Sept 28, 1995
Station 35 22

ENGINEERING DATA TRANSMITTAL

Page 1 of 1

1. EDT № 700688

2. To: (Receiving Organization) Distribution	3. From: (Originating Organization) Nuclear Analysis and Characterization	4. Related EDT No.:
5. Proj./Prog./Dept./Div.: Waste Tank	6. Cog. Engr.: S. H. Finfrock	7. Purchase Order No.:
8. Originator Remarks: Approval / Release		9. Equip./Component No.:
		10. System/Bldg./Facility:
		12. Major Assm. Dwg. No.:
11. Receiver Remarks:		13. Permit/Permit Application No.:
		14. Required Response Date:

15. DATA TRANSMITTED

(A) Item No.	(B) Document/Drawing No.	(C) Sheet No.	(D) Rev. No.	(E) Title or Description of Data Transmitted	(F) Impact Level	(G) Reason for Transmittal	(H) Originator Disposition	(I) Receiver Disposition
1	WHC-SD-WM-CSRS-028	All	0	System Design Description for the TMAD Code	Q	1/2	1	1

16. KEY

Impact Level (F)	Reason for Transmittal (G)	Disposition (H) & (I)
1, 2, 3, or 4 (see MRP 5.43)	1. Approval 2. Release 3. Information 4. Review 5. Post-Review 6. Dist. (Receipt Acknow. Required)	1. Approved 2. Approved w/comment 3. Disapproved w/comment 4. Reviewed no/comment 5. Reviewed w/comment 6. Receipt acknowledged

17. SIGNATURE/DISTRIBUTION (See Impact Level for required signatures)

(G) Reason	(H) Disp.	(J) Name	(K) Signature	(L) Date	(M) MSIN	(J) Name	(K) Signature	(L) Date	(M) MSIN	(G) Reason	(H) Disp.
1/2	1	Cog. Eng. SH Finfrock	[illegible]	9/27/95	H0-38						
1/2	1	Cog. Mgr. H Toffer	H Toffer	9/27/95	H0-38						
1	1	QA PJ Edwards	[illegible]	9/27/95	A4-79						
		Safety									
		Env.									
1		Ind. Rev: RD Crowe	[illegible]	9/27/95	H0-38						

18. SH Finfrock [illegible] 9/27/95 Signature of EDT Originator Date	19. Authorized Representative Date for Receiving Organization	20. H. Toffer H Toffer 9/27/95 Cognizant Manager Date	21. DOE APPROVAL (if required) Ltr No. ☐ Approved ☐ Approved w/comments ☐ Disapproved w/comments

BD-7400-172 (07/91)

RELEASE AUTHORIZATION

Document Number:	WHC-SD-WM-CSRS-028, Rev. 0
Document Title:	System Design Description for the TMAD code
Release Date:	9/28/95

This document was reviewed following the procedures described in WHC-CM-3-4 and is:

APPROVED FOR PUBLIC RELEASE

WHC Information Release Administration Specialist:

Chris Willingham 9/28/95

C. WILLINGHAM

SUPPORTING DOCUMENT

1. Total Pages 57		
2. Title System Design Description for the TMAD Code	3. Number WHC-SD-WM-CSRS-028	4. Rev No. 0
5. Key Words Neutron probe TMAD	6. Author Name: Scott H. Finfrock [illegible] Signature Organization/Charge Code 8M720 / N2546	

7. Abstract

This document serves as the System Design Description (SDD) for the TMAD code system, which includes the TMAD code and the LIBMAKR code. The SDD provides a detailed description of the theory behind the code, and the implementation of that theory. It is essential for anyone who is attempting to review or modify the code or who otherwise needs to understand the internal workings of the code. In addition, this document includes, in Appendix A, the System Requirements Specification for the TMAD system.

8. RELEASE STAMP

22

CONTENTS

LIST OF FIGURES

LIST OF TERMS

LOW	liquid observation well
MCNP	Monte Carlo n-particle
SDD	System Design Description
SRS	System Requirements Specification

SYSTEM DESIGN DESCRIPTION FOR THE TMAD CODE

1.0 INTRODUCTION

1.1 PURPOSE

This document serves as the System Design Description (SDD) for the TMAD code system (Finfrock 1995), which includes the TMAD code and the LIBMAKR code. The SDD provides a detailed description of the theory behind the code and the implementation of that theory. It is essential for anyone who is attempting to review or modify the code or who otherwise needs to understand the internal workings of the code. In addition, this document includes, in Appendix A, the System Requirements Specification (SRS) for the TMAD system.

1.2 SCOPE

The TMAD code was commissioned to facilitate the interpretation of moisture probe measurements in the Hanford Site waste tanks. In principle, the code is an interpolation routine that acts over a library of benchmark data based on two independent variables, typically anomaly size and moisture content. Two additional variables, anomaly type and detector type, can also be considered independent variables, but no interpolation is done over them. The dependant variable is detector response. The intent is to provide the code with measured detector responses from two or more detectors. The code will then interrogate (and interpolate upon) the benchmark data library and find the anomaly-type/anomaly-size/moisture-content combination that provides the closest match to the measured data.

1.3 OVERVIEW

Section 2 of this document will describe the breakdown of the system into its various components and the relationship between those components. Section 3 will identify the internal and external interfaces in the code system. Section 4 will explain the theory and function of the various modules and common data structures in the TMAD system. Appendix A will provide a description of the SRS for TMAD system.

1.4 DEFINITIONS

The following definitions will help the reader understand the neutron probe measurement process and the TMAD code:

TMAD	–	The name of the code being developed. The name was derived from the phrase "tank moisture and anomaly detection." TMAD is also used to refer to the system of codes that includes TMAD and LIBMAKR.

LIBMAKR –	The name of the auxiliary code that is being developed in conjunction with TMAD and that will be part of the TMAD code system. LIBMAKR is used to assemble the benchmark data into a library file.
Benchmark data –	Predictions of detector response to different moisture contents, anomaly types, and anomaly sizes. These data are produced by computer modeling of the waste tank and probes using the MCNP code (LANL 1986).
Neutron probe –	A device consisting of a neutron emitter and a neutron detector. The device is lowered to the bottom of a liquid observation well and then slowly raised. While it is being raised, it is continuously emitting neutrons. At predetermined intervals it measures the number of neutrons per second reaching the detector.
Scan –	The raw data recorded by the neutron probe (or other type of probe). It typically consists of a series of pairs of data points, with each pair consisting of a depth and a detector response.
Liquid observation well (LOW) –	A pipe, closed at the bottom and open at the top, that has been inserted into the waste. The LOW allows probes to be lowered into the waste without coming into direct contact with the waste. This avoids the need to decontaminate the probe after every scan.
MCNP –	The name of the code (LANL 1986) that will be used to model the detector response in the waste tank, thereby producing the benchmark data. The name is derived from the phrase "Monte Carlo neutron photon."

2.0 DECOMPOSITION DESCRIPTION

The TMAD system consists of two codes, TMAD and LIBMAKR. LIBMAKR is used to take the externally produced benchmark data and combine them into a library file that is useable by TMAD. TMAD then takes the library and externally produced scan data and processes them. The result of this processing is a prediction of the moisture content and anomaly size for each point in the scan data.

The benchmark data are produced externally by modeling each detector/anomaly/anomaly size/moisture content combination (typically using MCNP) and combining the results into properly formatted files (see Section 3.2.2.2). In order to create the TMAD library file, LIBMAKR performs a curve fit over the moisture content values of each detector/anomaly/anomaly size combination. These curves express the relationship between detector response and moisture content. Each curve is then stored in the library file in the form of its three characteristic coefficients. See Section 3.2.1.2 for a detailed description of the library file. Ideally the library file need only be created once after which TMAD can use it for any number of tank scan evaluations.

In order to determine the moisture content of a tank, TMAD must be provided with a library file and two or more detector scans. The scans consist of a series of detector readings, each representing a different depth in the tank. The scans are first interpolated to give them a common reference (i.e., the same number of points and the same depths for all of the scans). After the interpolation each point is processed to find the anomaly/anomaly size/moisture content combination that best represents the measured data.

The first step in this process is to take a point from one of the detector scans and compare it to all of the curves representing that detector. Because each moisture content curve is treated as a second order polynomial (most of the data relationships are first or second order, for a very few it may be desirable to consider higher order representations as a future upgrade), each curve will contain none, one, or two moisture values that would produce the probe measurement. By combining the possible solutions from all of the anomaly sizes (for a given detector-type/anomaly-type combination) none, one, or two curves can be produced that represent the relationship between the anomaly size and the moisture content for the given probe measurement. Note that at this point there are still an infinite number of possible solutions. This process is repeated for each detector type. At the end of this stage there are none, one, or two curves of possible solutions for each detector type.

The next step is to find the intersections between the curves. For a given anomaly type, every possible combination of two detectors is examined and the intersections are found between their representative curves. Again, because the curves are second order, there are two possible intersections for each pair of curves. One point is then chosen for each detector pair. The average of the resulting set of points is one possible unique solution. The standard deviation of the average, compared to the set, is then calculated in order to provide a measure of goodness for the solution. This procedure is then repeated for every possible combination of curve intersections. For N detectors, a set will consist of $\Sigma(N-1)$ points, that is, there are $\Sigma(N-1)$ detector pairs. If each detector is represented by two curves, and each pair of curves has two intersections, then there will be eight possible solution points for each detector pair. This means that there could be as many as $8^{\Sigma(N-1)}$ possible solutions for each anomaly. Obviously this is an extreme situation, and in most cases the number of solutions is much smaller, but the code has to be able to handle the extreme case.

Once all of the possible solutions have been identified for a given anomaly type, the process is repeated for the other anomaly types. Finally all of the possible solutions are rated according to their standard deviation, and the solution with the lowest standard deviation is selected as the solution for the given scan point. The entire process is then repeated for each subsequent scan point.

2.1 MODULE DECOMPOSITION AND DEPENDENCY DESCRIPTION

2.1.1 TMAD

The TMAD code reads in data scans and compares them to the benchmark data library. Based on this comparison, it determines the most likely moisture concentration and anomaly type and size at each point in the scans. The code consists of several modules, described below. Figure 2-1 shows a simple flow diagram for the TMAD code, and Figure 2-2 shows the dependencies between the modules.

TMAD modules:

Analyze	–	Determines moisture concentration and anomaly type and size at each point in the scan data set
Delscan	–	Removes a previously selected scan file from the list of scans to be evaluated
Findsols	–	Finds all possible solution points based on curves in the library data file
Getfit	–	Performs curve fits on the possible solutions, creating curves that relate moisture concentration to anomaly size
Getiscts	–	Finds the intersections between the moisture/size curves for each different detector type
Getlib	–	Reads in a previously created library file
Getscan	–	Selects a scan file and reads in data
Gsolve	–	Finds all of the possible combinations of solution points, calculates an average for each combination, and calculates an error associated with the average
Interp	–	Interpolates the scans to create a common positional basis
Intersct	–	Determines the points of intersection between two curves
Iistscan	–	Lists data from one scan
Pick_d	–	Allows the user to identify the detector type for a data set
Scanhead	–	Lists the currently selected scan files
Scanmenu	–	Presents a menu of scan-related functions
Scantype	–	Changes the detector type associated with a scan file
TMAD	–	The main module, provides a menu driver for the other modules.

Figure 2.1. TMAD Flow Diagram.

