NUREG/CR-6455
ORNL/TM-13206

Data Analysis for Steam Generator Tubing Samples

Prepared by
C. V. Dodd

Oak Ridge National Laboratory

Prepared for
U.S. Nuclear Regulatory Commission

AVAILABILITY NOTICE

Availability of Reference Materials Cited in NRC Publications

Most documents cited in NRC publications will be available from one of the following sources:

1. The NRC Public Document Room, 2120 L Street, NW., Lower Level, Washington, DC 20555-0001

2. The Superintendent of Documents, U.S. Government Printing Office, P. O. Box 37082, Washington, DC 20402-9328

3. The National Technical Information Service, Springfield, VA 22161-0002

Although the listing that follows represents the majority of documents cited in NRC publications, it is not intended to be exhaustive.

Referenced documents available for inspection and copying for a fee from the NRC Public Document Room include NRC correspondence and internal NRC memoranda; NRC bulletins, circulars, information notices, inspection and investigation notices; licensee event reports; vendor reports and correspondence; Commission papers; and applicant and licensee documents and correspondence.

The following documents in the NUREG series are available for purchase from the Government Printing Office: formal NRC staff and contractor reports, NRC-sponsored conference proceedings, international agreement reports, grantee reports, and NRC booklets and brochures. Also available are regulatory guides, NRC regulations in the *Code of Federal Regulations*, and *Nuclear Regulatory Commission Issuances*.

Documents available from the National Technical Information Service include NUREG-series reports and technical reports prepared by other Federal agencies and reports prepared by the Atomic Energy Commission, forerunner agency to the Nuclear Regulatory Commission.

Documents available from public and special technical libraries include all open literature items, such as books, journal articles, and transactions. *Federal Register* notices, Federal and State legislation, and congressional reports can usually be obtained from these libraries.

Documents such as theses, dissertations, foreign reports and translations, and non-NRC conference proceedings are available for purchase from the organization sponsoring the publication cited.

Single copies of NRC draft reports are available free, to the extent of supply, upon written request to the Office of Administration, Distribution and Mail Services Section, U.S. Nuclear Regulatory Commission, Washington, DC 20555-0001.

Copies of industry codes and standards used in a substantive manner in the NRC regulatory process are maintained at the NRC Library, Two White Flint North, 11545 Rockville Pike, Rockville, MD 20852-2738, for use by the public. Codes and standards are usually copyrighted and may be purchased from the originating organization or, if they are American National Standards, from the American National Standards Institute, 1430 Broadway, New York, NY 10018-3308.

NUREG/CR-6455
ORNL/TM-13206

Data Analysis for Steam Generator Tubing Samples

Manuscript Completed: January 1996
Date Published: July 1996

Prepared by
C. V. Dodd

Oak Ridge National Laboratory
Managed by Lockheed Martin Energy Research Corporation

Oak Ridge National Laboratory
Oak Ridge, TN 37831-6285

J. Muscara, NRC Project Manager

Prepared for
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001
NRC Job Code B0417

Abstract

The objective of the Improved Eddy-Current ISI for Steam Generators program is to upgrade and validate eddy-current inspections, including probes, instrumentation, and data processing techniques for inservice inspection of new, used, and repaired steam generator tubes; to improve defect detection, classification and characterization as affected by diameter and thickness variations, denting, probe wobble, tube sheet, tube supports, copper and sludge deposits, even when defect types and other variables occur in combination; to transfer this advanced technology to NRC's mobile NDE laboratory and staff. This report provides a description of the application of advanced eddy-current neural network analysis methods for the detection and evaluation of common steam generator tubing flaws including axial and circumferential outer-diameter stress-corrosion cracking and intergranular attack. The report describes the training of the neural networks on tubing samples with known defects and the subsequent evaluation results for unknown samples. Evaluations were done in the presence of artifacts. Computer programs are given in the appendix.

Contents

	Page
Abstract	iii
Introduction	1
Probe Design	2
Eddy-Current Data Acquisition	3
Metallographic Data Acquisition	5
Data Fitting on the Initial Training Set	5
Blind Test Results	6
Additional Training Results	7
Comparison of Eddy-Current and Metallographic Results	7
Summary and Conclusions	14
References	17
Appendix A: Computer Codes Used	A-1
Program bpnd.c	A-3
Program defplot.c	A-19
Program defplotm.c	A-28
Data File "samplename.dat"	A-36
Program metfile.c	A-36
Program caidepth.ma	A-37

Figures

1	Scan of a 40% electrodischarge machined notch with the P90 spherical probe	2
2	P90 Spherical probe	3
3	Circumferential notch standard used for training the neural network	4
4	Scanning of samples	4
5	A simplified neural network	5
6	Defect depth from eddy-current readings for C-outside diameter stress-corrosion cracking defect	10
7	Plot of metallographic data for C-outside diameter stress-corrosion cracking defect	10
8	Plot of eddy-current depth for intergranular attack defect	11
9	Depth from eddy-current readings for axial outside diameter stress-corrosion cracking defect	11

Page

10 Plot of metallographic data for an axial outside diameter stress-corrosion cracking defect 12

11 Calculated defect depth for axial outside diameter stress-corrosion cracking 12

12 Plot of metallographic data for axial outside diameter stress-corrosion cracking 13

13 Property magnetite computed from the eddy-current readings 13

14 Calculated defect depth with magnetite present 15

15 Calculated defect depth without magnetite 15

Tables

1 Eddy-current and metallographic depths 7

2 Summary of data points used for training 8

3 Eddy-current and metallographic data 16

A-1 Interactive commands for running the neural network training program A-3

Data Analysis for Steam Generator Tubing Samples*

C. V. Dodd

Introduction

The major cause of plant downtime in pressurized-water reactors (PWRs) is degradation in the tubing in steam generators. Due to the high inspection speeds, eddy-current testing has been the primary inspection tool for testing of steam generator tubing. The Nuclear Regulatory Commission has funded development work since 1977 at the Oak Ridge National Laboratory (ORNL) to improve the eddy-current inspection of the tubes in steam generators.

Recent developments have concentrated on the improvement of probes and data analysis methods for eddy-current testing. In order to test and further develop the eddy-current inspection methods developed at ORNL, a set of tubing samples was fabricated from Inconel 600, with a tube outside diameter (OD) of 0.875 in. and a wall thickness of 0.05 in. This set of 24 tubes included 16 tubes with axial OD stress-corrosion cracking (SCC), 5 tubes with circumferential OD SCC, and 3 tubes with intergranular attack (IGA). All of the defects were on the outer surface of the tubing. The majority of the degradation experienced in the field is on the tubing outer surface, and this surface is much more difficult to inspect. (With the proper technique, the inner surface can be inspected with much greater accuracy.) These samples represent a difficult challenge to the eddy-current inspection system.

Traditional efforts to fit eddy-current readings to the properties of the tube used standard statistical least-squares methods. More recently, neural networks have been shown to be much better at fitting the readings to the tube properties. The tube property examined in this study was the defect depth, and neural network inspection methods were successfully used to determine the depth.

All of the 24 sample tubes were tested using the ORNL-developed scanners along with two standard tubes. In a laboratory simulation of the actual steam generator, the samples and tubes were scanned using OD "artifacts" consisting of ferrite to simulate magnetite and steel to simulate the tube supports. After the data were acquired, 23 of the tubes were examined using metallography, and the metallographic results for 15 of the 23 tubes were supplied. These results, along with the readings from the standards, were used to train the neural network to recognize the defects in the tubing. The manner in which the metallographic results were reported allowed as many as 83 points from a single tube to be included in the training set for defect depth for the neural network. The eddy-current readings are capable of producing a profile or contour of the degradation on the tubing. If the metallography is performed in a manner to allow a contour to be obtained, then the data are much more valuable and will allow the fit to be improved considerably.

The neural network correlation had to be applied to the training program on a point-by-point basis, and as the fitting progressed, additional points could be located and included in the fits. In a few instances the metallographic data were furnished in a manner such that only one point per tube could be included for correlation with defect depth. Points in "clean" sections of the tube were also included to train on "good" tubing. This improves the signal-to-noise of the test. After the network was trained on the known samples, a "blind test" was performed on the remaining eight tubes to demonstrate the ability of eddy-currents to measure the defect depth. After these results were reported, the data from the eight additional samples were made available so that further training was possible. Some of these data were included in the network training, and another test was performed on the entire set of 24 samples. These results were then reported.

Plots of the defect depth obtained from the eddy-current readings and from metallography were made for the samples that had adequate metallographic data. The eddy-current coil tends to average over a volume of the tube, and the metallography only measures at discrete points (approximately every 0.1 in.). For "point defects" there will

---

*Research sponsored by the Office of Nuclear Regulatory Research, Division of Engineering, U.S. Nuclear Regulatory Commission, under Interagency Agreement DOE 1886-8109-8L with the U.S. Department of Energy under contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Corporation.

be no eddy-current response, and the metallographic measurement will produce a result only if a "cut" is made at that point. The effect of this is shown on the graphic plots. There are instances were the metallography and the eddy-current result give contours that match very well, and cases where the contours do not match. The latter case includes defects that are observed on a single cut with the metallography. Along with the computation of the defect depth, training was also performed for computation of the presence of OD artifacts consisting of tube supports and the presence of magnetite. A separate subroutine was written to compute these tube properties, using a different set of weights for each property.

The different computer programs used for training the neural network, for performing the calculations of the tube properties, and for manipulating the data are given in Appendix A.

Probe Design

The probe design was derived from both analytical and experimental techniques. The analytical methods used a "point defect" model based on early work by Dodd and Deeds,[1] and on volume integral models developed by Sabbagh and Associates.[2] Both methods gave slightly different results, and both have some limitations. The final design also used experimental verifications. The P90 probe represented the best comprise between coverage of the entire circumference of the tube with 16 coils and resolution of small defects. The design curves for the P90 probe are given in a report by Dodd and Pate,[3] and will not be repeated here. The result of experimental measurements made on electrodischarge machined (EDM) notches (see Figure 1) shows the response of the P90 probe as the notch was moved from the coil center. The coil has over 70% of the response when the notch is at the edge of the coil as it does when the notch is centered in the coil. The edge of the notch is 12° from the coil center, and with an array of coils, the maximum spacing would be 11.25°. This coverage represents the best compromise between 100% coverage for the entire circumference with a 16-coil array and resolution to small defects. Both the spherical- and cylindrical-shaped probes were tested and gave similar responses. The spherical-shaped probe tended to ride the surface better and had less liftoff noise. The cylindrical-shaped probe tended to chatter more as it moved along the surface of the tube. A drawing of the P90 probe is shown in Figure 2.

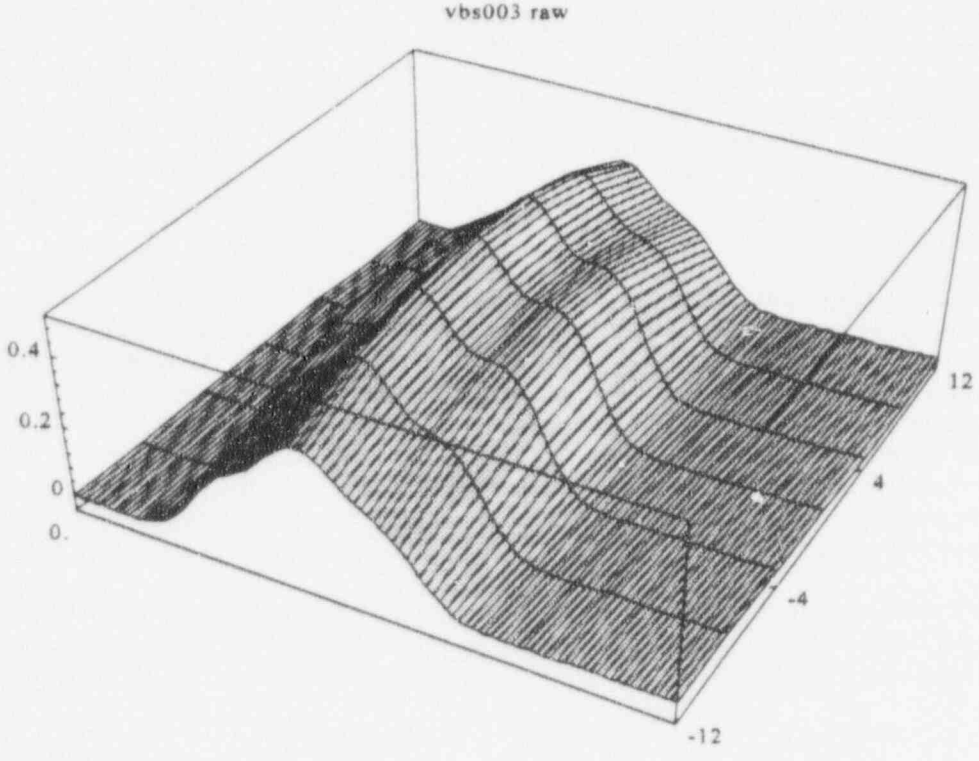

Figure 1. Scan of a 40% electrodischarge machined notch with the P90 spherical probe.

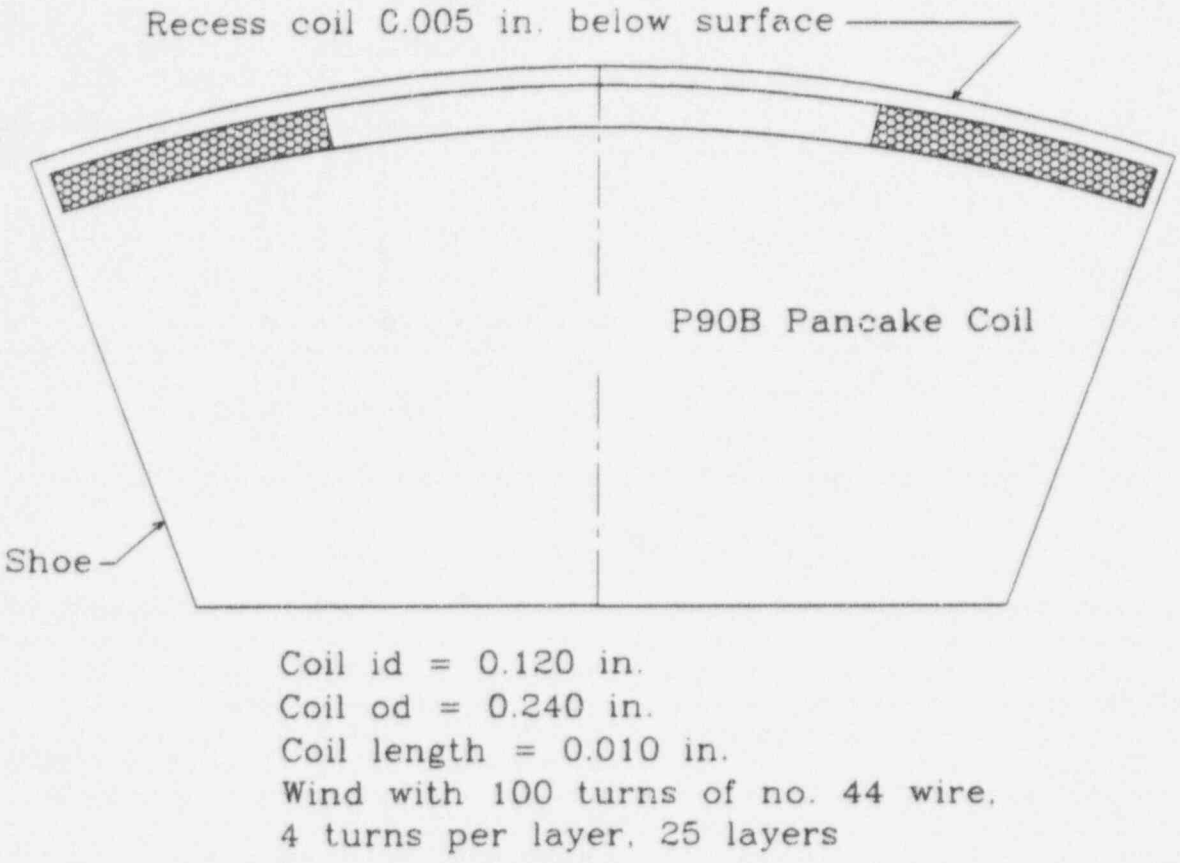

Figure 2. P90 spherical probe.

The coil is mounted on a spherical-shaped shoe, with a diameter slightly under that of the tube inside diameter (0.775 in.). The coil is wound flat and then contoured to the shoe. The coil is coated with a 0.005 in. layer of epoxy as a wear surface. The wear of this type of probe is less than that of the rotating pancake, and the probe velocity is much less.

