A[loop1][5] = dLm44_dk[loop1];

  /* calculate ATA[6][6] matrix from A[16][6] matrix */
  atamul(A, ATA);

  /* calculate ATAinv[6][6] matrix, clobbering ATA[6][6] */
  mInversion6x6(ATA, ATAinv);

  /* calculate ATAinvAT[6][16] matrix */
  matmul6616T(ATAinv, A, ATAinvAT);

  /* calculate x = ATAinvAT * y */
  matmul6161(ATAinvAT, y, x);

  /* calculate delta position */
  vpos_delta.r = 0.0;
  vpos_delta.i = range * x[0];
  vpos_delta.j = range * x[1];
  vpos_delta.k = range * x[2];

  /* calculate output position */
  Rfitted[0] = Rseed + vpos_delta;

  /* calculate delta orientation */
  qori_delta.k = x[5];
  qori_delta.j = x[4];
  qori_delta.i = x[3];
  qori_delta.r = 0 | qori_delta; /* normalize */

  /* calculate output orientation */
  Qfitted[0] = qori_delta * Qseed;

}
/* end of least-squares fitter */

/* calculate goodness of fit */
double GOF(double Lm44meas[4][4],double Lm44model[4][4]){
  return( ( (Lm44meas[0][0]-Lm44model[0][0])
```

```
                *(Lm44meas[0][0]-Lm44model[0][0])
              +(Lm44meas[0][1]-Lm44model[0][1])
                *(Lm44meas[0][1]-Lm44model[0][1])
              +(Lm44meas[0][2]-Lm44model[0][2])
                *(Lm44meas[0][2]-Lm44model[0][2])
              +(Lm44meas[0][3]-Lm44model[0][3])
                *(Lm44meas[0][3]-Lm44model[0][3])
              +(Lm44meas[1][0]-Lm44model[1][0])
                *(Lm44meas[1][0]-Lm44model[1][0])
              +(Lm44meas[1][1]-Lm44model[1][1])
                *(Lm44meas[1][1]-Lm44model[1][1])
              +(Lm44meas[1][2]-Lm44model[1][2])
                *(Lm44meas[1][2]-Lm44model[1][2])
              +(Lm44meas[1][3]-Lm44model[1][3])
                *(Lm44meas[1][3]-Lm44model[1][3])
              +(Lm44meas[2][0]-Lm44model[2][0])
                *(Lm44meas[2][0]-Lm44model[2][0])
              +(Lm44meas[2][1]-Lm44model[2][1])
                *(Lm44meas[2][1]-Lm44model[2][1])
              +(Lm44meas[2][2]-Lm44model[2][2])
                *(Lm44meas[2][2]-Lm44model[2][2])
              +(Lm44meas[2][3]-Lm44model[2][3])
                *(Lm44meas[2][3]-Lm44model[2][3])
              +(Lm44meas[3][0]-Lm44model[3][0])
                *(Lm44meas[3][0]-Lm44model[3][0])
              +(Lm44meas[3][1]-Lm44model[3][1])
                *(Lm44meas[3][1]-Lm44model[3][1])
              +(Lm44meas[3][2]-Lm44model[3][2])
                *(Lm44meas[3][2]-Lm44model[3][2])
              +(Lm44meas[3][3]-Lm44model[3][3])
                *(Lm44meas[3][3]-Lm44model[3][3])
             )/( Lm44meas[0][0]*Lm44meas[0][0]
                +Lm44meas[0][1]*Lm44meas[0][1]
                +Lm44meas[0][2]*Lm44meas[0][2]
                +Lm44meas[0][3]*Lm44meas[0][3]
                +Lm44meas[1][0]*Lm44meas[1][0]
                +Lm44meas[1][1]*Lm44meas[1][1]
                +Lm44meas[1][2]*Lm44meas[1][2]
                +Lm44meas[1][3]*Lm44meas[1][3]
                +Lm44meas[2][0]*Lm44meas[2][0]
                +Lm44meas[2][1]*Lm44meas[2][1]
                +Lm44meas[2][2]*Lm44meas[2][2]
                +Lm44meas[2][3]*Lm44meas[2][3]
                +Lm44meas[3][0]*Lm44meas[3][0]
                +Lm44meas[3][1]*Lm44meas[3][1]
                +Lm44meas[3][2]*Lm44meas[3][2]
                +Lm44meas[3][3]*Lm44meas[3][3]
                ) );
}

/* end of functions needed to calculate
   least-squares fit and goodness of fit */

main(int argc, char *argv[]){

int datptnumint;
char datptnumstr[100];
```

```
char datfilnamstr[100];
FILE *inpdat;
char inpstr[10240];
int i,j;
int R1XCQ[256],R1YCQ[256],R1ZCQ[256],R2XCQ[256],CURCQ[256];
double R1X[256][26],R1Y[256][26],R1Z[256][26],R2X[256][26];
double CUR[256][26];
double twopi;
Complex iw[4];
Complex Iiw[4][4]; /* jw times driver current */
Complex Txc_gain[4];
Complex Iiw_inverse[4][4]; /* inverse of (jw * driver current) */
Complex Rxc_atten[4];
Complex Gain[4], S[4][4], SIiw_inverse[4][4];
Complex Lmc[4][4]; /* complex mutual-inductance matrix */
double Lmc_phase[4][4]; /* Lmc phase errors from Lm */
double Lmmeas[4][4]; /* real measured mutual-inductance matrix */
double Lmseed[4][4]; /* Lm matrix calculated from seed P&O */
double Lmfit1[4][4],Lmfit2[4][4],Lmfit3[4][4];
Quaternion PPmech[12],Pseed,Oseed;
Quaternion Pfit1[1],Pfit2[1],Pfit3[1],Ofit1[1],Ofit2[1],Ofit3[1];
double Lm(int Step,int tx_coil_no,int rx_coil_no,
Quaternion qpos,Quaternion qori);
int Step;
double GOF(double Lm44meas[4][4],double Lm44model[4][4]);
double GOF1,GOF2,GOF3;

/* iw = i * 2pi * frequency_in_hertz */
twopi = 8.0 * atan(1.0);
iw[0].i = twopi * 11359.321;
iw[1].i = twopi * 11664.497;
iw[2].i = twopi * 11969.672;
iw[3].i = twopi * 12274.848;
iw[0].r = 0.0;
iw[1].r = 0.0;
iw[2].r = 0.0;
iw[3].r = 0.0;

Step = 0;

printf(
"curre135.cpp rev16, input data point number 1 through 11\n");

/* save the point number */
strcpy(datptnumstr,argv[1]);
datptnumint = atoi(datptnumstr);
/* print the point number */
printf("This run is for data point number ");
puts(datptnumstr);

/* generate the receiver-board SOP data filename */
strcpy(datfilnamstr,"tetra");
strcat(datfilnamstr,datptnumstr);
strcat(datfilnamstr,".dat");
/* open the file */
if((inpdat = fopen(datfilnamstr,"r"))==NULL)
    printf("file open failed\n");
```

```
else{

  /* read the input file */
  strcpy(inpstr,"this string does not start with a digit\n");

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  R1X[0][0] = 25.0;

  /* read R1X block of data */
  for(i=0;i<256;i++){
    /* parse the input record */
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
     %lf%lf%lf%lf%lf%lf",
     &R1XCQ[i],&R1X[i][0],&R1X[i][1],&R1X[i][2],&R1X[i][3],
     &R1X[i][4],
     &R1X[i][5],&R1X[i][6],&R1X[i][7],&R1X[i][8],&R1X[i][9],
     &R1X[i][10],&R1X[i][11],&R1X[i][12],&R1X[i][13],&R1X[i][14],
     &R1X[i][15],&R1X[i][16],&R1X[i][17],&R1X[i][18],&R1X[i][19],
     &R1X[i][20],&R1X[i][21],&R1X[i][22],&R1X[i][23],&R1X[i][24],
     &R1X[i][25]);

    /* get the next input record */
    fgets(inpstr,1000,inpdat);
  }

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  /* read R1Y block of data */
  for(i=0;i<256;i++){
    /* parse the input record */
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
     %lf%lf%lf%lf%lf%lf%lf",
     &R1YCQ[i],&R1Y[i][0],&R1Y[i][1],&R1Y[i][2],&R1Y[i][3],
     &R1Y[i][4],
     &R1Y[i][5],&R1Y[i][6],&R1Y[i][7],&R1Y[i][8],&R1Y[i][9],
     &R1Y[i][10],&R1Y[i][11],&R1Y[i][12],&R1Y[i][13],&R1Y[i][14],
     &R1Y[i][15],&R1Y[i][16],&R1Y[i][17],&R1Y[i][18],&R1Y[i][19],
     &R1Y[i][20],&R1Y[i][21],&R1Y[i][22],&R1Y[i][23],&R1Y[i][24],
     &R1Y[i][25]);

    /* get the next input record */
    fgets(inpstr,1000,inpdat);
  }

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }
```

```
  /* read R1Z block of data */
  for(i=0;i<256;i++){
    /* parse the input record */
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
     %lf%lf%lf%lf%lf%lf%lf",
     &R1ZCQ[i],&R1Z[i][0],&R1Z[i][1],&R1Z[i][2],&R1Z[i][3],
     &R1Z[i][4],
     &R1Z[i][5],&R1Z[i][6],&R1Z[i][7],&R1Z[i][8],&R1Z[i][9],
     &R1Z[i][10],&R1Z[i][11],&R1Z[i][12],&R1Z[i][13],&R1Z[i][14],
     &R1Z[i][15],&R1Z[i][16],&R1Z[i][17],&R1Z[i][18],&R1Z[i][19],
     &R1Z[i][20],&R1Z[i][21],&R1Z[i][22],&R1Z[i][23],&R1Z[i][24],
     &R1Z[i][25]);

    /* get the next input record */
    fgets(inpstr,1000,inpdat);
  }

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  /* read CUR block of data */
  for(i=0;i<256;i++){
    /* parse the input record */
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
     %lf%lf%lf%lf%lf%lf%lf",
     &CURCQ[i],&CUR[i][0],&CUR[i][1],&CUR[i][2],&CUR[i][3],
     &CUR[i][4],
     &CUR[i][5],&CUR[i][6],&CUR[i][7],&CUR[i][8],&CUR[i][9],
     &CUR[i][10],&CUR[i][11],&CUR[i][12],&CUR[i][13],&CUR[i][14],
     &CUR[i][15],&CUR[i][16],&CUR[i][17],&CUR[i][18],&CUR[i][19],
     &CUR[i][20],&CUR[i][21],&CUR[i][22],&CUR[i][23],&CUR[i][24],
&CUR[i][25]);

    /* get the next input record */
    fgets(inpstr,1000,inpdat);
  }

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  /* read R2X block of data
      (second R1X, after current, in tetra%d.dat) */
  for(i=0;i<256;i++){
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
     %lf%lf%lf%lf%lf%lf%lf",
     &R2XCQ[i],&R2X[i][0],&R2X[i][1],&R2X[i][2],&R2X[i][3],
     &R2X[i][4],
     &R2X[i][5],&R2X[i][6],&R2X[i][7],&R2X[i][8],&R2X[i][9],
     &R2X[i][10],&R2X[i][11],&R2X[i][12],&R2X[i][13],&R2X[i][14],
```

```
    &R2X[i][15],&R2X[i][16],&R2X[i][17],&R2X[i][18],&R2X[i][19],
    &R2X[i][20],&R2X[i][21],&R2X[i][22],&R2X[i][23],&R2X[i][24],
    &R2X[i][25]);

   /* get the next input record */
   fgets(inpstr,1000,inpdat);
 }

 /* we now have the file data in the arrays */

 /* The transmitter[0,1,2,3] currents are
      measured when CQ=[2,3,4,5], excluding the first measurement
      of each current:

     index    CUR[index][frq]
    -------  --------------------
     17-31    transmitter coil 0
     33-47    transmitter coil 1
     49-63    transmitter coil 2
     65-79    transmitter coil 3

    The transmitter[0,1,2,3] driver
      frequencies are SOP numbers:

     frq    driver frequency
    -----  ----------------------
      2     transmitter driver 0 I
      3     transmitter driver 0 Q
      4     transmitter driver 1 I
      5     transmitter driver 1 Q
      6     transmitter driver 2 I
      7     transmitter driver 2 Q
      8     transmitter driver 3 I
      9     transmitter driver 3 Q

*/

 /* Iiw[transmitter driver][frequency] */

 for(i=17;i<31;i++){
   Iiw[0][0].r = Iiw[0][0].r + CUR[i][2];
   Iiw[0][0].i = Iiw[0][0].i + CUR[i][3];
   Iiw[0][1].r = Iiw[0][1].r + CUR[i][4];
   Iiw[0][1].i = Iiw[0][1].i + CUR[i][5];
   Iiw[0][2].r = Iiw[0][2].r + CUR[i][6];
   Iiw[0][2].i = Iiw[0][2].i + CUR[i][7];
   Iiw[0][3].r = Iiw[0][3].r + CUR[i][8];
   Iiw[0][3].i = Iiw[0][3].i + CUR[i][9];
 }
 for(i=33;i<47;i++){
   Iiw[1][0].r = Iiw[1][0].r + CUR[i][2];
   Iiw[1][0].i = Iiw[1][0].i + CUR[i][3];
   Iiw[1][1].r = Iiw[1][1].r + CUR[i][4];
   Iiw[1][1].i = Iiw[1][1].i + CUR[i][5];
   Iiw[1][2].r = Iiw[1][2].r + CUR[i][6];
   Iiw[1][2].i = Iiw[1][2].i + CUR[i][7];
   Iiw[1][3].r = Iiw[1][3].r + CUR[i][8];
```

```
  Iiw[1][3].i = Iiw[1][3].i + CUR[i][9];
}
for(i=49;i<63;i++){
  Iiw[2][0].r = Iiw[2][0].r + CUR[i][2];
  Iiw[2][0].i = Iiw[2][0].i + CUR[i][3];
  Iiw[2][1].r = Iiw[2][1].r + CUR[i][4];
  Iiw[2][1].i = Iiw[2][1].i + CUR[i][5];
  Iiw[2][2].r = Iiw[2][2].r + CUR[i][6];
  Iiw[2][2].i = Iiw[2][2].i + CUR[i][7];
  Iiw[2][3].r = Iiw[2][3].r + CUR[i][8];
  Iiw[2][3].i = Iiw[2][3].i + CUR[i][9];
}
for(i=65;i<79;i++){
  Iiw[3][0].r = Iiw[3][0].r + CUR[i][2];
  Iiw[3][0].i = Iiw[3][0].i + CUR[i][3];
  Iiw[3][1].r = Iiw[3][1].r + CUR[i][4];
  Iiw[3][1].i = Iiw[3][1].i + CUR[i][5];
  Iiw[3][2].r = Iiw[3][2].r + CUR[i][6];
  Iiw[3][2].i = Iiw[3][2].i + CUR[i][7];
  Iiw[3][3].r = Iiw[3][3].r + CUR[i][8];
  Iiw[3][3].i = Iiw[3][3].i + CUR[i][9];
}
for(i=0;i<4;i++){
  for(j=0;j<4;j++){
    Iiw[i][j] = ((1.0/14.0) * Iiw[i][j]) * (-1.0*iw[j]);
  }
}

/* The (-1.0*iw[j]) factor aboves converts the currents into
 their time derivatives, based on e^(-iwt) time dependence. */

/* start of transmitter cable gain data
   This data is for Rcoil = 28 ohms and Lcoil = 410 uH,
   using 1 T section (2 T sections gives the same data). */
Txc_gain[0].r =  1.000115e+00;
Txc_gain[0].i = -1.1e-04;
Txc_gain[1].r =  1.000121e+00;
Txc_gain[1].i = -1.1e-04;
Txc_gain[2].r =  1.000128e+00;
Txc_gain[2].i = -1.2e-04;
Txc_gain[3].r =  1.000134e+00;
Txc_gain[3].i = -1.2e-04;

/* Multiply each complex current by transmitter cable gain */
for(i=0;i<4;i++){
  for(j=0;j<4;j++){
    Iiw[i][j] = Txc_gain[j] * Iiw[i][j];
  }
}

/* invert the current matrix */

Cmatinv4x4(Iiw, Iiw_inverse);

/* average 239 measurements for each receiver SOP */
/* calculate S[r][f] */
for(i=0;i<4;i++){
```

```
  for(j=0;j<4;j++){
    S[i][j].r = 0.0;
    S[i][j].i = 0.0;
  }
}
for(i=1;i<240;i++){
S[0][0].r += R2X[i][2];
S[0][0].i += R2X[i][3];
S[1][0].r += R1X[i][2];
S[1][0].i += R1X[i][3];
S[2][0].r += R1Y[i][2];
S[2][0].i += R1Y[i][3];
S[3][0].r += R1Z[i][2];
S[3][0].i += R1Z[i][3];

S[0][1].r += R2X[i][4];
S[0][1].i += R2X[i][5];
S[1][1].r += R1X[i][4];
S[1][1].i += R1X[i][5];
S[2][1].r += R1Y[i][4];
S[2][1].i += R1Y[i][5];
S[3][1].r += R1Z[i][4];
S[3][1].i += R1Z[i][5];

S[0][2].r += R2X[i][6];
S[0][2].i += R2X[i][7];
S[1][2].r += R1X[i][6];
S[1][2].i += R1X[i][7];
S[2][2].r += R1Y[i][6];
S[2][2].i += R1Y[i][7];
S[3][2].r += R1Z[i][6];
S[3][2].i += R1Z[i][7];

S[0][3].r += R2X[i][8];
S[0][3].i += R2X[i][9];
S[1][3].r += R1X[i][8];
S[1][3].i += R1X[i][9];
S[2][3].r += R1Y[i][8];
S[2][3].i += R1Y[i][9];
S[3][3].r += R1Z[i][8];
S[3][3].i += R1Z[i][9];
}
for(i=0;i<4;i++){
  for(j=0;j<4;j++){
    S[i][j] = (1.0/239.0) * S[i][j];
  }
}

/* frequency-dependent receiver cable attenuation */
Rxc_atten[0].r =  1.001140e+00;
Rxc_atten[0].i =  1.793272e-03;
Rxc_atten[1].r =  1.001126e+00;
Rxc_atten[1].i =  1.839486e-03;
Rxc_atten[2].r =  1.001109e+00;
Rxc_atten[2].i =  1.892287e-03;
Rxc_atten[3].r =  1.001093e+00;
Rxc_atten[3].i =  1.941929e-03;
```

```
  /* S[r][f] = S[r][f] Rxc_atten[f] */
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      S[i][j] = Rxc_atten[j] * S[i][j];
    }
  }

  /* SIiw_inverse[r][t] = S[r][f] Iiw_inverse[f][t] */
  Cmatmul4x4(S,Iiw_inverse,SIiw_inverse);

  /* channel-dependent receiver channel gains */
  Gain[0].r =  0.14938762;
  Gain[0].i = -0.000310136;
  Gain[1].r =  0.14950998;
  Gain[1].i = -0.0000908787;
  Gain[2].r =  0.15066835;
  Gain[2].i = -0.0001819950;
  Gain[3].r =  0.14910116;
  Gain[3].i = -0.00008113399;

  /* process to get complex measured mutual-inductance matrix */
  /* Lmt[r][t] = Gain[r] SIiw_inverse[r][t]
     Lmc[t][r] = Lmt[r][t].transpose
     in these loops, r=i and t=j, so Lmt[r][t] = Lmt[i][j] */
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      Lmc[j][i] = Gain[i] * SIiw_inverse[i][j];
    }
  }

  /* calculate phase errors in complex mutual-inductance matrix*/
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      Lmc_phase[i][j] = (360.0/twopi)
            * atan(Lmc[i][j].i/Lmc[i][j].r);
    }
  }

  printf("Phase of complex mutual-inductance matrix
Lmc_phase in degrees =\n");
  Rmatprint4x4(Lmc_phase);

  /* extract real mutual-inductance matrix */
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      Lmmeas[i][j] = Lmc[i][j].r;
    }
  }

  printf("Real measured mutual-inductance matrix
Lmmeas in henries =\n");
  Rmatprint4x4(Lmmeas);

  /* Mechanical starting orientation for seed */
  Oseed.i = 0.0;
  Oseed.j = 0.0;
```

```
Oseed.k = sin(twopi * 5.0/24.0);
Oseed.r = sqrt(1.0-Oseed.k*Oseed.k);