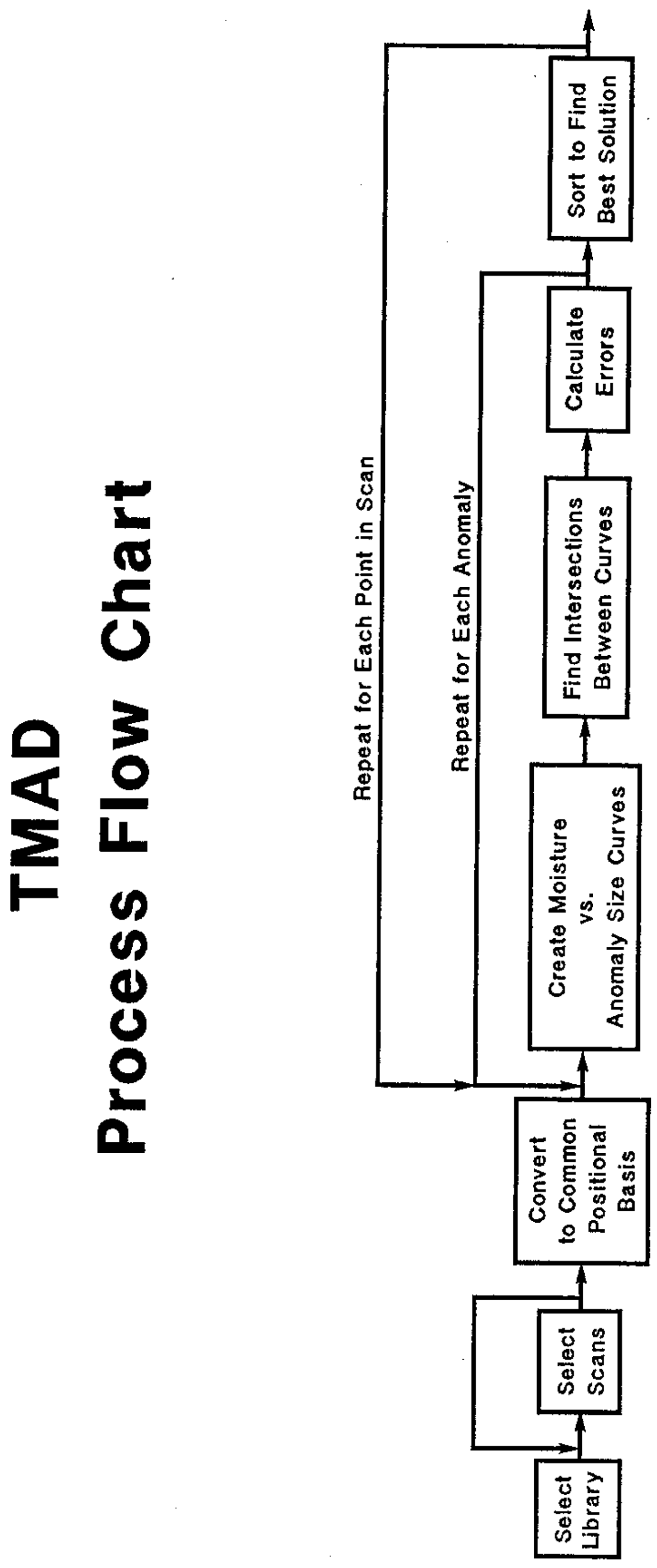
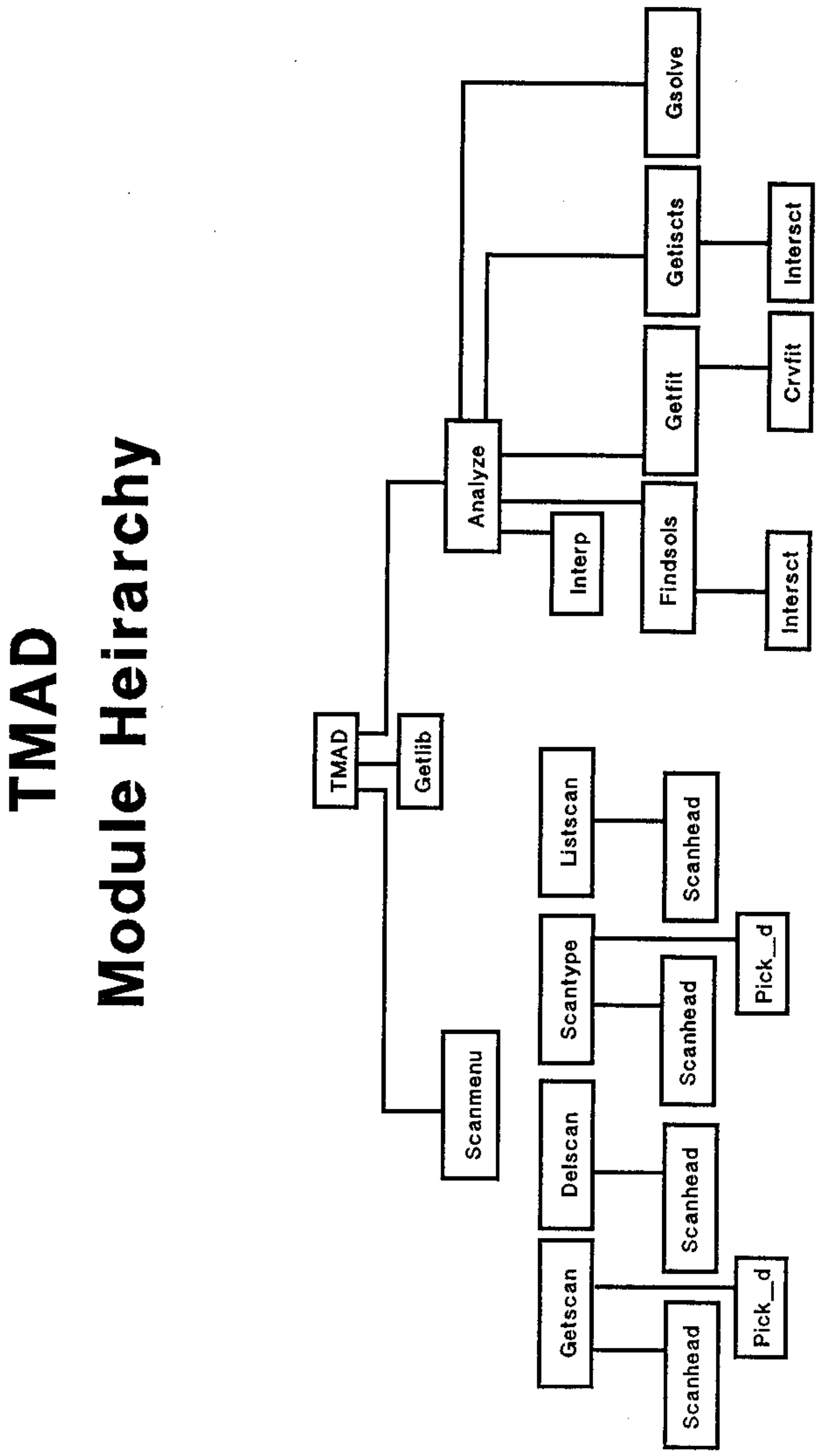

TMAD
Process Flow Chart
Select Library
Select Scans
Convert to Common Positional Basis
Create Moisture vs. Anomaly Size Curves
Find Intersections Between Curves
Calculate Errors
Sort to Find Best Solution
Repeat for Each Anomaly
Repeat for Each Point in Scan

Figure 2.2. TMAD Module Heirarchy.



TMAD also uses a set of external modules that were not written as part of this project. The external modules will be described briefly in Section 3.1.3.

2.1.2 LIBMAKR

The LIBMAKR code is used to create a library data file consisting of the benchmark data and curve fits associated with that data. Figure 2-3 shows a simple flow diagram for the LIBMAKR code, and Figure 2-4 shows the dependencies between the modules.

LIBMAKR modules:

Addlib	–	Reads in a set of benchmark data and adds it to the library file
Dellib	–	Deletes a specified set of benchmark data from the library file
Getlib	–	Reads in a previously created library file
Libmakr	–	The "main" routine, provides a menu driver for the other modules
Listlib	–	Lists the data in the current library file
Modalim	–	Allows the user to change the valid ranges of anomaly sizes
Modwgt	–	Allows the user to change the weighting factors for anomaly/detector pairs
Pick_a	–	Allows the user to identify the anomaly type for a data set
Pick_d	–	Allows the user to identify the detector type for a data set
Putlib	–	Writes out a library file.

LIBMAKR also uses a set of external modules that were not written as part of this project. The external modules will be described briefly in Section 3.1.3.

Figure 2.3. LIBMAKR Flow Diagram.

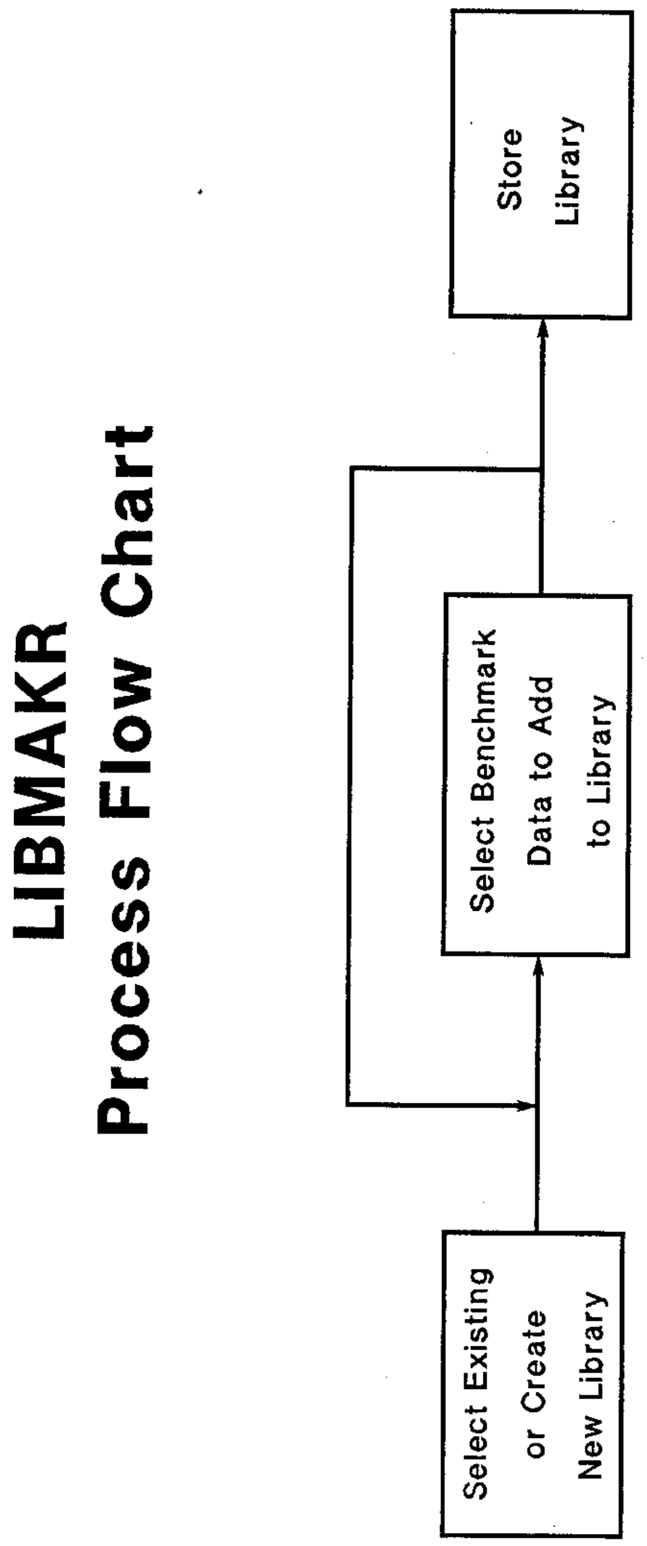
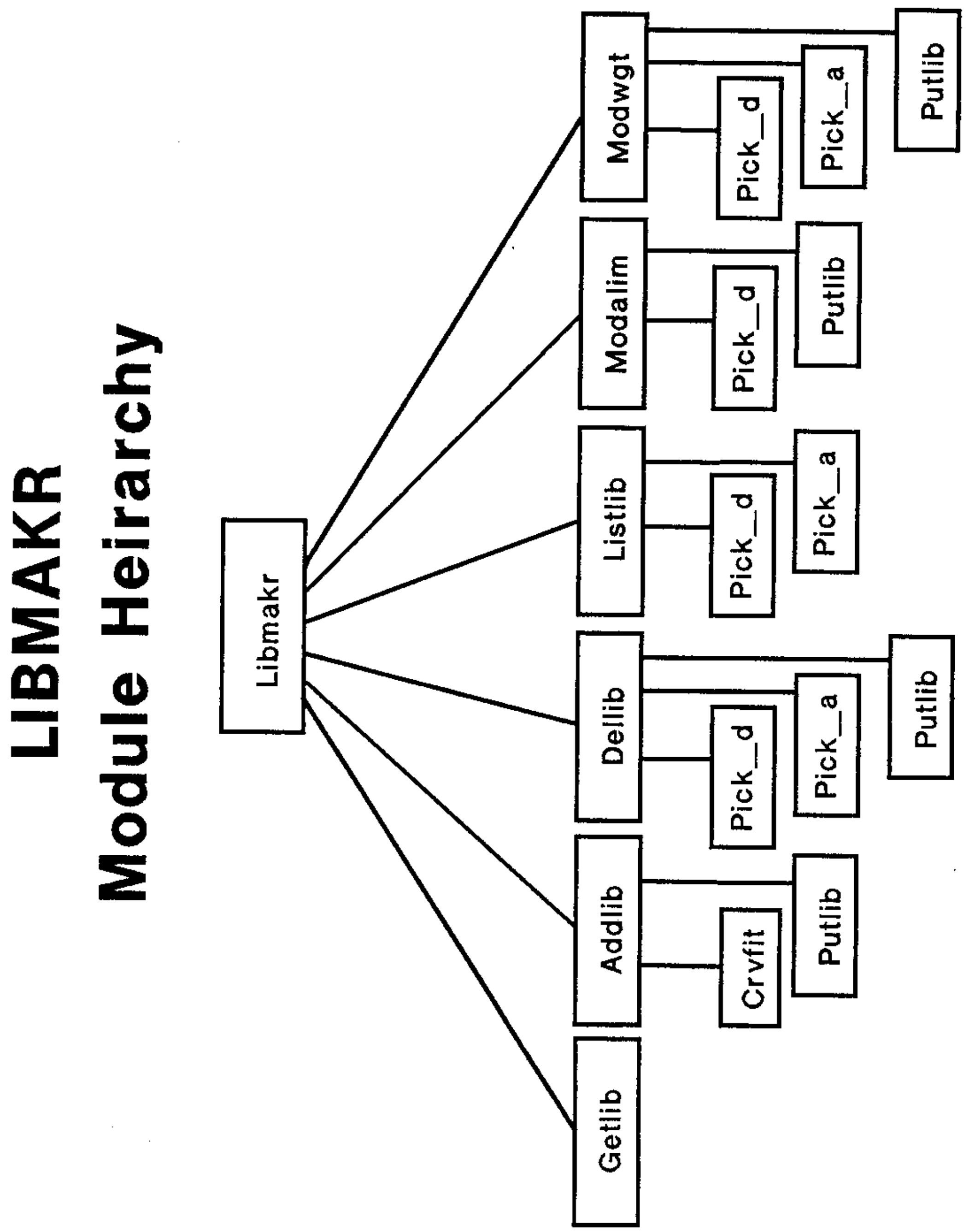