Eddy-Current Data Acquisition

Data were acquired from EDM notch standards and from the 24 sample tubes furnished by Pacific Northwest Laboratories (PNL). Two different EDM notch standards were used, one consisting of circumferential notches and the other consisting of axial notches. In both standards, the nominal notch depths were 20, 40, 60, 80 and 100%. The actual values were used for the fitting equations. In Figure 3 we show a drawing of the circumferential notch standard. This standard also incorporated liftoff in the form of an epoxy coating on the inside of the tube. The thicknesses of the steps in this coating were 0.004 and 0.008 in. The inclusion of liftoff is essential to the proper setup of pancake coils, and this should be done for all pancake coil inspections (not just ORNL-developed tests). Scans were made with the ferrite ring (to simulate magnetite) and the tube support ring in different locations on the standard. No readings were made on the copper ring. The initial training was done on this standard; then the standard with the axial defects was used. Several trial scans were made to center the coil on the axial defects, and then scans were made from -12° of the center value to +12°, in 4° increments. After centering, the axial data were also added to the data from the circumferential standard. Next, scans were made on the sample tubes furnished by PNL.

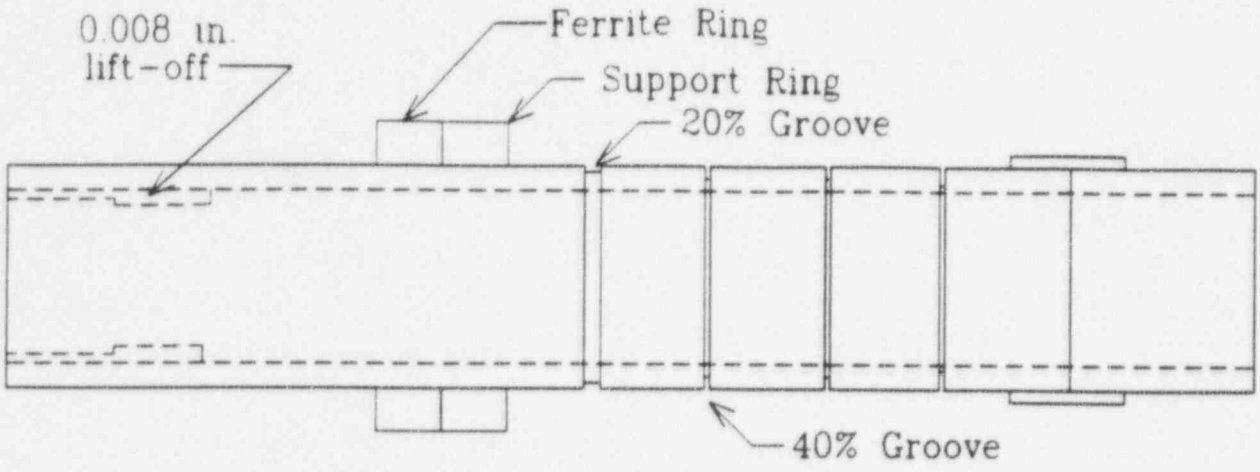

Figure 3. Circumferential notch standard used for training the neural network.

In Figure 4 we show a view of how the samples were scanned. Eighteen of the 24 tubes were scanned with the OD artifacts (the ferrite and the tube support) at different locations with respect to the "defective" regions of the tube. This was not done for various reasons on the remaining six samples. The data were recorded at four frequencies (520, 260, 180, and 60 KHz), every 0.030 in. along the tubing. The readings were repeated every 22.5° around the circumference of the tube to simulate the scan of a 16-coil array probe. The first set of measurements was made with the OD artifacts moved completely away from the defective region. Then, five additional sets of measurements were made as the artifacts were moved through the defect region. The junction of the ferrite and the tube support produces a larger signal than either by itself and is therefore more difficult to compensate for in an actual test. This "junction" frequently occurs in the steam generators, when the magnetite grows next to the tube support, or on top of the tube sheet. The scanner moves the probe toward the open end of the tube, the artifacts are moved toward the closed end of the tube, and the tube is rotated by the scanner. The probe motion is controlled by the x-axis of the scanner, the artifact motion by the y-axis of the scanner, and the sample rotation by the z-axis. The positive, x-axis motion of the scanner (ET Distance) can be related to the distance from the open end of the tube (Met. Distance) by the relationship:

$$\text{Met. Distance} = 9.492 - \text{ET Distance}$$

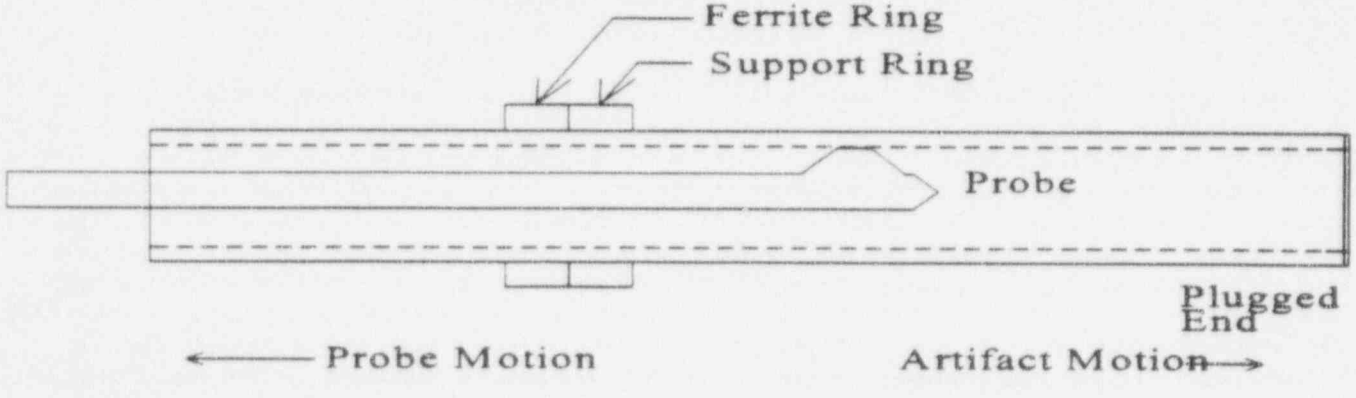

Figure 4. Scanning of samples.

Metallographic Data Acquisition

The metallographic data are converted to have the same orientation as the eddy-current data. Most of the metallography was done by PNL. The metallography on the axial OD SCC was performed in a manner that gave multiple defect depths at a number of axial and circumferential locations along the tube. Contour plots were made of the metallographic data from these tubes and compared to the eddy-current depth predictions. In addition, the metallography for the circumferential OD SCC was done at different points around the tube circumference. However, there is no information on the axial location of the circumferential defects. The eddy-current readings indicated that there was some axial variation, and pulled tube results have also confirmed this. For future metallographic results, all of the information should be obtained and given. For most of the IGA samples and for one of the axial OD SCC samples, only the maximum measured defect depth was given. This results in only a partial data set that does not allow optimum correlation with the more detailed eddy-current data. An even better data set would result from leak and burst tests on the sample and correlation of these results to the eddy-current readings.

Data Fitting on the Initial Training Set

The readings were fitted to the properties using a back-propagation neural network. A simplified neural network is shown in Figure 5. The input layer consists of the real and imaginary parts of the magnitude and phase for each of the four frequencies. There is also a bias layer that is used, so there are nine inputs. The number of neurons in the feature extraction layer can be varied. The output layer consists of one property that we wish to fit (there can be several, but for simplicity we used only one) and a bias layer. A separate network is used for the other properties that we may want to fit. The fitting program is given the value of the property to be fitted and the value of the readings. It will then determine the weights that will best match the properties to the readings using an iterative process. The program was first run for only the scans of the circumferential standard, shown in Figure 3. Next, the axial standard was added. Most of the training was done using the program bpnd.c for the defect depth, since depth was the major concern. However, the programs bpnt.c and bpnf.c were run to obtain fits of the readings to the presence of tube supports and to the presence of ferrite. Thus, one set of weights will give the defect depth, one set will give the tube supports, and one set will give the presence of ferrite.

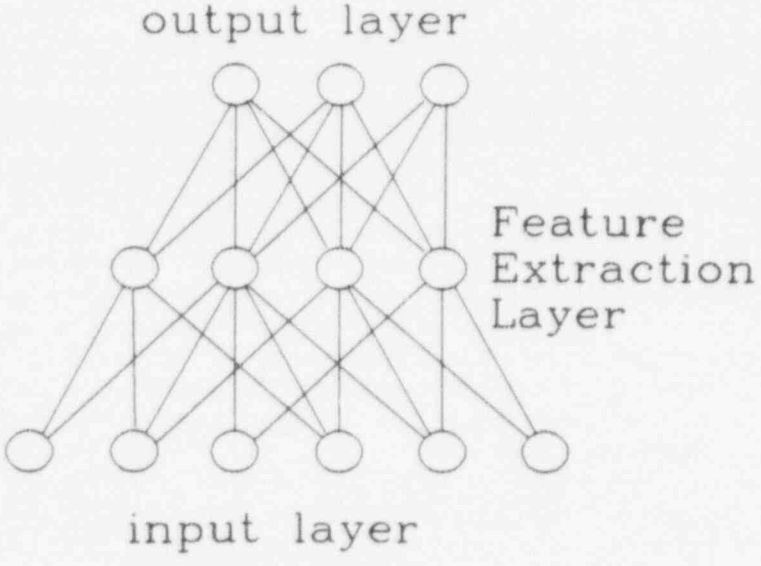

Figure 5. A simplified neural network.

After the training on the standards, the properties for the known samples were computed. While the defect depth was supposedly known, the exact location for the reading was not. In the future, better coordination between the metallography and the eddy-current measurements should be attempted. The circumferential defects were added first since they were the easiest to match. However, the metallography did not give any axial information, while the readings show a difference in axial position, along with possible multiple cracks, around the tube circumference. Since only one value was given for the IGA, this value was assigned to the deepest depth. However, for most of the axial defects, the depth was furnished as a function of distance from the tube end and as a function of circumferential position. While this information is much more difficult to obtain, it is much more valuable. For these defects, a metallographic depth contour plot was obtained, and this was compared to the eddy-current depth contour plot.

The rotation and axial distance of the two plots were adjusted to allow the two contour plots to fall on top of each other. When there is a good match between the positions of the eddy-current readings and the metallographic measurements, the rms error in defect depth can be small, but if we try to fit the readings to defects in the wrong place, this error will remain large. However, since we are using the readings from 40,000 locations for this fit, individual mismatches are not as noticeable. The axial OD SCC samples were added one at a time, and the training program run and evaluated for each. The large, single defects were run first. As the fit improved, more details from the eddy-current readings could be matched to the metallographic data, and these were included. This was a very time-consuming process since the code for the depth and location of each of the points had to be written. Along with the defect depths for the regions that were defective, regions that had no eddy-current readings were selected, and these were assigned a zero depth. A significant number of these zero points were chosen, and this had the effect of considerably reducing the noise. Also, small variations in conductivity, permeability, and wall thickness were reduced. After using these methods good results were achieved.

A study was performed on the relationship between the number of nodes in the intermediate layer and the rms error in the fit of the properties to the readings. It is generally believed that too many nodes will reduce the ability of the network to "generalize" and recognize new combinations of properties that it has not previously seen. The response would be more "memorized." Too few nodes will not allow a good fit of the properties that are present. After running the program with a number of different nodes in the middle layer, it was determined that 17 nodes in the hidden layer gave the smallest least-squares error (about 3% of the wall thickness).

Blind Test Results

After the weights were computed, the data were processed for the eight blind samples. The maximum defect depth for each sample was determined. These data are given in Table 1. Also included in the table are the metallographic results, furnished after the eddy-current results had been submitted. Note that metallography was not performed on sample E-11-06. It was decided to retain this sample for further study and reference purposes. The agreement is good between the two, particularly considering the nature of the defects and the nature of the scanning. If the defect is at the edge of the probe, the reading can be low by 30%. For a "point" defect, the eddy-currents will not even detect this type (nor will it be detrimental to the tube service). As can be seen from the plots that are shown later, many of the axial OD SCC types of defects fell in this category. In addition, since the metallography is only performed every 0.1 inches, for irregular-shaped defects the maximum depth determined from the eddy-current response can be considerably different from the metallographic value. In some instances, clear eddy-current readings were observed where no metallographic depths were given. A correlation between the eddy-current readings and the bottom-line material properties such as burst pressure and leak rate would be more useful. This fit would probably be better since the small "point defects" that have no effect on the eddy-current readings also have no effect on these properties.

Table 1. Eddy-current and metallographic depths

Sample	Depth (ET)	Depth (Met.)
2-12	24%	22%
4-11	91%	100%
5-08	48%	26%
B-10-10	43%	48%
B-63-07	77%	69%
E-11-06	73%	--
E-13-06	78%	73%
F-08	67%	38%

Additional Training Results

After the metallographic data were made available for the blind-test samples, some additional training was done. Three of the tubes with metallographic data were added to the training set. This allowed the data fit to be improved enough that a number of additional points from tubes with previous metallographic results could be added to the training set. After these points were added, additional training was done. In Table 2 we show the tube number and the training that has been done. Some of the tubes are used for zero values only, and others are used both for zero and defect values. The axial and circumferential standards contribute 7,525 data points to the data set, and the samples, at the present, contribute 31,765 points, for a total of 39,290 points. If the data are taken with OD artifacts, each location will contribute six points to the data set. If not, each location will only contribute one data point. Some of the tube data were not used because it was thought that the values would bias the results in an undesirable direction. Other tubes were not used because the defect reported by metallography did not give a clean eddy-current signal that could be identified. Tubes that had OD artifacts were added to the set before those that did not because the former furnished more data points to be fitted. Finally, tube E-11-06 has not been analyzed with metallography. Since the last training has been done, several defects have been identified that could now be added to the data set.

When the initial training was performed, a study was run to determine the optimum number of nodes in the hidden layer of the neural network. This study was repeated after the size of the data set was increased. The optimum number of nodes for the least rms error increased from 17 to 23. This was probably due to the increase in the complexity of the data set.

Comparison of Eddy-Current and Metallographic Results

The best way to compare the metallographic and eddy-current results is to use plots of each. This allows the reader to visually match the two methods of experimentally measured defect depth.

Since both methods are experimental, there are errors associated with each. The metallography will be more accurate for determining the defect depth at a given point. However, there is no guarantee that the point at which the depth is measured is the maximum depth. For very irregular defects, such as OD SCC, the chances are that the metallography was not done at the maximum depth. With the eddy-current measurement of depth, the defect influence on the eddy-currents is more of an average of the defect depth. The correlation to depth for irregular defects will probably not be any better than 20%. However, the eddy-current response is a much better predictor of the burst and leak performance of the tube than the maximum depth. Studies performed for the Alternate Plugging

Table 2. Summary of data points used for training

Tube number	Defect type	Defect points used	OD[a] art values	Zero values used
2-05	L-OD SCC[b]	2	1	Yes
2-12	L-OD SCC[b]	0	1	No
4-11	C-OD SCC[b]	5	1	Yes
5-01	IGA[c]	18	6	Yes
5-08	IGA[c]	0	6	No
5-14	IGA[c]	4	6	Yes
B-10-02	L-OD SCC[b]	24	6	Yes
B-10-10	L-OD SCC[b]	44	6	Yes
B-30-02	L-OD SCC[b]	16	6	Yes
B-45-06	L-OD SCC[b]	30	6	Yes
B-49-05	L-OD SCC[b]	33	6	Yes
B-49-09	L-OD SCC[b]	17	6	Yes
B-55-08	L-OD SCC[b]	24	6	Yes
B-59-10	L-OD SCC[b]	28	6	Yes
B-62-09	L-OD SCC[b]	29	1	Yes
B-63-07	L-OD SCC[b]	83	1	Yes
E-11-06	L-OD SCC[b]	0	6	No
E-13-06	L-OD SCC[b]	0	1	Yes
F-08	L-OD SCC[b]	0	6	Yes
L-14-06	L-OD SCC[b]	21	6	Yes
W-23-03	C-OD SCC[b]	5	6	Yes
W-23-09	C-OD SCC[b]	1	6	Yes
W-23-10	C-OD SCC[b]	0	6	No
W-40-07	C-OD SCC[b]	1	6	Yes

[a]OD = Outside diameter
[b]SCC = Stress-corrosion cracking
[c]IGA = Intergranular attack

Criteria (APC) have shown that OD SCC defects 100% deep can have little effect on the burst pressure and no leakage. Therefore, the eddy-current readings are more useful than the metallography for determining steam generator tube integrity.

In Figure 6 we show a contour plot of the calculated eddy-current depth measurements. The height of the plot represents the defect depth, and the distance along the tube and the circumference around the tube are along the other two axes. In Figure 7 we show a plot of the metallography data. There is very good correspondence between the two plots. There is a roll transition and a tube support ring at the edge of the defect. These artifacts have been well suppressed by the neural network.