/* Mechanical starting positions in meters for seed */
PPmech[0].r = 0.0;
PPmech[1].r = 0.0;
PPmech[2].r = 0.0;
PPmech[3].r = 0.0;
PPmech[4].r = 0.0;
PPmech[5].r = 0.0;
PPmech[6].r = 0.0;
PPmech[7].r = 0.0;
PPmech[8].r = 0.0;
PPmech[9].r = 0.0;
PPmech[10].r = 0.0;
PPmech[11].r = 0.0;
PPmech[0].i = 0.0408;
PPmech[1].i = 0.1830;
PPmech[3].i = 0.3252;
PPmech[2].i = 0.4675;
PPmech[5].i = 0.4675;
PPmech[6].i = 0.4675;
PPmech[8].i = 0.4675;
PPmech[9].i = 0.3252;
PPmech[10].i = 0.1830;
PPmech[11].i = 0.0408;
PPmech[7].i = 0.0408;
PPmech[4].i = 0.0408;
PPmech[0].j = 0.0513;
PPmech[1].j = 0.0513;
PPmech[3].j = 0.0513;
PPmech[2].j = 0.0513;
PPmech[5].j = 0.1935;
PPmech[6].j = 0.3358;
PPmech[8].j = 0.4780;
PPmech[9].j = 0.4780;
PPmech[10].j = 0.4780;
PPmech[11].j = 0.4780;
PPmech[7].j = 0.3358;
PPmech[4].j = 0.1935;
PPmech[0].k = 0.0675;
PPmech[1].k = 0.0675;
PPmech[2].k = 0.0675;
PPmech[3].k = 0.0675;
PPmech[4].k = 0.0675;
PPmech[5].k = 0.0675;
PPmech[6].k = 0.0675;
PPmech[7].k = 0.0675;
PPmech[8].k = 0.0675;
PPmech[9].k = 0.0675;
PPmech[10].k = 0.0675;
PPmech[11].k = 0.0675;
Pseed = PPmech[datptnumint];

/* Transmitter rotation tolerances:
   0.04 radian about Z axis
   0.002 radian about X,Y axes
```

```
     Receiver rotation tolerances:
     0.04 radian about Z axis
     0.002 radian about X,Y axes

     Receiver position tolerances:
     0.0003 M in X,Y,Z axes
     + effect of transmitter rotation tolerances */

  printf("Mechanical seed position for point %d
in meters =\n",datptnumint);
  Quatprint(Pseed);
  printf("Mechanical seed orientation =\n");
  Quatprint(Oseed);

  /* calculate P&O seed mutual-inductance matrix */
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      Lmseed[i][j] = Lm(Step,i,j,Pseed,Oseed);
    }
  }

  printf("Mechanical seed position mutual-inductance matrix
in henries =\n");
  Rmatprint4x4(Lmseed);

  /* end of code in curre126.cpp */

  least_squares_fit(Step,Pseed,Oseed,Lmmeas,Pfit1,Ofit1);

  printf("First-fit position in meters =\n");
  Quatprint(Pfit1[0]);
  printf("First-fit orientation =\n");
  Quatprint(Ofit1[0]);

  /* calculate first-fit mutual-inductance matrix */
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      Lmfit1[i][j] = Lm(Step,i,j,Pfit1[0],Ofit1[0]);
    }
  }

  printf("First-fit mutual-inductance matrix in henries =\n");
  Rmatprint4x4(Lmfit1);

  GOF1 =  GOF(Lmmeas, Lmfit1);
  printf("First goodness-of-fit = %11.4e\n", GOF1);

  least_squares_fit(Step,Pfit1[0],Ofit1[0],Lmmeas,Pfit2,Ofit2);

  printf("Second-fit position in meters =\n");
  Quatprint(Pfit2[0]);
  printf("Second-fit orientation =\n");
  Quatprint(Ofit2[0]);

  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
```

```
      Lmfit2[i][j] = Lm(Step,i,j,Pfit2[0],Ofit2[0]);
    }
  }

  printf("Second-fit mutual-inductance matrix in henries =\n");
  Rmatprint4x4(Lmfit2);

  GOF2 =  GOF(Lmmeas, Lmfit2);
  printf("Second goodness-of-fit = %11.4e\n", GOF2);

  least_squares_fit(Step,Pfit2[0],Ofit2[0],Lmmeas,Pfit3,Ofit3);

  printf("Third-fit position in meters =\n");
  Quatprint(Pfit3[0]);
  printf("Third-fit orientation =\n");
  Quatprint(Ofit3[0]);

  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      Lmfit3[i][j] = Lm(Step,i,j,Pfit3[0],Ofit3[0]);
    }
  }

  printf("Third-fit mutual-inductance matrix in henries =\n");
  Rmatprint4x4(Lmfit3);

  GOF3 =  GOF(Lmmeas, Lmfit3);
  printf("Third goodness-of-fit = %11.4e\n", GOF3);