LIBMAKR
Process Flow Chart
Select Existing or Create New Library
Select Benchmark Data to Add to Library
Store Library

Figure 2.4. LIBMAKR Module Heirarchy.



2.1.3 External Routines

Both TMAD and LIBMAKR call several external routines that were not written as part of this project but rather were purchased along with the text *Numerical Recipes* (Press et al. 1986).

External routines:

Crvfit	–	Serves as an interface with the svdfit module
Gammln	–	Calculates the log of the gamma function
Gammq	–	Calculates the incomplete gamma function Q(a,x) = 1-P(a,x)
Gcf	–	Calculates the incomplete gamma function Q(a,x)
Gser	–	Calculates the incomplete gamma function P(a,x)
Polyfunc	–	Defines the form of the equation used in the curve fitting module
Svbksb	–	A backsubstitution routine solves A•X = B given a decomposed array A and a vector B
Svdcmp	–	Decomposes matrix A into $U•W•V^T$ where W is a diagonal matrix
Svdfit	–	Performs the curve fit.

2.2 CONCURRENT PROCESS DECOMPOSITION

There are no concurrent processes in the TMAD system.

2.3 DATA ENTITY DECOMPOSITION

There are four classes of data important to the TMAD system: data files, common blocks, constants, and variables. Each of these classes is described, and its members listed, in the following sections.

2.3.1 Data Files

Data files are primarily used in the TMAD system for the input and output of large blocks of data. These files are described below.

Library	–	Unit number = 10; file name varies. This file stores the curve fit coefficients produced by LIBMAKR and used as input for TMAD.
Benchmark	–	Unit number = 11; file name varies. This file is used as an input file to convey benchmark data to LIBMAKR.

Scan	-	Unit number = 11; file name varies. This file is used as an input file to convey measured scan data to TMAD.
TMAD debug	-	Unit number = 30; file name = tmad.debug. This file records the results of various intermediate calculations performed by TMAD.
TMAD out	-	Unit number = 40; file name = tmad.out. This file records the output (moisture content) from TMAD for each point in the scan data.

2.3.2 Common Blocks

Common blocks, in the TMAD system, are used primarily for providing access to global variables (such as input parameters) and for passing data arrays between modules. The contents of the various common blocks employed by the TMAD system are described below.

charlib	-	Library text data such as file name and detector types. Variables included in this common block are libname libtitle, title, nanomaly, anomaly, ndetector, and detector.
datlib	-	Benchmark data library. Variables included in this common block are alsize, acoefs, chisqr, goodfit, flmoist, dlvalue, dlsigma, alimits, decay, latype, natypes, ndtypes, ldtype, nlsizes, nlmoist, icheck, imonth, iday, and iyear.
fndsldat	-	Solution points for a given detector. Variables included in this common block are xsolpts, ysolpts, nsolpts, and nsolcrvs.
getftdat	-	Curve fit parameters for detector solution curves. Variables included in this common block are aszcoefs, crvxmin, crvxmax, crvymin, and crvymax.
getisdat	-	Intercept points between different detector solution curves. Variables included in this common block are xpairpts, ypairpts, npairpts, and ncpts.
gsoldat	-	Solution points for sets of detectors. Variables included in this common block are gxsol, gysol, gerr, and jsol.
scanchar	-	Scan data points. Variables included in this common block are scantitl and scanfile.
scandat	-	Scan text data (file name and scan title). Variables included in this common block are nscan, npoints, depth, svalue, rdepth, rvalue, srcmod, npoints2, isdtype, imon, idy, and iyr.
wgtlib	-	Detector weighting factors. Variables included in this common block are weights.

2.3.3 Constants

The TMAD system uses a variety of constants, which are defined using the FORTRAN "parameter" command. A brief description of the constants is given below (note that in the code, constants are identified by being presented in all capital letters).

EPSILON	–	A number that is considered sufficiently close to zero that all positive numbers less than it can be treated as zero
MANOM	–	The maximum number of anomalies that can be represented in the data library
MAXSOL	–	The maximum number of solution points that will be considered by the code
MDETECT	–	The maximum number of detector types that can be represented in the data library
MMAX	–	The maximum number of curve coefficients for the curve fit routines
MMOIST	–	The maximum number of moisture contents for a given anomaly size that can be present in the data library
MPOINTS	–	The maximum number of data points in a curve
MSCAN	–	The maximum number of scans that TMAD can process at one time
MSIZE	–	The maximum number of anomaly sizes for a given anomaly that can be present in the data library
NCURVCFS	–	The number of coefficients used to describe a curve (this defines the "order" of the curve)
NMAX	–	The maximum number of iterations for the curve fit routines
TOL	–	Convergence limit used in the curve coefficient routines.

2.3.4 Variables

A large number of variables are defined in the TMAD system. These variables can be broken into two general classes, local and global. Local variables are those that are used by only one module, and global variables are those that are used by two or more modules, either through common blocks or direct passing. A brief description of the contents of the global variables is given below.

acoefs	–	Array of curve coefficients for the benchmark data library curves representing detector response versus moisture content for a given detector-type/anomaly-type/anomaly-size combination. This variable is included in the datlib common block.
alimits	–	Array of minimum and maximum allowed anomaly sizes for each anomaly. This variable is included in the datlib common block.
alsize	–	Array of anomaly size entries in the benchmark data library for each anomaly-type in the library. This variable is included in the datlib common block.
anomaly	–	Array of names of anomaly types that are represented in the benchmark data library. This variable is included in the charlib common block.
aszcoefs	–	Array of curve coefficients representing the moisture-content/anomaly-size curves. This variable is included in the getftdat common block.
a2coefs	–	Array of curve coefficients, similar to acoefs, with the first value set equal to the detector response and the other values set to zero (thereby representing a horizontal line). This array is passed to the subroutine intersct.
chi	–	Calculated value of χ^2 (see chisqr) for a moisture-content/anomaly-size curve. This variable is passed to the subroutine crvfit.
chisqr	–	Array of calculated values of χ^2 for the curve fit coefficients in the benchmark data library. This variable is included in the datlib common block.
crvxmax	–	Array of maximum anomaly sizes for moisture content versus anomaly size curves. This variable is included in the getftdat common block.
crvxmin	–	Array of minimum anomaly sizes for moisture-content versus anomaly size curves. This variable is included in the getftdat common block.
crvymax	–	Array of maximum moisture contents for moisture content versus anomaly size curves. This variable is included in the getftdat common block.
crvymin	–	Array of minimum moisture-contents for moisture content versus anomaly size curves. This variable is included in the getftdat common block.
decay		Array of the decay constants for the probes associated with the different detector types. This variable is included in the datlib common block.

depth	–	Array of vertical positions for each data point in each scan. This variable is included in the scandat common block.
detector	–	Array of names of detector types represented in the benchmark data library. This variable is included in the charlib common block.
dlsigma	–	Array of standard deviations (σ^2) for the data points in the benchmark data library. This variable is included in the datlib common block.
dlvalue	–	Array of predicted detector responses in the benchmark data library. This variable is included in the datlib common block.
flmoist	–	Array of moisture content values represented in the benchmark data library. This variable is included in the datlib common block.
gerr	–	Array of calculated errors for the solution points identified by the code. This variable is included in the gsoldat common block.
goodfit	–	Measure of "goodness of fit" of a curve fit. This variable is included in the datlib common block.
gxsol	–	Array of anomaly size values for the solution points identified by the code. This variable is included in the gsoldat common block.
gysol	–	Array of moisture content values for the solution points identified by the code. This variable is included in the gsoldat common block.
icheck	–	Array of flags indicating the anomaly-type/detector-type combinations that are represented in the benchmark data library. This variable is included in the datlib common block.
iday	–	Day of the effective date of the benchmark data library. This variable is included in the datlib common block.
idy	–	Array of the days of the month on which the scans were performed. This variable is included in the scandat common block.
idump	–	Number of solutions identified for a given point in the scans in excess of the maximum (MAXSOL). This variable is passed to the subroutine gsolve.
imon	–	Array of the months in which the scans were performed. This variable is included in the scandat common block.

imonth –	Month of the effective date of the benchmark data library. This variable is included in the datlib common block.
isdtype –	Array of detector types for each scan. This variable is included in the scandat common block.
isolcnt –	Number of solutions identified for a given point in the scans. This variable is passed to the subroutine gsolve.
iyear –	Year of the effective date of the benchmark data library. This variable is included in the datlib common block.
iyr –	Array of the years in which the scans were performed. This variable is included in the scandat common block.
izflag –	Flag indicating the status of the curve intersection calculation. This variable is returned by the subroutine intersct.
jsol –	Array of anomaly types for the solution points identified by the code. This variable is included in the gsoldat common block.
latype –	Array of anomaly types represented in the benchmark data library. This variable is included in the datlib common block.
ldtype –	Array of detector types represented in the benchmark data library. This variable is included in the datlib common block.
libname –	Name of the benchmark data library file. This variable is included in the charlib common block.
libtitle –	Title of the benchmark data library. This variable is included in the charlib common block.
nanomaly –	Number of names of anomaly types included in the benchmark data library. This variable is included in the charlib common block.
natypes –	Number of anomaly types represented in the benchmark data library. This variable is included in the datlib common block.
ncpts –	Number of intersections between the curves representing the different detectors for a given anomaly. This variable is included in the getisdat common block.
ndetector –	Number of names of detector types included in the benchmark data library. This variable is included in the charlib common block.