There was not enough metallographic data furnished with the IGA samples (usually only the maximum depth) to make a plot of this type of defect. In Figure 8 we show the plot of the eddy-current data for this type of defect. For IGA the conductivity and therefore the eddy-current signal depend on the manner in which the IGA was formed. This can be predicted to some extent by the location of the IGA in the generator. This particular IGA sample seems to have a relatively low conductivity and therefore a high eddy-current signal. The other IGA samples in the study had a higher conductivity and therefore a lower eddy-current signal. The mechanical properties do not seem to depend on the conductivity of the IGA but on the extent of the IGA.

The most interesting tubes are those with axial OD SCC. The metallography on all but one of the tubes was performed in a manner that allowed multiple metallographic data points to be obtained. This allowed a large number of readings to be obtained from each tube, greatly increasing the value of the sample to this study. In Figure 9 we show a tube with axial OD SCC. This tube, B-63-07, has several long, distinct cracks and allowed a fit to be obtained for 83 different positions along the tube. Unfortunately, the OD artifacts were not used on this tube, so only one point was obtained for the data set at each location. When Figure 9 is compared to the plot of the metallographic data, as shown in Figure 10, all of the cracks with any appreciable length can be easily located and identified. Both the size and location of the defects compare very well. This is an example where there is quite good agreement between the eddy-current data and the metallographic data. This good agreement is due to the nature of these particular cracks, which for the most part are large and fairly regular. For the small, pointlike defects, the agreement is not nearly as good. The eddy-current coil response is averaged over the effective coil field, and point defects do not contribute any significant amount to this signal.

In Figure 11 we show the defect depth computed from the eddy-current readings for a tube that has many small, short defects. Note that there is very little response from the eddy-current probe, which is giving readings in the 10% depth range. In Figure 12 we show the plot of the metallographic data for the same tube. This tube demonstrates the two cases where the eddy-current readings give the poorest representation of the metallographic data. The defects in this tube are all under 20%, which is about the threshold of detection for this type of pancake coil. They are also, for the most part, point defects detected only at one axial location and then not detected again. The method of plotting interpolates the defects back to zero over one chart division (about 0.1 in.). However, there are no data to support this conclusion, nor any other about the shape of the defect since the metallographic sections were taken at about 0.1 in. intervals. The redeeming factor is that this defective region has no significant effect on the properties of the tube that are of concern to us, such as burst pressure and leak rate. This sample is the exception rather than the rule, and most samples looked more like the ones in Figures 9 and 10.

As mentioned earlier, the neural network was trained to compute three different tubing properties: the defect depth, the presence of a tube support, and the presence of magnetite. The latter two properties are not considered as important as the defect depth, and less effort went into their training. The main goal was to determine if the artifacts were present or not. The network does such a good job of suppressing these artifacts for the defect depth computation that it is difficult to know the location of the probe in the tubing. The neural network was able to perform these computations using only 15 nodes in the hidden layer. An arbitrary value of ten was assigned if the artifact was present, and zero was assigned if the defect was absent. This allowed the color contour plots on the Hewlett Packard 755 workstation to be superimposed on the defect plots without overshadowing the defect depth plots. In Figure 13 we show the computed magnetite for the tube sample b-10-02. The magnetite extends from 4.0 to about 4.6 and is produced by a ferrite ring around the tube. The axial region 4.6 and greater has no OD artifact. Notice that the defect residual signal does not appear to any significant extent on the bare tube or on the magnetite, although there is a little noise riding on the magnetite.

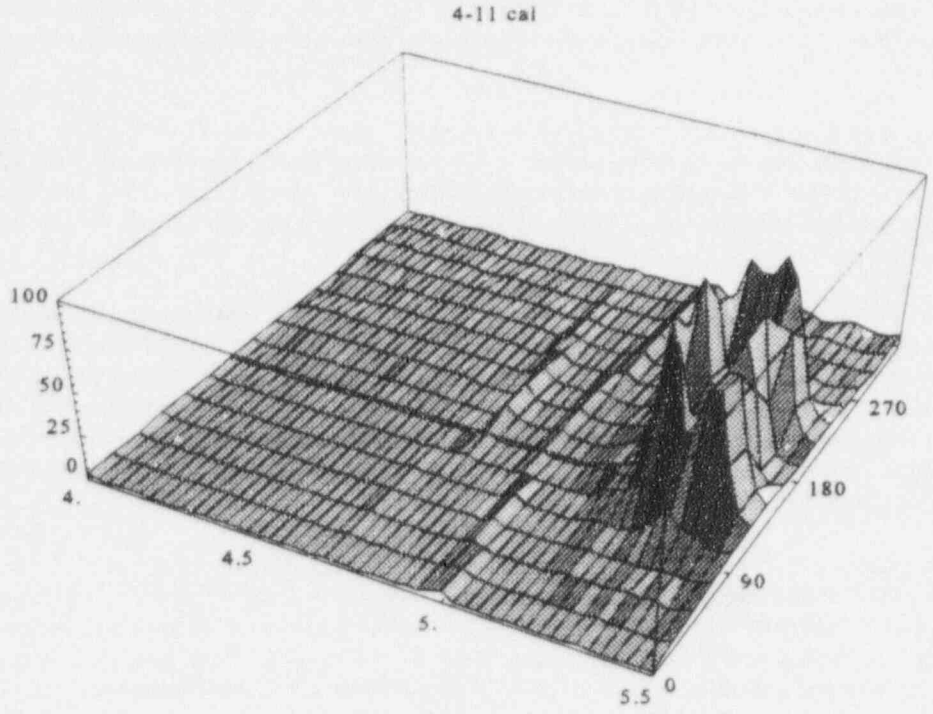

Figure 6. Defect depth from eddy-current readings for C-outside diameter stress-corrosion cracking defect.

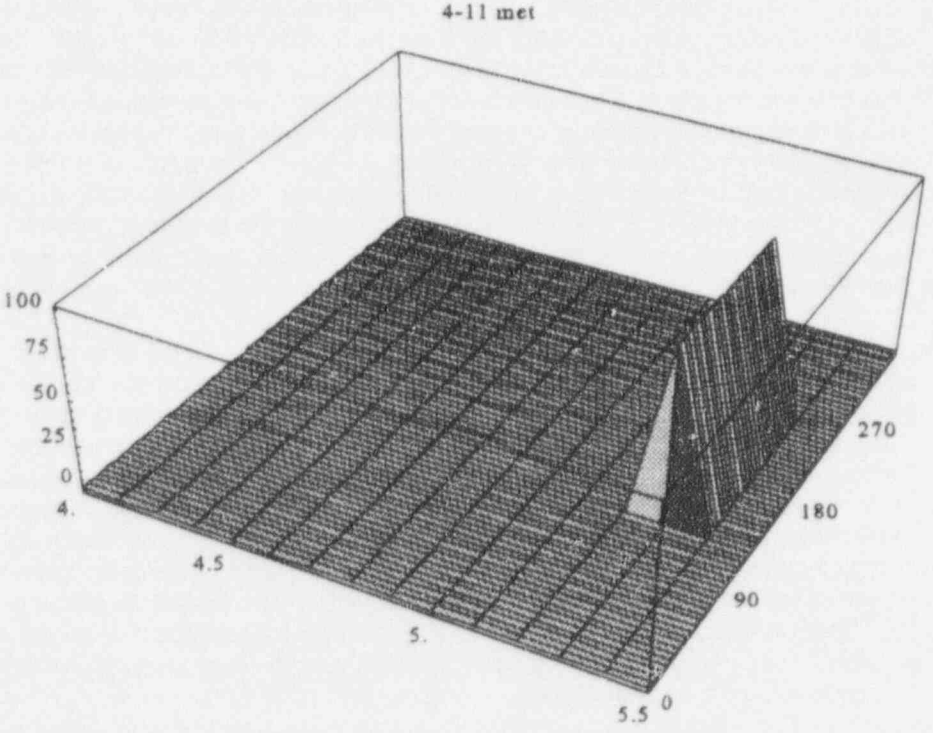

Figure 7. Plot of metallographic data for C-outside diameter stress-corrosion cracking defect.

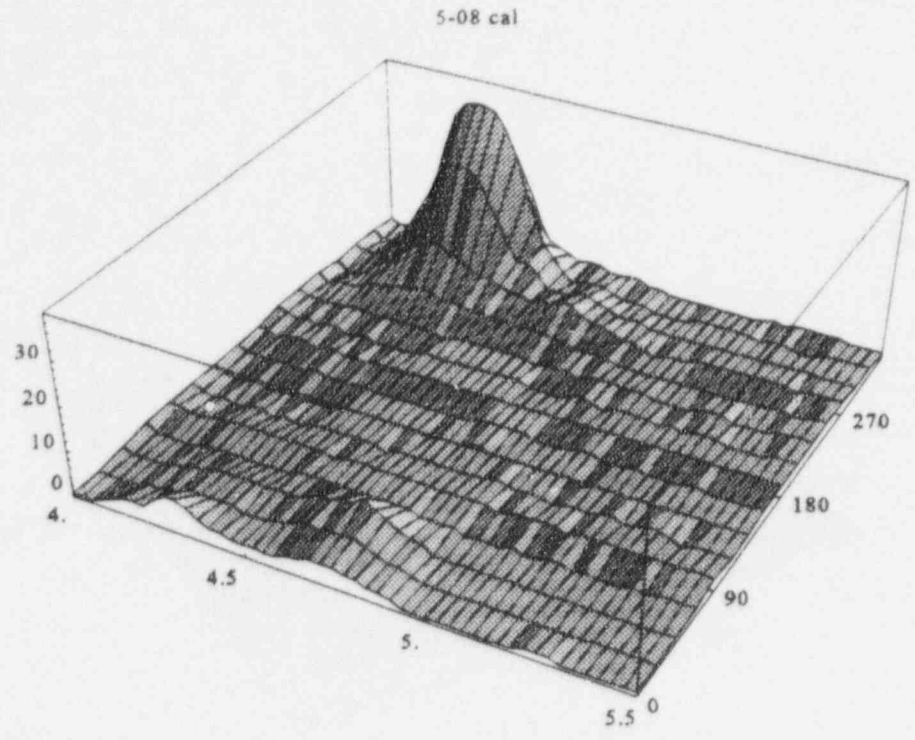

Figure 8. Plot of eddy-current depth for intergranular attack defect.

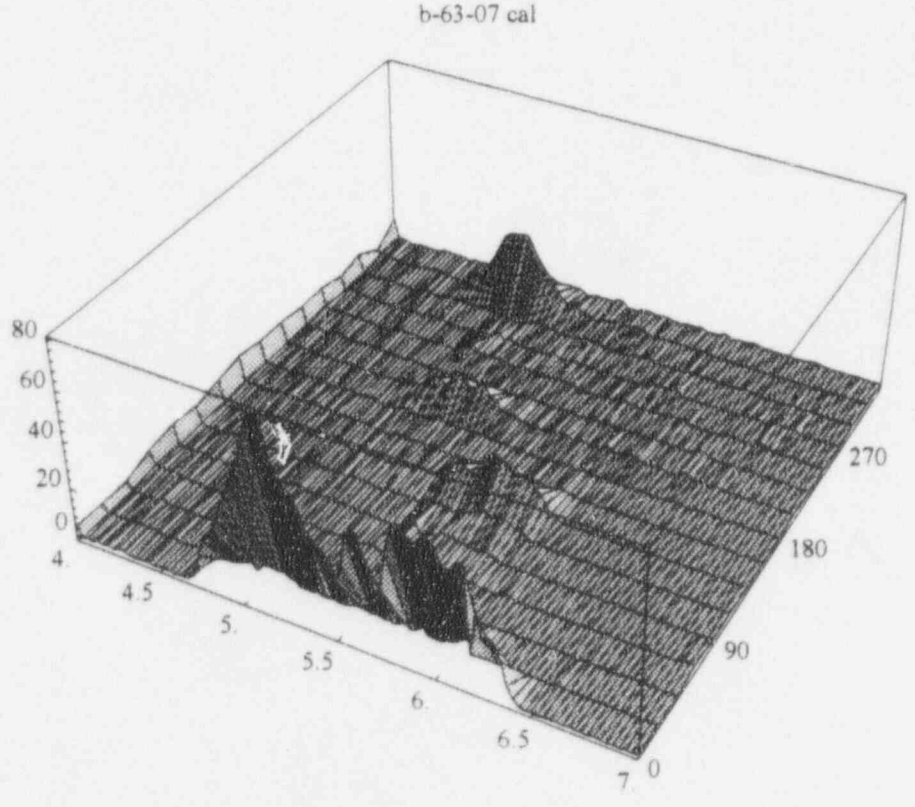

Figure 9. Depth from eddy-current readings for axial outside diameter stress-corrosion cracking defect.

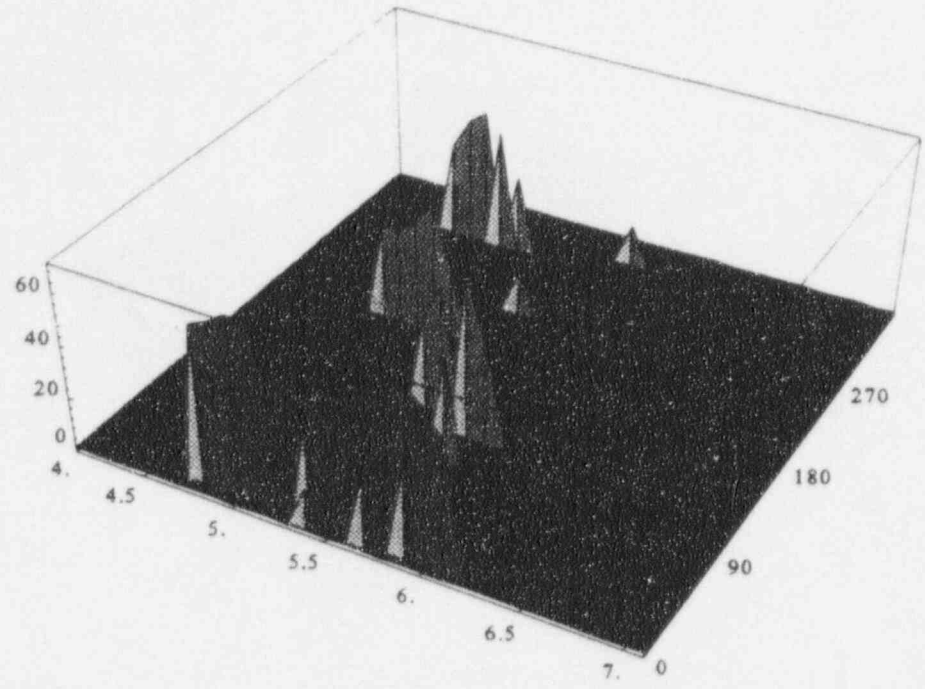

Figure 10. Plot of metallographic data for an axial outside diameter stress-corrosion cracking defect.

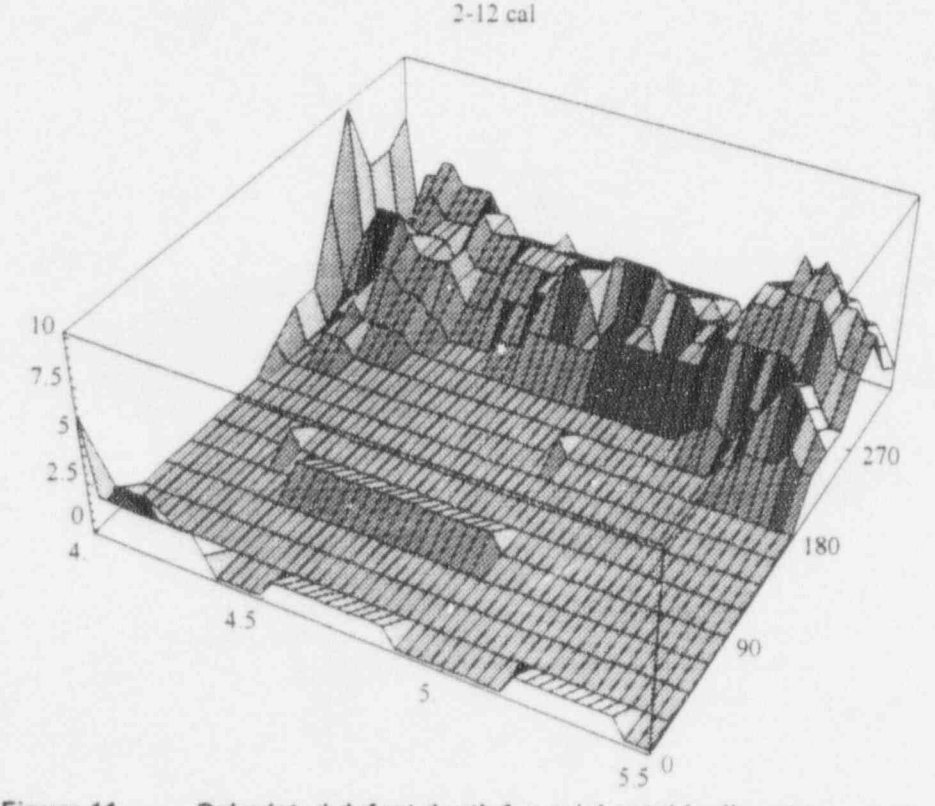

Figure 11. Calculated defect depth for axial outside diameter stress-corrosion cracking.