}
}
```

Appendix O

Lm matrix, P&O, and GOF results

The results of running the code in the preceding appendix appear on the following pages.

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 1
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+7.4221e-02 +1.4978e-01 +1.7317e-01 +2.4620e-01
+7.6439e-02 +1.4593e-01 +1.5709e-01 +2.0958e-01
+9.2754e-02 +1.6044e-01 +1.9667e-01 +2.2581e-01
+6.5646e-02 +1.5044e-01 +1.6916e-01 +2.6168e-01
Real measured mutual-inductance matrix Lmmeas in henries =
-3.4003e-07 -2.5099e-07 +7.2242e-07 -1.2769e-07
+2.6964e-07 -5.5503e-08 +2.7528e-07 -4.6249e-07
-2.1502e-07 +7.8101e-07 -1.3580e-06 +7.8160e-07
+2.9133e-07 -4.2162e-07 +2.2055e-07 -1.1635e-07
Mechanical seed position for point 1 in meters =
+0.0000e+00 +1.8300e-01 +5.1300e-02 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-3.4239e-07 -2.4963e-07 +7.2311e-07 -1.2705e-07
+2.6566e-07 -5.6364e-08 +2.7717e-07 -4.5927e-07
-2.0726e-07 +7.8102e-07 -1.3677e-06 +7.7906e-07
+2.8823e-07 -4.2089e-07 +2.2674e-07 -1.1638e-07
First-fit position in meters =
+0.0000e+00 +1.8310e-01 +5.0173e-02 +6.7626e-02
First-fit orientation =
+2.6471e-01 +2.1046e-03 -2.6915e-04 +9.6432e-01
First-fit mutual-inductance matrix in henries =
-3.4045e-07 -2.5187e-07 +7.2156e-07 -1.2655e-07
+2.6893e-07 -5.4928e-08 +2.7483e-07 -4.6126e-07
-2.1646e-07 +7.8205e-07 -1.3585e-06 +7.8025e-07
+2.9159e-07 -4.2148e-07 +2.2253e-07 -1.1589e-07
First goodness-of-fit =  3.3051e-06
Second-fit position in meters =
+0.0000e+00 +1.8309e-01 +5.0173e-02 +6.7626e-02
Second-fit orientation =
+2.6471e-01 +2.1056e-03 -2.9471e-04 +9.6433e-01
Second-fit mutual-inductance matrix in henries =
-3.4048e-07 -2.5188e-07 +7.2163e-07 -1.2657e-07
+2.6896e-07 -5.4942e-08 +2.7484e-07 -4.6129e-07
-2.1645e-07 +7.8211e-07 -1.3586e-06 +7.8030e-07
+2.9158e-07 -4.2151e-07 +2.2256e-07 -1.1589e-07
Second goodness-of-fit =  3.2982e-06
Third-fit position in meters =
+0.0000e+00 +1.8309e-01 +5.0173e-02 +6.7626e-02
Third-fit orientation =
+2.6471e-01 +2.1056e-03 -2.9468e-04 +9.6433e-01
Third-fit mutual-inductance matrix in henries =
-3.4048e-07 -2.5188e-07 +7.2163e-07 -1.2657e-07
+2.6896e-07 -5.4942e-08 +2.7484e-07 -4.6129e-07
-2.1645e-07 +7.8211e-07 -1.3586e-06 +7.8030e-07
+2.9158e-07 -4.2151e-07 +2.2256e-07 -1.1589e-07
Third goodness-of-fit =  3.2982e-06
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 2
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
-2.2947e-02 +7.5801e-02 +1.1080e-01 -2.0259e+00
-5.5996e-01 -3.3146e-02 +2.3228e-01 +1.4868e-01
-9.1022e-01 +2.4597e-01 +2.6860e-01 +2.2294e-01
-5.6760e-01 +1.4761e-01 +4.4244e-01 +3.1536e-01
Real measured mutual-inductance matrix Lmmeas in henries =
-3.7373e-08 -2.4934e-09 +3.9777e-08 +1.0627e-09
+1.4715e-08 +1.2881e-08 +7.8015e-09 -3.5144e-08
+5.7333e-09 +3.3710e-08 -8.9640e-08 +4.8742e-08
+1.7629e-08 -4.4274e-08 +4.0535e-08 -1.3973e-08
Mechanical seed position for point 2 in meters =
+0.0000e+00 +4.6750e-01 +5.1300e-02 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-3.7504e-08 -2.4837e-09 +3.9837e-08 +1.0054e-09
+1.4421e-08 +1.3041e-08 +7.8898e-09 -3.5097e-08
+6.6520e-09 +3.3619e-08 -9.0881e-08 +4.9242e-08
+1.7047e-08 -4.4264e-08 +4.1686e-08 -1.4280e-08
First-fit position in meters =
+0.0000e+00 +4.6890e-01 +5.0239e-02 +6.5000e-02
First-fit orientation =
+2.6397e-01 +9.8500e-03 +2.3764e-03 +9.6448e-01
First-fit mutual-inductance matrix in henries =
-3.7254e-08 -2.5734e-09 +3.9619e-08 +1.0333e-09
+1.4614e-08 +1.2800e-08 +7.7842e-09 -3.4937e-08
+5.6293e-09 +3.3921e-08 -8.9639e-08 +4.8784e-08
+1.7617e-08 -4.4237e-08 +4.0793e-08 -1.4023e-08
First goodness-of-fit =  1.1491e-05
Second-fit position in meters =
+0.0000e+00 +4.6889e-01 +5.0197e-02 +6.4947e-02
Second-fit orientation =
+2.6408e-01 +9.9509e-03 +2.4131e-03 +9.6445e-01
Second-fit mutual-inductance matrix in henries =
-3.7261e-08 -2.5733e-09 +3.9621e-08 +1.0376e-09
+1.4617e-08 +1.2801e-08 +7.7863e-09 -3.4942e-08
+5.6270e-09 +3.3929e-08 -8.9646e-08 +4.8785e-08
+1.7622e-08 -4.4246e-08 +4.0796e-08 -1.4023e-08
Second goodness-of-fit =  1.1474e-05
Third-fit position in meters =
+0.0000e+00 +4.6889e-01 +5.0196e-02 +6.4946e-02
Third-fit orientation =
+2.6408e-01 +9.9521e-03 +2.4132e-03 +9.6445e-01
Third-fit mutual-inductance matrix in henries =
-3.7261e-08 -2.5733e-09 +3.9621e-08 +1.0377e-09
+1.4617e-08 +1.2801e-08 +7.7862e-09 -3.4942e-08
+5.6270e-09 +3.3929e-08 -8.9646e-08 +4.8785e-08
+1.7623e-08 -4.4246e-08 +4.0796e-08 -1.4023e-08
Third goodness-of-fit =  1.1474e-05
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 3
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+7.3911e-02 +1.7295e-01 +1.4252e-01 +5.6659e-01
+5.8703e-02 +1.4432e-02 +1.6230e-01 +1.9551e-01
-1.2798e-01 +1.7440e-01 +2.2402e-01 +2.1174e-01
+5.7692e-02 +1.4480e-01 +2.2056e-01 +2.5476e-01
Real measured mutual-inductance matrix Lmmeas in henries =
-9.9258e-08 -2.5205e-08 +1.3744e-07 -8.7933e-09
+4.5113e-08 +1.9990e-08 +3.6257e-08 -9.9301e-08
+4.2223e-09 +1.2157e-07 -2.8031e-07 +1.4778e-07
+5.3237e-08 -1.1510e-07 +9.6039e-08 -3.4799e-08
Mechanical seed position for point 3 in meters =
+0.0000e+00 +3.2520e-01 +5.1300e-02 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-9.9755e-08 -2.5059e-08 +1.3746e-07 -8.7019e-09
+4.4311e-08 +2.0076e-08 +3.6850e-08 -9.9206e-08
+6.1922e-09 +1.2166e-07 -2.8301e-07 +1.4845e-07
+5.2167e-08 -1.1515e-07 +9.8146e-08 -3.5158e-08
First-fit position in meters =
+0.0000e+00 +3.2571e-01 +5.0600e-02 +6.7037e-02
First-fit orientation =
+2.6225e-01 +5.1490e-03 +8.9564e-04 +9.6498e-01
First-fit mutual-inductance matrix in henries =
-9.9211e-08 -2.5345e-08 +1.3712e-07 -8.7494e-09
+4.4884e-08 +1.9916e-08 +3.6194e-08 -9.8960e-08
+3.9971e-09 +1.2197e-07 -2.8027e-07 +1.4786e-07
+5.3195e-08 -1.1503e-07 +9.6519e-08 -3.4790e-08
First goodness-of-fit =  4.0953e-06
Second-fit position in meters =
+0.0000e+00 +3.2571e-01 +5.0585e-02 +6.7028e-02
Second-fit orientation =
+2.6230e-01 +5.1746e-03 +8.9262e-04 +9.6497e-01
Second-fit mutual-inductance matrix in henries =
-9.9217e-08 -2.5346e-08 +1.3712e-07 -8.7473e-09
+4.4889e-08 +1.9917e-08 +3.6193e-08 -9.8965e-08
+3.9951e-09 +1.2198e-07 -2.8028e-07 +1.4786e-07
+5.3198e-08 -1.1504e-07 +9.6523e-08 -3.4790e-08
Second goodness-of-fit =  4.0937e-06
Third-fit position in meters =
+0.0000e+00 +3.2571e-01 +5.0585e-02 +6.7028e-02
Third-fit orientation =
+2.6230e-01 +5.1748e-03 +8.9260e-04 +9.6497e-01
Third-fit mutual-inductance matrix in henries =
-9.9217e-08 -2.5346e-08 +1.3712e-07 -8.7473e-09
+4.4889e-08 +1.9917e-08 +3.6193e-08 -9.8965e-08
+3.9951e-09 +1.2198e-07 -2.8028e-07 +1.4786e-07
+5.3198e-08 -1.1504e-07 +9.6523e-08 -3.4790e-08
Third goodness-of-fit =  4.0937e-06
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 4
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+9.3944e-02 -1.9594e-02 +1.0527e-01 +2.0030e-01
+8.2863e-02 -3.1715e-02 +2.3135e-01 +1.6942e-01
+4.2508e-01 -2.7239e-02 +2.0721e-01 +2.2336e-01
+1.3422e-01 -8.9407e-03 +2.0313e-01 +2.1494e-01
Real measured mutual-inductance matrix Lmmeas in henries =
-3.2084e-07 -2.6076e-07 +7.9162e-08 +4.6450e-07
+2.5992e-07 -4.0649e-07 -7.2280e-08 +1.6812e-07
-6.7644e-09 +4.2979e-07 +5.5855e-07 -9.7488e-07
+1.2629e-07 +2.2856e-07 -5.6531e-07 +2.8118e-07
Mechanical seed position for point 4 in meters =
+0.0000e+00 +4.0800e-02 +1.9350e-01 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-3.1959e-07 -2.5710e-07 +8.2881e-08 +4.5446e-07
+2.5725e-07 -4.0297e-07 -6.6211e-08 +1.6391e-07
-8.2940e-09 +4.2462e-07 +5.4881e-07 -9.5825e-07
+1.2653e-07 +2.2600e-07 -5.6328e-07 +2.8020e-07
First-fit position in meters =
+0.0000e+00 +4.1000e-02 +1.9252e-01 +6.7788e-02
First-fit orientation =
+2.6340e-01 +1.6941e-03 -1.2512e-03 +9.6468e-01
First-fit mutual-inductance matrix in henries =
-3.2149e-07 -2.6064e-07 +7.9181e-08 +4.6346e-07
+2.6000e-07 -4.0609e-07 -7.1404e-08 +1.6926e-07
-7.8736e-09 +4.2894e-07 +5.5931e-07 -9.7535e-07
+1.2612e-07 +2.2841e-07 -5.6479e-07 +2.8176e-07
First goodness-of-fit =  2.7791e-06
Second-fit position in meters =
+0.0000e+00 +4.0993e-02 +1.9253e-01 +6.7779e-02
Second-fit orientation =
+2.6334e-01 +1.6817e-03 -1.2189e-03 +9.6470e-01
Second-fit mutual-inductance matrix in henries =
-3.2150e-07 -2.6061e-07 +7.9222e-08 +4.6340e-07
+2.5999e-07 -4.0608e-07 -7.1371e-08 +1.6922e-07
-7.8600e-09 +4.2890e-07 +5.5925e-07 -9.7525e-07
+1.2612e-07 +2.2841e-07 -5.6481e-07 +2.8176e-07
Second goodness-of-fit =  2.7698e-06
Third-fit position in meters =
+0.0000e+00 +4.0993e-02 +1.9253e-01 +6.7779e-02
Third-fit orientation =
+2.6334e-01 +1.6817e-03 -1.2189e-03 +9.6470e-01
Third-fit mutual-inductance matrix in henries =
-3.2150e-07 -2.6061e-07 +7.9222e-08 +4.6340e-07
+2.5999e-07 -4.0608e-07 -7.1371e-08 +1.6922e-07
-7.8600e-09 +4.2890e-07 +5.5925e-07 -9.7525e-07
+1.2612e-07 +2.2841e-07 -5.6481e-07 +2.8176e-07
Third goodness-of-fit =  2.7698e-06
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 5
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+5.1205e-02 +1.1938e-01 +1.0331e-03 -5.2743e-01
+3.6392e-02 +3.1328e-01 +1.8116e-01 +1.9131e-01
+1.7997e-01 +1.9395e-01 +2.9840e-01 +6.4350e-02
+2.9985e-02 +1.4323e-01 +4.8466e-01 +2.6863e-01
Real measured mutual-inductance matrix Lmmeas in henries =
-3.0467e-08 -3.7183e-09 +2.8996e-08 +5.7837e-09
+1.3076e-08 -7.4948e-09 +2.5778e-08 -3.1007e-08
+4.5631e-09 +4.7235e-08 -7.1155e-08 +1.8885e-08
+1.3346e-08 -3.5483e-08 +1.4898e-08 +6.6407e-09
Mechanical seed position for point 5 in meters =
+0.0000e+00 +4.6750e-01 +1.9350e-01 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-3.0472e-08 -3.8273e-09 +2.8848e-08 +5.8918e-09
+1.2471e-08 -7.4394e-09 +2.6305e-08 -3.0970e-08
+5.2803e-09 +4.7340e-08 -7.2179e-08 +1.9162e-08
+1.3089e-08 -3.5459e-08 +1.5578e-08 +6.3455e-09
First-fit position in meters =
+0.0000e+00 +4.6921e-01 +1.9163e-01 +6.5033e-02
First-fit orientation =
+2.6544e-01 +1.0552e-02 +1.3371e-03 +9.6407e-01
First-fit mutual-inductance matrix in henries =
-3.0349e-08 -3.7986e-09 +2.8836e-08 +5.7349e-09
+1.2992e-08 -7.6051e-09 +2.5773e-08 -3.0812e-08
+4.4461e-09 +4.7460e-08 -7.1161e-08 +1.8917e-08
+1.3268e-08 -3.5450e-08 +1.5125e-08 +6.5904e-09
First goodness-of-fit =  1.7794e-05
Second-fit position in meters =
+0.0000e+00 +4.6921e-01 +1.9158e-01 +6.4951e-02
Second-fit orientation =
+2.6552e-01 +1.0653e-02 +1.4280e-03 +9.6404e-01
Second-fit mutual-inductance matrix in henries =
-3.0356e-08 -3.7960e-09 +2.8836e-08 +5.7395e-09
+1.2995e-08 -7.6070e-09 +2.5777e-08 -3.0817e-08
+4.4456e-09 +4.7468e-08 -7.1168e-08 +1.8918e-08
+1.3273e-08 -3.5459e-08 +1.5128e-08 +6.5904e-09
Second goodness-of-fit =  1.7766e-05
Third-fit position in meters =
+0.0000e+00 +4.6921e-01 +1.9158e-01 +6.4950e-02
Third-fit orientation =
+2.6553e-01 +1.0655e-02 +1.4287e-03 +9.6404e-01
Third-fit mutual-inductance matrix in henries =
-3.0356e-08 -3.7959e-09 +2.8836e-08 +5.7395e-09
+1.2995e-08 -7.6070e-09 +2.5777e-08 -3.0817e-08
+4.4456e-09 +4.7468e-08 -7.1168e-08 +1.8918e-08
+1.3273e-08 -3.5459e-08 +1.5128e-08 +6.5904e-09
Third goodness-of-fit =  1.7766e-05
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 6
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+3.3861e-02 +4.9281e-01 -7.8720e-02 -2.2313e-01
-4.3638e-02 +3.7634e-02 +7.9524e-02 +3.1751e-01
+3.7564e-01 +1.7229e-01 +3.5207e-01 +8.0569e-01
-1.9661e-01 +1.2618e-01 -7.3177e-02 +1.1408e-02
Real measured mutual-inductance matrix Lmmeas in henries =
-2.1093e-08 -2.4583e-09 +1.6978e-08 +7.0162e-09
+9.2076e-09 -1.5058e-08 +2.2510e-08 -1.6679e-08
+3.8584e-09 +3.7821e-08 -3.7578e-08 -3.8059e-09
+8.3903e-09 -1.9981e-08 -2.3344e-09 +1.3406e-08
Mechanical seed position for point 6 in meters =
+0.0000e+00 +4.6750e-01 +3.3580e-01 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-2.1045e-08 -2.5038e-09 +1.6613e-08 +7.0876e-09
+8.7029e-09 -1.5032e-08 +2.2948e-08 -1.6615e-08
+4.1560e-09 +3.7878e-08 -3.8024e-08 -3.7216e-09
+8.3900e-09 -1.9906e-08 -2.0318e-09 +1.3110e-08
First-fit position in meters =
+0.0000e+00 +4.6899e-01 +3.3396e-01 +6.6025e-02
First-fit orientation =
+2.6432e-01 +9.9657e-03 +3.6508e-04 +9.6438e-01
First-fit mutual-inductance matrix in henries =
-2.1021e-08 -2.5207e-09 +1.6805e-08 +6.8835e-09
+9.1506e-09 -1.5151e-08 +2.2523e-08 -1.6533e-08
+3.7499e-09 +3.8014e-08 -3.7635e-08 -3.8175e-09
+8.3211e-09 -1.9906e-08 -2.1862e-09 +1.3330e-08
First goodness-of-fit =  3.3375e-05
Second-fit position in meters =
+0.0000e+00 +4.6898e-01 +3.3394e-01 +6.5941e-02
Second-fit orientation =
+2.6433e-01 +1.0004e-02 +4.8209e-04 +9.6438e-01
Second-fit mutual-inductance matrix in henries =
-2.1025e-08 -2.5182e-09 +1.6803e-08 +6.8863e-09
+9.1512e-09 -1.5153e-08 +2.2527e-08 -1.6535e-08
+3.7501e-09 +3.8018e-08 -3.7638e-08 -3.8176e-09
+8.3244e-09 -1.9911e-08 -2.1852e-09 +1.3330e-08
Second goodness-of-fit =  3.3353e-05
Third-fit position in meters =
+0.0000e+00 +4.6898e-01 +3.3394e-01 +6.5940e-02
Third-fit orientation =
+2.6433e-01 +1.0005e-02 +4.8334e-04 +9.6438e-01
Third-fit mutual-inductance matrix in henries =
-2.1025e-08 -2.5181e-09 +1.6803e-08 +6.8863e-09
+9.1512e-09 -1.5153e-08 +2.2527e-08 -1.6535e-08
+3.7500e-09 +3.8018e-08 -3.7638e-08 -3.8176e-09
+8.3244e-09 -1.9911e-08 -2.1852e-09 +1.3330e-08
Third goodness-of-fit =  3.3353e-05
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 7
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+6.2519e-02 -7.4648e-02 -2.5479e-02 +1.7619e-01
+5.8820e-02 -3.9797e-02 +3.2751e-01 +1.9055e-01
+1.9586e-01 -1.6807e-02 +1.7482e-01 +2.3359e-01
+2.1029e-01 -7.4073e-03 +1.7807e-01 -1.5429e-02
Real measured mutual-inductance matrix Lmmeas in henries =
-9.3761e-08 -3.5928e-08 +2.7909e-08 +1.0250e-07
+4.9549e-08 -1.1953e-07 -1.5558e-08 +8.1121e-08
+2.1251e-08 +7.3235e-08 +9.6070e-08 -1.9316e-07
+2.7803e-08 +7.9465e-08 -1.0941e-07 +7.9353e-09
Mechanical seed position for point 7 in meters =
+0.0000e+00 +4.0800e-02 +3.3580e-01 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-9.3103e-08 -3.5391e-08 +2.8248e-08 +1.0044e-07
+4.9139e-08 -1.1873e-07 -1.3896e-08 +7.9652e-08
+2.0739e-08 +7.2855e-08 +9.4300e-08 -1.9069e-07
+2.7677e-08 +7.8473e-08 -1.0926e-07 +8.9852e-09
First-fit position in meters =
+0.0000e+00 +4.1008e-02 +3.3450e-01 +6.8142e-02
First-fit orientation =
+2.6356e-01 +1.1333e-03 -1.3145e-03 +9.6464e-01
First-fit mutual-inductance matrix in henries =
-9.3823e-08 -3.5984e-08 +2.7778e-08 +1.0228e-07
+4.9615e-08 -1.1947e-07 -1.5277e-08 +8.1334e-08
+2.1020e-08 +7.3093e-08 +9.6178e-08 -1.9329e-07
+2.7706e-08 +7.9549e-08 -1.0931e-07 +8.0369e-09
First goodness-of-fit =  2.9259e-06
Second-fit position in meters =
+0.0000e+00 +4.1004e-02 +3.3450e-01 +6.8129e-02
Second-fit orientation =
+2.6352e-01 +1.1361e-03 -1.2912e-03 +9.6465e-01
Second-fit mutual-inductance matrix in henries =
-9.3823e-08 -3.5980e-08 +2.7786e-08 +1.0227e-07
+4.9614e-08 -1.1947e-07 -1.5268e-08 +8.1326e-08
+2.1024e-08 +7.3093e-08 +9.6166e-08 -1.9328e-07
+2.7705e-08 +7.9544e-08 -1.0932e-07 +8.0483e-09
Second goodness-of-fit =  2.9187e-06
Third-fit position in meters =
+0.0000e+00 +4.1004e-02 +3.3450e-01 +6.8129e-02
Third-fit orientation =
+2.6352e-01 +1.1361e-03 -1.2912e-03 +9.6465e-01
Third-fit mutual-inductance matrix in henries =
-9.3823e-08 -3.5980e-08 +2.7786e-08 +1.0227e-07
+4.9614e-08 -1.1947e-07 -1.5268e-08 +8.1326e-08
+2.1024e-08 +7.3093e-08 +9.6166e-08 -1.9328e-07
+2.7705e-08 +7.9544e-08 -1.0932e-07 +8.0483e-09
Third goodness-of-fit =  2.9187e-06
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 8
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+9.5973e-02 +5.2096e-01 -2.0566e-01 -1.8758e-01
+4.2756e-02 +3.3489e-02 +1.6635e-01 +2.2305e-01
+1.1312e+00 +1.9016e-02 +4.3685e-01 +3.0013e-01
-3.3648e-01 -2.3030e-01 +3.6899e-01 -1.4406e-01
Real measured mutual-inductance matrix Lmmeas in henries =
-1.3642e-08 -1.1295e-09 +9.4258e-09 +5.8821e-09
+5.8810e-09 -1.4146e-08 +1.4584e-08 -6.4596e-09
+3.0461e-09 +2.4475e-08 -1.6578e-08 -1.0606e-08
+5.0224e-09 -9.0940e-09 -7.3305e-09 +1.1174e-08
Mechanical seed position for point 8 in meters =
+0.0000e+00 +4.6750e-01 +4.7800e-01 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-1.3623e-08 -1.1120e-09 +8.9909e-09 +5.8013e-09
+5.5702e-09 -1.4157e-08 +1.4858e-08 -6.3768e-09
+3.0781e-09 +2.4475e-08 -1.6778e-08 -1.0519e-08
+5.0829e-09 -9.0386e-09 -7.1424e-09 +1.0896e-08
First-fit position in meters =
+0.0000e+00 +4.6880e-01 +4.7551e-01 +6.9552e-02
First-fit orientation =
+2.6419e-01 +7.5703e-03 -2.7664e-03 +9.6444e-01
First-fit mutual-inductance matrix in henries =
-1.3627e-08 -1.2251e-09 +9.2199e-09 +5.6868e-09
+5.8387e-09 -1.4230e-08 +1.4603e-08 -6.3248e-09
+2.9368e-09 +2.4635e-08 -1.6670e-08 -1.0638e-08
+4.9595e-09 -9.0095e-09 -7.2281e-09 +1.1077e-08
First goodness-of-fit =  9.3870e-05
Second-fit position in meters =
+0.0000e+00 +4.6879e-01 +4.7549e-01 +6.9501e-02
Second-fit orientation =
+2.6417e-01 +7.5354e-03 -2.6999e-03 +9.6444e-01
Second-fit mutual-inductance matrix in henries =
-1.3629e-08 -1.2240e-09 +9.2191e-09 +5.6884e-09
+5.8380e-09 -1.4231e-08 +1.4607e-08 -6.3268e-09
+2.9376e-09 +2.4637e-08 -1.6674e-08 -1.0637e-08
+4.9616e-09 -9.0121e-09 -7.2274e-09 +1.1077e-08
Second goodness-of-fit =  9.3842e-05
Third-fit position in meters =
+0.0000e+00 +4.6879e-01 +4.7549e-01 +6.9500e-02
Third-fit orientation =
+2.6417e-01 +7.5366e-03 -2.6992e-03 +9.6444e-01
Third-fit mutual-inductance matrix in henries =
-1.3629e-08 -1.2239e-09 +9.2191e-09 +5.6884e-09
+5.8380e-09 -1.4231e-08 +1.4607e-08 -6.3268e-09
+2.9376e-09 +2.4637e-08 -1.6674e-08 -1.0637e-08
+4.9616e-09 -9.0121e-09 -7.2274e-09 +1.1077e-08
Third goodness-of-fit =  9.3842e-05
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 9
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+1.3792e-01 +5.0806e-01 -6.0429e-02 +4.5129e-02
-5.4913e-02 +2.4205e-02 +1.8397e-01 -3.0014e-02
+9.5755e-01 +1.3247e-02 +4.1214e-01 +2.4008e-01
-6.2185e-02 -2.5980e-01 +2.5529e-01 +4.1987e-02
Real measured mutual-inductance matrix Lmmeas in henries =
-2.0755e-08 -3.1579e-09 +1.3354e-08 +1.1362e-08
+9.2043e-09 -2.6214e-08 +1.8428e-08 -1.9344e-09
+4.5654e-09 +3.5279e-08 -1.2192e-08 -2.6954e-08
+7.4979e-09 -5.7705e-09 -1.9246e-08 +1.7254e-08
Mechanical seed position for point 9 in meters =
+0.0000e+00 +3.2520e-01 +4.7800e-01 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-2.0788e-08 -3.0809e-09 +1.2828e-08 +1.1097e-08
+8.9387e-09 -2.6150e-08 +1.8739e-08 -1.9400e-09
+4.5154e-09 +3.5169e-08 -1.2464e-08 -2.6629e-08
+7.5727e-09 -5.7851e-09 -1.8970e-08 +1.6950e-08
First-fit position in meters =
+0.0000e+00 +3.2509e-01 +4.7640e-01 +7.0358e-02
First-fit orientation =
+2.6252e-01 +4.2134e-03 -3.2897e-03 +9.6491e-01
First-fit mutual-inductance matrix in henries =
-2.0836e-08 -3.3084e-09 +1.3086e-08 +1.1110e-08
+9.1955e-09 -2.6290e-08 +1.8480e-08 -1.8064e-09
+4.4602e-09 +3.5415e-08 -1.2286e-08 -2.6991e-08
+7.4225e-09 -5.6583e-09 -1.9149e-08 +1.7160e-08
First goodness-of-fit =  5.5690e-05
Second-fit position in meters =
+0.0000e+00 +3.2508e-01 +4.7640e-01 +7.0334e-02
Second-fit orientation =
+2.6247e-01 +4.1966e-03 -3.2653e-03 +9.6493e-01
Second-fit mutual-inductance matrix in henries =
-2.0838e-08 -3.3072e-09 +1.3087e-08 +1.1110e-08
+9.1949e-09 -2.6291e-08 +1.8484e-08 -1.8087e-09
+4.4612e-09 +3.5417e-08 -1.2290e-08 -2.6990e-08
+7.4239e-09 -5.6602e-09 -1.9150e-08 +1.7161e-08
Second goodness-of-fit =  5.5677e-05
Third-fit position in meters =
+0.0000e+00 +3.2508e-01 +4.7640e-01 +7.0334e-02
Third-fit orientation =
+2.6246e-01 +4.1967e-03 -3.2650e-03 +9.6493e-01
Third-fit mutual-inductance matrix in henries =
-2.0838e-08 -3.3072e-09 +1.3087e-08 +1.1110e-08
+9.1949e-09 -2.6291e-08 +1.8484e-08 -1.8087e-09
+4.4612e-09 +3.5417e-08 -1.2290e-08 -2.6990e-08
+7.4239e-09 -5.6602e-09 -1.9150e-08 +1.7161e-08
Third goodness-of-fit =  5.5677e-05
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 10
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+1.2544e-01 +1.5471e-02 -1.3606e-01 +8.0521e-02
+1.2888e-02 -2.7667e-02 +1.2369e-01 +1.7321e-01
+2.2253e-01 +1.1414e-02 -3.0270e-01 +2.5591e-01
+3.2757e-01 +1.0745e-01 +1.4639e-01 +1.8741e-01
Real measured mutual-inductance matrix Lmmeas in henries =
-2.9653e-08 -5.9648e-09 +1.5099e-08 +2.1008e-08
+1.3513e-08 -4.1312e-08 +1.3668e-08 +1.3176e-08
+6.8234e-09 +3.8926e-08 +7.0719e-09 -5.2257e-08
+1.0002e-08 +8.1108e-09 -3.5406e-08 +1.7473e-08
Mechanical seed position for point 10 in meters =
+0.0000e+00 +1.8300e-01 +4.7800e-01 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-2.9513e-08 -5.8263e-09 +1.4789e-08 +2.0621e-08
+1.3305e-08 -4.1166e-08 +1.4047e-08 +1.2960e-08
+6.6774e-09 +3.8778e-08 +6.6557e-09 -5.1602e-08
+1.0034e-08 +7.9845e-09 -3.5081e-08 +1.7299e-08
First-fit position in meters =
+0.0000e+00 +1.8293e-01 +4.7644e-01 +6.8904e-02
First-fit orientation =
+2.6223e-01 +2.4919e-03 -1.5740e-03 +9.6500e-01
First-fit mutual-inductance matrix in henries =
-2.9680e-08 -6.0470e-09 +1.4942e-08 +2.0854e-08
+1.3515e-08 -4.1413e-08 +1.3756e-08 +1.3281e-08
+6.7459e-09 +3.9025e-08 +7.0085e-09 -5.2278e-08
+9.9307e-09 +8.2065e-09 -3.5294e-08 +1.7410e-08
First goodness-of-fit =  1.3732e-05
Second-fit position in meters =
+0.0000e+00 +1.8292e-01 +4.7644e-01 +6.8883e-02
Second-fit orientation =
+2.6219e-01 +2.4846e-03 -1.5484e-03 +9.6501e-01
Second-fit mutual-inductance matrix in henries =
-2.9681e-08 -6.0451e-09 +1.4942e-08 +2.0852e-08
+1.3515e-08 -4.1414e-08 +1.3759e-08 +1.3279e-08
+6.7464e-09 +3.9025e-08 +7.0054e-09 -5.2275e-08
+9.9315e-09 +8.2049e-09 -3.5294e-08 +1.7411e-08
Second goodness-of-fit =  1.3727e-05
Third-fit position in meters =
+0.0000e+00 +1.8292e-01 +4.7644e-01 +6.8883e-02
Third-fit orientation =
+2.6219e-01 +2.4846e-03 -1.5482e-03 +9.6501e-01
Third-fit mutual-inductance matrix in henries =
-2.9681e-08 -6.0451e-09 +1.4942e-08 +2.0852e-08
+1.3515e-08 -4.1414e-08 +1.3759e-08 +1.3279e-08
+6.7464e-09 +3.9025e-08 +7.0054e-09 -5.2275e-08
+9.9315e-09 +8.2049e-09 -3.5294e-08 +1.7411e-08
Third goodness-of-fit =  1.3727e-05
```

```
curre135.cpp rev16, input data point number 1 through 11
This run is for data point number 11
Phase of complex mutual-inductance matrix Lmc_phase in degrees =
+7.3233e-03 +3.3639e-02 -1.5375e-01 +1.2007e-01
-8.0927e-04 +1.2172e-01 -4.8502e-01 +3.2636e-01
+2.0653e-01 +1.1669e-02 +8.9038e-02 +2.4280e-01
+3.6723e-01 -9.1903e-03 +1.4112e-01 +3.4962e-01
Real measured mutual-inductance matrix Lmmeas in henries =
-3.5812e-08 -7.0365e-09 +1.1340e-08 +3.1989e-08
+1.6518e-08 -4.8448e-08 -5.3787e-09 +3.6407e-08
+9.6299e-09 +2.2118e-08 +3.1203e-08 -6.3561e-08
+1.0515e-08 +3.2706e-08 -3.7457e-08 -4.8120e-09
Mechanical seed position for point 11 in meters =
+0.0000e+00 +4.0800e-02 +4.7800e-01 +6.7500e-02
Mechanical seed orientation =
+2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01
Mechanical seed position mutual-inductance matrix in henries =
-3.5492e-08 -6.9389e-09 +1.1284e-08 +3.1451e-08
+1.6428e-08 -4.8216e-08 -4.8356e-09 +3.5970e-08
+9.3888e-09 +2.2079e-08 +3.0710e-08 -6.2862e-08
+1.0450e-08 +3.2347e-08 -3.7310e-08 -4.5088e-09
First-fit position in meters =
+0.0000e+00 +4.1039e-02 +4.7641e-01 +6.8268e-02
First-fit orientation =
+2.6230e-01 +1.3764e-03 -1.1659e-03 +9.6499e-01
First-fit mutual-inductance matrix in henries =
-3.5776e-08 -7.0912e-09 +1.1282e-08 +3.1899e-08
+1.6556e-08 -4.8497e-08 -5.2497e-09 +3.6544e-08
+9.5470e-09 +2.2118e-08 +3.1204e-08 -6.3583e-08
+1.0458e-08 +3.2735e-08 -3.7394e-08 -4.8082e-09
First goodness-of-fit =  4.8006e-06
Second-fit position in meters =
+0.0000e+00 +4.1038e-02 +4.7642e-01 +6.8254e-02
Second-fit orientation =
+2.6227e-01 +1.3728e-03 -1.1465e-03 +9.6499e-01
Second-fit mutual-inductance matrix in henries =
-3.5775e-08 -7.0898e-09 +1.1283e-08 +3.1896e-08
+1.6556e-08 -4.8497e-08 -5.2469e-09 +3.6541e-08
+9.5470e-09 +2.2118e-08 +3.1201e-08 -6.3580e-08
+1.0458e-08 +3.2733e-08 -3.7395e-08 -4.8053e-09
Second goodness-of-fit =  4.7963e-06
Third-fit position in meters =
+0.0000e+00 +4.1038e-02 +4.7642e-01 +6.8254e-02
Third-fit orientation =
+2.6227e-01 +1.3728e-03 -1.1464e-03 +9.6499e-01
Third-fit mutual-inductance matrix in henries =
-3.5775e-08 -7.0898e-09 +1.1283e-08 +3.1896e-08
+1.6556e-08 -4.8497e-08 -5.2469e-09 +3.6541e-08
+9.5470e-09 +2.2118e-08 +3.1201e-08 -6.3580e-08
+1.0458e-08 +3.2733e-08 -3.7395e-08 -4.8053e-09
Third goodness-of-fit =  4.7963e-06
```