ndtypes –	Number of detector types represented in the benchmark data library. This variable is included in the datlib common block.
nlmoist –	Array of number of moisture content values represented for each anomaly type and size in the benchmark data library. This variable is included in the datlib common block.
nlsizes –	Array of number of anomaly size values represented for each anomaly type in the benchmark data library. This variable is included in the datlib common block.
npairpts –	Array of number of intersections between every pair of moisture-content/anomaly-size curves. This variable is included in the getisdat common block.
npts –	Number of points to be used by the curve fit routine in creating a moisture-content/anomaly-size curve. This variable is passed to the subroutine crvfit.
npoints –	Array of number of points in each scan. This variable is included in the scandat common block.
npoints2 –	Number of points at which each scan will be evaluated. This variable is included in the scandat common block.
nscan –	Number of scans to be evaluated. This variable is included in the scandat common block.
nsolcrvs –	Array of number of solution curves for each anomaly-type/detector-type combination. This variable is included in the fndsldat common block.
nsolpts –	Array of number of solutions for each anomaly-size/anomaly-type/detector-type combination. This variable is included in the fndsldat common block.
ntmppts –	Number of intersection points between a particular pair of moisture-content/anomaly-size curves. This variable is returned by the subroutine intersct.
rdepth –	Array of vertical positions at which the scans will be processed. This variable is included in the scandat common block.
rvalue –	Array of calculated (by interpolation) detector responses, for each scan, at the points at which the scans will be evaluated. This variable is included in the scandat common block.
scanfile –	Array of the names of the files containing the scans to be evaluated. This variable is included in the scanchar common block.

scantitl –	Array of titles of the scans to be evaluated. This variable is included in the scanchar common block.
sigma –	Array of standard deviations for the points used to create the moisture-content/anomaly-size curves. This variable is passed to the subroutine crvfit.
srcmod –	Array of source strength modifiers (based on the decay constant) for each scan. This variable is included in the scandat common block.
svalue –	Array of detector response values at each point in each scan. This variable is included in the scandat common block.
title –	Array of titles of the anomaly-type/detector-type combinations represented in the benchmark data library. This variable is included in the charlib common block.
weights –	Array of weighting factors for the different detector types. This variable is included in the wgtlib common block.
xpairpts –	Array of anomaly size values for the intersection points between pairs of moisture-content/anomaly-size curves. This variable is included in the getisdat common block.
xpts –	Array of anomaly size values to be used by the curve fit routine in creating a moisture-content/anomaly-size curve. This array is passed to the subroutine crvfit.
xsolpts –	Array of moisture content values for solution points for anomaly-size/anomaly-type/detector-type combinations. This variable is included in the fndsldat common block.
ypairpts –	Array of moisture content values for the intersection points between pairs of moisture-content/anomaly-size curves. This variable is included in the getisdat common block.
xtmppts –	Array of anomaly size values for the intersection points between a particular pair of moisture-content/anomaly-size curves. This variable is returned by the subroutine intersct.
ypts –	Array of moisture content values to be used by the curve fit routine in creating a moisture-content/anomaly-size curve. This variable is passed to the subroutine crvfit.
ysolpts –	Array of detector response values for solution points for anomaly-size/anomaly-type/detector-type combinations (a dummy array whose contents are not actually used). This variable is included in the fndsldat common block.

ytmppts –	Array of moisture content values for the intersection points between a particular pair of moisture-content/anomaly-size curves. This variable is returned by the subroutine intersct.

3.0 INTERFACE DESCRIPTION

This section describes the internal and external interfaces in the TMAD system.

3.1 MODULE INTERFACE

This section describes the intermodule interfaces in the TMAD system. Each interface description defines the data entities that must be passed to that subroutine, either through the subroutine call or through common blocks. The LIBMAKR code, the TMAD code, and the curve fitting routines are treated separately.

3.1.1 TMAD

This section describes the interface to each of the subroutines within the TMAD code.

3.1.1.1 Analyze. No data needs to be passed to this subroutine through the call command. The common blocks it employs are datlib, wgtlib, charlib, scandat, scanchar, and gsoldat.

3.1.1.2 Delscan. No data needs to be passed to this subroutine through the call command. It requires two common blocks, scandat and scanchar.

3.1.1.3 Findsols. The interface to this subroutine consists of four integer variables, all of which are passed to the subroutine. The first variable, *m*, represents the current scan point being evaluated; *j* is the number of the current anomaly type being considered; *i* is the number of the current scan being evaluated; and *id* is the detector type associated with the current scan. The common blocks accessed by this subroutine are datlib, wgtlib, scandat, scanchar, and fndsldat.

3.1.1.4 Getfit. The interface to this subroutine consists of three variables, all of which are passed to the subroutine. The first variable, *j*, is the number of the current anomaly type being condsidered; *i* is the number of the current scan being evaluated; and *id* is the detector type associated with the current scan. The common blocks used in this subroutine are datlib, wgtlib, fndsldat, and getftdat.

3.1.1.5 Getiscts. This subroutine requires only a single integer variable, *j*, the number of the current anomaly type being considered, which is passed to it. The common blocks it requires are charlib, scandat, scanchar, fndsldat, getftdat, and getisdat.

3.1.1.6 Getlib. No data needs to be passed to this subroutine through the call command. It needs the following common blocks: datlib, wgtlib, and charlib.

3.1.1.7 Getscan. No data needs to be passed to this subroutine through the call command. The common blocks this subroutine requires are scandat and scanchar.

3.1.1.8 Gsolve. The interface to this subroutine consists of three integer variables, *j*, *isolcnt*, and *idump*. The first variable, *j*, is the number of the current anomaly type being considered. It is passed to the subroutine. *Isolcnt* and *idump* are global variables that are passed to gsolve, modified, and returned to the calling routine. This subroutine uses the following common blocks: datlib, wgtlib, scandat, scanchar, gsoldat, and getisdat.

3.1.1.9 Interp. No data needs to be passed to this subroutine through the call command. The common blocks it employs are datlib, wgtlib, charlib, scandat, and scanchar.

3.1.1.10 Intersct. The interface to this subroutine consists of the following data entities: *xsol*, *ysol*, *nsol*, *ac1*, *ac2*, and *izflag*. The first two entities are floating-point arrays (with dimension of 2) that are equivalent to the global arrays xtmppts and ytmppts. The next entity, *nsol*, is an integer variable equivalent to the global variable ntmppts. The entities *ac1* and *ac2* are floating-point arrays (dimensioned to the constant "NCURVCFS"), each of which contains the coefficients representing a curve. *Izflag* is an integer variable. The two curve coefficient arrays, *ac1* and *ac2*, are passed to the subroutine and the other three entities are returned to the calling routine. This subroutine does not access any common blocks.

3.1.1.11 Listscan. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, scandat, and scanchar.

3.1.1.12 Pick_d. The interface to this subroutine consists of two integer variables, *idpick* and *mode*. *Mode* is passed to the subroutine and *idpick* is returned. This subroutine requires only the common block charlib.

3.1.1.13 Scanhead. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, scandat, and scanchar.

3.1.1.14 Scanmenu. No data needs to be passed to this subroutine through the call command. This subroutine does not use any common blocks.

3.1.1.15 Scantype. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, scandat, and scanchar.

3.1.2 LIBMAKR

This section describes the interface to each of the subroutines within the LIBMAKR code.

3.1.2.1 Addlib. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, datlib, and wgtlib.

3.1.2.2 Dellib. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, datlib, and wgtlib.

3.1.2.3 Getlib. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, datlib, and wgtlib.

3.1.2.4 Listlib. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, datlib, and wgtlib.

3.1.2.5 Modalim. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, datlib, and wgtlib.

3.1.2.6 Modwgt. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, datlib, and wgtlib.

3.1.2.7 Pick_a. The interface to this subroutine consists of two integer variables, *iapick* and *mode*. *Mode* is passed to the subroutine and *iapick* is returned. This subroutine requires only the common block charlib.

3.1.2.8 Pick_d. The interface to this subroutine consists of two integer variables, *idpick* and *mode*. *Mode* is passed to the subroutine and *idpick* is returned. This subroutine requires only the common block charlib.

3.1.2.9 Putlib. No data needs to be passed to this subroutine through the call command. The common blocks it employs are charlib, datlib, and wgtlib.

3.1.3 CURVE FIT ROUTINES

This section describes the interface to each of the curve fit routines.

3.1.3.1 Crvfit. This subroutine requires seven data entities be passed to it: *xpts*, *ypts*, *sigma*, *npts*, *ntmp*, *acoefs*, and *chi*. The first three are floating-point arrays (dimensioned to *npts*) that contain the *x* values, the *y* values, and the standard deviations, respectively, of each point in the curve fit. The next two variables, *npts* and *ntmp*, are integers representing the number of points to be used in the curve fit and the number of curve coefficients to be generated, respectively. *Acoefs* is a floating-point array (dimensioned to *ntmp*) that contains the curve coefficients generated by the curve fit. *Chi* is a floating-point variable that represents the χ^2 valued resulting from the curve fit. The first five items are all passed to the subroutine and the last two are returned to the calling routine. There are no common blocks accessed by this subroutine.

3.1.3.2 Gammln. This function is passed only one value, *xx*, a double precision, floating-point variable. The return value is also a double precision, floating-point value. There are no common blocks accessed by this subroutine.

3.1.3.3 Gammq. This function is passed two values, *a0* and *x0*, both of which are floating-point variables. The return value is also a floating-point value. There are no common blocks accessed by this subroutine.

3.1.3.4 Gcf. This subroutine is passed four data entities: *gammcf*, *a*, *x*, and *gln*. All four are double precision, floating-point variables. The second and third are passed to the subroutine, and the first and last are returned to the calling routine. There are no common blocks accessed by this subroutine.

3.1.3.5 Gser. This subroutine is passed four data entities: *gamser*, *a*, *x*, and *gln*. All four are double precision, floating-point variables. The second and third are passed to the subroutine, and the first and last are returned to the calling routine. There are no common blocks accessed by this subroutine.

3.1.3.6 Polyfunc. This subroutine requires that three data entities be passed to it. The first, *xval*, is a floating-point variable which is passed to the subroutine. The next, *pdat*, is a floating-point array (dimensioned to *np*) that is returned by the subroutine to the calling routine. The final item, *np*, is an integer variable that is passed to the subroutine. There are no common blocks accessed by this subroutine.

3.1.3.7 Svbksb. This subroutine is passed nine data entities: *uarray*, *warray*, *varray*, *ndata*, *maxa*, *mp*, *np*, *barray*, and *acoefs*. The first three are floating-point arrays (dimensioned to *mp* by *np*, *np*, and *np* by *np* respectively). The next four are all integer variables. The eighth item, *barray*, is a floating-point array (dimensioned to *mp*). The last item, *acoefs*, is also a floating-point array (dimensioned to *maxa*). The first eight items are all passed to the subroutine and the last (*acoefs*) is returned to the calling routine. There are no common blocks accessed by this subroutine.

3.1.3.8 Svdcmp. This subroutine is passed seven data entities: *acoefs*, *ndata*, *maxa*, *mp*, *np*, *warray*, and *varray*. The first entity, *acoefs*, is a floating-point array (dimensioned to *maxa*) that is passed to the subroutine, modified, and returned to the calling routine. The next four items are all integer variables that are passed to the subroutine. The last two, *warray* and *varray*, are both floating-point arrays (both dimensioned to *np* and *np* by *np*, respectively) that are returned to the calling routine. There are no common blocks accessed by this subroutine.