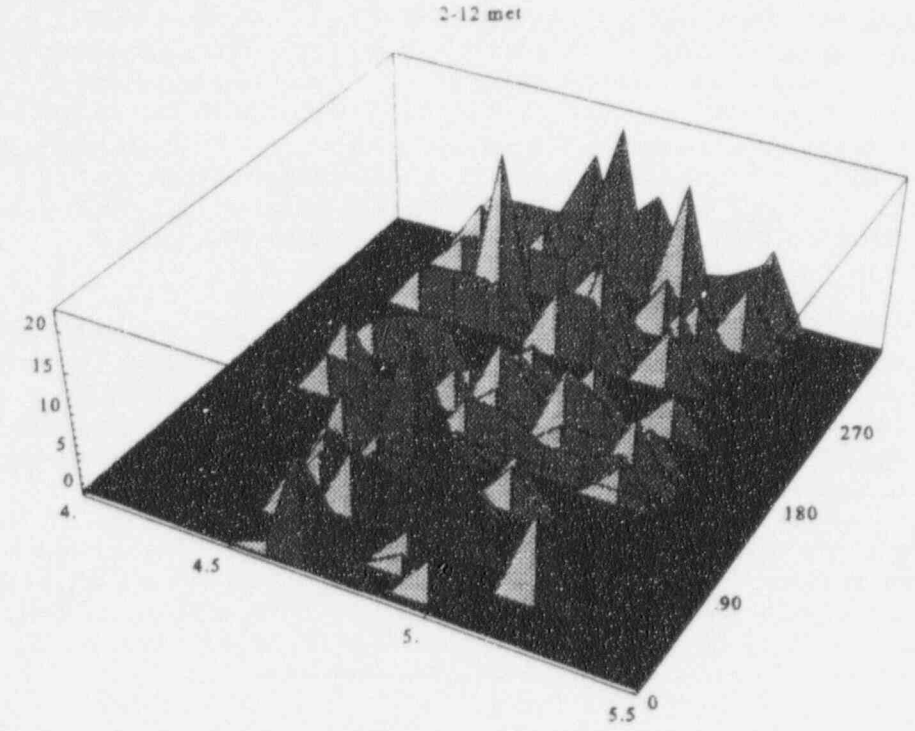

Figure 12. Plot of metallographic data for axial outside diameter stress-corrosion cracking.

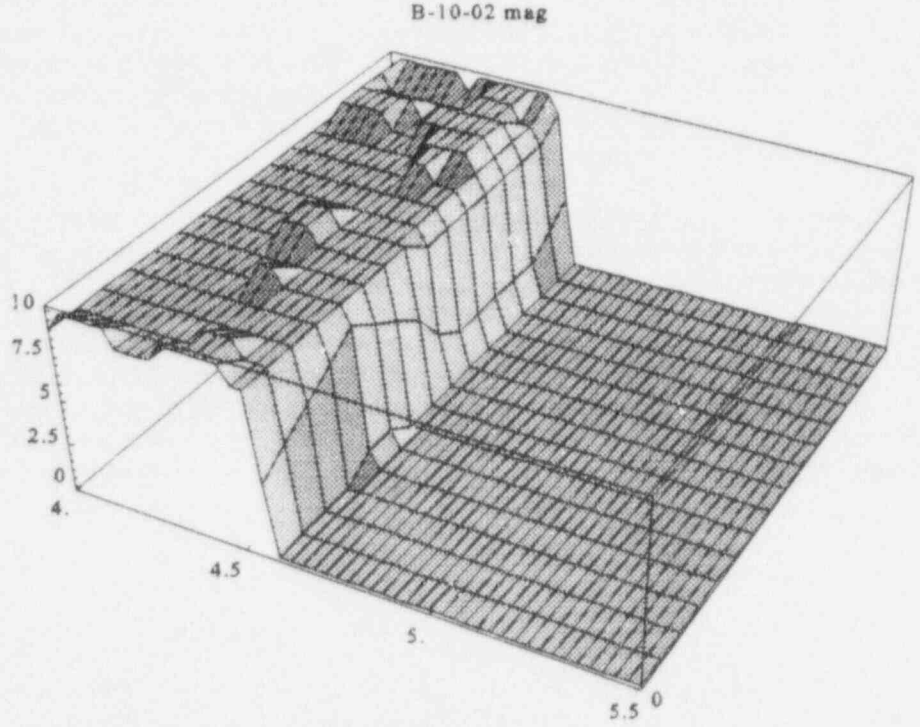

Figure 13. Property magnetite computed from the eddy-current readings.

The effect of the magnetite on the computation of the defect depth is of more importance since this is a measure of the ability of the neural network to compute the desired property in the presence of combinations of undesired properties. Magnetite was chosen as the example to display since it produces a larger signal on the raw readings than the tube-support ring does. In Figure 14 we show a plot of the defect depth with magnetite present. The raw readings in Figure 13 are the same raw readings that are used in Figure 14. These readings are processed by one set of weights to give Figure 13 and another set to give Figure 14. In Figure 15 we show a plot of the calculated defect depth without the ferrite ring, or with no magnetite. Notice that the presence of magnetite has very little effect on the calculation of the defect depth. This particular sample represents an extreme case. The defect chosen as magnetite is less on larger defects that have more eddy-current signal.

To summarize, Figure 13 is a computation of magnetite, which shows the location of the magnetite ring with very little defect residual present. Figure 14 is a computation of defect depth with very little magnetite residual present, and Figure 15 is a computation of the defect depth without the magnetite present. These figures demonstrate the ability of the neural network when it is trained on the proper samples. These results are much better than those obtained using least-squares techniques to perform the same type of analysis. The nonlinear features of neural networks are better matched to the nonlinear responses of eddy-currents.

After the new samples were added to the computation of the weights and additional points were added from the old samples, another prediction was done to determine the defect depth. The best estimate on all of the tubes using new weights is summarized in Table 3. As noted before, sample E-11-06 has not been sectioned. However, the eddy-current depth prediction is still given. The computer codes used for this prediction are in Appendix A. These results are quite good when the irregular nature of the defects is considered.

Summary and Conclusions

Neural networks were applied to fit data from standards and realistic laboratory samples to readings obtained from an eddy-current probe. This method gives results that are considerably better than those obtained for the least-squares fitting methods. The method of performing the metallography can increase the value of the data for this study considerably. Both the axial and circumferential location of the defects should be recorded along with the defect depth. This increases the usefulness several fold when compared to recording only the maximum depth. If the data were reported in a table similar to the defect tables in Appendix A, that would reduce the amount of work in making the metallographic defect plots considerably.

The maximum defect depth has been considered the property of most importance in the eddy-current test. However, the performance of the tube is influenced by many other properties, such as the defect length and any "bridging" that the defect may have. Also, the connectivity of multiple defects affects the tube burst pressure and the leak rate. These other properties also change the eddy-current readings in a manner that is in the same direction as the burst pressure (larger eddy-current signals and larger reduction in burst pressure). Therefore, the correlations should be done between the eddy-current readings and tube properties such as burst pressure and leak rate. The prediction of defect depth growth rates by corrosion experts has not been good enough to warrant the use of defect depth for this purpose. The only method accepted for the growth rates now is an extrapolation from depth measurements made from one cycle to the next. This same method would work for burst pressure and leak rate.

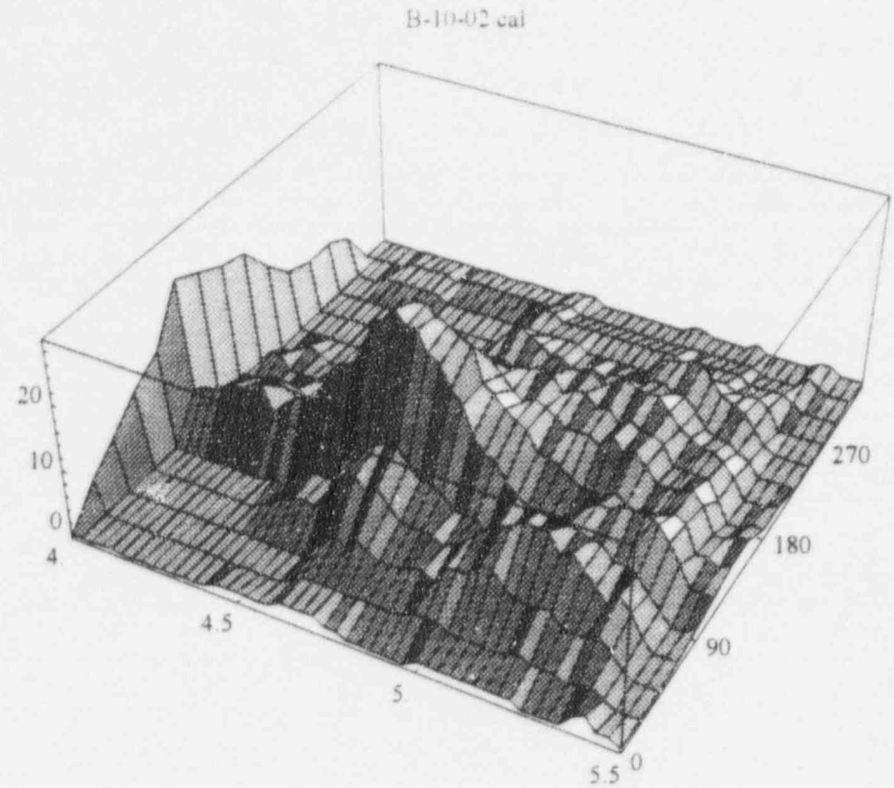

Figure 14. Calculated defect depth with magnetite present.

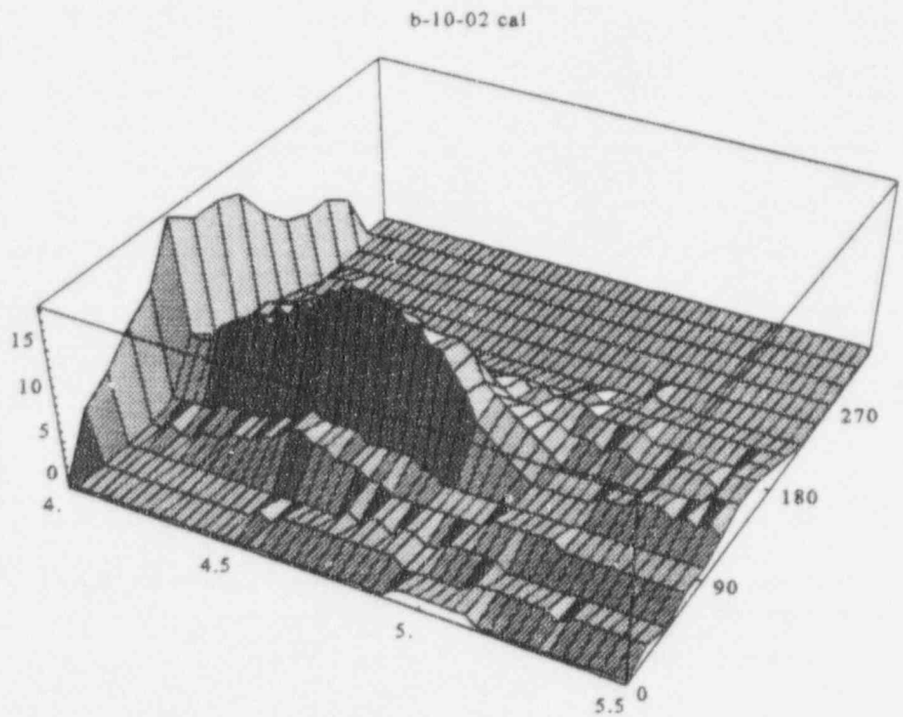

Figure 15. Calculated defect depth without magnetite.

Table 3. Eddy-current and metallographic data

Tube number	Defect type	ET depth	Met. Depth
2-05	L-OD[a] SCC[b]	57	62
2-12	L-OD[a] SCC[b]	26	22
4-11	C-OD[a] SCC[b]	100	100
5-01	IGA[c]	21	18
5-08	IGA[c]	52	26
5-14	IGA[c]	50	47
B-10-02	L-OD[a] SCC[b]	38	43
B-10-10	L-OD[a] SCC[b]	53	48
B-30-02	L-OD[a] SCC[b]	28	45
B-45-06	L-OD[a] SCC[b]	60	40
B-49-05	L-OD[a] SCC[b]	42	33
B-49-09	L-OD[a] SCC[b]	72	74
B-55-08	L-OD[a] SCC[b]	77	59
B-59-10	L-OD[a] SCC[b]	26	20
B-62-09	L-OD[a] SCC[b]	73	58
B-63-07	L-OD[a] SCC[b]	82	69
E-11-06	L-OD[a] SCC[b]	83	--
E13-06	L-OD[a] SCC[b]	85	73
F-08	L-OD[a] SCC[b]	64	38
L-14-06	L-OD[a] SCC[b]	100	100
W-23-03	C-OD[a] SCC[b]	100	100
W-23-09	C-OD[a] SCC[b]	100	100
W-23-10	C-OD[a] SCC[b]	100	88
W-40-07	C-OD[a] SCC[b]	100	100

[a]OD = outside diameter
[b]SCC = stress corrosion cracking
[c]IGA = intergranular attack

References

1. W. E. Deeds, C. V. Dodd, and G. W. Scott, *Computer-Aided Design of Multifrequency Eddy-Current Tests for Layered Conductors with Multiple Property Variations*, ORNL/TM-6858, Union Carbide Corp. Nuclear Div., Oak Ridge Natl. Lab., Oak Ridge, Tenn., October 1979.

2. H. A. Sabbagh, "Splines and their reciprocal-bases in volume-integral equations," *IEEE Trans. Magn.* **28**, 1138 (1992).

3. C. V. Dodd and J. R. Pate, Lockheed Martin Energy Research Corporation, Oak Ridge Natl. Lab., Oak Ridge, Tenn., *Evaluation and Field Validation of Eddy-Current Array Probes for Steam Generator Tube Inspection*, NUREG/CR-6357, July 1996 published date.

Appendix A

Computer Codes Used

Appendix A - Computer Codes Used

The computer codes used for this study are given in this appendix. They consist of neural network codes to fit the eddy-current readings to the desired properties, programs to move and display the data in the desired manners, and codes to perform the plots used in the report. Some programs are written in C and run on the Hewlett Packard (HP) 755 workstation. Others are written to run on a personal computer (PC). The eddy-current readings were acquired using the Hewlett Packard 755 to drive the MIZ-30 eddy-current instrument over a thin local area network (LAN). After the data were processed to determine the weights for the neural network, the network was run on the data files to compute the properties of interest. These property files were either displayed using graphics on the HP or written into files for transport to the PC. The files were transferred from one computer to the other over a thin LAN. The files were then plotted on the PC using the graphics packages in Mathematica. Most of the programs are written in ANSI C. The versions of C used are different on the HP and the PC.

Program bpnd.c

This program is used to compute the weights needed to fit the eddy-current readings to the defect depth. It is a modification of a program written by Dr. John Allen for general eddy-current data analysis. Similar programs are used to compute the weights for the tube supports and the magnetite. The program is started and run on an interactive basis. The user can read weight data from an existing file, or choose to start over. The user must use three layers in the network, and have nine nodes in the input (first) layer (since there are eight readings and one bias node) and two nodes in the output layer (since there is one output value and one bias node). This program will allow the number of nodes in the middle layer to be changed.

The user can then choose a number of options while running the program and can save the best weights as the program is run. These options are typed in after the program has started to run and their actions are summarized in the following Table A-1.

The user must specify the location of the null point (a position where there are no defects or artifacts) on the sample and must tell the program if the data contain readings for OD artifacts or not.