Appendix P

C++ code for data reduction to Lm matrix and noise index

```
/* curre136.cpp data reduction for thesis data point:
calculates normalized squared standard deviations of data points

   rev8, 26july2001pta: based on curre123.cpp rev8
   rev9, 26july2001pta: fixed 14/15 error
   rev10, 26july2001pta
   rev11, 31july2001pta
   rev12, 1aug2001pta
   rev13, 1aug2001pta
   rev14, 1aug2001pta
   rev15, 3sep2001pta
   rev16, 4sep2001pta
   rev17, 6sep2001pta

input from stdin:

  data point number 1 through 11

output to stdout:

  mutual inductance matrix and noise indices

   */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <ptacompl.h>
#include <ptaquats.h>

/* function to print a quaternion */
Quatprint(Quaternion R){
  printf("%+11.4e ",  R.r);
  printf("%+11.4e ",  R.i);
  printf("%+11.4e ",  R.j);
  printf("%+11.4e\n", R.k);
}

/* function to print a real 4x4 matrix */
```

```
Rmatprint4x4(double Rmatrix[4][4]){
  printf("%+11.4e ",  Rmatrix[0][0]);
  printf("%+11.4e ",  Rmatrix[0][1]);
  printf("%+11.4e ",  Rmatrix[0][2]);
  printf("%+11.4e\n", Rmatrix[0][3]);
  printf("%+11.4e ",  Rmatrix[1][0]);
  printf("%+11.4e ",  Rmatrix[1][1]);
  printf("%+11.4e ",  Rmatrix[1][2]);
  printf("%+11.4e\n", Rmatrix[1][3]);
  printf("%+11.4e ",  Rmatrix[2][0]);
  printf("%+11.4e ",  Rmatrix[2][1]);
  printf("%+11.4e ",  Rmatrix[2][2]);
  printf("%+11.4e\n", Rmatrix[2][3]);
  printf("%+11.4e ",  Rmatrix[3][0]);
  printf("%+11.4e ",  Rmatrix[3][1]);
  printf("%+11.4e ",  Rmatrix[3][2]);
  printf("%+11.4e\n", Rmatrix[3][3]);
}

/* function to print a complex 4x4 matrix */
Cmatprint4x4(Complex Cmatrix[4][4]){
  printf("%+11.4e%+11.4ei\n      ",
  Cmatrix[0][0].r,Cmatrix[0][0].i);
  printf("%+11.4e%+11.4ei\n            ",
  Cmatrix[0][1].r,Cmatrix[0][1].i);
  printf("%+11.4e%+11.4ei\n                  ",
  Cmatrix[0][2].r,Cmatrix[0][2].i);
  printf("%+11.4e%+11.4ei\n",
  Cmatrix[0][3].r,Cmatrix[0][3].i);
  printf("%+11.4e%+11.4ei\n      ",
  Cmatrix[1][0].r,Cmatrix[1][0].i);
  printf("%+11.4e%+11.4ei\n            ",
  Cmatrix[1][1].r,Cmatrix[1][1].i);
  printf("%+11.4e%+11.4ei\n                  ",
  Cmatrix[1][2].r,Cmatrix[1][2].i);
  printf("%+11.4e%+11.4ei\n"  ,
  Cmatrix[1][3].r,Cmatrix[1][3].i);
  printf("%+11.4e%+11.4ei\n      ",
  Cmatrix[2][0].r,Cmatrix[2][0].i);
  printf("%+11.4e%+11.4ei\n            ",
  Cmatrix[2][1].r,Cmatrix[2][1].i);
  printf("%+11.4e%+11.4ei\n                  ",
  Cmatrix[2][2].r,Cmatrix[2][2].i);
  printf("%+11.4e%+11.4ei\n",
  Cmatrix[2][3].r,Cmatrix[2][3].i);
  printf("%+11.4e%+11.4ei\n      ",
  Cmatrix[3][0].r,Cmatrix[3][0].i);
  printf("%+11.4e%+11.4ei\n            ",
  Cmatrix[3][1].r,Cmatrix[3][1].i);
  printf("%+11.4e%+11.4ei\n                  ",
  Cmatrix[3][2].r,Cmatrix[3][2].i);
  printf("%+11.4e%+11.4ei\n",
  Cmatrix[3][3].r,Cmatrix[3][3].i);
}

/* start of functions for 4x4 complex matrix inverse */
```

```
Cmatmul4x4(Complex l[4][4],Complex r[4][4],Complex prod[4][4]){

  prod[0][0] = l[0][0]*r[0][0]+l[0][1]*r[1][0]
  +l[0][2]*r[2][0]+l[0][3]*r[3][0];
  prod[0][1] = l[0][0]*r[0][1]+l[0][1]*r[1][1]
  +l[0][2]*r[2][1]+l[0][3]*r[3][1];
  prod[0][2] = l[0][0]*r[0][2]+l[0][1]*r[1][2]
  +l[0][2]*r[2][2]+l[0][3]*r[3][2];
  prod[0][3] = l[0][0]*r[0][3]+l[0][1]*r[1][3]
  +l[0][2]*r[2][3]+l[0][3]*r[3][3];

  prod[1][0] = l[1][0]*r[0][0]+l[1][1]*r[1][0]
  +l[1][2]*r[2][0]+l[1][3]*r[3][0];
  prod[1][1] = l[1][0]*r[0][1]+l[1][1]*r[1][1]
  +l[1][2]*r[2][1]+l[1][3]*r[3][1];
  prod[1][2] = l[1][0]*r[0][2]+l[1][1]*r[1][2]
  +l[1][2]*r[2][2]+l[1][3]*r[3][2];
  prod[1][3] = l[1][0]*r[0][3]+l[1][1]*r[1][3]
  +l[1][2]*r[2][3]+l[1][3]*r[3][3];

  prod[2][0] = l[2][0]*r[0][0]+l[2][1]*r[1][0]
  +l[2][2]*r[2][0]+l[2][3]*r[3][0];
  prod[2][1] = l[2][0]*r[0][1]+l[2][1]*r[1][1]
  +l[2][2]*r[2][1]+l[2][3]*r[3][1];
  prod[2][2] = l[2][0]*r[0][2]+l[2][1]*r[1][2]
  +l[2][2]*r[2][2]+l[2][3]*r[3][2];
  prod[2][3] = l[2][0]*r[0][3]+l[2][1]*r[1][3]
  +l[2][2]*r[2][3]+l[2][3]*r[3][3];

  prod[3][0] = l[3][0]*r[0][0]+l[3][1]*r[1][0]
  +l[3][2]*r[2][0]+l[3][3]*r[3][0];
  prod[3][1] = l[3][0]*r[0][1]+l[3][1]*r[1][1]
  +l[3][2]*r[2][1]+l[3][3]*r[3][1];
  prod[3][2] = l[3][0]*r[0][2]+l[3][1]*r[1][2]
  +l[3][2]*r[2][2]+l[3][3]*r[3][2];
  prod[3][3] = l[3][0]*r[0][3]+l[3][1]*r[1][3]
  +l[3][2]*r[2][3]+l[3][3]*r[3][3];
}

/* 3x3 matrix inverter and supporting routines from
"Advanced Engineering Mathematics", Seventh Edition,
by Erwin Kreyszig, John Wiley & Sons, Inc., 1993,
pp. 367-383. */

/* extension to 4x4 by PTA based on data from
"Linear Algebra and its Applications", Third Edition,
by Gilbert Strang, Harcourt Brace Jovanovich, Publishers, 1988,
pp. 231-232. */

/* determinant of 3x3 matrix */
Cmatdet3x3(Complex a[3][3], Complex determinant[1]){

    determinant[0] = ( a[0][0]*(a[1][1]*a[2][2]-a[2][1]*a[1][2])
                     -a[1][0]*(a[0][1]*a[2][2]-a[2][1]*a[0][2])
                   +a[2][0]*(a[0][1]*a[1][2]-a[1][1]*a[0][2]) );
}
```

```
/* determinant of 4x4 matrix */
Cmatdet4x4(Complex a[4][4], Complex determinant[1]){
Complex det0,det1,det2,det3;

  det0 = ( a[1][0]*(a[2][1]*a[3][2]-a[3][1]*a[2][2])
         -a[2][0]*(a[1][1]*a[3][2]-a[3][1]*a[1][2])
         +a[3][0]*(a[1][1]*a[2][2]-a[2][1]*a[1][2]) )*a[0][3];

  det1 = ( a[0][0]*(a[2][1]*a[3][2]-a[3][1]*a[2][2])
         -a[2][0]*(a[0][1]*a[3][2]-a[3][1]*a[0][2])
         +a[3][0]*(a[0][1]*a[2][2]-a[2][1]*a[0][2]) )*a[1][3];

  det2 = ( a[0][0]*(a[1][1]*a[3][2]-a[3][1]*a[1][2])
         -a[1][0]*(a[0][1]*a[3][2]-a[3][1]*a[0][2])
         +a[3][0]*(a[0][1]*a[1][2]-a[1][1]*a[0][2]) )*a[2][3];

  det3 = ( a[0][0]*(a[1][1]*a[2][2]-a[2][1]*a[1][2])
         -a[1][0]*(a[0][1]*a[2][2]-a[2][1]*a[0][2])
         +a[2][0]*(a[0][1]*a[1][2]-a[1][1]*a[0][2]) )*a[3][3];

  determinant[0] = det3 - det2 + det1 - det0;
}

Cmatinv4x4(Complex a[4][4], Complex ainv[4][4]){
  int error;
  Complex adet[1],adetinv,minor[3][3],minordet[1],minors[4][4];

  /* calculate minors of column 0 */

  /* minor of a[0][0] */
  minor[0][0] = a[1][1];
  minor[0][1] = a[1][2];
  minor[0][2] = a[1][3];
  minor[1][0] = a[2][1];
  minor[1][1] = a[2][2];
  minor[1][2] = a[2][3];
  minor[2][0] = a[3][1];
  minor[2][1] = a[3][2];
  minor[2][2] = a[3][3];
  Cmatdet3x3(minor,minordet);
  minors[0][0] = minordet[0];

  /* minor of a[1][0] */
  minor[0][0] = a[0][1];
  minor[0][1] = a[0][2];
  minor[0][2] = a[0][3];
  Cmatdet3x3(minor,minordet);
  minors[1][0] = minordet[0];

  /* minor of a[2][0] */
  minor[1][0] = a[1][1];
  minor[1][1] = a[1][2];
  minor[1][2] = a[1][3];
  Cmatdet3x3(minor,minordet);
  minors[2][0] = minordet[0];

  /* minor of a[3][0] */
```

```
minor[2][0] = a[2][1];
minor[2][1] = a[2][2];
minor[2][2] = a[2][3];
Cmatdet3x3(minor,minordet);
minors[3][0] = minordet[0];

/* calculate minors of column 1 */

/* minor of a[0][1] */
minor[0][0] = a[1][0];
minor[0][1] = a[1][2];
minor[0][2] = a[1][3];
minor[1][0] = a[2][0];
minor[1][1] = a[2][2];
minor[1][2] = a[2][3];
minor[2][0] = a[3][0];
minor[2][1] = a[3][2];
minor[2][2] = a[3][3];
Cmatdet3x3(minor,minordet);
minors[0][1] = minordet[0];

/* minor of a[1][1] */
minor[0][0] = a[0][0];
minor[0][1] = a[0][2];
minor[0][2] = a[0][3];
Cmatdet3x3(minor,minordet);
minors[1][1] = minordet[0];

/* minor of a[2][1] */
minor[1][0] = a[1][0];
minor[1][1] = a[1][2];
minor[1][2] = a[1][3];
Cmatdet3x3(minor,minordet);
minors[2][1] = minordet[0];

/* minor of a[3][1] */
minor[2][0] = a[2][0];
minor[2][1] = a[2][2];
minor[2][2] = a[2][3];
Cmatdet3x3(minor,minordet);
minors[3][1] = minordet[0];

/* calculate minors of column 2 */

/* minor of a[0][2] */
minor[0][0] = a[1][0];
minor[0][1] = a[1][1];
minor[0][2] = a[1][3];
minor[1][0] = a[2][0];
minor[1][1] = a[2][1];
minor[1][2] = a[2][3];
minor[2][0] = a[3][0];
minor[2][1] = a[3][1];
minor[2][2] = a[3][3];
Cmatdet3x3(minor,minordet);
minors[0][2] = minordet[0];
```

```
/* minor of a[1][2] */
minor[0][0] = a[0][0];
minor[0][1] = a[0][1];
minor[0][2] = a[0][3];
Cmatdet3x3(minor,minordet);
minors[1][2] = minordet[0];

/* minor of a[2][2] */
minor[1][0] = a[1][0];
minor[1][1] = a[1][1];
minor[1][2] = a[1][3];
Cmatdet3x3(minor,minordet);
minors[2][2] = minordet[0];

/* minor of a[3][2] */
minor[2][0] = a[2][0];
minor[2][1] = a[2][1];
minor[2][2] = a[2][3];
Cmatdet3x3(minor,minordet);
minors[3][2] = minordet[0];

/* calculate minors of column 3 */

/* minor of a[0][3] */
minor[0][0] = a[1][0];
minor[0][1] = a[1][1];
minor[0][2] = a[1][2];
minor[1][0] = a[2][0];
minor[1][1] = a[2][1];
minor[1][2] = a[2][2];
minor[2][0] = a[3][0];
minor[2][1] = a[3][1];
minor[2][2] = a[3][2];
Cmatdet3x3(minor,minordet);
minors[0][3] = minordet[0];

/* minor of a[1][3] */
minor[0][0] = a[0][0];
minor[0][1] = a[0][1];
minor[0][2] = a[0][2];
Cmatdet3x3(minor,minordet);
minors[1][3] = minordet[0];

/* minor of a[2][3] */
minor[1][0] = a[1][0];
minor[1][1] = a[1][1];
minor[1][2] = a[1][2];
Cmatdet3x3(minor,minordet);
minors[2][3] = minordet[0];

/* minor of a[3][3] */
minor[2][0] = a[2][0];
minor[2][1] = a[2][1];
minor[2][2] = a[2][2];
Cmatdet3x3(minor,minordet);
minors[3][3] = minordet[0];
```

```
  /* if not singular, convert minors to ainv */
  Cmatdet4x4(a,adet);
  error=1; /* singular matrix if |adet| = 0 */
  if((1.0|adet[0]) !=0){
    error = 0;
    adetinv = 1.0/adet[0];
    /* transpose, calculate cofactor signs,
       and divide by determinant */
    ainv[0][0] =       adetinv*minors[0][0];
    ainv[0][1] = -1.0*(adetinv*minors[1][0]);
    ainv[0][2] =       adetinv*minors[2][0];
    ainv[0][3] = -1.0*(adetinv*minors[3][0]);

    ainv[1][0] = -1.0*(adetinv*minors[0][1]);
    ainv[1][1] =       adetinv*minors[1][1];
    ainv[1][2] = -1.0*(adetinv*minors[2][1]);
    ainv[1][3] =       adetinv*minors[3][1];

    ainv[2][0] =       adetinv*minors[0][2];
    ainv[2][1] = -1.0*(adetinv*minors[1][2]);
    ainv[2][2] =       adetinv*minors[2][2];
    ainv[2][3] = -1.0*(adetinv*minors[3][2]);

    ainv[3][0] = -1.0*(adetinv*minors[0][3]);
    ainv[3][1] =       adetinv*minors[1][3];
    ainv[3][2] = -1.0*(adetinv*minors[2][3]);
    ainv[3][3] =       adetinv*minors[3][3];
  }
  return(error);
}

/* end of functions for 4x4 complex matrix inverse */

main(int argc, char *argv[]){

int datptnumint;
char datptnumstr[100];
char datfilnamstr[100];
FILE *inpdat;
char inpstr[10240];
int i,j,k;
int R1XCQ[256],R1YCQ[256],R1ZCQ[256],R2XCQ[256],CURCQ[256];
double R1X[256][26],R1Y[256][26],R1Z[256][26],
R2X[256][26],CUR[256][26];
double twopi;
Complex iw[4];
Complex Iiw[4][4]; /* jw times driver current */
Complex Txc_gain[4];
Complex Iiw_inv[4][4];
  /* inverse of (jw times driver current) */
Complex Iiw_times_Iiw_inv[4][4]; /* should be unit matrix */
Complex Rxc_atten[4];
Complex Gain[4], S[4][4], SIiw_inv[4][4];
Complex Lmc[4][4]; /* complex mutual-inductance matrix */
double Lmc_phase[4][4]; /* Lmc phase errors from Lm */
double Lm[4][4]; /* real mutual-inductance matrix */
Quaternion Omech, Pmech[12];
```

```
/* means of sums of data elements */
Complex msCUR[4][4], msLm[4][4];
/* squares of magnitudes of means of sums of data elements */
double smmsCUR[4][4], smmsLm[4][4];
/* means of sums of squares of magnitudes of data elements */
double mssmCUR[4][4], mssmLm[4][4];
/* sums over 4x4 matrix */
double sm44CUR,sm44Lm,ms44CUR,ms44Lm,msMsm44CUR,msMsm44Lm;
/* standard deviation of Lm 4x4 matrix */
double stdevLm[4][4];
/* standard deviation of CUR 4x4 matrix */
double stdevCUR[4][4];
/* normalized squared variations in matrices */
double CUR_Ni,Lm_Ni;

/* iw = i * 2pi * frequency_in_hertz */
twopi = 8.0 * atan(1.0);
iw[0].i = twopi * 11359.321;
iw[1].i = twopi * 11664.497;
iw[2].i = twopi * 11969.672;
iw[3].i = twopi * 12274.848;
iw[0].r = 0.0;
iw[1].r = 0.0;
iw[2].r = 0.0;
iw[3].r = 0.0;

printf(
"curre136.cpp rev17, input data point number 1 through 11\n");

/* save the point number */
strcpy(datptnumstr,argv[1]);
datptnumint = atoi(datptnumstr);
/* print the point number */
printf("This run is for data point number ");
puts(datptnumstr);

/* generate the receiver-board SOP data filename */
strcpy(datfilnamstr,"tetra");
strcat(datfilnamstr,datptnumstr);
strcat(datfilnamstr,".dat");
/* open the file */
if((inpdat = fopen(datfilnamstr,"r"))==NULL)printf(
"file open failed\n");
else{

  /* read the input file */
  strcpy(inpstr,"this string does not start with a digit\n");

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  R1X[0][0] = 25.0;

  /* read R1X block of data */
  for(i=0;i<256;i++){
```

```
    /* parse the input record */
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf",
     &R1XCQ[i],&R1X[i][0],&R1X[i][1],&R1X[i][2],&R1X[i][3],
     &R1X[i][4],
     &R1X[i][5],&R1X[i][6],&R1X[i][7],&R1X[i][8],&R1X[i][9],
     &R1X[i][10],&R1X[i][11],&R1X[i][12],&R1X[i][13],&R1X[i][14],
     &R1X[i][15],&R1X[i][16],&R1X[i][17],&R1X[i][18],&R1X[i][19],
     &R1X[i][20],&R1X[i][21],&R1X[i][22],&R1X[i][23],&R1X[i][24],
     &R1X[i][25]);
    /* get the next input record */
    fgets(inpstr,1000,inpdat);
  }

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  /* read R1Y block of data */
  for(i=0;i<256;i++){
    /* parse the input record */
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf",
     &R1YCQ[i],&R1Y[i][0],&R1Y[i][1],&R1Y[i][2],&R1Y[i][3],
     &R1Y[i][4],
     &R1Y[i][5],&R1Y[i][6],&R1Y[i][7],&R1Y[i][8],&R1Y[i][9],
     &R1Y[i][10],&R1Y[i][11],&R1Y[i][12],&R1Y[i][13],&R1Y[i][14],
     &R1Y[i][15],&R1Y[i][16],&R1Y[i][17],&R1Y[i][18],&R1Y[i][19],
     &R1Y[i][20],&R1Y[i][21],&R1Y[i][22],&R1Y[i][23],&R1Y[i][24],
     &R1Y[i][25]);
    /* get the next input record */
    fgets(inpstr,1000,inpdat);
  }

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  /* read R1Z block of data */
  for(i=0;i<256;i++){
    /* parse the input record */
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf",
     &R1ZCQ[i],&R1Z[i][0],&R1Z[i][1],&R1Z[i][2],&R1Z[i][3],
     &R1Z[i][4],
     &R1Z[i][5],&R1Z[i][6],&R1Z[i][7],&R1Z[i][8],&R1Z[i][9],
     &R1Z[i][10],&R1Z[i][11],&R1Z[i][12],&R1Z[i][13],&R1Z[i][14],
     &R1Z[i][15],&R1Z[i][16],&R1Z[i][17],&R1Z[i][18],&R1Z[i][19],
     &R1Z[i][20],&R1Z[i][21],&R1Z[i][22],&R1Z[i][23],&R1Z[i][24],
     &R1Z[i][25]);
    /* get the next input record */
    fgets(inpstr,1000,inpdat);
```

```
  }

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  /* read CUR block of data */
  for(i=0;i<256;i++){
    /* parse the input record */
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf",
     &CURCQ[i],&CUR[i][0],&CUR[i][1],&CUR[i][2],&CUR[i][3],
     &CUR[i][4],
     &CUR[i][5],&CUR[i][6],&CUR[i][7],&CUR[i][8],&CUR[i][9],
     &CUR[i][10],&CUR[i][11],&CUR[i][12],&CUR[i][13],&CUR[i][14],
     &CUR[i][15],&CUR[i][16],&CUR[i][17],&CUR[i][18],&CUR[i][19],
     &CUR[i][20],&CUR[i][21],&CUR[i][22],&CUR[i][23],&CUR[i][24],
     &CUR[i][25]);
    /* get the next input record */
    fgets(inpstr,1000,inpdat);
  }

  /* skip records which do not start with a digit */
  while(isdigit(inpstr[0])==0){
  fgets(inpstr,10200,inpdat);
  }

  /* read R2X block of data
    (second R1X, after current, in tetra%d.dat) */
  for(i=0;i<256;i++){
    sscanf(inpstr,
     "%d%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf
%lf%lf%lf%lf%lf%lf%lf%lf%lf%lf",
     &R2XCQ[i],&R2X[i][0],&R2X[i][1],&R2X[i][2],&R2X[i][3],
     &R2X[i][4],
     &R2X[i][5],&R2X[i][6],&R2X[i][7],&R2X[i][8],&R2X[i][9],
     &R2X[i][10],&R2X[i][11],&R2X[i][12],&R2X[i][13],&R2X[i][14],
     &R2X[i][15],&R2X[i][16],&R2X[i][17],&R2X[i][18],&R2X[i][19],
     &R2X[i][20],&R2X[i][21],&R2X[i][22],&R2X[i][23],&R2X[i][24],
     &R2X[i][25]);
    fgets(inpstr,1000,inpdat);
  }

  /* we now have the file data in the arrays */

  /* process transmitter currents */

  /* The transmitter[0,1,2,3] currents are measured
     when CQ=[2,3,4,5],
     excluding the first measurement of each current:

      index    CUR[index][frq]
     -------  --------------------
      17-31    transmitter coil 0
      33-47    transmitter coil 1
```

```
     49-63    transmitter coil 2
     65-79    transmitter coil 3

    The transmitter[0,1,2,3] driver frequencies are
    SOP numbers:

     frq    driver frequency
    -----  ----------------------
      2     transmitter driver 0 I
      3     transmitter driver 0 Q
      4     transmitter driver 1 I
      5     transmitter driver 1 Q
      6     transmitter driver 2 I
      7     transmitter driver 2 Q
      8     transmitter driver 3 I
      9     transmitter driver 3 Q

*/

 for(i=0;i<4;i++){
   for(j=0;j<4;j++){
     mssmCUR[i][j] = 0.0;
   }
 }

 /* Iiw[transmitter driver][frequency] */

 for(i=17;i<31;i++){
   Iiw[0][0].r = Iiw[0][0].r + CUR[i][2];
   Iiw[0][0].i = Iiw[0][0].i + CUR[i][3];
   Iiw[0][1].r = Iiw[0][1].r + CUR[i][4];
   Iiw[0][1].i = Iiw[0][1].i + CUR[i][5];
   Iiw[0][2].r = Iiw[0][2].r + CUR[i][6];
   Iiw[0][2].i = Iiw[0][2].i + CUR[i][7];
   Iiw[0][3].r = Iiw[0][3].r + CUR[i][8];
   Iiw[0][3].i = Iiw[0][3].i + CUR[i][9];
   mssmCUR[0][0] += CUR[i][2]*CUR[i][2]+CUR[i][3]*CUR[i][3];
   mssmCUR[0][1] += CUR[i][4]*CUR[i][4]+CUR[i][5]*CUR[i][5];
   mssmCUR[0][2] += CUR[i][6]*CUR[i][6]+CUR[i][7]*CUR[i][7];
   mssmCUR[0][3] += CUR[i][8]*CUR[i][8]+CUR[i][9]*CUR[i][9];
 }
 for(i=33;i<47;i++){
   Iiw[1][0].r = Iiw[1][0].r + CUR[i][2];
   Iiw[1][0].i = Iiw[1][0].i + CUR[i][3];
   Iiw[1][1].r = Iiw[1][1].r + CUR[i][4];
   Iiw[1][1].i = Iiw[1][1].i + CUR[i][5];
   Iiw[1][2].r = Iiw[1][2].r + CUR[i][6];
   Iiw[1][2].i = Iiw[1][2].i + CUR[i][7];
   Iiw[1][3].r = Iiw[1][3].r + CUR[i][8];
   Iiw[1][3].i = Iiw[1][3].i + CUR[i][9];
   mssmCUR[1][0] += CUR[i][2]*CUR[i][2]+CUR[i][3]*CUR[i][3];
   mssmCUR[1][1] += CUR[i][4]*CUR[i][4]+CUR[i][5]*CUR[i][5];
   mssmCUR[1][2] += CUR[i][6]*CUR[i][6]+CUR[i][7]*CUR[i][7];
   mssmCUR[1][3] += CUR[i][8]*CUR[i][8]+CUR[i][9]*CUR[i][9];
 }
 for(i=49;i<63;i++){
   Iiw[2][0].r = Iiw[2][0].r + CUR[i][2];
```

```
  Iiw[2][0].i = Iiw[2][0].i + CUR[i][3];
  Iiw[2][1].r = Iiw[2][1].r + CUR[i][4];
  Iiw[2][1].i = Iiw[2][1].i + CUR[i][5];
  Iiw[2][2].r = Iiw[2][2].r + CUR[i][6];
  Iiw[2][2].i = Iiw[2][2].i + CUR[i][7];
  Iiw[2][3].r = Iiw[2][3].r + CUR[i][8];
  Iiw[2][3].i = Iiw[2][3].i + CUR[i][9];
  mssmCUR[2][0] += CUR[i][2]*CUR[i][2]+CUR[i][3]*CUR[i][3];
  mssmCUR[2][1] += CUR[i][4]*CUR[i][4]+CUR[i][5]*CUR[i][5];
  mssmCUR[2][2] += CUR[i][6]*CUR[i][6]+CUR[i][7]*CUR[i][7];
  mssmCUR[2][3] += CUR[i][8]*CUR[i][8]+CUR[i][9]*CUR[i][9];
}
for(i=65;i<79;i++){
  Iiw[3][0].r = Iiw[3][0].r + CUR[i][2];
  Iiw[3][0].i = Iiw[3][0].i + CUR[i][3];
  Iiw[3][1].r = Iiw[3][1].r + CUR[i][4];
  Iiw[3][1].i = Iiw[3][1].i + CUR[i][5];
  Iiw[3][2].r = Iiw[3][2].r + CUR[i][6];
  Iiw[3][2].i = Iiw[3][2].i + CUR[i][7];
  Iiw[3][3].r = Iiw[3][3].r + CUR[i][8];
  Iiw[3][3].i = Iiw[3][3].i + CUR[i][9];
  mssmCUR[3][0] += CUR[i][2]*CUR[i][2]+CUR[i][3]*CUR[i][3];
  mssmCUR[3][1] += CUR[i][4]*CUR[i][4]+CUR[i][5]*CUR[i][5];
  mssmCUR[3][2] += CUR[i][6]*CUR[i][6]+CUR[i][7]*CUR[i][7];
  mssmCUR[3][3] += CUR[i][8]*CUR[i][8]+CUR[i][9]*CUR[i][9];
}

for(i=0;i<4;i++){
  for(j=0;j<4;j++){
    mssmCUR[i][j] = (1.0/14.0) * mssmCUR[i][j];
    msCUR[i][j] = (1.0/14.0) * Iiw[i][j];
    smmsCUR[i][j] =
      msCUR[i][j].r*msCUR[i][j].r +
      msCUR[i][j].i*msCUR[i][j].i;
    Iiw[i][j] = ((1.0/14.0) * Iiw[i][j]);
  }
}

printf("transmitter current matrix =\n");
Cmatprint4x4(Iiw);

sm44CUR =0.0;
ms44CUR =0.0;
for(i=0;i<4;i++){
  for(j=0;j<4;j++){
    sm44CUR += smmsCUR[i][j];
    ms44CUR += mssmCUR[i][j];
    stdevCUR[i][j] = (mssmCUR[i][j]-smmsCUR[i][j])/sqrt(14.0);
    Iiw[i][j] = Iiw[i][j] * (-1.0*iw[j]);
  }
}

/* The (-1.0*iw[j]) factor aboves converts the currents into
   their time derivatives, based on e^(-iwt) time dependence. */

printf(
"Standard deviation of transmitter current matrix =\n");
```

```
 Rmatprint4x4(stdevCUR);

 msMsm44CUR = ms44CUR - sm44CUR;
 CUR_Ni = msMsm44CUR/ms44CUR;
 printf(
 "transmitter curent noise index CUR_Ni = %11.4e\n",CUR_Ni);

 printf(
 "(cable-uncorrected jw * transmitter current) matrix =\n");
 Cmatprint4x4(Iiw);

 /* start of transmitter cable gain data from curre109.c
    This data is for Rcoil = 28 ohms and Lcoil = 410 uH,
    using 1 T section (2 T sections gives the same data). */
 Txc_gain[0].r =  1.000115e+00;
 Txc_gain[0].i = -1.1e-04;
 Txc_gain[1].r =  1.000121e+00;
 Txc_gain[1].i = -1.1e-04;
 Txc_gain[2].r =  1.000128e+00;
 Txc_gain[2].i = -1.2e-04;
 Txc_gain[3].r =  1.000134e+00;
 Txc_gain[3].i = -1.2e-04;

 /* Multiply each complex current by transmitter cable gain */
 for(i=0;i<4;i++){
   for(j=0;j<4;j++){
     Iiw[i][j] = Txc_gain[j] * Iiw[i][j];
   }
 }
 /* end of data from curre109.c */

 printf(
 "(cable-corrected jw * transmitter current) matrix Iiw =\n");
 Cmatprint4x4(Iiw);

 /* invert the current matrix */

 Cmatinv4x4(Iiw, Iiw_inv);

 printf(
 "(cable-corrected jw * transmitter current).inv matrix
Iiw_inv =\n");
 Cmatprint4x4(Iiw_inv);

 Cmatmul4x4(Iiw_inv, Iiw, Iiw_times_Iiw_inv);

 printf("Iiw_inv * Iiw =\n");
 Cmatprint4x4(Iiw_times_Iiw_inv);

 Cmatmul4x4(Iiw, Iiw_inv, Iiw_times_Iiw_inv);

 printf("Iiw * Iiw_inv =\n");
 Cmatprint4x4(Iiw_times_Iiw_inv);

 /* average 239 measurements for each Lm[4][4] element */

 for(i=0;i<4;i++){
```

```
  for(j=0;j<4;j++){
    Lm[i][j] = 0.0;
    mssmLm[i][j] = 0.0;
  }
}

for(k=1;k<240;k++){
/* calculate S[r][t] */
S[0][0].r = R2X[k][2];
S[0][0].i = R2X[k][3];
S[1][0].r = R1X[k][2];
S[1][0].i = R1X[k][3];
S[2][0].r = R1Y[k][2];
S[2][0].i = R1Y[k][3];
S[3][0].r = R1Z[k][2];
S[3][0].i = R1Z[k][3];

S[0][1].r = R2X[k][4];
S[0][1].i = R2X[k][5];
S[1][1].r = R1X[k][4];
S[1][1].i = R1X[k][5];
S[2][1].r = R1Y[k][4];
S[2][1].i = R1Y[k][5];
S[3][1].r = R1Z[k][4];
S[3][1].i = R1Z[k][5];

S[0][2].r = R2X[k][6];
S[0][2].i = R2X[k][7];
S[1][2].r = R1X[k][6];
S[1][2].i = R1X[k][7];
S[2][2].r = R1Y[k][6];
S[2][2].i = R1Y[k][7];
S[3][2].r = R1Z[k][6];
S[3][2].i = R1Z[k][7];

S[0][3].r = R2X[k][8];
S[0][3].i = R2X[k][9];
S[1][3].r = R1X[k][8];
S[1][3].i = R1X[k][9];
S[2][3].r = R1Y[k][8];
S[2][3].i = R1Y[k][9];
S[3][3].r = R1Z[k][8];
S[3][3].i = R1Z[k][9];

/* frequency-dependent receiver cable attenuation */
Rxc_atten[0].r =  1.001140e+00;
Rxc_atten[0].i =  1.793272e-03;
Rxc_atten[1].r =  1.001126e+00;
Rxc_atten[1].i =  1.839486e-03;
Rxc_atten[2].r =  1.001109e+00;
Rxc_atten[2].i =  1.892287e-03;
Rxc_atten[3].r =  1.001093e+00;
Rxc_atten[3].i =  1.941929e-03;

/* S[r][f] = S[r][f] Rxc_atten[f] */
for(i=0;i<4;i++){
  for(j=0;j<4;j++){
```

```
      S[i][j] = Rxc_atten[j] * S[i][j];
    }
  }

  /* SIiw_inv[r][t] = S[r][f] Iiw_inv[f][t] */
  Cmatmul4x4(S,Iiw_inv,SIiw_inv);

  /* channel-dependent receiver channel gains */
  Gain[0].r =  0.14938762;
  Gain[0].i = -0.000310136;
  Gain[1].r =  0.14950998;
  Gain[1].i = -0.0000908787;
  Gain[2].r =  0.15066835;
  Gain[2].i = -0.0001819950;
  Gain[3].r =  0.14910116;
  Gain[3].i = -0.00008113399;

  /* Lmt[r][t] = Gain[r] SIiw_inv[r][t]
     Lmc[t][r] = Lmt[r][t].transpose
     in these loops, r=i and t=j, so Lmt[r][t] = Lmt[i][j] */
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      Lmc[j][i] = Gain[i] * SIiw_inv[i][j];
    }
  }

  /* extract mean (real Lm) and mean ((real Lm) squared)
     matrices */
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      Lm[i][j] += Lmc[i][j].r/239.0;  /* real Lm */
      mssmLm[i][j] += Lmc[i][j].r*Lmc[i][j].r/239.0;
    }
  }

 }

  sm44Lm =0.0;
  ms44Lm =0.0;
  for(i=0;i<4;i++){
    for(j=0;j<4;j++){
      smmsLm[i][j] = Lm[i][j] * Lm[i][j];
      ms44Lm += mssmLm[i][j];
      sm44Lm += smmsLm[i][j];
      stdevLm[i][j] =
            sqrt((240.0/239.0)*(mssmLm[i][j]-smmsLm[i][j]));
    }
  }
  msMsm44Lm = ms44Lm - sm44Lm;

  printf(
"Real measured mutual-inductance matrix Lmmeas in henries =\n");
  Rmatprint4x4(Lm);

  printf("Standard deviation of Lmmeas in henries =\n");
  Rmatprint4x4(stdevLm);
```

```
  Lm_Ni = msMsm44Lm / ms44Lm;
  printf("receiver noise index = Lm_Ni = %11.4e\n",Lm_Ni);

}
}
```