3.1.3.9 Svdfit. This subroutine requires twelve data entities in the interface: *xdat*, *ydat*, *sig*, *ndata*, *acoefs*, *maxa*, *uarray*, *varray*, *warray*, *mp*, *np*, and *chisq*. The first three are floating-point arrays (dimensioned to *ndata*) that are passed to the subroutine. The next, *ndata*, is an integer variable that is passed to the subroutine. The fifth item, *acoefs*, is a floating-point array (dimensioned to *maxa*) that is returned to the calling routine. The sixth item, *maxa*, is an integer variable that is passed to the subroutine. The next three items (*uarray*, *varray*, and *warray*) are all floating-point arrays (dimensioned to *mp* by *np*, *np* by *np*, and *np*, respectively) that are returned to the calling routine. The tenth and eleventh items, *mp* and *np*, are integer variables that are passed to the subroutine. The final entity, *chisq*, is a floating-point variable that is returned to the calling routine. There are no common blocks accessed by this subroutine.

3.2 PROCESS INTERFACE

The TMAD system consists of two independent processes, the TMAD code and the LIBMAKR code. Both of these processes have interfaces consisting of input files, output files, and a menu-driven user interface. Each of these are described in detail in the following sections.

3.2.1 TMAD Interfaces

This section describes the interfaces associated with the TMAD code.

3.2.1.1 User Interface. The TMAD code provides a menu-driven user interface to provide flexibility and ease of use.

Immediately upon initiating the TMAD code, the user is prompted to enter the name of the benchmark data library file. Once this has been entered, the code checks to see whether the file exists. If the file does not exist, the user is given the option of aborting or entering a new filename. This process is repeated until the user either enters a valid filename or opts to abort.

Once the library file has been identified, the user is presented with the main menu, which includes the following options:

- Done
- Select scan data to analyze
- Analyze scan data.

The first option, "done," exits the program. The second option, "select," takes the user to the input scan menu (described below), and the third option, "snalyze," initiates processing of the scan data. If either the second or third options are selected, the user will be returned to the main menu after that option has been executed. This process will continue until the user selects the "done" option.

The input scan menu provides the user with the following options:

- Return to previous menu
- Select a scan
- Unselect a scan
- Change scan detector type
- List selected scan data.

The first option, "return," returns the user to the TMAD main menu. If the user selects any of the other four options, the user will return to the input scan menu after the option has been executed. This process will continue until the user selects the "return" option.

The second option, "select," allows the user to select a scan data file for inclusion in the evaluation. After the user selects this option, the code will present a list of previously selected scans (if any) and will then prompt the user for a scan data filename. The code will check to see whether this file exists, and if it doesn't, the user will be returned to the TMAD main menu. If the file does exist, the user will be prompted for the detector type associated with the file. Typically the "select" option will be exercised several times.

The "unselect" option is provided in the event that the user decides to replace one scan data set with another in the course of an evaluation. When this option is selected, the user is presented with a list of previously selected scans and is prompted to select one. The designated scan is then removed from the list of scans to be evaluated.

The "change detector type" option allows the user to change the detector type that had been previously specified for a scan. When this option is selected, the user is presented with a list of possible detector types and prompted to select one. After a detector type has been selected, the code checks to be sure that no scan has already been specified as this type. If the choice is valid, the change is made, otherwise an error message is generated.

The final option, "list," allows the user to list the data in one or more of the selected scans.

3.2.1.2 Benchmark Data Library File. The benchmark data library file is produced by the LIBMAKR code and serves as an interface between the two processes. The file contains all of the benchmark data necessary for the interpolation process performed by TMAD. The details of the file's contents are presented below.

File entries:

```
Line 1 (format=a): libtitle
Line 2 (unformatted): imonth, iday, iyear
Line 3 (format=2i4): ndetector, nanomaly
Line 4 (format=a): detector(il)
     Line 4 is repeated ndetector times
Line 5 (format=a): anomaly(jl)
     Line 5 is repeated nanomaly times
Line 6 (format=2e15.6): decay(il)
     Line 6 is repeated ndetector times
Line 7 (format=2e15.6): alimits(1,jl),alimits(2,jl)
     Line 7 is repeated nanomaly times
Line 8 (format=i4): ndtypes
Line 9 (format=2i4): ld,natypes(i)
Line 10 (format=i4): la
Line 11 (format=a): title(la,ld)
Line 12 (format=i4): nlsizes(la,ld)
Line 13 (format=e15.6): weights(la,ld)
Line 14 (format=e15.6,i4): alsize(k,la,ld), nlmoist(k,la,ld)
Line 15 (format=10e15.6): acoefs(m,k,la,ld), m = 1,NCURVCFS
Line 16 (format=2e15.6): chisqr(k,la,ld), goodfit(k,la,ld)
Line 17 (format=3e15.6): flmoist(l,k,la,ld), dlvalue(l,k,la,ld),
     dlsigma(l,k,la,ld)
     Line 17 is repeated nlmoist(k,la,ld) times
     Lines 14-17 are repeated nlsizes(la,ld) times
Lines 10-17 are repeated natypes(i) times
Lines 9-17 are repeated ndtypes times.
```

Variable descriptions:

acoefs(m,k,la,ld)	–	Curve coefficients; type = floating point
alimits(1,jl)	–	Minimum anomaly size; type = floating point
alimits(2,jl)	–	Maximum anomaly size; type = floating point
alsize(k,la,ld)	–	Anomaly size; type = floating point
anomaly(jl)	–	Anomaly type name; type = character*32
chisqr(k,la,ld)	–	Chi squared value; type = floating point
decay(il)	–	Decay constant (1/days); type = floating point
detector(il)	–	Detector type name; type = character*32
dlsigma(l,k,la,ld)	–	Standard deviation; type = floating point
dlvalue(l,k,la,ld)	–	Detector response; type = floating point
flmoist(l,k,la,ld)	–	Moisture concentration; type = floating point
goodfit(k,la,ld)	–	Goodness of fit; type = floating point
i	–	Detector type counter, increments from 1 to ndtypes; type = integer
il	–	Detector type name counter, increments from 1 to ndetector; type = integer
iday	–	Day of month corresponding to benchmark model; type = integer
imonth	–	Month of year corresponding to benchmark model; type = integer
iyear	–	Year corresponding to benchmark model; type = integer
j	–	Anomaly type counter, increments from 1 to natypes(i); type = integer
jl	–	Anomaly type name counter, increments from 1 to nanomaly; type = integer
k	–	Anomaly size counter, increments from 1 to nlsizes(la,ld); type = integer

l	– Moisture concentration counter, increments from 1 to nlmoist(k,la,ld); type = integer
la	– Anomaly counter, increments from 1 to natypes(i); type = integer
ld	– Detector counter, increments from 1 to ndtypes (equivalent to i); type = integer
libtitle	– Title of library; type = character*60
nanomaly	– Number of anomaly type names; type = integer
natypes(i)	– Number of anomaly types in library; type = integer
ndetector	– Number of detector type names; type = integer
ndtypes	– Number of detector types in library; type = integer.
nlmoist(k,la,ld)	– Number of moisture concentrations; type = integer.
nlsizes(la,ld)	– Number of anomaly sizes; type = integer
title(la,ld)	– Title of detector-type/anomaly-type model; type = character*60
weights(la,ld)	– Weight factors; type = floating point.

3.2.1.3 Scan Data File. The scan data file contains the measured detector responsed for a particular scan. A typical application of TMAD will actually access two or more different scan data files, but each will have an identical format, as described below.

File entries:

```
Line 1 (format=a): scantitle
Line 2 (unformatted): imon, idy, iyr
Line 3 (unformatted): npoints(nscan)
Line 4 (unformatted): depth(i,nscan),svalue(i,nscan)
     Line 4 is repeated npoints(nscan) times
```

Variable descriptions:

depth(i,nscan)	– Depth (in meters) at which detector response was measured; type = floating point
i	– Data point counter, increments from 1 to npoints(nscan); type = integer

idy	–	Day of month corresponding to the scan; type = integer.
imon	–	Month of year corresponding to the scan; type = integer.
iyr	–	Year corresponding to the scan; type = integer
npoints(nscan)	–	Number of points in the scan; type = integer
nscan	–	Scan counter, indicates the number of the current scan; type = integer
scantitle	–	Title of scan data file; type = character*72
svalue(i,nscan)	–	Detector response; type = floating point.

3.2.1.4 TMAD.OUT Output File. The file tmad.out contains the results of the TMAD evaluation. The details of the file are described below.

The first line in the tmad.out file is a header line that identifies the version of TMAD that produced the file. The next line describes the columns that are to follow. Specifically, the column headings are: "point," "depth m," "depth ft," "anomaly," "size," "%H2O," "sigma," and "confidence." Following the header line is one line for each point that was evaluated in the scan. The entries in those lines are described below.

Point	–	This is a counter that increments from 1 to the total number of points.
Depth-m	–	The depth (in meters) to which the given data point corresponds.
Depth-ft	–	The depth (in feet) to which the given data point corresponds.
Anomaly	–	The name of the anomaly type predicted for this point.
Size	–	The anomaly size predicted for this point.
%H2O	–	The moisture concentration (weight percent) predicted for this point.
Sigma	–	The standard deviation associated with the anomaly size and moisture content predictions for this point.
Confidence	–	A calculated confidence factor associated with the anomaly size and moisture content predicted for this point.

The format for the data lines in this file is

i5, f8.3, 1x, f8.3, 1x, a13, f5.1, 5x, f5.1, 5x, g9.3, 1x, g9.3.

3.2.1.5 TMAD.DEBUG Output File. The file tmad.debug contains a report of intermediate calculations completed in the course of the evaluation. It is useful for verification purposes and, on occasion, to provide input to perform (by hand) a more detailed assessment of the confidence factor. The file is a mixture of alpha and numeric outputs and is not intended to be used as input for a code. As such the format of the file will not be discussed in detail, but the contents will be summarized below.

The first section of the file summarizes the interpolation of the various scans to a common positional basis. The minimum and maximum depths of the new basis are given, as is the number of points to be used in the new basis. The depth at each point in the new basis is then listed, as is the corresponding (interpolated) detector response for each scan.

The next section is a summary of the scan and benchmark data. This section includes information such as the number of scans and anomaly types. For each scan the associated detector type is identified and the number of anomalies for that detector type that are included in the library. For each anomaly type, the anomaly type name and the number of anomaly sizes in the library is given. The file also includes a list of all of the anomalies that are common in the library for the detector types represented by the selected scans.

Following this, the file includes a summary of the calculations for each point in the new basis. The first item reported is the number of the point and the depth (both in meters and in feet). For each point in the scan, the code loops over the possible anomaly types. The current anomaly type is written to the file. For each anomaly type, the code loops over the selected scans. For each scan, the associated detector type and the measured detector response are written to the file. For each scan type, the code then loops over the available anomaly sizes. Each anomaly size is written to the file. Any solution points found for the given anomaly size are then recorded. Once all of the anomaly sizes have been tested, the code performs a curve fit on the identified solution points, and the results, in the form of curve coefficients, are written to the file. This process is repeated until all of the anomaly types have been evaluated.

The code then loops over the anomaly types again, searching for intersections between the curves representing the different scans. At this stage, the anomaly type is again written to the file. Each pair of curves compared are identified in the file along with any intersections found. The code then combines the intersection points into all possible clusters, such that each scan is only represented once per cluster, and calculates an average value and associated error. The intersection points, the average, and the error are reported for each cluster. This process is then repeated until all of the anomaly types have been evaluated.

In the final stage, the code takes all of the possible solution points (the cluster average values) and compares them based on the calculated error. Every identified solution point is then written to the file in order of increasing error. The whole process is then repeated for the next point in the scan.

3.2.2 LIBMAKR INTERFACES

This section describes the interfaces associated with the LIBMAKR code.