Table A-1. Interactive commands for running the neural network training program

Command	Action
c	Concise mode of output. This is the default and recommended.
e	Program will calculate the true rms error for all of the points fitted rather than a random average. If this error is less than previous errors, the weight file will be saved. If e is typed again, the program will switch back to the random rms error. When the program is in the true rms mode, a t will be printed after the calculated error, or ** will be printed if the error is a new minimum.
j	Jog the weights. The user will be prompted for a value to jog the weights. Start with a small value.
l	The user will be prompted for a new value for the learning rate.
m	The user will be prompted for a new value for the momentum.
q	Quit the program. This is the preferred way to stop execution.
s	Save the weight values to a file (not stop).
u	Undo a bad step.
v	Verbose mode. This increases the data printed out as the program is run by a large amount.

```
/*
 * bpnd.c
 * Version 18 July 1995
 * Program to do neural network fit for defect depth.  Each sample has
 * a function to pick the values to be fitted for that sample.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curses.h>
#include <math.h>

float set_depth_1010(float *);
float set_depth_4909(float *);
float set_depth_6307(float *);
float set_depth_205(float *);
float set_depth_4905(float *);
float set_depth_3002(float *);
float set_depth_514(float *);
float set_depth_1002(float *);
float set_depth_6209(float *);
float set_depth_5508(float *);
float set_depth_4506(float *);
float set_depth_5910(float *);
float set_depth_1306(float *);
float set_depth_411(float *);
float set_depth_1406(float *);
float set_depth_501(float *);
float set_depth_4007(float *);
float set_depth_2303(float *);
float set_depth_2309(float *);
float set_depth_z11954(float *);
float set_depth_z11956(float *);
int find_null(int,FILE *);

int nfiles = 32;

char file[][80] = {
  "/hd1/metdat/vbs003d.dat",
  "/hd1/metdat/vbm001.dat",
  "/hd1/metdat/vbm002.dat",
  "/hd1/metdat/vbm003.dat",
  "/hd1/metdat/vbm004.dat",
  "/hd1/metdat/vbm005.dat",
  "/hd1/metdat/vbm006.dat",
  "/hd1/metdat/vbm007.dat",
  "/hd1/metdat/vbm008.dat",
  "/hd1/metdat/vbm009.dat",
  "/hd1/metdat/vbm010.dat",
  "/hd1/metdat/vbm011.dat",
  "/hd1/metdat/vbm012.dat",
  "/hd1/metdat/w-23-09",
  "/hd1/metdat/w-23-03",
  "/hd1/metdat/w-40-07",
  "/hd1/metdat/5-01",
  "/hd1/metdat/l-14-06",
  "/hd1/metdat/4-11",
```

```
  "/hd1/metdat/e-13-06",
  "/hd1/metdat/b-59-10",
  "/hd1/metdat/b-45-06",
  "/hd1/metdat/b-55-08",
  "/hd1/metdat/b-62-09",
  "/hd1/metdat/b-10-02",
  "/hd1/metdat/5-14",
  "/hd1/metdat/b-30-02",
  "/hd1/metdat/b-49-05",
  "/hd1/metdat/2-05",
  "/hd1/metdat/b-63-07",
  "/hd1/metdat/b-49-09",
  "/hd1/metdat/b-10-10"
 };

int naxes[] = {
  2,1,1,1,1,
  1,1,1,1,1,
  1,1,1,3,3,
  3,3,3,2,3,
  3,3,3,3,3,
  3,3,3,2,2,
  3,3};

int standard[] = {
  11954,11956,11956,11956,11956,11956,
  11956,11956,11956,11956,11956,11956,
  11956,11956,11956,2309,2303,
  4007,501,1406,411,1306,
  5910,4506,5508,6209,1002,
  514,3002,4905,205,6307,
  4909,1010};

float nullpos[][3] = {
  {5.25,0.,0.},
  {3.88,0.,0.},
  {3.88,0.,0.},
  {4.63,0.,0.},
  {4.63,0.,0.},
  {5.38,0.,0.},
  {5.38,0.,0.},
  {3.88,0.,0.},
  {3.88,0.,0.},
  {4.63,0.,0.},
  {4.63,0.,0.},
  {5.38,0.,0.},
  {5.38,0.,0.},
  {3.98,0.,0.},
  {3.93,0.,0.},
  {3.33,0.,0.},
  {5.37,0.,0.},
  {2.78,0.,0.},
  {6.40,0.,0.},
  {6.43,0.,0.},
  {4.48,0.,0.},
  {3.33,0.,0.},
  {4.23,0.,0.},
```

```
 {4.08,0.,0.},
 {3.53,0.,0.},
 {3.21,0.,0.},
 {3.43,0.,0.},
 {3.13,0.,0.},
 {3.47,0.,0.},
 {3.79,0.,0.},
 {3.63,0.,0.},
 {2.73,0.,0.}
 };

float nullval[4][2];
float depth;
int usepoint;
int plotpoint;


void main(void)
{
 FILE *strdat, *strwt, *strout,*strot1;
 char wtsin[80], wtsout[80];
 int found_null;
 int eoftst;
 float x[3];
 int is_null_pos;
 int npt = 0;
 int i, j, k, l;
 int nlayers;
 int ninput, nhidden, noutput;
 int restore_from_file;
 int pattern;
 int ch;
 int tmpdatr,tmpdati;
 int data[4][2];
 int print_period = 1;
 int calculate_error = 0;
 int undo_bad_step = 0;
 long nn = 0;
 double w0[20][33], w1[2][20];
 double d0[20][33], d1[2][20];
 double last_w0[20][33], last_w1[2][20];
 double prev_epoch_w0[20][33], prev_epoch_w1[2][20];
 double prev_epoch_d0[20][33], prev_epoch_d1[2][20];
 float input[100000][33];
 float pos;
 float *p0_input, *p1_input;
 double hidden[20], output[2];
 float actual_depth[100000];
 double sserr, sum, rmserr;
 double prev_epoch_err = 1000.;
 double lowest_error = 15.;
 int lowest;
 double hidden_error[20], output_error[2];
 float desired_value[100000][2];
 double alpha, eta0;
 float caldep;
 float jog_amt;
```

```
 float datr,dati;
 float rdatr,rdati;
 float datm,datp;
 int use;

/* Pancake coil */

 float normfac[4] = {807.,1130.,1143.,471.};
 float phase[4] = {98.,-14.,-82.,-125.};

/* TEMPORARY */

 normfac[0] *= 1.;
 normfac[1] *= 1.;
 normfac[2] *= 1.;
 normfac[3] *= 1.;

 initscr();
 scrollok(stdscr, TRUE);
 move(0, 0);
 clrtobot();
 erase();
 refresh();

 for(i=0;i<4;i++) {
  phase[i] *= 3.1415927/180.;
 }

 printw("Enter 1 to restore from file, 0 otherwise  ");
 refresh();
 scanw("%d", &restore_from_file);

 if(restore_from_file) {
  printw("Enter filename  ");
  refresh();
  scanw("%s", &wtsin[0]);
  if((strwt = fopen(&wtsin[0], "r")) == NULL) {
   printw("Error:  failure to open input data file.\n");
   refresh();
   exit(1);
  }
  fscanf(strwt, "%ld", &nn);
  fscanf(strwt, "%d", &nlayers);
  if(nlayers != 3) {
   printw("Error: Number of layers in file not equal to 3.\n");
   refresh();
   exit(1);
  }
  fscanf(strwt, "%d %d %d", &ninput, &nhidden, &noutput);
  fscanf(strwt, "%lf %lf", &alpha, &eta0);

  for(i=0; i<ninput; i++) {
   for(j=1; j<nhidden; j++) {
       fscanf(strwt, "%lf", &w0[j][i]);
       last_w0[j][i] = w0[j][i];
       d0[j][i] = 0.;
   }
```

```
    }
    for(i=0; i<nhidden; i++) {
     for(j=1; j<noutput; j++) {
          fscanf(strwt, "%lf", &w1[j][i]);
          last_w1[j][i] = w1[j][i];
          d1[j][i] = 0.;
     }
    }
    fclose(strwt);
  }
  else {
    printw("Enter number of elements in input layer ");
    refresh();
    scanw("%d", &ninput);
    printw("Enter number of elements in hidden layer ");
    refresh();
    scanw("%d", &nhidden);
    printw("Enter number of elements in output layer ");
    refresh();
    scanw("%d", &noutput);

    printw("Enter momentum  ");
    refresh();
    scanw("%lf", &alpha);
    printw("Enter learning rate  ");
    refresh();
    scanw("%lf", &eta0);
printw("alpha = %f, eta0 = %f\n", alpha, eta0);

    for(i=0; i<500; i++) {
     j = rand();
    }

    for(i=0; i<ninput; i++) {
     for(j=1; j<nhidden; j++) {
          w0[j][i] = 0.1 * (1./32767.) * rand() - 0.5;
          last_w0[j][i] = w0[j][i];
          d0[j][i] = 0.;
     }
    }
    for(i=0; i<nhidden; i++) {
     for(j=1; j<noutput; j++) {
          w1[j][i] = (1./32767.) * rand() - 0.5;
          last_w1[j][i] = w1[j][i];
          d1[j][i] = 0.;
     }
    }
  }

  nodelay(stdscr, TRUE);

  strot1 = fopen("act.dat","w");
  if(strot1 == NULL) {
   printf("Unable to open output data file.\n");
   exit(-1);
  }
  strout = fopen("raw.dat","w");
```

```
  if(strout == NULL) {
   printf("Unable to open output data file.\n");
   exit(-1);
  }

  i = 0;
  for(k=0;k<nfiles;k++) {
  /*  printw("Reading file %d,%s\n",k,file[k]);*/
   refresh();
   if((strdat = fopen(file[k], "r")) == NULL) {
    printw("Error:  failure to open input data file %s.\n",file[k]);
    refresh();
    exit(1);
   }

/* Read null values */

   found_null = find_null(k,strdat);

   if(!found_null) {
    printw("Null position not found in file %s\n",file[k]);
    printw("Null position = (%f, %f, %f)\n",nullpos[0],
           nullpos[1],nullpos[2]);
    refresh();
    exit(-1);
   }
   else {
   /*  printw("Found null in file %d\n",k);*/
    refresh();
   }
   rewind(strdat);

   eoftst = fscanf(strdat,"%d %d",&data[0][0],&data[0][1]);

   while(eoftst != EOF) {
    for(j=0;j<3;j++) {
       fscanf(strdat,"%d %d",&data[j+1][0],&data[j+1][1]);
    }
    for(j=0;j<naxes[k];j++) {
       fscanf(strdat,"%f",&x[j]);
    }
/* Set the defect depth using the set_depth functions */
    i++;
    input[i][0] = 1.0;
    for(j=0;j<4;j++) {
       rdatr = ((data[j][0] - nullval[j][0]) * cos(phase[j])
          - (data[j][1] - nullval[j][1]) * sin(phase[j]))
         /normfac[j];
       rdati = ((data[j][1] - nullval[j][1]) * cos(phase[j])
          + (data[j][0] - nullval[j][0]) * sin(phase[j]))
         /normfac[j];

       input[i][2*j+1] = rdatr;
       input[i][2*j+2] = rdati;
    }
    depth = 0.0;
    usepoint = 0;
```

```
plotpoint = 0;

if(standard[k] == 11956) {
    actual_depth[i] = set_depth_z11956(x);
}
else if(standard[k] == 11954) {
    actual_depth[i] = set_depth_z11954(x);
}
else if(standard[k] == 1010) {
    actual_depth[i] = set_depth_1010(x);
}
else if(standard[k] == 4909) {
    actual_depth[i] = set_depth_4909(x);
}
else if(standard[k] == 6307) {
    actual_depth[i] = set_depth_6307(x);
}
else if(standard[k] == 2309) {
    actual_depth[i] = set_depth_2309(x);
}
else if(standard[k] == 2303) {
    actual_depth[i] = set_depth_2303(x);
}
else if(standard[k] == 4007) {
    actual_depth[i] = set_depth_4007(x);
}
else if(standard[k] == 501) {
    actual_depth[i] = set_depth_501(x);
}
else if(standard[k] == 1406) {
    actual_depth[i] = set_depth_1406(x);
}
else if(standard[k] == 411) {
    actual_depth[i] = set_depth_411(x);
}
else if(standard[k] == 1306) {
    actual_depth[i] = set_depth_1306(x);
}
else if(standard[k] == 5910) {
    actual_depth[i] = set_depth_5910(x);
}
else if(standard[k] == 4506) {
    actual_depth[i] = set_depth_4506(x);
}
else if(standard[k] == 5508) {
    actual_depth[i] = set_depth_5508(x);
}
else if(standard[k] == 6209) {
    actual_depth[i] = set_depth_6209(x);
}
else if(standard[k] == 1002) {
    actual_depth[i] = set_depth_1002(x);
}
else if(standard[k] == 514) {
    actual_depth[i] = set_depth_514(x);
}
else if(standard[k] == 3002) {
```

```
            actual_depth[i] = set_depth_3002(x);
        }
        else if(standard[k] == 4905) {
            actual_depth[i] = set_depth_4905(x);
        }
        else if(standard[k] == 205) {
            actual_depth[i] = set_depth_205(x);
        }
        else {
            printf("unknown standard.\n");
            exit(-1);
        }
        desired_value[i][0] = 0.;
        desired_value[i][1] = 0.001 * actual_depth[i] + 0.2;
        if(plotpoint){
            fprintf(strout,"%6.3f",x[0]);
            fprintf(strout,"%6.1f\n",10.*input[i][4]);
            fprintf(strot1,"%6.3f",x[0]);
            fprintf(strot1,"%6.1f\n",actual_depth[i]);
        }
        if(!usepoint){
            i--;
        }
        eoftst = fscanf(strdat,"%d %d",&data[0][0],&data[0][1]);
    }
    fclose(strdat);
}

fclose(strout);
fclose(strot1);
npt = i;
printw("npt = %d\n",npt);
refresh();

lbl1:
sserr = 0.;
hidden[0] = 1.0;

for(i=0; i<1000000; i++) {
    pattern = (int)(npt/32767. * rand());
    if(pattern >= npt) {
        pattern = npt - 1;
    }

    p0_input = &input[pattern][0];

    for(j=1; j<nhidden; j++) {
        p1_input = p0_input;
        sum = 0.;
        for(k=0; k<ninput; k++) {
            sum += *(p1_input) * w0[j][k];
            p1_input++;
        }
        hidden[j] = 1.0/(1.0 + exp(-sum));
    }

    for(i=1; i<noutput; i++) {
```

```
      sum = 0.;
      for(j=0; j<nhidden; j++) {
        sum += hidden[j] * w1[i][j];
      }
      output[i] = 1.0/(1.0 + exp(-sum));
      output_error[i] = output[i] * (1.0 - output[i])
                  * (desired_value[pattern][i] - output[i]);
    }

    for(j=0; j<nhidden; j++) {
      hidden_error[j] = 0.;
      for(i=1; i<noutput; i++) {
        hidden_error[j] += output_error[i] * w1[i][j];
      }
      hidden_error[j] *= hidden[j] * (1.0 - hidden[j]);
    }

    for(j=1; j<nhidden; j++) {
      p1_input = p0_input;
      for(k=0; k<ninput; k++) {
        w0[j][k] += *(p1_input) * eta0 * hidden_error[j]
                + alpha * d0[j][k];
        d0[j][k] = w0[j][k] - last_w0[j][k];
        last_w0[j][k] = w0[j][k];
        p1_input++;
      }
    }

    for(i=1; i<noutput; i++) {
      for(j=0; j<nhidden; j++) {
        w1[i][j] += hidden[j] * eta0 * output_error[i]
                + alpha * d1[i][j];
        d1[i][j] = w1[i][j] - last_w1[i][j];
        last_w1[i][j] = w1[i][j];
      }
    }

    caldep = (output[1] - 0.2)/0.001;

      sserr += (caldep - actual_depth[pattern])
          * (caldep - actual_depth[pattern]);
  }
  rmserr = sqrt(sserr/1000000.);
  nn += 1;
  if((nn % print_period) == 0) {
    if( ! calculate_error){
        printw("%ld %8.5f\n", nn, rmserr);
        refresh();
      }
    if(calculate_error) {
        rmserr = 0.;
        for(pattern = 0; pattern<npt; pattern++) {
            for(j=1; j<nhidden; j++) {
                sum = 0.;
                for(k=0; k<ninput; k++) {
                    sum += input[pattern][k] * w0[j][k];
                }
```

```
            hidden[j] = 1.0/(1.0 + exp(-sum));
          }
          for(i=1; i<noutput; i++) {
            sum = 0.;
            for(j=0; j<nhidden; j++) {
              sum += hidden[j] * w1[i][j];
            }
            output[i] = 1.0/(1.0 + exp(-sum));
          }

          rmserr += (desired_value[pattern][1] - output[1])
              * (desired_value[pattern][1] - output[1]);
          }

        rmserr = sqrt(rmserr/npt)/0.001;
        if(rmserr < lowest_error) {
          lowest_error = rmserr;
          lowest = 1;
          printw("%ld %8.5f **\n", nn, rmserr);
        }
        else {
          lowest = 0;
          printw("%ld %8.5f \t\n", nn, rmserr);
        }
        if(undo_bad_step) {
        if(rmserr < prev_epoch_err) {
          prev_epoch_err = rmserr;
          for(i=1; i<noutput; i++) {
            for(j=0; j<nhidden; j++) {
              prev_epoch_w1[i][j] = w1[i][j];
              prev_epoch_d1[i][j] = d1[i][j];
            }
          }
          for(j=1; j<nhidden; j++) {
            for(k=0; k<ninput; k++) {
              prev_epoch_w0[j][k] = w0[j][k];
              prev_epoch_d0[j][k] = d0[j][k];
            }
          }
        }
        else {
          printw("last epoch changes reversed.\n");
          for(i=1; i<noutput; i++) {
            for(j=0; j<nhidden; j++) {
              w1[i][j] = prev_epoch_w1[i][j];
              last_w1[i][j] = w1[i][j];
              d1[i][j] = prev_epoch_d1[i][j];
            }
          }
          for(j=1; j<nhidden; j++) {
            for(k=0; k<ninput; k++) {
              w0[j][k] = prev_epoch_w0[j][k];
              last_w0[j][k] = w0[j][k];
              d0[j][k] = prev_epoch_d0[j][k];
/*printw("j k w0 %d %d %f", j, k, w0[j][k]);*/
            }
          }
```

```
            }
            }
            refresh();
          }
           }
  ch = getch();
  if((ch == 's') || (lowest)) {
   if(ch == 's') {
    nodelay(stdscr, FALSE);
    printw("\nEnter filename  ");
    refresh();
    scanw("%s", &wtsout[0]);
   }
   else {
    strcpy(&wtsout[0], "best.wts");
   }
   if((strout = fopen(&wtsout[0], "w")) == NULL) {
    printw("error:  failure to open output data file.\n");
    refresh();
    exit(1);
   }
   fprintf(strout, "%ld\n", nn);
   fprintf(strout, "3 ");
   fprintf(strout, "%d %d %d\n", ninput, nhidden, noutput);
   fprintf(strout, "%f %f\n", alpha, eta0);
   for(i=0; i<ninput; i++) {
    for(j=1; j<nhidden; j++) {
        fprintf(strout, "%f ", w0[j][i]);
    }
    fprintf(strout, "\n");
   }
   for(i=0; i<nhidden; i++) {
    for(j=1; j<noutput; j++) {
        fprintf(strout, "%f ", w1[j][i]);
    }
    fprintf(strout, "\n");
   }
   fclose(strout);
   nodelay(stdscr, TRUE);
   goto lbl1;
  }
  else if(ch == 'j') {
   nodelay(stdscr, FALSE);
   printw("\nenter amount to jog weights  ");
   refresh();
   scanw("%f", &jog_amt);

   for(i=1; i<noutput; i++) {
    for(j=0; j<nhidden; j++) {
        w1[i][j] += (jog_amt * (rand() - 16384))/16384.;
        d1[i][j] = 0.;
        last_w1[i][j] = w1[i][j];
    }
   }

   for(j=1; j<nhidden; j++) {
    for(k=0; k<ninput; k++) {
```

```
            w0[j][k] += (jog_amt * (rand() - 16384))/16384.;
            d0[j][k] = 0.;
            last_w0[j][k] = w0[j][k];
        }
      }

      nodelay(stdscr, TRUE);
      goto lbl1;
    }
    else if(ch == 'q') {
      goto lbl2;
    }
    else if(ch == 'l') {
      nodelay(stdscr, FALSE);
      printw("\ncurrent learning rate = %f\n", eta0);
      printw("enter learning rate  ");
      refresh();
      scanw("%lf", &eta0);

      for(i=1; i<noutput; i++) {
        for(j=0; j<nhidden; j++) {
            d1[i][j] = 0.;
        }
      }

      for(j=1; j<nhidden; j++) {
        for(k=0; k<ninput; k++) {
            d0[j][k] = 0.;
        }
      }

      nodelay(stdscr, TRUE);
      goto lbl1;
    }
    else if(ch == 'm') {
      nodelay(stdscr, FALSE);
      printw("\ncurrent momentum = %f\n", alpha);
      printw("enter momentum  ");
      refresh();
      scanw("%lf", &alpha);
      nodelay(stdscr, TRUE);
      goto lbl1;
    }
    else if(ch == 'c') {
      printw("\nconcise mode\n");
      refresh();
      print_period = 5;
      goto lbl1;
    }
    else if(ch == 'v') {
      printw("\nverbose mode\n");
      refresh();
      print_period = 1;
      goto lbl1;
    }
    else if(ch == 'e') {
      calculate_error = calculate_error ^ 1;
```

```
    printw("\n");
    goto lbl1;
  }
  else if(ch == 'u') {
    undo_bad_step++;
    undo_bad_step = undo_bad_step %2;
    printw("\n");
    goto lbl1;
  }
  else if(ch == 'p') {
    strout = fopen("bpntmp.dat","w");
    if(strout == NULL) {
      printw("unable to open output data file.\n");
      refresh();
      goto lbl1;
    }
    for(k=0;k<nfiles;k++) {
      strdat = fopen(file[k],"r");
      if(strdat == NULL) {
          printw("unable to open input data file %s\n",file[k]);
          refresh();
          goto lbl1;
      }
      fclose(strdat);
    }
    fclose(strout);
  }
  else {
    goto lbl1;
  }

 lbl2:
  endwin();
}

float set_depth_z11956(float *xp)
{
/* set actual depths for standard z-11956 */
  float x,y,z;
  float width=0.01;
  float p20=3.484;
  float p40=4.234;
  float p60=4.9838;
  float p80=5.7365;

  x = xp[0];
  y = xp[1];
  z = xp[2];

  if(fabs(x - p20) < 0.15) {
    if(fabs(x - p20) < 0.07) {
      depth  = 21.;
    }
    else {
      depth = (0.15 - fabs(p20 - x)) * 21./0.08;
    }
  }
```

```
  else if(fabs(x - p40) < 0.2) {
   depth = 38.*exp(-(x-p40)*(x-p40)/width);
  }
  else if(fabs(x - p60) < 0.2) {
   depth = 56.*exp(-(x-p60)*(x-p60)/width);
  }
  else if(fabs(x - p80) < 0.2) {
   depth = 75.*exp(-(x-p80)*(x-p80)/width);
  }
  usepoint = 1;
  return depth;
}

float set_depth_z11954(float *xp)
{
/* set actual depths for axial notches in standard z-11954 */
 float x,y,z;
 float p100 = 2.162;
 float p80 = 3.41;
 float p60 = 4.66;
 float p40 = 5.91;
 float p20 = 7.16;

 x = xp[0];

 if(fabs(x-p100) < 0.175) {
  if(fabs(x-p100) < 0.125) {
   depth = 100.;
  }
  else {
   depth = (0.175-fabs(x-p100)) * 100./0.05;
  }
 }
 else if(fabs(x-p80) < 0.175) {
  if(fabs(x-p80) < 0.125) {
   depth = 81.;
  }
  else {
   depth = (0.175 - fabs(x-p80)) * 81./0.05;
  }
 }
 else if(fabs(x-p60) < 0.175) {
  if(fabs(x-p60) < 0.125) {
   depth = 61.;
  }
  else {
   depth = (0.175 - fabs(x-p60)) * 61./0.05;
  }
 }
 else if(fabs(x-p40) < 0.175) {
  if(fabs(x-p40) < 0.125) {
   depth = 41.;
  }
  else {
   depth = (0.175 - fabs(x-p40)) * 41./0.05;
  }
 }
```

```
  else if(fabs(x-p20) < 0.175) {
   if(fabs(x-p20) < 0.125) {
    depth = 20.;
   }
   else {
    depth = (0.175 - fabs(x-p20)) * 20./0.05;
   }
  }
  usepoint=1;
  return depth;
}

float set_depth_2309(float *xp)
{
/* set actual depths for sample w-23-09 */
  float x,y,z;
  float width=0.01;
  float p0=4.30;
  float p1=4.80;

  x = xp[0];
  y = xp[1];
  z = xp[2];

  if (fabs(z - 135.0)< 0.001){
   usepoint = 1;
   if(fabs(x - p1) < 0.3) {
    depth = 100.*exp(-(x-p1)*(x-p1)/width);
   }
  }
  if(fabs(x - p0) < 0.15) {
   usepoint = 1;
  }
  return depth;
}
```