Appendix Q

Lm matrix and noise-index results

The results of running the code in the preceding appendix appear on the following pages. Each row of each complex matrix is printed as four staggered entries to fit the page width.

Note that the frequency-times-current matrix `Iiw` is well-conditioned for all eleven locations. The condition number [14] for `Iiw`, calculated with Octave, is between 3.7 and 3.8 in all eleven locations. A perfectly-conditioned matrix has a condition number of 1, and a singular matrix has a condition number of infinity, so `Iiw` is well-conditioned.

For each column of `Iiw`, three of the elements are similar in magnitude, and the fourth element is eight or more times the magnitude of the other elements. No two columns have the same element as the largest in magnitude. Thus, the columns are linearly independent and `Iiw` is well-conditioned and invertible. For `Iiw` left-multiplied by its inverse `Iiw_inv` and for `Iiw` right-multiplied by its inverse `Iiw_inv`, the difference between the product and the expected unit matrix is less than 10^{-15} in all elements (The diagonal elements were checked using a variant code, not included here, which subtracted 1 from each diagonal element). Thus, the inverse `Iiw_inv` is accurate.

Output for location 1:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 1
transmitter current matrix =
+1.0590e+07-3.7569e+06i
      +8.0242e+05-9.8672e+05i
            +5.6985e+05+1.1287e+06i
                  -1.3718e+06+2.2271e+05i
-1.0086e+06-3.9386e+05i
      -2.0401e+07+3.1598e+07i
            -1.1629e+06+3.8189e+06i
                  -3.4443e+06-2.6572e+06i
-8.7833e+05-4.5563e+05i
      +3.6609e+06-2.5662e+05i
            -1.9122e+07-2.9664e+07i
                  -2.8149e+06-2.6255e+06i
-8.9040e+05-4.4265e+05i
      +3.6703e+06-3.2275e+05i
            -1.3308e+06+3.3415e+06i
                  +3.4858e+07-9.3026e+06i
Standard deviation of transmitter current matrix =
+1.8717e+04 +6.1126e+03 +9.3377e+03 +8.1835e+03
+9.3186e+03 +2.6680e+04 +9.1015e+03 +1.1052e+04
+1.1243e+04 +8.9385e+03 +2.1091e+04 +9.0575e+03
+6.6810e+03 +1.2660e+04 +7.2360e+03 +4.4096e+04
transmitter curent noise index CUR_Ni =  1.9620e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.6814e+11-7.5582e+11i
      -7.2317e+10-5.8810e+10i
            +8.4887e+10-4.2857e+10i
                  +1.7176e+10+1.0580e+11i
-2.8111e+10+7.1988e+10i
      +2.3158e+12+1.4952e+12i
            +2.8721e+11+8.7460e+10i
                  -2.0494e+11+2.6564e+11i
-3.2519e+10+6.2689e+10i
      -1.8808e+10-2.6831e+11i
            -2.2309e+12+1.4382e+12i
                  -2.0249e+11+2.1710e+11i
-3.1593e+10+6.3550e+10i
      -2.3655e+10-2.6899e+11i
            +2.5130e+11+1.0008e+11i
                  -7.1747e+11-2.6885e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.6825e+11-7.5588e+11i
      -7.2332e+10-5.8809e+10i
            +8.4893e+10-4.2873e+10i
                  +1.7191e+10+1.0581e+11i
-2.8106e+10+7.1999e+10i
      +2.3163e+12+1.4952e+12i
            +2.8725e+11+8.7437e+10i
                  -2.0493e+11+2.6570e+11i
-3.2516e+10+6.2700e+10i
      -1.8840e+10-2.6834e+11i
            -2.2311e+12+1.4386e+12i
                  -2.0250e+11+2.1715e+11i
-3.1590e+10+6.3561e+10i
```

```
      -2.3687e+10-2.6902e+11i
            +2.5135e+11+1.0007e+11i
                  -7.1789e+11-2.6887e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3066e-13+1.1770e-12i
      -2.9034e-14+3.8042e-14i
            -3.0643e-14+4.0461e-14i
                  -3.2190e-14+4.2983e-14i
+3.6782e-14+1.3475e-14i
      +3.0781e-13-1.8732e-13i
            +4.0688e-14+1.8274e-14i
                  +4.2497e-14+2.1132e-14i
+7.9800e-16-3.7624e-14i
      +2.2545e-15-3.9886e-14i
            -3.0882e-13-2.1033e-13i
                  +5.6949e-15-4.3980e-14i
-3.4264e-14+1.1116e-14i
      -3.6732e-14+1.0510e-14i
            -3.9179e-14+9.7074e-15i
                  -1.0084e-13+3.4173e-13i
Iiw_inv * Iiw =
+1.0000e+00+1.4745e-17i
      +3.1225e-17-8.6736e-18i
            +0.0000e+00-1.9949e-17i
                  +0.0000e+00+1.3878e-17i
+1.3010e-18+1.3010e-18i
      +1.0000e+00-3.9899e-17i
            +1.3878e-17+1.7347e-18i
                  -3.4694e-18+2.7756e-17i
-1.7347e-18+4.3368e-19i
      +5.2042e-18+1.2035e-17i
            +1.0000e-00-1.9082e-17i
                  +0.0000e+00+3.4694e-18i
+0.0000e+00+0.0000e+00i
      -1.3878e-17-3.4694e-18i
            +2.0817e-17+0.0000e+00i
                  +1.0000e-00-3.1225e-17i
Iiw * Iiw_inv =
+1.0000e+00+1.6046e-17i
      -5.6379e-18+4.3368e-19i
            +1.2143e-17+3.4694e-18i
                  +0.0000e+00+5.2042e-18i
-1.3878e-17-1.5613e-17i
      +1.0000e-00-3.8164e-17i
            +2.1684e-17-1.2143e-17i
                  +0.0000e+00+0.0000e+00i
+0.0000e+00-1.7347e-17i
      -1.3878e-17-3.2960e-17i
            +1.0000e-00-2.2551e-17i
                  +6.9389e-18+1.3878e-17i
-6.9389e-18-1.3878e-17i
      +0.0000e+00+1.3878e-17i
            -6.9389e-18+1.3878e-17i
                  +1.0000e-00-2.7756e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-3.4003e-07 -2.5099e-07 +7.2242e-07 -1.2769e-07
+2.6964e-07 -5.5503e-08 +2.7528e-07 -4.6249e-07
```