3.2.2.1 User Interface. The LIBMAKR code provides a menu-driven user interface to facilitate performing a number of different library building functions with the code.

Immediately upon initiating the TMAD code, the user is prompted to enter the name of the benchmark data library file. Once this has been entered, the code checks to see whether the file exists. If the file does not exist, the user is given the option of aborting, creating a new library, or entering a new filename. This process is repeated until the user enters a valid filename or opts to either abort or create a new library. If the user elects to create a new library, the code prompts for the effective date of the source code in the library.

Once the library file has been identified, the user is presented with the main menu, which includes the following options:

- Done
- Add data to library
- Delete data from library
- Modify weights
- List data in library
- Modify anomaly limits.

The first option, "done," exits the program. If the user selects any of the other options then, after the option is executed, the user is returned to the main menu. This process is repeated until the user selects the "Done" option.

The next option, "add," allows the user to identify a file containing benchmark data to be added to the library. The format of this file will be described in the next section. The user is first prompted for a filename. The code checks to see whether the file exists, and if it does not, the user is returned to the LIBMAKR main menu. Once a benchmark data file has been read, the user is prompted for a detector type and anomaly type to associate with the new data set. In both cases the user may select from those types already existing in the library, or the user may identify a new type. If a new anomaly type is specified, the user is prompted for minimum and maximum sizes for the anomaly.

The third option, "delete," allows the user to delete a data set from the library. If this option is selected then the user is presented with a list of all of the detector and anomaly types represented in the library. The user may select a specific detector/anomaly pair or all data associated with one detector type or one anomaly type. All selected data are then removed from the library.

The next option, "modify weights," allows the user to change the weighting factor associated with a particular detector and/or anomaly type (the default value is 1.0). The user is first prompted to enter a new weight value, and then to select the data sets to apply it to. The selection process here is the same as for the "delete" option.

The fifth option, "list," allows the user to list the data associated with a particular detector and/or anomaly type. The selection process here is the same as for the "delete" option.

The last option, "modify limits," allows the user to change the minimum and/or maximum sizes specified for a particular anomaly. After selecting an anomaly type to modify, the user is presented with the current minimum and maximum size limits and is prompted for new values.

3.2.2.2 Benchmark Data File. This file contains the benchmark data associated with a particular detector-type/anomaly-type combination. In most applications of LIBMAKR there will be many such files but all of them will have the same format, which is discussed below.

File entries:

```
Line 1 (format=a): tmptitle
Line 2 (unformatted): nsizes
Line 3 (unformatted): asize(i), nmoist(i)
Line 4 (unformatted): fmoist(j,i), dvalue(j,i), sigma(j,i)
     Line 4 is repeated nmoist(i) times.
     Lines 3 through 4 are repeated nsizes times.
```

Variable descriptions:

asize(i)	–	Anomaly size; type = integer
dvalue(j,i)	–	Predicted detector response; type = floating point
fmoist(j,i)	–	Moisture concentration (weight percent); type = floating point
i	–	Anomaly size counter, increments from 1 to nsizes; type = integer
j	–	Moisture content counter, increments from 1 to nmoist(i); type = integer
nmoist(i)	–	Number of moisture content values associated with anomaly size "i;" type = integer
nsizes	–	Number of anomaly sizes represented in data set; type = integer
sigma(j,i)	–	Standard deviation associated with predicted detector response; type = floating point
tmptitle	–	Data set title; type = character*60.

3.2.2.3 Benchmark Data Library File. The benchmark data library file is produced by the LIBMAKR code and serves as an interface between the two processes. The file contains all of the benchmark data necessary for the interpolation process performed by TMAD. The details of the file's contents are presented in Section 3.2.1.2 above.

4.0 DETAILED DESIGN

This section provides the internal details of each design entity.

4.1 MODULE DETAIL DESIGN

This section provides the internal details of each module in the TMAD system. The modules are broken up into two groups; those associated with the TMAD code and those associated with the LIBMAKR code.

4.1.1 TMAD Module Design

The following sections detail the design of the TMAD modules.

4.1.1.1 Analyze. The "analyze" module serves as the driver for the routines that find the possible solutions in the benchmark library that would produce the measured detector response. The first step in this process is to call the "interp" routine, which interpolates the various scans so that they are all on a common positional basis. Next the routine creates a list of all the anomalies that are included in the data library for all of the specified detector types.

After the initialization, the routine loops through each point in the common positional basis and finds the best solution at each point. The solutions are first found for each anomaly type from those listed. Solutions are found by developing curves that relate the moisture concentration to the anomaly size for each detector type. The intersections between these curves represent a possible solution. The "analyze" module calls the "findsols" routine to find all of the moisture concentrations that correspond to the measured detector response for each of the anomaly sizes in the library. The routine "getfit" is then called to perform a curve fit over these points and produce coefficients that can be used to represent the curve. Next, the routine "getiscts" finds the intersections between these curves and calculates the resulting solution point and associated error.

Once all of the possible solution points have been determined, the code sorts them to find the best (i.e., lowest error) solution. A confidence factor, which relates the best solution to the second best, is calculated. In the event that no solution is found, then the predicted anomaly size, predicted moisture concentration, and confidence factor are all set to -1, the error is set to 100.0, and the anomaly type description is set to "no solution". After the confidence factor is determined then the best solution, for that point in the scan, is printed and control returned to the calling routine.

4.1.1.2 Delscan. The "delscan" module is used to remove a particular scan from the list of scans to be included in the analysis.

The module first calls the subroutine "scanhead," which prints a menu of the scans that have been selected. The user is then prompted to select one of the scans for deletion. Once a scan has been selected, the code deletes it from the appropriate data arrays and returns control to the calling routine.

4.1.1.3 Findsols. This module takes as input the measured detector response for a particular detector type. For each detector type, an intersection calculation is performed with the detector response curve fit data in the data library. The result of this operation is an array of X,Y points where X corresponds to the anomaly size and Y corresponds to the moisture concentration.

For each anomaly size in the data library (for the given detector type), the subroutine "intersct" is called. This routine returns the number of intersections (0, 1, or 2), the corresponding moisture concentration, and, if there are no intersections, a warning flag. If there are no intersections, the module also prints out an appropriate warning message.

The module then checks for situations in which most of the anomaly sizes have two intersections but one size has only one solution. This occurs when one of the detector responses is just equal to the maximum (or minimum) of the curve. In this case, the singular point is repeated so that the particular anomaly size is treated as having two identical solutions. This is necessary for the curve fitting procedure in the "getfit" module.

Finally, control is returned to the calling routine.

4.1.1.4 Getfit. The "getfit" module performs a curve fit on the sets of possible solutions for a given anomaly-type/detector-type pair. This is done after first filtering the sets for points that are outside of a prespecified valid range. Sets that do not have at least two consecutive valid points are discarded (because it is impossible to establish a curve with just one point).

The routine first examines each point in a particular solution curve; (the location of each point corresponds to an anomaly size and a solution curve is a family of related solution points for a given detector-type/anomaly-type combination). Each point falls into one of three categories: no solution (i.e., no moisture content at this point agrees with the measured data), a valid solution, or an invalid solution (a solution can be found only by extrapolating to a moisture content that is outside the range of the benchmark data for this detector/anomaly combination). Any invalid point that is immediately adjacent to a valid point is flagged as valid (to avoid discarding important end points).

Next, the series of points is searched to find the longest consecutive string of points that does not include a "no solution" point. If this string includes less than two points, then the entire curve is discarded. Assuming that there are more than two points, the string is examined and any points that had been flagged as invalid are then flagged valid and the series of points becomes the basis for the curve fit.

At this point the code determines the legitimate range for the anomaly size and the moisture content. The legitimate range for the anomaly size is determined by finding the minimum range covered by the series of valid points and then extending it 30% up and down. The extension allows for errors in the measured or modeled data that might cause the predicted results to fall outside the limits. This process is repeated for the moisture content except that the range is extended by only 10%. Appropriate values for the size of the extension were determined empirically and may be dependant on the data set.

Next, the subroutine "crvfit" is called to produce a curve fit for the series of points. The function "gammq" produces an estimate of the goodness of fit of the resultant curve. Finally, control is returned to the calling routine.

4.1.1.5 Getiscts. The "getiscts" module searches through all of the moisture/size curves for a given anomaly type and finds all of the valid intersections between curves representing different scans.

The module first creates and initializes an array containing the number of intersections between curves representing every pair of scans (i.e., the number of intersections between scan 1 curves and scan 2 curves, the number of intersections between scan 1 curves and scan 3 curves, and so on).

Next, the code loops over every scan, and every curve of every scan, and identifies each pair of curves that represent two different scans. Those two curves are then passed to the subroutine "intersct," which finds the actual intersection coordinates. Once an intersection has been found, it is compared to minimum and maximum values for the moisture content and anomaly size. If the point is within the specified range, it is stored. This process is repeated until all valid intersection points have been found and identified, and control is then returned to the calling routine.

4.1.1.6 Getlib. This module is responsible for reading in a library file so that it can be modified. It starts out by zeroing the array that is used to flag the anomaly/detector pairs that are represented in the library. The user is prompted for the name of the library file that is to be read in, and if the specified file does not exist, the user is given the option of exiting the module or entering a new file name.

Once a valid file is identified, the file is opened and the following data read in from the data library file (described in Section 3.2.1.2).

- The data library file title
- The effective date of the benchmark data
- The number of possible detector and anomaly types
- The names of the possible detector types
- The names of the possible anomaly types
- The decay constants for each detector type
- The minimum and maximum anomaly sizes for each anomaly type
- The number of detector types represented in the library.

For each detector, the code reads the detector number and the number of anomalies for that detector. Next, the following data are listed for each detector/anomaly pair selected.

- The anomaly number
- The title line from the benchmark data file from which the data came
- The number of anomaly sizes in the library
- The importance weight associated with this detector/anomaly pair.

Each detector/anomaly pair will have one or more sets of data, each corresponding to a different anomaly size. For each anomaly size, the following data will be read:

- The anomaly size
- The number of moisture concentrations in the data set
- The curve fit coefficients for this data set.
- The standard deviation and goodness of fit values for the curve fit.

The curve corresponding to a given detector/anomaly/anomaly-size combination is constructed (in the module "addlib") from a set of data points relating the detector response to the moisture concentration at a number of different moisture concentrations. For each moisture concentration, the following data are read:

- The moisture concentration
- The detector response
- The standard deviation in the detector response value.

Finally, the library data file is closed, and control is returned to the calling routine.

4.1.1.7 Getscan. The "getscan" module is responsible for reading in scan data files that are to be processed by TMAD.

The routine first checks to be sure that TMAD does not already have the maximum number of scans it can handle. If it does, an error message is printed and control is returned to the calling routine.

If TMAD can accept another scan, the routine calls the subroutine "scanhead," which prints a list of the titles of the scans that have already been read in. The user is then prompted to enter the name of the file containing the scan data. The subroutine checks to see whether this file exists. If it does not, an error message is printed and control is returned to the calling routine. If the filename is valid, the file title is read in and printed for the user, and the user is prompted to specify the appropriate detector type and anomaly type (through the subroutines "pick_d" and "pick_a," respectively) for this scan.

Next, the date that the scan was performed is read in. After the date is read in, the routine reads in the scan data. These data are then sorted, if necessary, to be sure that it is in order of increasing depth. At this point the code also checks to see whether there has already been a scan entered for this detector-type/anomaly-type combination. If so, an error message is printed and this scan is deleted from memory.

Finally, control is returned to calling routine.

4.1.1.8 Gsolve. The "gsolve" module takes all of the intersections between pairs of scan curves and forms clusters consisting of one point for every pair of scans. These clusters are averaged, and the result is one possible solution for the given point in the scan. This is done for every possible cluster.