About 20 pages of code are omitted at this point. They are all similar to those already given and are used to specify the position and depth of the defects in the individual samples.

```
int find_null(int k,FILE *strdat)
{
  int data[4][2];
  float x[3];
  int found_null = 0;
  int is_null_pos;
  int eoftst;
  int j;

  eoftst = fscanf(strdat,"%d %d",&data[0][0],&data[0][1]);

  while((eoftst != EOF) && (!found_null)) {
   for(j=0;j<3;j++) {
    fscanf(strdat,"%d %d",&data[j+1][0],&data[j+1][1]);
   }
   for(j=0;j<naxes[k];j++) {
    fscanf(strdat,"%f",&x[j]);
   }
   is_null_pos = 0;
```

```
    for(j=0;j<naxes[k];j++) {
     if(fabs(nullpos[k][j] - x[j]) < 0.005) {
         is_null_pos++;
     }
    }

    if(is_null_pos == naxes[k]) {
     found_null = 1;
     for(j=0;j<4;j++) {
         nullval[j][0] = data[j][0];
         nullval[j][1] = data[j][1];
     }
    }
    eoftst = fscanf(strdat,"%d %d",&data[0][0],&data[0][1]);
  }
  return found_null;
}
```

Program defplot.c

This program is run on the HP and calculates all of the property data for all of the different samples from the neural network weight files. It is intended for use with the HP plotting package, xgraph. The values of NART in the program determine the location of the OD artifacts, with NART going from zero for no artifact to five for the artifact moved all the way across the sample. The plot produced will be a three-dimensional or "C-Scan" plot on the HP computer if the variable delz is assigned a non-zero value (usually 0.1). When the program is run with delz = 0.0, then each value of the rotation of the probe is assigned a different color. The depth, length, and location of any defects are read from the graph that is plotted with the command "xgraph samplename.def." A scrip command named "plotdef" has been written on the HP to plot all of the samples one after the other. As soon as one is finished, a control-c can be typed and the next one will be plotted.