```
-2.1502e-07 +7.8101e-07 -1.3580e-06 +7.8160e-07
+2.9133e-07 -4.2162e-07 +2.2055e-07 -1.1635e-07
Standard deviation of Lmmeas in henries =
+1.4768e-10 +7.3377e-10 +3.6686e-10 +3.8020e-10
+4.4261e-11 +2.0661e-10 +1.0748e-10 +1.0698e-10
+6.2692e-11 +2.6025e-10 +1.7075e-10 +1.2904e-10
+5.2126e-11 +3.0112e-10 +1.5165e-10 +1.4076e-10
receiver noise index = Lm_Ni =  2.5581e-07
```

Output for location 2:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 2
transmitter current matrix =
+1.0519e+07-3.6364e+06i
      +8.1149e+05-9.7148e+05i
            +5.5066e+05+1.1311e+06i
                  -1.3680e+06+1.9991e+05i
-9.9976e+05-4.0439e+05i
      -2.0930e+07+3.1368e+07i
            -1.2544e+06+3.8199e+06i
                  -3.4035e+06-2.7559e+06i
-8.6842e+05-4.6382e+05i
      +3.6922e+06-1.8443e+05i
            -1.8670e+07-3.0043e+07i
                  -2.7696e+06-2.7078e+06i
-8.8072e+05-4.5183e+05i
      +3.7037e+06-2.4821e+05i
            -1.4150e+06+3.3333e+06i
                  +3.5090e+07-8.6887e+06i
Standard deviation of transmitter current matrix =
+8.2184e+03 +6.9335e+03 +8.4219e+03 +1.0506e+04
+1.3458e+04 +1.5474e+04 +1.3384e+04 +1.2214e+04
+1.0578e+04 +1.1250e+04 +1.9507e+04 +1.0401e+04
+1.1524e+04 +6.1026e+03 +7.4355e+03 +3.3864e+04
transmitter curent noise index CUR_Ni =  1.7740e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.5954e+11-7.5080e+11i
      -7.1200e+10-5.9474e+10i
            +8.5064e+10-4.1414e+10i
                  +1.5418e+10+1.0551e+11i
-2.8862e+10+7.1356e+10i
      +2.2990e+12+1.5339e+12i
            +2.8729e+11+9.4339e+10i
                  -2.1255e+11+2.6250e+11i
-3.3104e+10+6.1981e+10i
      -1.3517e+10-2.7060e+11i
            -2.2595e+12+1.4041e+12i
                  -2.0884e+11+2.1361e+11i
-3.2248e+10+6.2859e+10i
      -1.8191e+10-2.7144e+11i
            +2.5069e+11+1.0642e+11i
                  -6.7011e+11-2.7063e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.5965e+11-7.5086e+11i
      -7.1215e+10-5.9473e+10i
            +8.5070e+10-4.1429e+10i
                  +1.5433e+10+1.0552e+11i
-2.8858e+10+7.1367e+10i
      +2.2994e+12+1.5339e+12i
            +2.8733e+11+9.4316e+10i
                  -2.1254e+11+2.6256e+11i
-3.3101e+10+6.1992e+10i
      -1.3549e+10-2.7063e+11i
            -2.2596e+12+1.4046e+12i
                  -2.0884e+11+2.1366e+11i
-3.2245e+10+6.2870e+10i
```

```
      -1.8223e+10-2.7147e+11i
            +2.5073e+11+1.0640e+11i
                  -6.7053e+11-2.7066e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.2511e-13+1.1916e-12i
      -2.8755e-14+3.8345e-14i
            -3.0339e-14+4.0782e-14i
                  -3.1855e-14+4.3329e-14i
+3.6983e-14+1.3147e-14i
      +3.0407e-13-1.9134e-13i
            +4.0909e-14+1.7925e-14i
                  +4.2738e-14+2.0777e-14i
+3.8584e-16-3.7690e-14i
      +1.8271e-15-3.9968e-14i
            -3.1145e-13-2.0468e-13i
                  +5.2468e-15-4.4094e-14i
-3.4207e-14+1.1554e-14i
      -3.6681e-14+1.0960e-14i
            -3.9138e-14+1.0173e-14i
                  -9.4454e-14+3.4268e-13i
Iiw_inv * Iiw =
+1.0000e+00+4.0766e-17i
      +3.4694e-18+0.0000e+00i
            -1.7347e-18-8.6736e-19i
                  +0.0000e+00+6.9389e-18i
-4.3368e-19-1.3010e-18i
      +1.0000e-00+7.1124e-17i
            -1.5613e-17+6.9389e-18i
                  +0.0000e+00+0.0000e+00i
+1.3010e-18-3.9031e-18i
      -3.2960e-17+1.2577e-17i
            +1.0000e+00+5.2042e-17i
                  +0.0000e+00+1.7347e-18i
+3.4694e-18+6.9389e-18i
      +0.0000e+00-2.0817e-17i
            +2.0817e-17+0.0000e+00i
                  +1.0000e+00-1.7347e-17i
Iiw * Iiw_inv =
+1.0000e+00+4.0766e-17i
      +9.5410e-18+1.7347e-18i
            +4.5536e-18+8.6736e-19i
                  -6.9389e-18+6.0715e-18i
-1.6480e-17+3.4694e-18i
      +1.0000e-00+7.6328e-17i
            -9.5410e-18-1.7347e-18i
                  +1.3878e-17+1.3878e-17i
-1.4745e-17-1.3878e-17i
      +2.6021e-18-2.2551e-17i
            +1.0000e+00+4.6838e-17i
                  -6.9389e-18+0.0000e+00i
+0.0000e+00+0.0000e+00i
      -6.9389e-18+1.3878e-17i
            +6.9389e-18+2.7756e-17i
                  +1.0000e+00-1.7347e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-3.7373e-08 -2.4934e-09 +3.9777e-08 +1.0627e-09
+1.4715e-08 +1.2881e-08 +7.8015e-09 -3.5144e-08
```

```
+5.7333e-09 +3.3710e-08 -8.9640e-08 +4.8742e-08
+1.7629e-08 -4.4274e-08 +4.0535e-08 -1.3973e-08
Standard deviation of Lmmeas in henries =
+1.9374e-10 +5.4672e-10 +2.9723e-10 +2.8621e-10
+1.0685e-10 +1.8949e-10 +1.0295e-10 +9.3255e-11
+2.9419e-10 +1.7605e-10 +1.0327e-10 +8.9723e-11
+2.5240e-10 +2.2606e-10 +1.2058e-10 +1.1352e-10
receiver noise index = Lm_Ni =  4.1673e-05
```

Output for location 3:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 3
transmitter current matrix =
+1.0583e+07-3.7441e+06i
      +8.0346e+05-9.8510e+05i
            +5.6769e+05+1.1288e+06i
                  -1.3715e+06+2.2054e+05i
-1.0077e+06-3.9490e+05i
      -2.0457e+07+3.1571e+07i
            -1.1721e+06+3.8179e+06i
                  -3.4399e+06-2.6671e+06i
-8.7743e+05-4.5660e+05i
      +3.6642e+06-2.4909e+05i
            -1.9071e+07-2.9693e+07i
                  -2.8105e+06-2.6338e+06i
-8.8951e+05-4.4367e+05i
      +3.6735e+06-3.1477e+05i
            -1.3390e+06+3.3397e+06i
                  +3.4879e+07-9.2417e+06i
Standard deviation of transmitter current matrix =
+1.2030e+04 +7.5452e+03 +9.0291e+03 +6.5750e+03
+9.7379e+03 +1.8446e+04 +1.5486e+04 +9.1749e+03
+1.0669e+04 +5.7292e+03 +2.0737e+04 +8.9913e+03
+6.2892e+03 +1.3002e+04 +1.3061e+04 +2.6910e+04
transmitter curent noise index CUR_Ni =  1.7286e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.6723e+11-7.5535e+11i
      -7.2198e+10-5.8886e+10i
            +8.4897e+10-4.2695e+10i
                  +1.7009e+10+1.0578e+11i
-2.8185e+10+7.1923e+10i
      +2.3138e+12+1.4993e+12i
            +2.8713e+11+8.8152e+10i
                  -2.0570e+11+2.6531e+11i
-3.2589e+10+6.2625e+10i
      -1.8256e+10-2.6855e+11i
            -2.2332e+12+1.4343e+12i
                  -2.0313e+11+2.1676e+11i
-3.1666e+10+6.3487e+10i
      -2.3070e+10-2.6923e+11i
            +2.5117e+11+1.0070e+11i
                  -7.1277e+11-2.6900e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.6734e+11-7.5541e+11i
      -7.2213e+10-5.8885e+10i
            +8.4902e+10-4.2710e+10i
                  +1.7024e+10+1.0579e+11i
-2.8181e+10+7.1934e+10i
      +2.3143e+12+1.4992e+12i
            +2.8718e+11+8.8129e+10i
                  -2.0570e+11+2.6537e+11i
-3.2585e+10+6.2635e+10i
      -1.8287e+10-2.6858e+11i
            -2.2333e+12+1.4347e+12i
                  -2.0313e+11+2.1682e+11i
-3.1663e+10+6.3498e+10i
```

```
      -2.3102e+10-2.6926e+11i
            +2.5121e+11+1.0069e+11i
                  -7.1319e+11-2.6903e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3003e-13+1.1784e-12i
      -2.9007e-14+3.8076e-14i
            -3.0617e-14+4.0497e-14i
                  -3.2157e-14+4.3027e-14i
+3.6805e-14+1.3438e-14i
      +3.0744e-13-1.8778e-13i
            +4.0715e-14+1.8239e-14i
                  +4.2526e-14+2.1098e-14i
+7.5659e-16-3.7647e-14i
      +2.2111e-15-3.9910e-14i
            -3.0919e-13-2.0982e-13i
                  +5.6522e-15-4.4010e-14i
-3.4264e-14+1.1161e-14i
      -3.6733e-14+1.0554e-14i
            -3.9180e-14+9.7520e-15i
                  -1.0022e-13+3.4185e-13i
Iiw_inv * Iiw =
+1.0000e-00-8.6302e-17i
      +8.6736e-18-8.6736e-18i
            +6.9389e-18+2.6021e-18i
                  +0.0000e+00+2.0817e-17i
-4.3368e-18-5.2042e-18i
      +1.0000e-00-6.9389e-18i
            -2.2551e-17+6.9389e-18i
                  -6.9389e-18+0.0000e+00i
+5.2042e-18-4.1200e-18i
      +3.4694e-18-1.0950e-17i
            +1.0000e-00+4.5103e-17i
                  +0.0000e+00+3.4694e-18i
-3.4694e-18+0.0000e+00i
      +2.7756e-17-3.4694e-18i
            +0.0000e+00+0.0000e+00i
                  +1.0000e+00+4.1633e-17i
Iiw * Iiw_inv =
+1.0000e-00-8.6736e-17i
      +5.4210e-18-1.3010e-18i
            -4.3368e-19-4.3368e-18i
                  +0.0000e+00-8.6736e-19i
+1.3878e-17+1.7347e-18i
      +1.0000e-00-6.9389e-18i
            +2.1684e-17+3.4694e-18i
                  -1.3878e-17-1.3878e-17i
-2.3419e-17+1.3878e-17i
      +1.7347e-17+1.9082e-17i
            +1.0000e-00+4.5103e-17i
                  -6.9389e-18-1.3878e-17i
-6.9389e-18+1.3878e-17i
      +6.9389e-18+1.3878e-17i
            -6.9389e-18+0.0000e+00i
                  +1.0000e-00+4.1633e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-9.9258e-08 -2.5205e-08 +1.3744e-07 -8.7933e-09
+4.5113e-08 +1.9990e-08 +3.6257e-08 -9.9301e-08
```

```
+4.2223e-09 +1.2157e-07 -2.8031e-07 +1.4778e-07
+5.3237e-08 -1.1510e-07 +9.6039e-08 -3.4799e-08
Standard deviation of Lmmeas in henries =
+1.0763e-10 +7.2689e-10 +3.8001e-10 +3.6444e-10
+6.0461e-11 +2.1177e-10 +1.1214e-10 +1.0653e-10
+7.8580e-11 +2.3909e-10 +1.3049e-10 +1.1346e-10
+5.9414e-11 +3.0908e-10 +1.6170e-10 +1.4456e-10
receiver noise index = Lm_Ni =  6.0840e-06
```

Output for location 4:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 4
transmitter current matrix =
+1.0616e+07-3.8087e+06i
      +7.9783e+05-9.9223e+05i
            +5.7708e+05+1.1269e+06i
                  -1.3727e+06+2.3160e+05i
-1.0123e+06-3.8902e+05i
      -2.0189e+07+3.1697e+07i
            -1.1261e+06+3.8188e+06i
                  -3.4615e+06-2.6188e+06i
-8.8272e+05-4.5196e+05i
      +3.6497e+06-2.8668e+05i
            -1.9297e+07-2.9518e+07i
                  -2.8340e+06-2.5947e+06i
-8.9473e+05-4.3843e+05i
      +3.6577e+06-3.5349e+05i
            -1.2976e+06+3.3452e+06i
                  +3.4780e+07-9.5373e+06i
Standard deviation of transmitter current matrix =
+1.2611e+04 +8.4584e+03 +1.2148e+04 +9.6951e+03
+2.3074e+04 +2.3620e+04 +7.5267e+03 +7.1886e+03
+1.0727e+04 +1.4546e+04 +1.8731e+04 +1.1553e+04
+1.0615e+04 +6.1543e+03 +1.0565e+04 +2.1774e+04
transmitter curent noise index CUR_Ni =  1.8701e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.7184e+11-7.5767e+11i
      -7.2721e+10-5.8473e+10i
            +8.4752e+10-4.3401e+10i
                  +1.7862e+10+1.0587e+11i
-2.7765e+10+7.2253e+10i
      +2.3231e+12+1.4796e+12i
            +2.8720e+11+8.4689e+10i
                  -2.0197e+11+2.6697e+11i
-3.2258e+10+6.3002e+10i
      -2.1011e+10-2.6749e+11i
            -2.2200e+12+1.4513e+12i
                  -2.0011e+11+2.1857e+11i
-3.1292e+10+6.3859e+10i
      -2.5908e+10-2.6807e+11i
            +2.5158e+11+9.7586e+10i
                  -7.3556e+11-2.6824e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.7195e+11-7.5772e+11i
      -7.2736e+10-5.8472e+10i
            +8.4758e+10-4.3417e+10i
                  +1.7877e+10+1.0588e+11i
-2.7760e+10+7.2265e+10i
      +2.3235e+12+1.4796e+12i
            +2.8725e+11+8.4666e+10i
                  -2.0197e+11+2.6703e+11i
-3.2255e+10+6.3013e+10i
      -2.1043e+10-2.6752e+11i
            -2.2201e+12+1.4517e+12i
                  -2.0011e+11+2.1863e+11i
-3.1288e+10+6.3870e+10i
```

```
      -2.5940e+10-2.6810e+11i
            +2.5163e+11+9.7569e+10i
                  -7.3598e+11-2.6827e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3325e-13+1.1711e-12i
      -2.9130e-14+3.7883e-14i
            -3.0747e-14+4.0300e-14i
                  -3.2305e-14+4.2814e-14i
+3.6702e-14+1.3602e-14i
      +3.0926e-13-1.8561e-13i
            +4.0598e-14+1.8401e-14i
                  +4.2400e-14+2.1260e-14i
+9.4708e-16-3.7607e-14i
      +2.4079e-15-3.9859e-14i
            -3.0776e-13-2.1249e-13i
                  +5.8532e-15-4.3945e-14i
-3.4275e-14+1.0965e-14i
      -3.6738e-14+1.0348e-14i
            -3.9180e-14+9.5441e-15i
                  -1.0323e-13+3.4122e-13i
Iiw_inv * Iiw =
+1.0000e-00-2.7192e-16i
      +8.6736e-18-8.6736e-19i
            +1.2143e-17+1.0408e-17i
                  -2.7756e-17+6.9389e-18i
+1.3010e-18+4.3368e-18i
      +1.0000e+00-3.4001e-16i
            -1.5613e-17-2.4286e-17i
                  +2.0817e-17+5.5511e-17i
-3.4694e-18-8.6736e-19i
      +5.2042e-18-1.0950e-17i
            +1.0000e-00-1.5439e-16i
                  +0.0000e+00+3.4694e-18i
+0.0000e+00-3.4694e-18i
      +1.3878e-17+2.4286e-17i
            -6.9389e-18+0.0000e+00i
                  +1.0000e+00-7.9797e-17i
Iiw * Iiw_inv =
+1.0000e+00-2.7279e-16i
      -5.6379e-18+8.6736e-19i
            -4.1200e-18-2.6021e-18i
                  +0.0000e+00-4.3368e-18i
-7.8063e-18-8.6736e-18i
      +1.0000e+00-3.3827e-16i
            +1.2143e-17+2.6021e-17i
                  -2.7756e-17-2.7756e-17i
-1.9949e-17+8.6736e-18i
      -2.6888e-17-1.0408e-17i
            +1.0000e-00-1.5439e-16i
                  +6.9389e-18-1.3878e-17i
-1.3878e-17+0.0000e+00i
      -2.7756e-17-1.3878e-17i
            +6.9389e-18-1.3878e-17i
                  +1.0000e+00-7.9797e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-3.2084e-07 -2.6076e-07 +7.9162e-08 +4.6450e-07
+2.5992e-07 -4.0649e-07 -7.2280e-08 +1.6812e-07
```

```
-6.7644e-09 +4.2979e-07 +5.5855e-07 -9.7488e-07
+1.2629e-07 +2.2856e-07 -5.6531e-07 +2.8118e-07
Standard deviation of Lmmeas in henries =
+1.0827e-10 +6.9778e-10 +3.5475e-10 +3.9584e-10
+2.9771e-11 +2.1159e-10 +1.0659e-10 +1.1285e-10
+3.6427e-11 +2.2809e-10 +1.2572e-10 +1.4792e-10
+3.1073e-11 +2.9617e-10 +1.6161e-10 +1.5637e-10
receiver noise index = Lm_Ni =  4.1843e-07
```

Output for location 5:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 5
transmitter current matrix =
+1.0595e+07-3.7672e+06i
      +8.0141e+05-9.8792e+05i
            +5.7128e+05+1.1284e+06i
                  -1.3720e+06+2.2459e+05i
-1.0093e+06-3.9287e+05i
      -2.0357e+07+3.1618e+07i
            -1.1553e+06+3.8189e+06i
                  -3.4476e+06-2.6493e+06i
-8.7912e+05-4.5491e+05i
      +3.6584e+06-2.6283e+05i
            -1.9160e+07-2.9634e+07i
                  -2.8185e+06-2.6190e+06i
-8.9125e+05-4.4184e+05i
      +3.6675e+06-3.2898e+05i
            -1.3238e+06+3.3423e+06i
                  +3.4841e+07-9.3505e+06i
Standard deviation of transmitter current matrix =
+1.8059e+04 +9.7407e+03 +1.0192e+04 +7.8161e+03
+7.8307e+03 +2.7508e+04 +1.0479e+04 +1.0969e+04
+1.2096e+04 +1.1155e+04 +2.5373e+04 +1.5071e+04
+5.7032e+03 +7.5254e+03 +9.1807e+03 +2.2400e+04
transmitter curent noise index CUR_Ni =  1.8873e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.6888e+11-7.5618e+11i
      -7.2405e+10-5.8735e+10i
            +8.4864e+10-4.2965e+10i
                  +1.7322e+10+1.0581e+11i
-2.8040e+10+7.2036e+10i
      +2.3173e+12+1.4920e+12i
            +2.8721e+11+8.6885e+10i
                  -2.0433e+11+2.6589e+11i
-3.2468e+10+6.2745e+10i
      -1.9263e+10-2.6813e+11i
            -2.2287e+12+1.4410e+12i
                  -2.0199e+11+2.1737e+11i
-3.1536e+10+6.3611e+10i
      -2.4111e+10-2.6880e+11i
            +2.5137e+11+9.9560e+10i
                  -7.2116e+11-2.6871e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.6899e+11-7.5624e+11i
      -7.2420e+10-5.8734e+10i
            +8.4870e+10-4.2980e+10i
                  +1.7337e+10+1.0583e+11i
-2.8036e+10+7.2047e+10i
      +2.3178e+12+1.4919e+12i
            +2.8726e+11+8.6861e+10i
                  -2.0432e+11+2.6595e+11i
-3.2465e+10+6.2756e+10i
      -1.9294e+10-2.6816e+11i
            -2.2288e+12+1.4415e+12i
                  -2.0199e+11+2.1743e+11i
-3.1532e+10+6.3622e+10i
```

```
      -2.4143e+10-2.6883e+11i
            +2.5141e+11+9.9543e+10i
                  -7.2158e+11-2.6874e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3119e-13+1.1758e-12i
      -2.9051e-14+3.8012e-14i
            -3.0669e-14+4.0428e-14i
                  -3.2213e-14+4.2953e-14i
+3.6763e-14+1.3502e-14i
      +3.0811e-13-1.8697e-13i
            +4.0669e-14+1.8301e-14i
                  +4.2476e-14+2.1160e-14i
+8.3276e-16-3.7617e-14i
      +2.2877e-15-3.9877e-14i
            -3.0859e-13-2.1078e-13i
                  +5.7302e-15-4.3968e-14i
-3.4268e-14+1.1084e-14i
      -3.6734e-14+1.0476e-14i
            -3.9180e-14+9.6737e-15i
                  -1.0134e-13+3.4164e-13i
Iiw_inv * Iiw =
+1.0000e+00+1.5396e-16i
      +5.2042e-18+1.0408e-17i
            -2.0817e-17+1.7347e-17i
                  -2.7756e-17-2.0817e-17i
+3.9031e-18+4.3368e-18i
      +1.0000e+00-5.3776e-17i
            +1.7347e-17-6.9389e-18i
                  +3.4694e-18+2.7756e-17i
-1.3010e-18+2.3852e-18i
      +8.6736e-18-1.2468e-17i
            +1.0000e+00+2.2551e-17i
                  +0.0000e+00+6.9389e-18i
+0.0000e+00+0.0000e+00i
      -1.3878e-17-6.9389e-18i
            +2.0817e-17+0.0000e+00i
                  +1.0000e+00-4.1633e-17i
Iiw * Iiw_inv =
+1.0000e-00+1.5439e-16i
      -7.1557e-18+2.1684e-18i
            -6.5052e-19-2.6021e-18i
                  +0.0000e+00+1.7347e-18i
+2.6021e-18-1.0408e-17i
      +1.0000e+00-5.7246e-17i
            +0.0000e+00-1.3878e-17i
                  +0.0000e+00+1.3878e-17i
+1.9949e-17-5.2042e-18i
      -3.4694e-18+5.2042e-18i
            +1.0000e+00+2.2551e-17i
                  +0.0000e+00+0.0000e+00i
+1.3878e-17-1.3878e-17i
      -6.9389e-18+0.0000e+00i
            +1.3878e-17+2.7756e-17i
                  +1.0000e+00-3.8164e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-3.0467e-08 -3.7183e-09 +2.8996e-08 +5.7837e-09
+1.3076e-08 -7.4948e-09 +2.5778e-08 -3.1007e-08
```