The routine first initializes an array consisting of counters for every possible combination of two scans. This array is used identify the solution point from each scan pair that is going to be used for a given cluster. The code steps through every possible combination of clusters and performs a weighted average on each. This average is then adjusted, if necessary, to ensure that it falls within the allowed ranges for moisture content and anomaly size. The result is one possible solution for the given point in the scan. The standard deviation associated with the solution is calculated and the solution is stored in an array. This process is repeated for every possible cluster of points, and then control is returned to the calling routine.

4.1.1.9 Interp. This module establishes a common positional basis for all of the scans. It interpolates on the detector responses for all of the scans to provide estimated detector responses at the same positions for all of the scans.

The routine first finds the minimum and maximum depths that are common to all of the scans. These points become the boundaries for the new positional basis. Each of the scans is then searched to determine which has the most points falling between the minimum and maximum depth. These points become the remainder of the positional basis. Each of the scans is then interpolated to estimate the detector response at the each point in the new positional basis.

Next, the scan values are adjusted to account for any source decay in the probe. Typically, neutron probes use a radioactive source that decays over time. As a result, the strength of the source (and consequently the magnitude of the response) will be different from what was modeled in the benchmark data library. To avoid this problem, the measured response is adjusted to the appropriate value based on the difference between the source strength at the time of the scan and that modeled in the library. This is done by modifying every detector response by a modifier that is calculated as follows:

$$S=\lambda(t_s-t_l)$$

Where:

S	source modifier for the given detector
λ	decay constant (in inverse days) for the given detector
t_s	scan date (in days)
t_l	library date (in days)

Note that the date values are calculated using the following equation:

$t = y*365.25 + m*30.44 + d$

where:

t	relative date
y	year (e.g., 1995)
m	month (e.g., 12)
d	day (e.g., 31)

Finally, control is returned to the calling routine.

4.1.1.10 Intersct. The module finds the intersections, if any, between two second-order curves as defined by two sets of curve coefficients. The intersections are found by solving the following equation:

$$C0 + C1*X + C2*X^2 = D0 + D1*X + D2*X^2$$

where C0 is the 0th order coefficient for the first curve, C1 is the 1st order coefficient, C2 is the 2nd order coefficient, D0, D1, and D2 are corresponding coefficients for the second equation, and X is the x coordinate of the intersection point.

First, the code calculates the difference between the 0th, 1st, and 2nd order coefficients. If the 2nd order coefficients are different, the above equation can be solved as a quadratic equation. In this case there are two solutions (which may be different, identical, or imaginary). If the 2nd order coefficients are the same, there is only one solution (at most), and the equation can be reduced to

$$X = -(C0 - D0)/(C1 - D1)$$

If the 1st order coefficients are identical, there is no unique solution. In this case, if the 0th order coefficients are identical, the lines are identical and there is an infinite number of intersections. Otherwise, if the 0th order coefficients are different, the curves are essentially parallel and there are no intersections between them.

In any of the above situations in which there is no real solution, a flag is set with a value corresponding to the reason no intersection was found (imaginary, identical curves, or non-intersecting curves).

Once the x coordinates of the intersection points are found, the y values are calculated using the first curve coefficients, and control is returned to the calling routine.

4.1.1.11 Listscan. The "listscan" module prints out the data in one of the scans that has been entered into TMAD. The module first lists the titles of each of the scans that have been entered into the system. The user is then prompted to select one of the scans and the data from this scan is then listed to the screen. Control is then returned to the calling routine.

4.1.1.12 Pick_d. This module allows the user to select a detector type to associate with the scan file. The calling routine passes a mode flag to this module to indicate whether "all" is a valid option for the user to select. The module creates a menu listing all of the possible detector types from the data library (and possibly "all" depending on the value of the mode flag). The user then selects one of the detector types. Finally, the code returns a number corresponding to the selected detector type (or a zero if "all" was selected) to the calling routine.

4.1.1.13 Scanhead. This module is used to create a list or menu of all of the scans that have currently been read into the TMAD system. The routine lists out the filename and title associated with each scan and then returns control to the calling routine.

4.1.1.14 Scanmenu. The "scanmenu" module provides a menu-driven user interface to select the specific scan-related function to perform. The following options are provided to the user:

- Return to previous menu
- Select a scan
- Unselect a scan
- Change scan detector type
- List selected scan data.

If the first option is selected, control is returned to the calling routine. If the second through fifth option is selected, the code calls the "getscan," "delscan," "scantype," or "listscan" subroutine, respectively. After control is returned from any of these subroutines, the menu is presented again. This is repeated until the "return" option is selected.

4.1.1.15 Scantype. This module allows the user to change the detector type that has been associated with a particular scan. The subroutine first lists the available scans by calling the subroutine "scanhead" and prompts the user to select one of the scans to change. The code next lists the current detector type assigned to that scan and prompts the user to select a new detector type (by calling the subroutine "pick_d"). Once a new detector type has been selected, the code checks to be sure that detector type has not already been assigned. If it has been assigned, the code prints an error message, otherwise the change is made, and control is returned to the calling routine.

4.1.1.16 TMAD. This is the main TMAD module which serves as a driver for the other modules in the TMAD code. The first function this module performs is to open the output files and to load the benchmark library file by calling the subroutine "getlib." Next, the code provides a menu of functions to the user. There are three possible choices the user can select: done, select scan data, and analyze scan data. The first choice exits the program. The second choice calls the "scanmenu" subroutine, which offers the user several choices for loading scan data. The third choice initiates the TMAD analysis on the selected scan data by calling the subroutine "analyze." If either the second or third option is specified, the option is completed and the menu is presented again. This is repeated until the "done" option is selected.

4.1.2 LIBMAKR Module Design

The following sections detail the design of the LIBMAKR modules.

4.1.2.1 Addlib. The "addlib" module's main function is to read in benchmark data from a file and insert it into a library file. "Addlib" first prompts the user to input the name of the file containing the benchmark data. If the file exists, the data are read in, otherwise an error message is generated and control is returned to the calling module. After the data has been read in, it is passed to the curve-fitting routines, which return the coefficients that define the fitted curve. The user will then be prompted to identify the anomaly type and the detector type that correspond to the input data. In both cases, the user will be given the alternative of specifying one of the types already in the library or of specifying a new type. If a new detector type is specified then the user is also prompted for the decay constant (in inverse days) for the detector. If a new anomaly type is specified then the user is prompted for the minimum and maximum acceptable anomaly sizes. Once this has been done, the data and the curve-fit coefficients will be transferred to the library data structure and written to the library data file. Finally, control will be returned to the calling module.

4.1.2.2 Dellib. "Dellib" deletes existing benchmark data from a library file. The user is first prompted for an anomaly-type/detector-type combination to delete. As an option, the user may delete all anomaly types for a given detector or all detector types for a given anomaly. It is important to note that the code does not actually delete the data but rather flags it as unavailable. It also does not remove the specified anomaly type or detector type from the list of available types. After the data has been flagged, the library file is rewritten.

4.1.2.3 Getlib. This module is responsible for reading in a library file so that it can be modified. It starts out by zeroing the array that is used to flag which anomaly/detector pairs are represented in the library. The user is then prompted for the name of the library file that is to be read in. If the specified file does not exist, the user is given the option of exiting the module, entering a new file name, or creating a new library file using the specified name. If the user opts to create a new file, the user is prompted for a title line and an effective date for the library data (neutron probes typically use a radioactive source that decays over time).

If an existing file is identified, the file is opened and the data read in from the data library file (described in Section 3.2.1.2). The following data are read in from the library file:

- The data library file title
- The effective date of the benchmark data
- The number of possible detector and anomaly types
- The names of the possible detector types
- The names of the possible anomaly types
- The decay constants for each of the detector types
- The minimum and maximum anomaly sizes for each anomaly type
- The number of detector types represented in the library.

Then for each detector the code reads the detector number and the number of anomalies for that detector.

Next, for each detector/anomaly pair selected, the following data are listed:

- The anomaly number
- The title line from the benchmark data file from which the data came
- The number of anomaly sizes in the library.

- The importance weight associated with this detector/anomaly pair.

Each detector/anomaly pair will have one or more sets of data, each corresponding to a different anomaly size. For each anomaly size the following data will be read:

- The anomaly size
- The number of moisture concentrations in the data set
- The curve fit coefficients for this data set
- The standard deviation and goodness of fit values for the curve fit.

The curve corresponding to a given detector/anomaly/anomaly-size combination is constructed (in the module "addlib") from a set of data points relating the detector response to moisture concentration at a number of different moisture concentrations. For each moisture concentration, the following data are read:

- The moisture concentration
- The detector response
- The standard deviation in the detector response value.

Finally, the library data file is closed and control is returned to the calling routine.

4.1.2.4 LIBMAKR. This is the main module of the LIBMAKR code. It starts by calling the "getlib" routine in order to determine the data library file to be used. The code then presents the user with a menu of functions to select from:

- Done (exit program)
- Add data to library (call module "addlib")
- Delete data from library (call module "dellib")
- Modify weights (call module "modwgt")
- List data in library (call module "listlib")
- Modify anomaly limits (call module "modalim")

After the selected function has been completed, the menu is presented again. This is repeated until the "done" option is selected, at which point execution is terminated.

4.1.2.5 Listlib. This module lists the library data to the screen. The user is prompted for the detector type and the anomaly type of the data to be listed ("all" is an acceptable option for either or both). The module then lists the specified data to the screen, as described below.

The first item to be presented is the title of the active library file (i.e., the library file that was selected at the beginning of execution in the module "getlib"). Then, for each detector/anomaly pair selected, the following data are listed:

- The title line of the benchmark data file from which the data came
- The name of the selected detector type
- The name of the selected anomaly type
- The effective date of the benchmark data

- The number of anomaly sizes in the library and the minimum and maximum acceptable values for the sizes
- The importance weight associated with this detector/anomaly pair.

Each detector/anomaly pair will have one or more sets of data, each corresponding to a different anomaly size. For each anomaly size the following data will be presented:

- The anomaly size and the number of moisture concentrations in the data set
- The curve fit coefficients for this data set
- The standard deviation and goodness of fit values for the curve fit.

The curve corresponding to a given detector/anomaly/anomaly-size combination is constructed (in the module "addlib") from a set of data points relating the detector response to moisture concentration at a number of different moisture concentrations. For each moisture concentration, the following data are listed:

- The moisture concentration
- The detector response
- The standard deviation in the detector response value.

After all of the data are listed, control is returned to the calling routine.

4.1.2.6 Modalim. This module allows the user to change the anomaly size limits (minimum and maximum) that have been assigned to a given anomaly type. The user is first prompted for the anomaly type whose limits are going to be changed. The code then lists the current limits and prompts the user for the new values. The new data are written to the library file, and control is returned to the calling module.

4.1.2.7 Modwgt. This module allows the user to change the importance weighting factor that has been assigned to a given detector/anomaly pair. The user is first prompted to input the new weight value. The user then selects the detector type and the anomaly type ("all" may be selected for either) to which the new weight is to be applied. The code searches through the data library and finds all detector/anomaly pairs that are to be changed. For each such pair, the weight is changed to the new value and the title of the combination is written to the screen. The modified library data are rewritten to the library file, and control is returned to the calling module.

4.1.2.8 Pick_a. This module allows the user to select an anomaly type for one of the input or library modification routines. The calling routine passes a mode flag to this module to indicate whether "all" is a valid user option. The module creates a menu listing all of the possible anomaly types from the data library (and possible "all" depending on the value of the mode flag). The user then selects one of the anomaly types. Finally, the code returns a number corresponding to the selected anomaly type (or a zero if "all" was selected) to the calling routine.

4.1.2.9 Pick_d. This module allows the user to select a detector type for one of the input or library modification routines. The calling routine passes a mode flag to this module to indicate whether "all" is a valid user option.

The module creates a menu listing all of the possible detector types from the data library (and possibly "all" depending on the value of the mode flag). The user then selects one of the detector types. Finally, the code returns a number corresponding to the selected detector type (or a zero if "all" was selected) to the calling routine.