Running this program will generate 72 files, with each property for each sample. The weights are read from a file designated by DWTS, TWTS, and MWTS for the defect depth, the tube support, and the magnetite, respectively. When the defect depth is being calculated, a bias of 15 to 20% is added to the depth value to achieve a better match to the metallography values. The program bpnd.c (and variations for the tube support and magnetite) must have been run to determine these values. The neural network weight files automatically specify the needed information about the configuration of the network used in the training.

```
/*
 * defplot.c   version  June 6, 1995  - for OD artifact scans
 * Program calculates defect depth (sample.def), tube support
 * (sample.tsp), magnetite (sample.mag) in a format that can be read
 * into xgraph for plotting.  xgraph sample.def sample.mag sample.tsp
 * will display all data together.  Data are plotted in a C-scan type
 * of display if delz is greater than 0.0
 */

#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

#define DWTS "best.wts"
#define TWTS "besttsp.wts"
```

```
#define MWTS "bestmag.wts"
#define NART 0   /* allowable values 0 through 5  */
#define PATHNAME "/hd1/metdat/"

float bpn_defect_depth(void);
float bpn_tube_support(void);
float bpn_magnetite(void);
float bpn_copper(void);

int find_null(int,FILE *);
int nfiles = 24;
char file[][80] = {"b-10-02","b-59-10","b-45-06","b-49-05","b-30-02",
                   "b-10-10","b-49-09","b-62-09","b-55-08","e-13-06",
                   "l-14-06","w-23-03","w-40-07","w-23-09","w-23-10",
                   "5-01","5-14","5-08","2-05","2-12",
                    "4-11","f-08","e-11-06","b-63-07"};
float nullpos[][3] = { {3.53,0.,0.},
                       {4.48,0.,0.},
                  {3.33,0.,0.},
                       {3.13,0.,0.},
                       {3.43,0.,0.},
                       {2.73,0.,0.},
                       {3.63,0.,0.},
                       {4.08,0.,0.},
                       {4.23,0.,0.},
                       {6.43,0.,0.},
                       {2.78,0.,0.},
                       {3.93,0.,0.},
                       {3.33,0.,0.},
                       {3.98,0.,0.},
                       {3.93,0.,0.},
                       {5.37,0.,0.},
                       {3.21,0.,0.},
                       {4.30,0.,0.},
                       {3.47,0.,0.},
                       {7.93,0.,0.},
                       {6.40,0.,0.},
                       {2.88,0.,0.},
                       {5.13,0.,0.},
                       {3.79,0.,0.}};
int naxes[] = {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,2,2,2,2,2};

float nullval[4][2];
double rdg[32];
float pe[3][33];

int readdefwt = 1;
int readtspwt = 1;
int readmagwt = 1;
int readcuwt = 1;

void main(void)
{
  long int np;
  int found_null;
  int k,i2, i3;
  float pos;
```

```
 float actual_depth[100000];
 double dfdep;
 float bpndepth, bpntsp, bpnmag;
 float bpndeptho = 0.;
 float bpntspo = 0.;
 float bpnmago = 0.;
 float artval;
 int layer, i, j, ncoil;
 int eoftst;
 float data[4][2];
 int use;
 float x[3], xo[3];
 float xp, xpo;
 float delx; /* offset value for x axis to produce 3D effect */
 float delz = 0.;/* offset value for rotational axis for 3D effect .1*/
 int rotax;     /* Denotes axis is the rotational axis  */
 int artax;     /* denotes the y or OD artifact axis     */
 float raw, delraw;
 float rawo;
 float bpndderr;
 float delta[32];
 char full_name[80],rawname[80],defname[80],tspname[80],magname[80];
 float y0[] = {3.725,2.775,3.925,4.125,3.825,4.525,3.625,3.175,3.025,
          1.925,4.475,3.475,4.075,3.425,3.475,4.475,5.125,3.975};
 int nart = NART;
 float delart[] = {0.0,0.0,0.3625,0.725,1.2275,1.73};

 FILE *strdat, *strotdc, *strottc, *strotmc, *strotr;

 float normfac[4] = {807.,1130.,1143.,471.};
 float phase[4] = {98.,-14.,-82.,-125.};

 delx = 0.05 * delz;  /* smaller x axis than for z axis for 3D effect */
 delraw = 0.2 * delz; /* smaller value needed for raw readings   */
 for(i=0;i<4;i++) {
  phase[i] *= 3.1415927/180.;
 }
 for(k=0;k<nfiles;k++) {
  np = 0;
  ncoil = 0;
  rotax = naxes[k] - 1;
  artax = naxes[k] - 2;
  xo[rotax] = -999.;
  strcpy(defname, file[k]);
  strcpy(tspname, file[k]);
  strcpy(magname, file[k]);
  strcpy(rawname, file[k]);
  strcat(defname,".def");
  strcat(tspname,".tsp");
  strcat(magname,".mag");
  strcat(rawname,".raw");

  strotdc = fopen(defname,"w");
  if(strotdc == NULL) {
   printf("Unable to open calculated defect data file.\n");
   exit(-1);
  }
```

```
    strottc = fopen(tspname,"w");
    if(strottc == NULL) {
      printf("Unable to open calculated tube support data file.\n");
      exit(-1);
    }
    strotmc = fopen(magname,"w");
    if(strotmc == NULL) {
      printf("Unable to open calculated magnetite data file.\n");
      exit(-1);
    }
    strotr = fopen(rawname,"w");
    if(strotr == NULL) {
      printf("Unable to open raw output data file.\n");
      exit(-1);
    }
/*    printf("%s, ", file[k]);
    printf("%f  \n",y0[k]);     */
    fprintf(strotdc,"\"Def %d %s\n",nart, file[k]);
    fprintf(strottc,"\"Tsp %d %s\n",nart, file[k]);
    fprintf(strotmc,"\"Mag %d %s\n",nart, file[k]);
    fprintf(strotr,"\"Raw %d %s\n",nart, file[k]);
    strcpy(full_name, PATHNAME);
    strcat(full_name, file[k]);
    strdat = fopen(full_name,"r");
    if(strdat == NULL) {
      printf("Unable to open input file %s\n",full_name);
      exit(-1);
     }

    found_null = find_null(k,strdat);
    if(!found_null) {
     printf("Null position not found.\n");
     exit(-1);
    }
/*  set the artifact position       */
    if(nart == 0){
     artval = 0.1;
    }
    else{
     artval = y0[k]-delart[nart];
    }
/* read the data files and perform the calculations      */
    rewind(strdat);
    eoftst = fscanf(strdat,"%f %f",&data[0][0],&data[0][1]);
    while(eoftst != EOF) {

     for(i2 = 0; i2 < 3; i2++) {
        fscanf(strdat, "%f %f", &data[i2+1][0],&data[i2+1][1]);
     }
     for(i2=0;i2<4;i2++) {

        rdg[2*i2+1] = ((data[i2][0] - nullval[i2][0]) * cos(phase[i2])
                - (data[i2][1] - nullval[i2][1]) * sin(phase[i2]))
          /normfac[i2];
        rdg[2*i2+2] = ((data[i2][1] - nullval[i2][1]) * cos(phase[i2])
                + (data[i2][0] - nullval[i2][0]) * sin(phase[i2]))
          /normfac[i2];
```

```
    }
    for(i2=0;i2<naxes[k];i2++) {
        fscanf(strdat,"%f",&x[i2]);
    }

    use = 1;
    if(use) {
        pe[0][0] = 1.;
        pe[1][0] = 1.;
        pe[2][0] = 1.;

        for(i=0; i<8; i++) {
         pe[0][i+1] = ((float)rdg[i+1]);
        }
        if((fabs(x[artax] - artval) < 0.005) || (artax == 0)) {
         bpndepth = bpn_defect_depth();
         bpntsp = bpn_tube_support();
         bpnmag = bpn_magnetite();
         np++;
/*  Section to generate C-scan for rotational axis          */
         xp=x[0]+delx*x[rotax];
         bpndepth = bpndepth + delz*x[rotax];
         bpntsp = bpntsp + delz*x[rotax];
         bpnmag = bpnmag + delz*x[rotax];
         raw = rdg[4] + delraw*x[rotax];
/*  End of C-scan section; now print it to a file */
         if(xo[rotax] == x[rotax] ){
          fprintf(strotdc,"%5.3f ",10.*xp);
          fprintf(strottc,"%5.3f ",10.*xp);
          fprintf(strotmc,"%5.3f ",10.*xp);
          fprintf(strotr,"%5.3f ",10.*xp);
          fprintf(strotdc,"%4.1f \n",bpndepth);
          fprintf(strottc,"%4.1f \n",bpntsp);
          fprintf(strotmc,"%4.1f \n",bpnmag);
          fprintf(strotr,"%6.4f \n",raw);
         }
        else{
         if(np == 1){
          bpndeptho = bpndepth;
          bpntspo = bpntsp;
          bpnmago = bpnmag;
          rawo = raw;
          xpo = xp;
          xo[rotax] = x[rotax];
          if(fabs(delz) < 0.001){
           fprintf(strotdc,"%5.3f ",10.*xp);
           fprintf(strottc,"%5.3f ",10.*xp);
           fprintf(strotmc,"%5.3f ",10.*xp);
           fprintf(strotr,"%5.3f ",10.*xp);
           fprintf(strotdc,"%4.1f \n",bpndepth);
           fprintf(strottc,"%4.1f \n",bpntsp);
           fprintf(strotmc,"%4.1f \n",bpnmag);
           fprintf(strotr,"%6.4f \n",raw);
          }
    }
      if(fabs(delz) > 0.001){
       fprintf(strotdc,"%5.3f ",10.*xpo);
```

```
            fprintf(strottc,"%5.3f ",10.*xpo);
            fprintf(strotmc,"%5.3f ",10.*xpo);
            fprintf(strotr,"%5.3f ",10.*xpo);
            fprintf(strotdc,"%4.1f \n",bpndeptho);
            fprintf(strottc,"%4.1f \n",bpntspo);
            fprintf(strotmc,"%4.1f \n",bpnmago);
            fprintf(strotr,"%6.4f \n",rawo);
            fprintf(strotdc,"%5.3f ",10.*xp);
            fprintf(strottc,"%5.3f ",10.*xp);
            fprintf(strotmc,"%5.3f ",10.*xp);
            fprintf(strotr,"%5.3f ",10.*xp);
            fprintf(strotdc,"%4.1f \n",bpndepth);
            fprintf(strottc,"%4.1f \n",bpntsp);
            fprintf(strotmc,"%4.1f \n",bpnmag);
            fprintf(strotr,"%6.4f \n",raw);
           }
           bpndeptho = bpndepth;
           bpntspo = bpntsp;
           bpnmago = bpnmag;
           rawo = raw;
           xpo = xp;
           xo[rotax] = x[rotax];
           if(fabs(delz) < 0.001){  /* Plot different colors if not 3D */
            fprintf(strotdc,"\n\"%dscn\n", ncoil);
            fprintf(strottc,"\n\"%dscn\n", ncoil);
            fprintf(strotmc,"\n\"%dscn\n", ncoil);
            fprintf(strotr,"\n\"%dscn\n", ncoil);
            ncoil ++;
           }
          }
          }
          eoftst = fscanf(strdat,"%f %f",&data[0][0],&data[0][1]);
     }
    }
    printf("%s, ", file[k]);
    fclose(strdat);
    fclose(strotdc);
    fclose(strottc);
    fclose(strotmc);
    fclose(strotr);
  }

  printf("\n %d readings\n", np);
}

float bpn_defect_depth(void)
{
  FILE *strwt;
  static float weight[2][33][20];
  static int ddnlayers, ddnpe[3];
  long nn;
  int i, j, layer;
  float sum, bpndepth, alpha, eta;

  if(readdefwt){
   if((strwt = fopen(DWTS, "r"))== NULL){
    printf(" Failed to open input data file for weights.\n");
```

```
      exit(1);
    }
  /*  Read the weight file                    */
    fscanf(strwt, "%ld", &nn);
    fscanf(strwt, "%d", &ddnlayers);
    printf( DWTS);
    printf(" %d layers: ", ddnlayers);
    for(i=0; i<ddnlayers; i++) {
     fscanf(strwt, "%d", &ddnpe[i]);
     printf("lyr%d %d ", i, ddnpe[i]);
    }
    printf("\n");
    fscanf(strwt, "%f %f", &alpha, &eta);

    for(layer=0; layer<ddnlayers-1; layer++) {
     for(i=0; i<ddnpe[layer]; i++) {
         for(j=1; j<ddnpe[layer+1]; j++) {
          fscanf(strwt, "%f", &weight[layer][i][j]);
         }
     }
    }
    readdefwt --;
    fclose (strwt);
  }

    for(layer=1; layer<ddnlayers; layer++) {
       for(i=1; i<ddnpe[layer]; i++){
          sum = 0.;
          for(j=0; j<ddnpe[layer-1]; j++) {
             sum += pe[layer-1][j] * weight[layer-1][j][i];
          }
          pe[layer][i] = 1.0/(1.0 + exp(-sum));
       }
    }
    bpndepth = (pe[2][1] - 0.2)/0.001; /* match to scaling */
    return(bpndepth);
}

float bpn_tube_support(void)
{
 FILE *strwt;
 static float weight[2][33][20];
 static int ddnlayers, ddnpe[3];
 long nn;
 int i, j, layer;
 float sum, bpntsp, alpha, eta;

 if(readtspwt){
  if((strwt = fopen(TWTS, "r"))== NULL){
   printf(" Failed to open input data file for weights.\n");
   exit(1);
  }
/*  Read the weight file                    */
  fscanf(strwt, "%ld", &nn);
  fscanf(strwt, "%d", &ddnlayers);
  printf( TWTS);
  printf(" %d layers: ", ddnlayers);
```

```
    for(i=0; i<ddnlayers; i++) {
     fscanf(strwt, "%d", &ddnpe[i]);
     printf("lyr%d %d ", i, ddnpe[i]);
    }
    printf("\n");
    fscanf(strwt, "%f %f", &alpha, &eta);

    for(layer=0; layer<ddnlayers-1; layer++) {
     for(i=0; i<ddnpe[layer]; i++) {
         for(j=1; j<ddnpe[layer+1]; j++) {
          fscanf(strwt, "%f", &weight[layer][i][j]);
         }
     }
    }
    readtspwt --;
    fclose (strwt);
  }

    for(layer=1; layer<ddnlayers; layer++) {
        for(i=1; i<ddnpe[layer]; i++){
            sum = 0.;
            for(j=0; j<ddnpe[layer-1]; j++) {
                sum += pe[layer-1][j] * weight[layer-1][j][i];
            }
            pe[layer][i] = 1.0/(1.0 + exp(-sum));
        }
    }
    bpntsp = (pe[2][1] - 0.2)/0.01; /* match to scaling */
    return(bpntsp);
}

float bpn_magnetite(void)
{
 FILE *strwt;
 static float weight[2][33][20];
 static int ddnlayers, ddnpe[3];
 long nn;
 int i, j, layer;
 float sum, bpnmag, alpha, eta;

 if(readmagwt){
  if((strwt = fopen(MWTS, "r"))== NULL){
   printf(" Failed to open input data file for weights.\n");
   exit(1);
  }
/*  Read the weight file                    */
  fscanf(strwt, "%ld", &nn);
  fscanf(strwt, "%d", &ddnlayers);
  printf( MWTS);
  printf(" %d layers; ", ddnlayers);
  for(i=0; i<ddnlayers; i++) {
   fscanf(strwt, "%d", &ddnpe[i]);
   printf("lyr%d %d ", i, ddnpe[i]);
  }
  printf("\n");
  fscanf(strwt, "%f %f", &alpha, &eta);
```

```
    for(layer=0; layer<ddnlayers-1; layer++) {
      for(i=0; i<ddnpe[layer]; i++) {
          for(j=1; j<ddnpe[layer+1]; j++) {
            fscanf(strwt, "%f", &weight[layer][i][j]);
          }
      }
    }
    readmagwt --;
    fclose (strwt);
  }

    for(layer=1; layer<ddnlayers; layer++) {
        for(i=1; i<ddnpe[layer]; i++){
            sum = 0.;
            for(j=0; j<ddnpe[layer-1]; j++) {
                sum += pe[layer-1][j] * weight[layer-1][j][i];
            }
            pe[layer][i] = 1.0/(1.0 + exp(-sum));
        }
    }
    bpnmag = (pe[2][1] - 0.2)/0.01; /* match to scaling */
    return(bpnmag);
}

int find_null(int k,FILE *strdat)
{
 int data[4][2];
 float x[3];
 int found_null = 0;
 int is_null_pos;
 int eoftst;
 int j;

 eoftst = fscanf(strdat,"%d %d",&data[0][0],&data[0][1]);

 while((eoftst != EOF) && (!found_null)) {
  for(j=0;j<3;j++) {
   fscanf(strdat,"%d %d",&data[j+1][0],&data[j+1][1]);
  }
  for(j=0;j<naxes[k];j++) {
   fscanf(strdat,"%f",&x[j]);
  }
  is_null_pos = 0;
  for(j=0;j<naxes[k];j++) {
   if(fabs(nullpos[k][j] - x[j]) < 0.005) {
       is_null_pos++;
   }
  }

  if(is_null_pos == naxes[k]) {
   found_null = 1;
   for(j=0;j<4;j++) {
       nullval[j][0] = data[j][0];
       nullval[j][1] = data[j][1];
   }
  }
  eoftst = fscanf(strdat,"%d %d",&data[0][0],&data[0][1]);
```

```
  }
  return found_null;
}
```