```
+4.5631e-09 +4.7235e-08 -7.1155e-08 +1.8885e-08
+1.3346e-08 -3.5483e-08 +1.4898e-08 +6.6407e-09
Standard deviation of Lmmeas in henries =
+1.4378e-10 +6.8524e-10 +3.8740e-10 +3.9671e-10
+3.3419e-11 +2.0717e-10 +1.1778e-10 +1.1313e-10
+5.2548e-11 +2.1619e-10 +1.2237e-10 +1.2164e-10
+6.3085e-11 +2.6628e-10 +1.5010e-10 +1.4279e-10
receiver noise index = Lm_Ni =  8.1335e-05
```

Output for location 6:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 6
transmitter current matrix =
+1.0599e+07-3.7738e+06i
      +8.0097e+05-9.8862e+05i
            +5.7231e+05+1.1282e+06i
                  -1.3722e+06+2.2573e+05i
-1.0098e+06-3.9231e+05i
      -2.0331e+07+3.1632e+07i
            -1.1507e+06+3.8190e+06i
                  -3.4500e+06-2.6444e+06i
-8.7965e+05-4.5450e+05i
      +3.6569e+06-2.6654e+05i
            -1.9182e+07-2.9615e+07i
                  -2.8208e+06-2.6150e+06i
-8.9187e+05-4.4131e+05i
      +3.6661e+06-3.3300e+05i
            -1.3196e+06+3.3428e+06i
                  +3.4832e+07-9.3811e+06i
Standard deviation of transmitter current matrix =
+8.8966e+03 +1.0291e+04 +4.7885e+03 +9.5946e+03
+6.0184e+03 +3.1552e+04 +1.3019e+04 +1.5443e+04
+6.6924e+03 +1.1813e+04 +1.6005e+04 +9.1695e+03
+4.2752e+03 +1.2507e+04 +8.9072e+03 +4.7071e+04
transmitter curent noise index CUR_Ni =  1.9318e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.6934e+11-7.5645e+11i
      -7.2456e+10-5.8703e+10i
            +8.4852e+10-4.3042e+10i
                  +1.7410e+10+1.0583e+11i
-2.8000e+10+7.2072e+10i
      +2.3183e+12+1.4900e+12i
            +2.8722e+11+8.6538e+10i
                  -2.0395e+11+2.6608e+11i
-3.2439e+10+6.2783e+10i
      -1.9535e+10-2.6801e+11i
            -2.2273e+12+1.4426e+12i
                  -2.0168e+11+2.1756e+11i
-3.1498e+10+6.3655e+10i
      -2.4406e+10-2.6869e+11i
            +2.5140e+11+9.9241e+10i
                  -7.2352e+11-2.6864e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.6946e+11-7.5651e+11i
      -7.2472e+10-5.8702e+10i
            +8.4858e+10-4.3058e+10i
                  +1.7425e+10+1.0584e+11i
-2.7995e+10+7.2083e+10i
      +2.3187e+12+1.4900e+12i
            +2.8726e+11+8.6515e+10i
                  -2.0394e+11+2.6614e+11i
-3.2435e+10+6.2794e+10i
      -1.9567e+10-2.6804e+11i
            -2.2274e+12+1.4431e+12i
                  -2.0169e+11+2.1761e+11i
-3.1494e+10+6.3666e+10i
```

```
      -2.4438e+10-2.6872e+11i
            +2.5145e+11+9.9224e+10i
                  -7.2394e+11-2.6867e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3148e-13+1.1750e-12i
      -2.9066e-14+3.7991e-14i
            -3.0680e-14+4.0411e-14i
                  -3.2228e-14+4.2932e-14i
+3.6753e-14+1.3518e-14i
      +3.0829e-13-1.8675e-13i
            +4.0658e-14+1.8317e-14i
                  +4.2464e-14+2.1175e-14i
+8.5253e-16-3.7614e-14i
      +2.3076e-15-3.9872e-14i
            -3.0846e-13-2.1105e-13i
                  +5.7501e-15-4.3962e-14i
-3.4268e-14+1.1066e-14i
      -3.6734e-14+1.0456e-14i
            -3.9179e-14+9.6530e-15i
                  -1.0164e-13+3.4156e-13i
Iiw_inv * Iiw =
+1.0000e-00-4.6838e-17i
      +5.2042e-18+8.6736e-19i
            +1.3878e-17-2.2551e-17i
                  +0.0000e+00-1.3878e-17i
+1.3010e-18+2.1684e-18i
      +1.0000e-00-1.0061e-16i
            -1.0408e-17+3.4694e-18i
                  +3.4694e-18+0.0000e+00i
+3.0358e-18+8.6736e-19i
      +3.4694e-18+3.7947e-18i
            +1.0000e-00-7.4593e-17i
                  -1.3878e-17+0.0000e+00i
-3.4694e-18-3.4694e-18i
      +0.0000e+00+1.0408e-17i
            -6.9389e-18+0.0000e+00i
                  +1.0000e+00-5.2042e-17i
Iiw * Iiw_inv =
+1.0000e-00-4.6404e-17i
      +2.1684e-19+1.3010e-18i
            +1.1276e-17+7.8063e-18i
                  -6.9389e-18-6.0715e-18i
+2.6021e-18+1.7347e-18i
      +1.0000e-00-1.0755e-16i
            +6.9389e-18-1.3878e-17i
                  +0.0000e+00-2.7756e-17i
+5.2042e-18-5.2042e-18i
      +6.0715e-18+5.2042e-18i
            +1.0000e-00-7.4593e-17i
                  +0.0000e+00+0.0000e+00i
+6.9389e-18+0.0000e+00i
      +2.0817e-17+0.0000e+00i
            +1.3878e-17+1.3878e-17i
                  +1.0000e+00-4.5103e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-2.1093e-08 -2.4583e-09 +1.6978e-08 +7.0162e-09
+9.2076e-09 -1.5058e-08 +2.2510e-08 -1.6679e-08
```

```
+3.8584e-09 +3.7821e-08 -3.7578e-08 -3.8059e-09
+8.3903e-09 -1.9981e-08 -2.3344e-09 +1.3406e-08
Standard deviation of Lmmeas in henries =
+1.2320e-10 +3.8834e-10 +2.4498e-10 +3.0070e-10
+3.4232e-11 +1.2918e-10 +7.7834e-11 +8.2737e-11
+3.4589e-11 +1.3944e-10 +8.5477e-11 +8.8995e-11
+5.5058e-11 +1.5404e-10 +8.5335e-11 +9.8568e-11
receiver noise index = Lm_Ni =  7.8533e-05
```

Output for location 7:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 7
transmitter current matrix =
+1.0615e+07-3.8085e+06i
      +7.9783e+05-9.9231e+05i
            +5.7735e+05+1.1273e+06i
                  -1.3729e+06+2.3161e+05i
-1.0123e+06-3.8912e+05i
      -2.0190e+07+3.1703e+07i
            -1.1268e+06+3.8207e+06i
                  -3.4622e+06-2.6195e+06i
-8.8273e+05-4.5188e+05i
      +3.6507e+06-2.8670e+05i
            -1.9307e+07-2.9531e+07i
                  -2.8347e+06-2.5955e+06i
-8.9470e+05-4.3858e+05i
      +3.6587e+06-3.5360e+05i
            -1.2984e+06+3.3471e+06i
                  +3.4785e+07-9.5385e+06i
Standard deviation of transmitter current matrix =
+9.7229e+03 +7.5515e+03 +9.3257e+03 +6.5127e+03
+1.1733e+04 +2.9920e+04 +1.5501e+04 +8.5203e+03
+6.3819e+03 +6.4623e+03 +4.1337e+04 +1.2933e+04
+1.3146e+04 +1.7549e+04 +7.7215e+03 +2.7031e+04
transmitter curent noise index CUR_Ni =  2.0693e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.7182e+11-7.5762e+11i
      -7.2726e+10-5.8473e+10i
            +8.4781e+10-4.3421e+10i
                  +1.7863e+10+1.0588e+11i
-2.7772e+10+7.2247e+10i
      +2.3235e+12+1.4797e+12i
            +2.8735e+11+8.4742e+10i
                  -2.0203e+11+2.6702e+11i
-3.2252e+10+6.3003e+10i
      -2.1012e+10-2.6756e+11i
            -2.2209e+12+1.4520e+12i
                  -2.0018e+11+2.1862e+11i
-3.1302e+10+6.3857e+10i
      -2.5915e+10-2.6814e+11i
            +2.5173e+11+9.7648e+10i
                  -7.3566e+11-2.6828e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.7193e+11-7.5768e+11i
      -7.2742e+10-5.8472e+10i
            +8.4787e+10-4.3437e+10i
                  +1.7878e+10+1.0589e+11i
-2.7767e+10+7.2259e+10i
      +2.3239e+12+1.4797e+12i
            +2.8740e+11+8.4718e+10i
                  -2.0203e+11+2.6708e+11i
-3.2249e+10+6.3014e+10i
      -2.1044e+10-2.6759e+11i
            -2.2210e+12+1.4525e+12i
                  -2.0018e+11+2.1868e+11i
-3.1299e+10+6.3868e+10i
```

```
      -2.5948e+10-2.6817e+11i
            +2.5177e+11+9.7630e+10i
                  -7.3608e+11-2.6831e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3327e-13+1.1712e-12i
      -2.9131e-14+3.7880e-14i
            -3.0747e-14+4.0299e-14i
                  -3.2307e-14+4.2814e-14i
+3.6695e-14+1.3608e-14i
      +3.0923e-13-1.8557e-13i
            +4.0593e-14+1.8405e-14i
                  +4.2395e-14+2.1264e-14i
+9.4745e-16-3.7590e-14i
      +2.4117e-15-3.9846e-14i
            -3.0761e-13-2.1240e-13i
                  +5.8554e-15-4.3930e-14i
-3.4274e-14+1.0959e-14i
      -3.6737e-14+1.0345e-14i
            -3.9179e-14+9.5414e-15i
                  -1.0322e-13+3.4117e-13i
Iiw_inv * Iiw =
+1.0000e+00-4.3368e-19i
      -5.2042e-18-1.2143e-17i
            +3.4694e-18+4.2501e-17i
                  -2.7756e-17+6.9389e-18i
+3.4694e-18+4.7705e-18i
      +1.0000e+00-2.0817e-17i
            +3.4694e-18-8.6736e-18i
                  +1.0408e-17-2.7756e-17i
-5.2042e-18+3.2526e-18i
      -1.0408e-17-2.8731e-18i
            +1.0000e+00+7.4593e-17i
                  -2.7756e-17-3.4694e-18i
+0.0000e+00-3.4694e-18i
      +0.0000e+00+3.4694e-18i
            +0.0000e+00-1.3878e-17i
                  +1.0000e+00+8.6736e-17i
Iiw * Iiw_inv =
+1.0000e+00-4.3368e-19i
      -6.5052e-19+4.7705e-18i
            -4.3368e-18-2.6021e-18i
                  -6.9389e-18-3.4694e-18i
-8.6736e-18+5.2042e-18i
      +1.0000e+00-1.9082e-17i
            +2.6021e-18-1.7347e-18i
                  +2.7756e-17+2.7756e-17i
+6.9389e-18+1.5613e-17i
      +2.6021e-18+1.7347e-18i
            +1.0000e+00+7.6328e-17i
                  +6.9389e-18+1.3878e-17i
-6.9389e-18+0.0000e+00i
      -1.3878e-17+0.0000e+00i
            +1.3878e-17+0.0000e+00i
                  +1.0000e+00+8.6736e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-9.3761e-08 -3.5928e-08 +2.7909e-08 +1.0250e-07
+4.9549e-08 -1.1953e-07 -1.5558e-08 +8.1121e-08
```

```
+2.1251e-08 +7.3235e-08 +9.6070e-08 -1.9316e-07
+2.7803e-08 +7.9465e-08 -1.0941e-07 +7.9353e-09
Standard deviation of Lmmeas in henries =
+1.2348e-10 +6.7017e-10 +3.5722e-10 +4.1831e-10
+3.1161e-11 +1.9810e-10 +1.0523e-10 +1.1648e-10
+3.4924e-11 +2.1921e-10 +1.1597e-10 +1.3076e-10
+3.3791e-11 +2.6473e-10 +1.4134e-10 +1.5066e-10
receiver noise index = Lm_Ni =  8.7731e-06
```

Output for location 8:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 8
transmitter current matrix =
+1.0601e+07-3.7798e+06i
      +8.0040e+05-9.8932e+05i
            +5.7301e+05+1.1276e+06i
                  -1.3722e+06+2.2676e+05i
-1.0102e+06-3.9171e+05i
      -2.0304e+07+3.1645e+07i
            -1.1457e+06+3.8177e+06i
                  -3.4519e+06-2.6396e+06i
-8.8016e+05-4.5401e+05i
      +3.6552e+06-2.7030e+05i
            -1.9198e+07-2.9589e+07i
                  -2.8231e+06-2.6110e+06i
-8.9229e+05-4.4084e+05i
      +3.6644e+06-3.3692e+05i
            -1.3151e+06+3.3422e+06i
                  +3.4821e+07-9.4107e+06i
Standard deviation of transmitter current matrix =
+5.7000e+03 +7.9602e+03 +6.7534e+03 +6.7233e+03
+5.3986e+03 +1.8419e+04 +8.5905e+03 +1.5659e+04
+6.5308e+03 +1.4889e+04 +1.8774e+04 +8.3524e+03
+8.2741e+03 +9.9342e+03 +1.0800e+04 +2.8690e+04
transmitter curent noise index CUR_Ni =  1.6230e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.6978e+11-7.5664e+11i
      -7.2507e+10-5.8661e+10i
            +8.4802e+10-4.3094e+10i
                  +1.7489e+10+1.0583e+11i
-2.7958e+10+7.2104e+10i
      +2.3193e+12+1.4881e+12i
            +2.8712e+11+8.6165e+10i
                  -2.0358e+11+2.6623e+11i
-3.2404e+10+6.2819e+10i
      -1.9810e+10-2.6789e+11i
            -2.2253e+12+1.4439e+12i
                  -2.0138e+11+2.1773e+11i
-3.1464e+10+6.3685e+10i
      -2.4693e+10-2.6857e+11i
            +2.5136e+11+9.8904e+10i
                  -7.2580e+11-2.6856e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.6989e+11-7.5670e+11i
      -7.2522e+10-5.8660e+10i
            +8.4808e+10-4.3110e+10i
                  +1.7504e+10+1.0585e+11i
-2.7953e+10+7.2116e+10i
      +2.3197e+12+1.4880e+12i
            +2.8717e+11+8.6141e+10i
                  -2.0357e+11+2.6629e+11i
-3.2401e+10+6.2830e+10i
      -1.9842e+10-2.6792e+11i
            -2.2254e+12+1.4443e+12i
                  -2.0138e+11+2.1778e+11i
-3.1461e+10+6.3696e+10i
```

```
      -2.4725e+10-2.6860e+11i
            +2.5140e+11+9.8887e+10i
                  -7.2622e+11-2.6859e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3181e-13+1.1744e-12i
      -2.9079e-14+3.7973e-14i
            -3.0691e-14+4.0390e-14i
                  -3.2244e-14+4.2911e-14i
+3.6743e-14+1.3535e-14i
      +3.0848e-13-1.8653e-13i
            +4.0644e-14+1.8333e-14i
                  +4.2449e-14+2.1192e-14i
+8.7030e-16-3.7622e-14i
      +2.3278e-15-3.9877e-14i
            -3.0842e-13-2.1138e-13i
                  +5.7716e-15-4.3969e-14i
-3.4271e-14+1.1044e-14i
      -3.6734e-14+1.0435e-14i
            -3.9179e-14+9.6316e-15i
                  -1.0195e-13+3.4151e-13i
Iiw_inv * Iiw =
+1.0000e-00-8.5435e-17i
      -2.0817e-17-5.2042e-18i
            +6.9389e-18+1.6480e-17i
                  +0.0000e+00+1.3878e-17i
+3.4694e-18-8.6736e-19i
      +1.0000e-00-1.0235e-16i
            +0.0000e+00-1.7347e-17i
                  +2.0817e-17+2.7756e-17i
+2.6021e-18-1.7347e-18i
      -1.2143e-17-1.0083e-17i
            +1.0000e-00+7.8063e-17i
                  +0.0000e+00+6.9389e-18i
-3.4694e-18-3.4694e-18i
      +1.3878e-17+1.0408e-17i
            -6.9389e-18+1.3878e-17i
                  +1.0000e+00+5.5511e-17i
Iiw * Iiw_inv =
+1.0000e-00-8.5001e-17i
      -7.5894e-18+3.9031e-18i
            -8.8905e-18-5.2042e-18i
                  -6.9389e-18-1.7347e-18i
+6.0715e-18+1.9082e-17i
      +1.0000e-00-1.0582e-16i
            +1.4745e-17-8.6736e-18i
                  -2.7756e-17-1.3878e-17i
-1.9949e-17+1.5613e-17i
      +4.3368e-18-1.9082e-17i
            +1.0000e-00+7.9797e-17i
                  -6.9389e-18+0.0000e+00i
-1.3878e-17+0.0000e+00i
      -1.3878e-17-1.3878e-17i
            +6.9389e-18+0.0000e+00i
                  +1.0000e+00+5.2042e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-1.3642e-08 -1.1295e-09 +9.4258e-09 +5.8821e-09
+5.8810e-09 -1.4146e-08 +1.4584e-08 -6.4596e-09
```

```
+3.0461e-09 +2.4475e-08 -1.6578e-08 -1.0606e-08
+5.0224e-09 -9.0940e-09 -7.3305e-09 +1.1174e-08
Standard deviation of Lmmeas in henries =
+1.7599e-10 +3.3374e-10 +2.2084e-10 +2.3645e-10
+4.2737e-11 +1.2825e-10 +8.0538e-11 +8.4509e-11
+6.1582e-11 +1.2433e-10 +7.6005e-11 +8.0405e-11
+8.3790e-11 +1.4817e-10 +8.9693e-11 +9.7462e-11
receiver noise index = Lm_Ni =  1.7070e-04
```

Output for location 9:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 9
transmitter current matrix =
+1.0605e+07-3.7878e+06i
      +7.9967e+05-9.9018e+05i
            +5.7437e+05+1.1278e+06i
                  -1.3724e+06+2.2809e+05i
-1.0108e+06-3.9102e+05i
      -2.0271e+07+3.1659e+07i
            -1.1406e+06+3.8192e+06i
                  -3.4545e+06-2.6338e+06i
-8.8081e+05-4.5354e+05i
      +3.6539e+06-2.7481e+05i
            -1.9232e+07-2.9578e+07i
                  -2.8261e+06-2.6067e+06i
-8.9294e+05-4.4021e+05i
      +3.6627e+06-3.4147e+05i
            -1.3106e+06+3.3441e+06i
                  +3.4810e+07-9.4455e+06i
Standard deviation of transmitter current matrix =
+1.6741e+04 +8.4902e+03 +1.0862e+04 +8.4054e+03
+8.2306e+03 +3.2690e+04 +1.1439e+04 +1.0976e+04
+1.6327e+04 +1.1611e+04 +3.4102e+04 +9.7319e+03
+1.0059e+04 +1.6794e+04 +1.1262e+04 +2.4996e+04
transmitter curent noise index CUR_Ni =  2.1709e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.7034e+11-7.5690e+11i
      -7.2570e+10-5.8608e+10i
            +8.4818e+10-4.3197e+10i
                  +1.7591e+10+1.0584e+11i
-2.7908e+10+7.2143e+10i
      +2.3203e+12+1.4857e+12i
            +2.8723e+11+8.5783e+10i
                  -2.0313e+11+2.6643e+11i
-3.2370e+10+6.2866e+10i
      -2.0141e+10-2.6779e+11i
            -2.2245e+12+1.4464e+12i
                  -2.0104e+11+2.1796e+11i
-3.1419e+10+6.3732e+10i
      -2.5026e+10-2.6844e+11i
            +2.5150e+11+9.8570e+10i
                  -7.2849e+11-2.6847e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.7046e+11-7.5696e+11i
      -7.2585e+10-5.8607e+10i
            +8.4824e+10-4.3213e+10i
                  +1.7606e+10+1.0586e+11i
-2.7903e+10+7.2154e+10i
      +2.3208e+12+1.4856e+12i
            +2.8728e+11+8.5760e+10i
                  -2.0313e+11+2.6649e+11i
-3.2367e+10+6.2877e+10i
      -2.0173e+10-2.6782e+11i
            -2.2246e+12+1.4468e+12i
                  -2.0104e+11+2.1802e+11i
-3.1415e+10+6.3742e+10i
```

```
      -2.5059e+10-2.6847e+11i
            +2.5155e+11+9.8553e+10i
                  -7.2891e+11-2.6850e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3223e-13+1.1735e-12i
      -2.9095e-14+3.7950e-14i
            -3.0709e-14+4.0369e-14i
                  -3.2263e-14+4.2885e-14i
+3.6731e-14+1.3558e-14i
      +3.0870e-13-1.8627e-13i
            +4.0632e-14+1.8354e-14i
                  +4.2435e-14+2.1213e-14i
+8.9639e-16-3.7608e-14i
      +2.3515e-15-3.9863e-14i
            -3.0814e-13-2.1163e-13i
                  +5.7946e-15-4.3950e-14i
-3.4273e-14+1.1020e-14i
      -3.6737e-14+1.0410e-14i
            -3.9180e-14+9.6071e-15i
                  -1.0230e-13+3.4143e-13i
Iiw_inv * Iiw =
+1.0000e-00+1.0192e-16i
      +0.0000e+00+1.9949e-17i
            -1.3878e-17+1.2143e-17i
                  +2.7756e-17+1.3878e-17i
-1.3010e-18+4.3368e-19i
      +1.0000e+00+6.2450e-17i
            +0.0000e+00+0.0000e+00i
                  -6.9389e-18+5.5511e-17i
+1.3010e-18-2.1684e-19i
      -1.7347e-17+1.2848e-17i
            +1.0000e+00+4.6838e-17i
                  +2.7756e-17-1.7347e-17i
+0.0000e+00+0.0000e+00i
      -2.7756e-17-1.3878e-17i
            +0.0000e+00+0.0000e+00i
                  +1.0000e-00-2.0817e-17i
Iiw * Iiw_inv =
+1.0000e-00+1.0322e-16i
      -5.8547e-18-2.1684e-18i
            +1.3010e-18-1.7347e-18i
                  +1.3878e-17+8.6736e-18i
+7.8063e-18-1.0408e-17i
      +1.0000e+00+5.8981e-17i
            +8.6736e-19-1.9082e-17i
                  -1.3878e-17+0.0000e+00i
+3.4694e-18+5.2042e-18i
      +8.6736e-19-5.2042e-18i
            +1.0000e+00+4.5103e-17i
                  -1.3878e-17+0.0000e+00i
+6.9389e-18+0.0000e+00i
      +1.3878e-17+1.3878e-17i
            +0.0000e+00-1.3878e-17i
                  +1.0000e-00-1.3878e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-2.0755e-08 -3.1579e-09 +1.3354e-08 +1.1362e-08
+9.2043e-09 -2.6214e-08 +1.8428e-08 -1.9344e-09
```

```
+4.5654e-09 +3.5279e-08 -1.2192e-08 -2.6954e-08
+7.4979e-09 -5.7705e-09 -1.9246e-08 +1.7254e-08
Standard deviation of Lmmeas in henries =
+1.7191e-10 +5.3464e-10 +3.2582e-10 +3.7447e-10
+5.3678e-11 +1.6635e-10 +9.8771e-11 +1.0861e-10
+9.4139e-11 +1.7761e-10 +1.0648e-10 +1.1670e-10
+8.0851e-11 +2.1304e-10 +1.2798e-10 +1.3880e-10
receiver noise index = Lm_Ni =  1.6035e-04
```

Output for location 10:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 10
transmitter current matrix =
+1.0607e+07-3.7931e+06i
      +7.9922e+05-9.9061e+05i
            +5.7503e+05+1.1276e+06i
                  -1.3725e+06+2.2899e+05i
-1.0111e+06-3.9046e+05i
      -2.0250e+07+3.1669e+07i
            -1.1371e+06+3.8191e+06i
                  -3.4562e+06-2.6301e+06i
-8.8123e+05-4.5304e+05i
      +3.6526e+06-2.7763e+05i
            -1.9249e+07-2.9564e+07i
                  -2.8278e+06-2.6036e+06i
-8.9342e+05-4.3976e+05i
      +3.6613e+06-3.4440e+05i
            -1.3075e+06+3.3445e+06i
                  +3.4802e+07-9.4678e+06i
Standard deviation of transmitter current matrix =
+9.9754e+03 +9.4465e+03 +8.0659e+03 +8.7013e+03
+9.3241e+03 +1.1403e+04 +1.2251e+04 +8.0258e+03
+1.9957e+04 +7.1959e+03 +1.3663e+04 +1.2426e+04
+1.0719e+04 +8.3247e+03 +1.3504e+04 +2.7550e+04
transmitter curent noise index CUR_Ni =  1.7043e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.7072e+11-7.5708e+11i
      -7.2602e+10-5.8575e+10i
            +8.4803e+10-4.3246e+10i
                  +1.7661e+10+1.0585e+11i
-2.7868e+10+7.2166e+10i
      +2.3210e+12+1.4841e+12i
            +2.8723e+11+8.5516e+10i
                  -2.0284e+11+2.6656e+11i
-3.2335e+10+6.2896e+10i
      -2.0348e+10-2.6770e+11i
            -2.2235e+12+1.4477e+12i
                  -2.0080e+11+2.1810e+11i
-3.1387e+10+6.3766e+10i
      -2.5241e+10-2.6834e+11i
            +2.5153e+11+9.8335e+10i
                  -7.3020e+11-2.6841e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.7084e+11-7.5714e+11i
      -7.2617e+10-5.8574e+10i
            +8.4809e+10-4.3262e+10i
                  +1.7676e+10+1.0587e+11i
-2.7863e+10+7.2177e+10i
      +2.3215e+12+1.4840e+12i
            +2.8727e+11+8.5492e+10i
                  -2.0284e+11+2.6662e+11i
-3.2332e+10+6.2907e+10i
      -2.0380e+10-2.6773e+11i
            -2.2236e+12+1.4481e+12i
                  -2.0080e+11+2.1815e+11i
-3.1383e+10+6.3777e+10i
```

```
      -2.5273e+10-2.6837e+11i
            +2.5157e+11+9.8318e+10i
                  -7.3062e+11-2.6844e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3250e-13+1.1729e-12i
      -2.9106e-14+3.7930e-14i
            -3.0719e-14+4.0349e-14i
                  -3.2273e-14+4.2869e-14i
+3.6722e-14+1.3567e-14i
      +3.0885e-13-1.8610e-13i
            +4.0622e-14+1.8366e-14i
                  +4.2425e-14+2.1224e-14i
+9.0694e-16-3.7602e-14i
      +2.3668e-15-3.9858e-14i
            -3.0803e-13-2.1183e-13i
                  +5.8086e-15-4.3944e-14i
-3.4274e-14+1.1008e-14i
      -3.6737e-14+1.0394e-14i
            -3.9180e-14+9.5913e-15i
                  -1.0253e-13+3.4138e-13i
Iiw_inv * Iiw =
+1.0000e-00+1.2273e-16i
      +3.2960e-17-6.0715e-18i
            +1.0408e-17+2.6888e-17i
                  +0.0000e+00+0.0000e+00i
+3.0358e-18+1.2577e-17i
      +1.0000e+00+1.1623e-16i
            +2.2551e-17-1.0408e-17i
                  +3.4694e-18+2.7756e-17i
+5.6379e-18-2.1684e-18i
      +1.7347e-18+4.2826e-18i
            +1.0000e+00+1.5959e-16i
                  +2.7756e-17-6.9389e-18i
+0.0000e+00+0.0000e+00i
      -1.3878e-17+0.0000e+00i
            +2.0817e-17-1.3878e-17i
                  +1.0000e-00+7.9797e-17i
Iiw * Iiw_inv =
+1.0000e+00+1.2317e-16i
      -2.1250e-17-5.6379e-18i
            -1.4528e-17+8.6736e-19i
                  +0.0000e+00-8.6736e-19i
-1.3010e-17+0.0000e+00i
      +1.0000e+00+1.1970e-16i
            -9.5410e-18-1.5613e-17i
                  +0.0000e+00+2.7756e-17i
-4.3368e-18-5.2042e-18i
      -2.0817e-17-1.5613e-17i
            +1.0000e+00+1.5786e-16i
                  -1.3878e-17-1.3878e-17i
+0.0000e+00+0.0000e+00i
      -1.3878e-17-2.7756e-17i
            +6.9389e-18+1.3878e-17i
                  +1.0000e-00+7.9797e-17i
Real measured mutual-inductance matrix Lmmeas in henries =
-2.9653e-08 -5.9648e-09 +1.5099e-08 +2.1008e-08
+1.3513e-08 -4.1312e-08 +1.3668e-08 +1.3176e-08
```

```
+6.8234e-09 +3.8926e-08 +7.0719e-09 -5.2257e-08
+1.0002e-08 +8.1108e-09 -3.5406e-08 +1.7473e-08
Standard deviation of Lmmeas in henries =
+1.5213e-10 +3.8079e-10 +2.2307e-10 +2.7313e-10
+6.0432e-11 +1.1711e-10 +6.8535e-11 +7.7697e-11
+1.0216e-10 +1.2432e-10 +7.3695e-11 +8.2410e-11
+8.3645e-11 +1.5490e-10 +9.1084e-11 +1.0553e-10
receiver noise index = Lm_Ni =  4.1150e-05
```

Output for location 11:

```
curre136.cpp rev17, input data point number 1 through 11
This run is for data point number 11
transmitter current matrix =
+1.0609e+07-3.7974e+06i
      +7.9868e+05-9.9099e+05i
            +5.7563e+05+1.1273e+06i
                  -1.3725e+06+2.2976e+05i
-1.0113e+06-3.9000e+05i
      -2.0230e+07+3.1677e+07i
            -1.1338e+06+3.8190e+06i
                  -3.4576e+06-2.6267e+06i
-8.8165e+05-4.5263e+05i
      +3.6515e+06-2.8035e+05i
            -1.9264e+07-2.9550e+07i
                  -2.8295e+06-2.6008e+06i
-8.9373e+05-4.3933e+05i
      +3.6602e+06-3.4728e+05i
            -1.3045e+06+3.3448e+06i
                  +3.4794e+07-9.4883e+06i
Standard deviation of transmitter current matrix =
+8.5477e+03 +9.7534e+03 +6.0413e+03 +7.2925e+03
+7.7357e+03 +2.8193e+04 +6.4868e+03 +1.5628e+04
+1.1622e+04 +1.0331e+04 +2.8540e+04 +1.0952e+04
+7.4453e+03 +1.1291e+04 +1.0232e+04 +2.1086e+04
transmitter curent noise index CUR_Ni =  1.7998e-10
(cable-uncorrected jw * transmitter current) matrix =
-2.7103e+11-7.5719e+11i
      -7.2630e+10-5.8536e+10i
            +8.4783e+10-4.3292e+10i
                  +1.7721e+10+1.0585e+11i
-2.7835e+10+7.2178e+10i
      +2.3216e+12+1.4826e+12i
            +2.8722e+11+8.5269e+10i
                  -2.0258e+11+2.6667e+11i
-3.2305e+10+6.2926e+10i
      -2.0547e+10-2.6762e+11i
            -2.2224e+12+1.4488e+12i
                  -2.0059e+11+2.1823e+11i
-3.1356e+10+6.3788e+10i
      -2.5452e+10-2.6826e+11i
            +2.5156e+11+9.8107e+10i
                  -7.3179e+11-2.6835e+12i
(cable-corrected jw * transmitter current) matrix Iiw =
-2.7114e+11-7.5725e+11i
      -7.2645e+10-5.8535e+10i
            +8.4788e+10-4.3307e+10i
                  +1.7736e+10+1.0587e+11i
-2.7831e+10+7.2190e+10i
      +2.3220e+12+1.4826e+12i
            +2.8727e+11+8.5245e+10i
                  -2.0258e+11+2.6673e+11i
-3.2302e+10+6.2937e+10i
      -2.0579e+10-2.6765e+11i
            -2.2225e+12+1.4493e+12i
                  -2.0059e+11+2.1828e+11i
-3.1353e+10+6.3799e+10i
```

```
      -2.5485e+10-2.6829e+11i
            +2.5160e+11+9.8089e+10i
                  -7.3221e+11-2.6838e+12i
(cable-corrected jw * transmitter current).inv matrix Iiw_inv =
-4.3276e-13+1.1725e-12i
      -2.9114e-14+3.7916e-14i
            -3.0729e-14+4.0335e-14i
                  -3.2285e-14+4.2856e-14i
+3.6714e-14+1.3581e-14i
      +3.0899e-13-1.8595e-13i
            +4.0615e-14+1.8380e-14i
                  +4.2419e-14+2.1239e-14i
+9.1736e-16-3.7603e-14i
      +2.3812e-15-3.9857e-14i
            -3.0794e-13-2.1203e-13i
                  +5.8237e-15-4.3943e-14i
-3.4276e-14+1.0996e-14i
      -3.6739e-14+1.0381e-14i
            -3.9182e-14+9.5777e-15i
                  -1.0275e-13+3.4135e-13i
Iiw_inv * Iiw =
+1.0000e+00+2.3419e-16i
      +1.5613e-17-6.0715e-18i
            -3.4694e-18-1.7347e-18i
                  -2.7756e-17-6.9389e-18i
-8.6736e-19-3.0358e-18i
      +1.0000e+00+2.3419e-16i
            +3.2960e-17+5.2042e-18i
                  -3.4694e-18+0.0000e+00i
+7.8063e-18-3.4694e-18i
      -1.7347e-17+1.4311e-17i
            +1.0000e+00+1.2837e-16i
                  +0.0000e+00-1.7347e-17i
+3.4694e-18-3.4694e-18i
      +1.3878e-17+0.0000e+00i
            +0.0000e+00-1.3878e-17i
                  +1.0000e+00+2.1511e-16i
Iiw * Iiw_inv =
+1.0000e+00+2.3462e-16i
      +4.3368e-18-1.3010e-18i
            -9.9747e-18+0.0000e+00i
                  -6.9389e-18-2.6021e-18i
+0.0000e+00-1.0408e-17i
      +1.0000e+00+2.3592e-16i
            -2.0817e-17-3.1225e-17i
                  +1.3878e-17+1.3878e-17i
-5.2042e-18-3.2960e-17i
      +6.9389e-18-1.5613e-17i
            +1.0000e+00+1.3357e-16i
                  +6.9389e-18+1.3878e-17i
+0.0000e+00-1.3878e-17i
      -6.9389e-18-1.3878e-17i
            -6.9389e-18-1.3878e-17i
                  +1.0000e+00+2.1164e-16i
Real measured mutual-inductance matrix Lmmeas in henries =
-3.5812e-08 -7.0365e-09 +1.1340e-08 +3.1989e-08
+1.6518e-08 -4.8448e-08 -5.3787e-09 +3.6407e-08
```

```
+9.6299e-09 +2.2118e-08 +3.1203e-08 -6.3561e-08
+1.0515e-08 +3.2706e-08 -3.7457e-08 -4.8120e-09
Standard deviation of Lmmeas in henries =
+1.6831e-10 +4.2066e-10 +2.4338e-10 +2.9754e-10
+4.5744e-11 +1.1046e-10 +6.5059e-11 +7.2707e-11
+1.0008e-10 +1.3398e-10 +7.5801e-11 +8.9677e-11
+7.8955e-11 +1.7590e-10 +9.7438e-11 +1.1078e-10
receiver noise index = Lm_Ni =  3.2438e-05
```

Appendix R

C++ code for calculating orientation errors

```
/* curre137.cpp data reduction for thesis data point:
calculates orientation errors of data points

   rev1, 2aug2001pta: based on curre136.cpp rev14
   rev2, 2aug2001pta
   rev3, 27aug2001pta

input from stdin:

  none

output to stdout:

  orientation errors

   */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <ptacompl.h>
#include <ptaquats.h>

/* function to print a quaternion */
Quatprint(Quaternion R){
  printf("%+11.4e ",  R.r);
  printf("%+11.4e ",  R.i);
  printf("%+11.4e ",  R.j);
  printf("%+11.4e\n", R.k);
}

main(int argc, char *argv[]){

int i,j,k;
double twopi, Aerr[12];
Quaternion Onom, Ofit[12], Oerr[12];

/* Mechanical starting orientation for seed */
twopi = 8.0 * atan(1.0);
```

```
Onom.i = 0.0;
Onom.j = 0.0;
Onom.k = sin(twopi * 5.0/24.0);
Onom.r = 0.0;
Onom.r = sqrt(1.0-(2|Onom));

Ofit[1].r = .26471;
Ofit[1].i = .0021056;
Ofit[1].j = -.00029468;
Ofit[1].k = 0.0;
Ofit[1].k = sqrt(1.0-(2|Ofit[1]));

Ofit[2].r = .26408;
Ofit[2].i = .0099521;
Ofit[2].j = .0024132;
Ofit[2].k = 0.0;
Ofit[2].k = sqrt(1.0-(2|Ofit[2]));

Ofit[3].r = .26230;
Ofit[3].i = .0051748;
Ofit[3].j = .0008926;
Ofit[3].k = 0.0;
Ofit[3].k = sqrt(1.0-(2|Ofit[3]));

Ofit[4].r = .26334;
Ofit[4].i = .0016817;
Ofit[4].j = -.0012189;
Ofit[4].k = 0.0;
Ofit[4].k = sqrt(1.0-(2|Ofit[4]));

Ofit[5].r = .26553;
Ofit[5].i = .010655;
Ofit[5].j = .0014287;
Ofit[5].k = 0.0;
Ofit[5].k = sqrt(1.0-(2|Ofit[5]));

Ofit[6].r = .26433;
Ofit[6].i = .010005;
Ofit[6].j = .00048334;
Ofit[6].k = 0.0;
Ofit[6].k = sqrt(1.0-(2|Ofit[6]));

Ofit[7].r = .26352;
Ofit[7].i = .0011361;
Ofit[7].j = -.0012912;
Ofit[7].k = 0.0;
Ofit[7].k = sqrt(1.0-(2|Ofit[7]));

Ofit[8].r = .26417;
Ofit[8].i = .0075366;
Ofit[8].j = -.0026992;
Ofit[8].k = 0.0;
Ofit[8].k = sqrt(1.0-(2|Ofit[8]));

Ofit[9].r = .26246;
Ofit[9].i = .0041967;
Ofit[9].j = -.003265;
```

```
Ofit[9].k = 0.0;
Ofit[9].k = sqrt(1.0-(2|Ofit[9]));

Ofit[10].r = .26219;
Ofit[10].i = .0024846;
Ofit[10].j = -.0015482;
Ofit[10].k = 0.0;
Ofit[10].k = sqrt(1.0-(2|Ofit[10]));

Ofit[11].r = .26227;
Ofit[11].i = .0013728;
Ofit[11].j = -.0011464;
Ofit[11].k = 0.0;
Ofit[11].k = sqrt(1.0-(2|Ofit[11]));

printf("curre137.cpp rev3\n");
printf("\n    Onom = ");
Quatprint(Onom);
printf("\n");
for(i=1;i<12;i++){
   printf("Ofit[%2d] = ",i);
   Quatprint(Ofit[i]);
}
printf("\n");
for(i=1;i<12;i++){
   Oerr[i] = Ofit[i]/Onom;
   printf("Oerr[%2d] = ",i);
   Quatprint(Oerr[i]);
}
printf("\n");
for(i=1;i<12;i++){
   Oerr[i] = Ofit[i]/Onom;
   Aerr[i] = 2.0*sqrt(Oerr[i].i*Oerr[i].i
     +Oerr[i].j*Oerr[i].j+Oerr[i].k*Oerr[i].k);
   printf("Aerr[%2d] = %11.4e\n",i,Aerr[i]);
}

}
```

Appendix S

Orientation errors results

The results of running the code in the preceding appendix appear on the following page.

`Onom` is the nominal ideal mechanical orientation quaternion.

`Ofit` is the fitted orientation quaternion from the third fit.

`Oerr` is the quaternion orientation error between `Ofit` and `Onom`.

`Aerr` is the magnitude of the rotation in radians expressed by `Oerr`.

```
curre137.cpp rev3

    Onom = +2.5882e-01 +0.0000e+00 +0.0000e+00 +9.6593e-01

Ofit[ 1] = +2.6471e-01 +2.1056e-03 -2.9468e-04 +9.6433e-01
Ofit[ 2] = +2.6408e-01 +9.9521e-03 +2.4132e-03 +9.6445e-01
Ofit[ 3] = +2.6230e-01 +5.1748e-03 +8.9260e-04 +9.6497e-01
Ofit[ 4] = +2.6334e-01 +1.6817e-03 -1.2189e-03 +9.6470e-01
Ofit[ 5] = +2.6553e-01 +1.0655e-02 +1.4287e-03 +9.6404e-01
Ofit[ 6] = +2.6433e-01 +1.0005e-02 +4.8334e-04 +9.6438e-01
Ofit[ 7] = +2.6352e-01 +1.1361e-03 -1.2912e-03 +9.6465e-01
Ofit[ 8] = +2.6417e-01 +7.5366e-03 -2.6992e-03 +9.6444e-01
Ofit[ 9] = +2.6246e-01 +4.1967e-03 -3.2650e-03 +9.6493e-01
Ofit[10] = +2.6219e-01 +2.4846e-03 -1.5482e-03 +9.6501e-01
Ofit[11] = +2.6227e-01 +1.3728e-03 -1.1464e-03 +9.6499e-01

Oerr[ 1] = +9.9998e-01 +8.2961e-04 +1.9576e-03 -6.1044e-03
Oerr[ 2] = +9.9993e-01 +2.4482e-04 +1.0238e-02 -5.4646e-03
Oerr[ 3] = +9.9998e-01 +4.7715e-04 +5.2295e-03 -3.6092e-03
Oerr[ 4] = +9.9999e-01 +1.6126e-03 +1.3089e-03 -4.6840e-03
Oerr[ 5] = +9.9992e-01 +1.3777e-03 +1.0662e-02 -6.9697e-03
Oerr[ 6] = +9.9993e-01 +2.1226e-03 +9.7892e-03 -5.7232e-03
Oerr[ 7] = +9.9999e-01 +1.5412e-03 +7.6320e-04 -4.8704e-03
Oerr[ 8] = +9.9995e-01 +4.5578e-03 +6.5812e-03 -5.5524e-03
Oerr[ 9] = +9.9998e-01 +4.2399e-03 +3.2087e-03 -3.7751e-03
Oerr[10] = +9.9999e-01 +2.1385e-03 +1.9992e-03 -3.4927e-03
Oerr[11] = +9.9999e-01 +1.4626e-03 +1.0293e-03 -3.5748e-03

Aerr[ 1] =  1.2928e-02
Aerr[ 2] =  2.3215e-02
Aerr[ 3] =  1.2744e-02
Aerr[ 4] =  1.0248e-02
Aerr[ 5] =  2.5624e-02
Aerr[ 6] =  2.3073e-02
Aerr[ 7] =  1.0330e-02
Aerr[ 8] =  1.9485e-02
Aerr[ 9] =  1.3042e-02
Aerr[10] =  9.1146e-03
Aerr[11] =  7.9946e-03
```