Program defplotm.c

This program is similar to the program defplot.c except it arranges the data in a form for plotting using Mathematica. The property data are displayed in an array of integers. For defects, it goes from 0 to 100. These files that are generated are named the same as the files generated by plotdef.c, and will overwrite them. The files must be transfered to the PC over the network for plotting.

```
/*
 * defplotm.c   version  July 12, 1995  - for OD artifact scans
 * Program calculates defect depth (sample.def), tube support
 * (sample.tsp), magnetite (sample.mag) in a format that can be read
 * into mathematica for plotting.
 */

#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

#define DWTS "best.wts"
#define TWTS "besttsp.wts"
#define MWTS "bestmag.wts"
#define NART 3   /* allowable values 0 through 5  */
#define PATHNAME "/hd1/metdat/"

float bpn_defect_depth(void);
float bpn_tube_support(void);
float bpn_magnetite(void);
float bpn_copper(void);

int find_null(int,FILE *);
int nfiles = 24;
char file[][80] = {"b-10-02","b-59-10","b-45-06","b-49-05","b-30-02",
                   "b-10-10","b-49-09","b-62-09","b-55-08","e-13-06",
                   "l-14-06","w-23-03","w-40-07","w-23-09","w-23-10",
                   "5-01","5-14","5-08","2-05","2-12",
                    "4-11","f-08","e-11-06","b-63-07"};
float nullpos[][3] = { {3.53,0.,0.},
                       {4.48,0.,0.},
                  {3.33,0.,0.},
                       {3.13,0.,0.},
                       {3.43,0.,0.},
                       {2.73,0.,0.},
                       {3.63,0.,0.},
                       {4.08,0.,0.},
                       {4.23,0.,0.},
                       {6.43,0.,0.},
                       {2.78,0.,0.},
                       {3.93,0.,0.},
```

```
                {3.33,0.,0.},
                {3.98,0.,0.},
                {3.93,0.,0.},
                {5.37,0.,0.},
                {3.21,0.,0.},
                {4.30,0.,0.},
                {3.47,0.,0.},
                {7.93,0.,0.},
                {6.40,0.,0.},
                {2.88,0.,0.},
                {5.13,0.,0.},
                {3.79,0.,0.}};
int naxes[] = {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,2,2,2,2,2};

float nullval[4][2];
double rdg[32];
float pe[3][33];

int readdefwt = 1;
int readtspwt = 1;
int readmagwt = 1;
int readcuwt = 1;

void main(void)
{
 long int np;
 int found_null;
 int k,i2, i3;
 float pos;
 float actual_depth[100000];
 double dfdep;
 float bpndepth, bpntsp, bpnmag;
 float artval;
 int layer, i, j, ncoil;
 int eoftst;
 float data[4][2];
 int use;
 float x[3], xo[3];
 float xp, xpo;
 float delx; /* offset value for x axis to produce 3D effect */
 float delz = 0.;/* offset value for rotational axis for 3D effect .1*/
 int rotax;     /* Denotes axis is the rotational axis  */
 int artax;     /* denotes the y or OD artifact axis     */
 float raw, delraw;
 float rawo;
 float bpndderr;
 float delta[32];

 char full_name[80],rawname[80],defname[80],tspname[80],magname[80];
 float y0[] = {3.725,2.775,3.925,4.125,3.825,4.525,3.625,3.175,3.025,
          1.925,4.475,3.475,4.075,3.425,3.475,4.475,5.125,3.975};
 int nart = NART;
 float delart[] = {0.0,0.0,0.3625,0.725,1.2275,1.73};

 FILE *strdat, *strotdc, *strottc, *strotmc, *strotr;

 float normfac[4] = {807.,1130.,1143.,471.};
```

```
float phase[4] = {98.,-14.,-82.,-125.};
int  depth, tsp, mag;



delx = 0.05 * delz;  /* smaller x axis than for z axis for 3D effect */
delraw = 0.2 * delz; /* smaller value needed for raw readings   */
for(i=0;i<4;i++) {
 phase[i] *= 3.1415927/180.;
}
for(k=0;k<nfiles;k++) {
 np = 0;
 ncoil = 0;
 rotax = naxes[k] - 1;
 artax = naxes[k] - 2;
 xo[rotax] = -999.;
 strcpy(defname, file[k]);
 strcpy(tspname, file[k]);
 strcpy(magname, file[k]);
 strcpy(rawname, file[k]);
 strcat(defname,".def");
 strcat(tspname,".tsp");
 strcat(magname,".mag");
 strcat(rawname,".raw");

 strotdc = fopen(defname,"w");
 if(strotdc == NULL) {
  printf("Unable to open calculated defect data file.\n");
  exit(-1);
 }
 strottc = fopen(tspname,"w");
 if(strottc == NULL) {
  printf("Unable to open calculated tube support data file.\n");
  exit(-1);
 }
 strotmc = fopen(magname,"w");
 if(strotmc == NULL) {
  printf("Unable to open calculated magnetite data file.\n");
  exit(-1);
 }
 strotr = fopen(rawname,"w");
 if(strotr == NULL) {
  printf("Unable to open raw output data file.\n");
  exit(-1);
 }
 strcpy(full_name, PATHNAME);
 strcat(full_name, file[k]);
 strdat = fopen(full_name,"r");
 if(strdat == NULL) {
  printf("Unable to open input file %s\n",full_name);
  exit(-1);
 }

 found_null = find_null(k,strdat);
 if(!found_null) {
  printf("Null position not found.\n");
  exit(-1);
```

```
    }
/*  set the artifact position       */
    if(nart == 0){
     artval = 0.1;
    }
    else{
     artval = y0[k]-delart[nart];
    }
/* read the data files and perform the calculations      */
    rewind(strdat);
    eoftst = fscanf(strdat,"%f %f",&data[0][0],&data[0][1]);
    while(eoftst != EOF) {

     for(i2 = 0; i2 < 3; i2++) {
         fscanf(strdat, "%f %f", &data[i2+1][0],&data[i2+1][1]);
     }
     for(i2=0;i2<4;i2++) {

         rdg[2*i2+1] = ((data[i2][0] - nullval[i2][0]) * cos(phase[i2])
                 - (data[i2][1] - nullval[i2][1]) * sin(phase[i2]))
          /normfac[i2];
         rdg[2*i2+2] = ((data[i2][1] - nullval[i2][1]) * cos(phase[i2])
                 + (data[i2][0] - nullval[i2][0]) * sin(phase[i2]))
          /normfac[i2];
    }
    for(i2=0;i2<naxes[k];i2++) {
        fscanf(strdat,"%f",&x[i2]);
    }

    use = 1;
    if(use) {
        pe[0][0] = 1.;
        pe[1][0] = 1.;
        pe[2][0] = 1.;

        for(i=0; i<8; i++) {
          pe[0][i+1] = ((float)rdg[i+1]);
        }
        if((fabs(x[artax] - artval) < 0.005) || (artax == 0)) {
          bpndepth = bpn_defect_depth();
          bpntsp = bpn_tube_support();
          bpnmag = bpn_magnetite();
          np++;
          if(np == 1){
           xo[rotax] = x[rotax];
     }
          depth = bpndepth ;
          tsp = bpntsp;
          mag = bpnmag;
          raw = rdg[4];
          if(depth > 100){
           depth = 100;
          }
          if(depth < 0){
           depth = 0;
          }
          if(xo[rotax] == x[rotax] ){
```

```
            fprintf(strotdc,"%d ",depth);
            fprintf(strottc,"%d ",tsp);
            fprintf(strotmc,"%d ",mag);
            fprintf(strotr,"%6.4f ",raw);
           }
          else{
           xc[rotax] = x[rotax];
           fprintf(strotdc,"\n");
           fprintf(strottc,"\n");
           fprintf(strotmc,"\n");
           fprintf(strotr,"\n");
           fprintf(strotdc,"%d ",depth);
           fprintf(strottc,"%d ",tsp);
           fprintf(strotmc,"%d ",mag);
           fprintf(strotr,"%6.4f ",raw);
           ncoil ++;
          }
          }
          eoftst = fscanf(strdat,"%f %f",&data[0][0],&data[0][1]);
     }
    }
    printf("%s, ", file[k]);
    fclose(strdat);
    fclose(strotdc);
    fclose(strottc);
    fclose(strotmc);
    fclose(strotr);
  }

  printf("\n %d readings\n", np);
}

float bpn_defect_depth(void)
{
  FILE *strwt;
  static float weight[2][33][20];
  static int ddnlayers, ddnpe[3];
  long nn;
  int i, j, layer;
  float sum, bpndepth, alpha, eta;

  if(readdefwt){
   if((strwt = fopen(DWTS, "r"))== NULL){
    printf(" Failed to open input data file for weights.\n");
    exit(1);
   }
/*  Read the weight file                    */
   fscanf(strwt, "%ld", &nn);
   fscanf(strwt, "%d", &ddnlayers);
   printf( DWTS);
   printf(" %d layers; ", ddnlayers);
   for(i=0; i<ddnlayers; i++) {
    fscanf(strwt, "%d", &ddnpe[i]);
    printf("lyr%d %d ", i, ddnpe[i]);
   }
   printf("\n");
   fscanf(strwt, "%f %f", &alpha, &eta);
```

```
    for(layer=0; layer<ddnlayers-1; layer++) {
     for(i=0; i<ddnpe[layer]; i++) {
        for(j=1; j<ddnpe[layer+1]; j++) {
         fscanf(strwt, "%f", &weight[layer][i][j]);
        }
     }
    }
    readdefwt --;
    fclose (strwt);
   }

    for(layer=1; layer<ddnlayers; layer++) {
       for(i=1; i<ddnpe[layer]; i++){
          sum = 0.;
          for(j=0; j<ddnpe[layer-1]; j++) {
             sum += pe[layer-1][j] * weight[layer-1][j][i];
          }
          pe[layer][i] = 1.0/(1.0 + exp(-sum));
       }
    }
    bpndepth = (pe[2][1] - 0.2)/0.001; /* match to scaling */
    return(bpndepth);
}

float bpn_tube_support(void)
{
 FILE *strwt;
 static float weight[2][33][20];
 static int ddnlayers, ddnpe[3];
 long nn;
 int i, j, layer;
 float sum, bpntsp, alpha, eta;

 if(readtspwt){
  if((strwt = fopen(TWTS, "r"))== NULL){
   printf(" Failed to open input data file for weights.\n");
   exit(1);
  }
/*  Read the weight file                    */
  fscanf(strwt, "%ld", &nn);
  fscanf(strwt, "%d", &ddnlayers);
  printf( TWTS);
  printf(" %d layers: ", ddnlayers);
  for(i=0; i<ddnlayers; i++) {
   fscanf(strwt, "%d", &ddnpe[i]);
   printf("lyr%d %d ", i, ddnpe[i]);
  }
  printf("\n");
  fscanf(strwt, "%f %f", &alpha, &eta);

  for(layer=0; layer<ddnlayers-1; layer++) {
   for(i=0; i<ddnpe[layer]; i++) {
      for(j=1; j<ddnpe[layer+1]; j++) {
       fscanf(strwt, "%f", &weight[layer][i][j]);
      }
   }
  }
```

```
  readtspwt --;
  fclose (strwt);
 }

  for(layer=1; layer<ddnlayers; layer++) {
     for(i=1; i<ddnpe[layer]; i++){
        sum = 0.;
        for(j=0; j<ddnpe[layer-1]; j++) {
           sum += pe[layer-1][j] * weight[layer-1][j][i];
        }
        pe[layer][i] = 1.0/(1.0 + exp(-sum));
     }
  }
  bpntsp = (pe[2][1] - 0.2)/0.01; /* match to scaling */
  return(bpntsp);
}

float bpn_magnetite(void)
{
 FILE *strwt;
 static float weight[2][33][20];
 static int ddnlayers, ddnpe[3];
 long nn;
 int i, j, layer;
 float sum, bpnmag, alpha, eta;

 if(readmagwt){
  if((strwt = fopen(MWTS, "r"))== NULL){
   printf(" Failed to open input data file for weights.\n");
   exit(1);
  }
/*  Read the weight file                    */
  fscanf(strwt, "%ld", &nn);
  fscanf(strwt, "%d", &ddnlayers);
  printf( MWTS);
  printf(" %d layers: ", ddnlayers);
  for(i=0; i<ddnlayers; i++) {
   fscanf(strwt, "%d", &ddnpe[i]);
   printf("lyr%d %d ", i, ddnpe[i]);
  }
  printf("\n");
  fscanf(strwt, "%f %f", &alpha, &eta);

  for(layer=0; layer<ddnlayers-1; layer++) {
   for(i=0; i<ddnpe[layer]; i++) {
      for(j=1; j<ddnpe[layer+1]; j++) {
       fscanf(strwt, "%f", &weight[layer][i][j]);
      }
   }
  }
  readmagwt --;
  fclose (strwt);
 }

  for(layer=1; layer<ddnlayers; layer++) {
     for(i=1; i<ddnpe[layer]; i++){
        sum = 0.;
```

```
            for(j=0; j<ddnpe[layer-1]; j++) {
                sum += pe[layer-1][j] * weight[layer-1][j][i];
            }
            pe[layer][i] = 1.0/(1.0 + exp(-sum));
        }
    }
    bpnmag = (pe[2][1] - 0.2)/0.01; /* match to scaling */
    return(bpnmag);
}

int find_null(int k,FILE *strdat)
{
 int data[4][2];
 float x[3];
 int found_null = 0;
 int is_null_pos;
 int eoftst;
 int j;

 eoftst = fscanf(strdat,"%d %d",&data[0][0],&data[0][1]);

 while((eoftst != EOF) && (!found_null)) {
  for(j=0;j<3;j++) {
   fscanf(strdat,"%d %d",&data[j+1][0],&data[j+1][1]);
  }
  for(j=0;j<naxes[k];j++) {
   fscanf(strdat,"%f",&x[j]);
  }
  is_null_pos = 0;
  for(j=0;j<naxes[k];j++) {
   if(fabs(nullpos[k][j] - x[j]) < 0.005) {
       is_null_pos++;
   }
  }

  if(is_null_pos == naxes[k]) {
   found_null = 1;
   for(j=0;j<4;j++) {
       nullval[j][0] = data[j][0];
       nullval[j][1] = data[j][1];
   }
  }
  eoftst = fscanf(strdat,"%d %d",&data[0][0],&data[0][1]);
 }
 return found_null;
}
```

Data File "samplename.dat"

This data file is created using a text editor (WordPerfect) and the metallographic data that are supplied. It is typed in using the integer format using:

axialdistance rotationalvalue defectdepth

The axial value is in terms of 0.1-in. steps from the start of the metallographic data. In some cases, an interpolation of the data furnished had to be done. The rotationalvalue is the degrees of rotation from an arbitrary point furnished by the metallography. The defectdepth is the wall percentage reported at that location. Only non-zero values need to be typed in, since the program makes all other values equal to zero. A short sample file (b-30-02.dat) for the sample b-30-02 is shown:

2 356 17
3 356 11
4 356 27
5 21 21
6 13 45
7 21 33

Program metfile.c

This program takes the metallographic depth data from the file "samplename.dat" and writes a file to be plotted by Mathematica. The scan direction and the rotational direction are reversed to match the scan and rotational direction of the eddy-current data file. The rotational starting point is matched to the eddy-current data by varying the integer rot, and the axial starting point is varied by shifting the data either to the left (pof) or the right (nof). The program writes all zero defect depths in the entire data array. Then it reads the axial value of the defect depth, the rotational value of the defect, and then the defect depth from the data file "samplename.dat." Then the file "samplename.met" is written, with the directions and offsets properly changed.

```
/*      metfile.c  Version 6-21-95 creates a file of metallography data
        to be plotted by Mathematica  */

#include <stdio.h>
#include <stdlib.h>

int main()

{
 int rot = 350; /* amount of rotational offset */
 int pof = 7;   /* move defect to left by this amount  */
 int nof = 0;   /*  move defect to right by this amount */
 int metlen = 31; /* length of metallographic plot  */
 static int data[361][50];
 int i,j, a, b, c;
 FILE *datin, *datout;
 datin = fopen("d:\\metdat\\b-63-07.dat","r");
 if(datin == NULL){
  printf("failed to open input data file\n");
  exit(-1);
 }
 datout = fopen("d:\\metdat\\b-63-07.met","w");
 if(datout == NULL){
```

```
    printf("failed to open output data file\n");
    exit(-1);
  }
  for(i=0;i<360;i++){
    for(j=0;j<50;j++){
      data[i][j]=0;
    }
  }
  while(fscanf(datin,"%d %d %d",&j, &i, &c) != EOF){
    data[i][j+pof] = c;
/*    printf("%d  %d %d\n",i,j,data[i][j]);  */
  }
  for(i=0;i<360;i++){
    a = i + rot;
    if(a > 359){
      a = a -360;
    }
    a = 360 -a ; /* change the direction of rotation   */
    for(j=0;j<metlen;j++){
      b = metlen + nof -j;
      fprintf(datout,"%d ",data[a][b]);
    }
    fprintf(datout,"\n");
  }
/*  printf("Program end  \n");  */
  return(0);
}
```

Program caldepth.ma

This program is a Mathematica Notebook, and is written to run under Version 2.2.3, modified for Windows 95. It reads the data stored on the PC and makes the three-dimensional color contour plots. The data consist of the eddy-current readings computed on the HP and transferred over the network, and the metallography generated on the PC. The starting point for the x and y axis is arbitrary. The increments used for the eddy-current data are 0.030-in. in the axial direction and 22.5° in the y or circumferential direction. For the metallographic plots, the axial direction is about 0.1 in., and the circumferential direction is 1.0°. The appearance of the plots does not match exactly due to the difference in the density of measurements used by the two techniques. In particular, with the data given every 1° for the metallographic data, the high-plot density makes the zero metallographic data appear black.

```
data=ReadList["d:\\metdat\\I-14-06.d27",
        Number,RecordLists->True];
ListPlot3D[data,PlotRange->All,
Ticks->{{{1,4.0},{17,4.5},{33,5.0},{50,5.5},{67,6.},{83,6.5},{100,7.0},
        {117,7.5},{133,8.0}}
            ,{{1,0},{5,90},{9,180},{13,270}},Automatic},
            PlotLabel -> "I-14-06 cal",
            DefaultFont -> {"Times-Bold",12}];
data=ReadList["d:\\metdat\\I-14-06.met",
        Number,RecordLists->True];
ListPlot3D[data,
            PlotLabel -> "I-14-06 met",
            DefaultFont -> {"Times-Bold",12},
            PlotRange->All,
```

```
Ticks->{{{1.4.0},{5.4.5},{10.5.0},{15.5.5},{20.6.0},{25.6.5},{30.7.0},
        {35.7.5},{40.8.0}}
          ,{{1,0},{90,90},{180,180},{270,270}},Automatic}]
```

INTERNAL DISTRIBUTION

1. W. R. Corwin
2. L. D. Chitwood
3. D. F. Craig
4. S. A. David
5-9. C. V. Dodd
10. H. W. Hayden, Jr.
11-16. D. J. McGuire
17. R. K. Nanstad
18. J. R. Pate
19. C. E. Pugh
20. W. A. Simpson
21-22. M&C Records Office
23. ORNL Patent Section
24. Central Research Library
25. Document Reference Section
26-27. Laboratory Records Department
28. Laboratory Records (RC)
29. Nuclear Safety Information Center

EXTERNAL DISTRIBUTION

30-31. NRC, OFFICE OF NUCLEAR REGULATORY RESEARCH, Washington, DC 20555

J. Craig
M. E. Mayfield
J. Muscara
C. Z. Serpan

32.-33. ELECTRIC POWER RESEARCH INSTITUTE, P.O. Box 217097, Charlotte, NC 28221

M. Elmo
K. Kryzwosz

34. BATTELLE PACIFIC NORTHWEST LABORATORIES, P.O. Box 999, Richland, WA 99352

B. Ferris

35. Steven D. Brown, 23 Greensburg Street, Delmont Borough, PA 15626

36. DOE, OAK RIDGE OPERATIONS OFFICE, Oak Ridge, TN 37831-8600

Office of Assistant Manager for Energy Research and Development

37. DOE, OFFICE OF SCIENTIFIC AND TECHNICAL INFORMATION, P. O. Box 62, Oak Ridge, TN 37831

38-286. Given distribution as shown in category RF (NTIS-10)

NRC FORM 335 (2-89) NRCM 1102, 3201, 3202

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

(See instructions on the reverse)

1. REPORT NUMBER (Assigned by NRC, Add Vol., Supp., Rev., and Addendum Numbers, if any.)

NUREG/CR-6455
ORNL/TM-13206

2. TITLE AND SUBTITLE

Data Analysis for Steam Generator Tubing Samples

3. DATE REPORT PUBLISHED

MONTH: July
YEAR: 1996

4. FIN OR GRANT NUMBER

B0417

5. AUTHOR(S)

C. V. Dodd

6. TYPE OF REPORT

Technical

7. PERIOD COVERED *(Inclusive Dates)*

8. PERFORMING ORGANIZATION – NAME AND ADDRESS *(If NRC, provide Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address; if contractor, provide name and mailing address.)*

Oak Ridge National Laboratory
Oak Ridge, TN 37831-6285

9. SPONSORING ORGANIZATION – NAME AND ADDRESS *(If NRC, type "Same as above"; if contractor, provide NRC Division, Office or Region, U.S. Nuclear Regulatory Commission, and mailing address.)*

Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, DC 20555-0001

10. SUPPLEMENTARY NOTES

J. Muscara, NRC Project Manager

11. ABSTRACT *(200 words or less)*

The objective of the Improved Eddy-Current ISI for Steam Generators program is to upgrade and validate eddy-current inspections, including probes, instrumentation, and data processing techniques for inservice inspection of new, used, and repaired steam generator tubes; to improve defect detection, classification and characterization as affected by diameter and thickness variations, denting, probe wobble, tube sheet, tube supports, copper and sludge deposits, even when defect types and other variables occur in combination; to transfer this advanced technology to NRC's mobile NDE laboratory and staff. This report provides a description of the application of advanced eddy-current neural network analysis methods for the detection and evaluation of common steam generator tubing flaws including axial and circumferential outer-diameter stress-corrosion cracking and intergranular attack. The report describes the training of the neural networks on tubing samples with known defects and the subsequent evaluation results for unknown samples. Evaluations were done in the presence of artifacts. Computer programs are given in the appendix.

12. KEY WORDS/DESCRIPTORS *(List words or phrases that will assist researchers in locating the report.)*

Array Probe
Computer Programs
Neural Networks
Steam Generators
Tubing Inspection

13. AVAILABILITY STATEMENT

Unlimited

14. SECURITY CLASSIFICATION

(This Page)
Unclassified

(This Report)
Unclassified

15. NUMBER OF PAGES

16. PRICE

Federal Recycling Program