4.1.2.10 Putlib. This module writes out the data library to a file. It first opens the data library file and writes the header information to the file (see Section 3.2.1.2 for detailed descriptions of the library file data). The following data are then written to the library file:

- The data library title
- The effective date of the benchmark data
- The number of possible detector and anomaly types
- The names of the possible detector types
- The names of the possible anomaly types
- The decay constants for each of the detector types
- The minimum and maximum anomaly sizes for each anomaly type.

Next, the module searches through the library data to determine the number of detector types and detector/anomaly pairs for which there is data in the library (this is done in order to discard data that was flagged to be deleted). The remainder of the library data are then written to the file, including the number of detector types represented in the library. For each detector type, the detector number and the number of anomaly types for that detector are listed. For each detector/anomaly pair the code writes:

- The title line of the benchmark data file from which the data came
- The number of anomaly sizes in the library
- The importance weight associated with this detector/anomaly pair.

Each detector/anomaly pair will have one or more sets of data, each corresponding to a different anomaly size. For each anomaly size, the following data will be written to the file:

- The anomaly size
- The number of moisture concentrations in the data set
- The curve fit coefficients for this data set
- The standard deviation and goodness of fit values for the curve fit.

The curve corresponding to a given detector/anomaly/anomaly-size combination is constructed (in the module "addlib") from a set of data points relating the detector response to moisture concentration at a number of different moisture concentrations. For each moisture concentration, the following data are written:

- The moisture concentration
- The detector response
- The standard deviation in the detector response value.

Finally, the file is closed and control is returned to the calling routine.

4.1.3 Curve Fit Module Design

This section describes the internal design of the curve fit modules. These subroutines are not described in great detail because they were purchased and not developed as part of this project. Any significant modification to these subroutines should be accompanied by a thorough design description for the affected modules.

4.1.3.1 Crvfit. This module serves as a front-end interface for the curve fitting routines. "Crvfit" receives the parameters from the calling routine and then passes them to the "svdfit" routine. Once control is returned from "svdfit," it is returned to the calling routine.

4.1.3.2 Gammln. This routine calculates the natural log of the gamma function of the parameter passed to the routine. The gamma function is approximated using a technique derived by Lanczos and documented in Press et al. (1986, pp 156-157).

4.1.3.3 Gammq. This module calculates the incomplete gamma function $Q(a,x) = 1 - P(a,x)$ as explained in Press et al. (1989, 160-163). The routine uses two different methods to calculated the result. If $x < a+1$ then $P(a,x)$ is calculated using a series representation as implemented in the routine "gser." The desired result, $Q(a,x)$, is then calculated by taking the complement of $P(a,x)$.

Alternately, if $x \geq a+1$, then $Q(a,x)$ is calculated directly using a continued fraction representation as implemented in the routine "gcf."

Finally, the result is returned to the calling routine.

4.1.3.4 Gcf. This module calculates the incomplete gamma function $Q(a,x)$, using a continued fraction representation, as described in Press et al. (1989, 160-163). After performing the calculation, the result is returned to the calling function.

4.1.3.5 Gser. This module calculates the incomplete gamma function $P(a,x)$, using a series representation, as described in Press et al. (1989, 160-163). After performing the calculation, the result is returned to the calling function.

4.1.3.6 Polyfunc. This module defines the form of the equation used in the curve fitting algorithm. The curve fitting routines assume that the resulting curve will be a function of x whose value is determined, for any value of x, by taking the scaler-product of two vectors, **C** and **X**. The vector **C** is an array of coefficients, and **X** is an array of corresponding functions of x. In this case, the curve is taken to be a second order polynomial, so $\mathbf{X}(1) = 1$, $\mathbf{X}(2) = x$, and $\mathbf{X}(3) = x^2$. After the vector is defined, it is returned to the calling routine.

4.1.3.7 Svbksb. The purpose of this module is to solve the matrix equation $\mathbf{A} \bullet \mathbf{X} = \mathbf{B}$ for **X**, where **A** is a decomposed array and **B** is a vector. This is done using singular value decomposition. A discussion of this technique can be found in Press et al. (1989, 52-64). After the calculation is complete, the result is returned to the calling routine.

4.1.3.8 Svdcmp. The purpose of this module is to decompose the matrix **A** into three matrices U, **V**, and **W**, according to the equation $\mathbf{A} = \mathbf{U} \bullet \mathbf{W} \bullet \mathbf{V}^T$ where **W** is a diagonal matrix. A discussion of this technique can be found in Press et al. (1989, 52-64). After the calculation is complete the result is returned to the calling routine.

4.1.3.9 Svdfit. This module performs the curve fit on the data points. The fit is performed using χ^2 minimization and singular value decomposition of the arrays. This technique is discussed in Press et al. (1989, 515-520). After the calculation is complete, the result is returned to the calling routine.

5.0 REFERENCES

Finfrock, S. H., 1995, *System Program Management Plan for the TMAD Code*, WHC-Sd-WM-CSRS-027, Westinghouse Hanford Company, Richland, Washington.

LANL, 1986, *MCNP - A General Monte Carlo Code for Neutrons and Photon Transport, Version 3A*, LA-7396-M, Rev. 2, Los Alamos National Laboratory, Los Alamos, New Mexico.

Press, 1989, W. H., B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, New York.

This page intentionally left blank.

APPENDIX A

SYSTEM REQUIREMENTS SPECIFICATIONS FOR THE TMAD CODE

This page intentionally left blank.

1.0 INTRODUCTION

This appendix serves as the System Requirements Specification for the TMAD code.

2.0 GENERAL DESCRIPTION

This section describes the general factors that affect the code and its requirements.

2.1 PRODUCT PERSPECTIVE

The TMAD system does not interface with any other software systems. It does, however, make use of data from other projects. Specifically, it requires predicted detector response data from a probe computer modeling effort, and it needs measured detector response data from scans.

There are two general performance expectations placed on the TMAD system. First, the staff time required for the user to execute the code (not necessarily the computer time required) must be less than 1% of the time required to perform the same calculations by hand. Secondly, the moisture contents and anomaly sizes predicted by those calculations must agree with the hand calculations to .1% and .1 cm respectively. Any differences larger than that must be demonstrably the result of changes in the algorithm (such as using a curve fit rather than linear interpolation) from that used in the hand calculations.

2.2 PRODUCT FUNCTIONS

The TMAD system will perform the following functions:

- Create a data library from benchmark calculation data
- Accept measured scan data from at least five different types of detectors
- Allow for different weighting factors to be applied to different detector types
- Interpolate between the benchmark data points, using a curve fitting algorithm, to find all possible solutions for a given set of detector responses
- Establish an error factor for each possible solution and sort the solutions accordingly
- Produce results in a tabular form that can be inspected visually or imported into a graphics package for graphical display.

2.3 USER CHARACTERISTICS

The TMAD system is intended to be used by a very limited group of users. Most of the users will be intimately familiar with the techniques associated with acquiring the data to be processed, both the benchmark and the measured data, and with the basic theory of the code. Some users may be only moderately familiar with these subjects, but they will be working under the immediate supervision of someone who is. As a result, no special user training will be required.

It is anticipated that the TMAD system will be used periodically, as opposed to routinely, with some users falling into the infrequent-use category. The user documentation will need to be sufficiently clear and succinct that the infrequent users can quickly refresh their understanding of the code.

2.4 GENERAL CONSTRAINTS

The basic theory of the code is largely untested, and as a result, much of the theory development will progress along with the code development. Furthermore, there are limits on the resources that can be dedicated to this project (one staff year and one calendar year being upper limits). For these reasons, the code design will be limited to primarily functional developments while more aesthetic features, such as graphical user interfaces, will be relegated to future upgrades.

2.5 ASSUMPTIONS AND DEPENDENCIES

The only major assumption or dependency for the TMAD system is that the benchmark and scan data will be readily available.

3.0 SPECIFIC REQUIREMENTS

This section includes the details necessary for the system developer to create a design.

3.1 FUNCTIONAL REQUIREMENTS

This section provides specific descriptions of the functions that must be accomplished by the TMAD system.

Create a data library from benchmark calculation data.

- The code must be able to read in multiple benchmark data files (as described in Section 3.2.1.1). These files must be combined and written out to a single library file.

- The capability must exist to modify existing library files, including deleting or replacing data, and modifying dates, weight factors, and identifiers for the various data sets.
- The code must be able to read in multiple scan data files (as described in Section 3.2.1.2).
- The capability must exist to change the selection of scan data files that are to be used for the analysis.
- The code must be able to find the intersections between a straight line (corresponding to the detector response) and a curve (representing the detector response versus moisture content relationship for a particular detector/anomaly/anomaly-size combination).
- A curve fit must be performed on the intersection points to establish a curve that represents a moisture versus anomaly size relationship, a process that must be repeated for all anomaly/ detector combinations.
- The code must be able to find the intersections between two moisture/anomaly curves, representing two different detectors, for a given anomaly, a process that must be repeated for all combinations of two different detectors.
- The code must be able to cluster the intersections so that all possible combinations (such that there is no more than one point per detector) are identified.
- Weighted averages must then be performed on the clusters to find one possible solution point.
- The error (defined as the standard deviation) must be calculated for each solution point.
- The process (starting with the moisture/anomaly curve intersections) must be repeated for all anomaly types.
- All resulting solution points must be sorted according to error.
- A confidence factor, defined as the ratio of the error to the next best error minus one, must be established for each solution point.
- Accept measured scan data from at least five different types of detectors.
- The process, starting with the straight line/curve intersections, must be repeated for every point in the scan.
- The results of the TMAD analysis must be printed to files as described in Sections 3.2.1.3 and 3.2.1.4.

3.2 EXTERNAL INTERFACE REQUIREMENTS

3.2.1 User Interfaces

There are six distinct user interfaces with the TMAD system. They include two input files, two interactive interfaces, and two output files. The requirements for each will be described in the following sections.

3.2.1.1 Benchmark Data Input Files. The benchmark data files will serve as the data interface to the LIBMAKR code. They will be ASCII formatted and contain the following information:

- Title
- Number of anomaly sizes in file
- Anomaly size, number of moisture content values for this size
- Moisture content, detector response, standard deviation.

The last item will be repeated once for each moisture content value. The last two items, with the last again being repeated multiple times, will be repeated for each anomaly size.

3.2.1.2 Scan Data Input Files. The scan data files will serve as the data interface to the TMAD code. They will be ASCII formatted and contain the following information:

- Title
- Date (month, day, year)
- Number of points
- Depth (in meters) and detector response.

The last item will be repeated once for every point.

3.2.1.3 Predicted Moisture Content Output File. This file, produced by the TMAD code, will contain the results of the TMAD calculations. There will be a title line followed by one line for every point processed in the calculations. The results line will contain the depth of the point being processed, the type of anomaly predicted for that point, the size of the anomaly, the moisture content at that point, the error associated with the prediction, and the confidence factor associated with the prediction.

3.2.1.4 Intermediate Calculations Output File. This file, produced by TMAD, will maintain a record of the intermediate stages in the calculations to facilitate code verification. This file shall contain all of the intermediate solution points, both for a given detector/anomaly pair and for the set of detector/anomaly pairs, and the results of the curve fit, curve intersection, and solution average calculations. The file should contain sufficient text so that the particular outputs can be identified and so that specific points can be located.

3.2.1.5 LIBMAKR Interactive Interface. The LIBMAKR code shall have an interactive interface to facilitate the library generation functions. This interface shall be text oriented and accessible through any telnet-type connection to the host machine.

The interface shall have the following features:

- User input of library file name
- Choice of creating a new library or editing an existing one
- User input of benchmark data file names
- User assignment of detector type, anomaly type, date, and weighting factors to input data files
- Ability to remove data sets that have already been placed in the library
- Ability to list some or all of the data in the library.

In addition, the code shall check the validity of input filenames, and if a nonexistent file has been specified, the user shall be informed and given the option of entering a new name.

3.2.1.6 TMAD Interactive Interface. The LIBMAKR code shall have an interactive interface to facilitate the scan data file selection functions. This interface shall be text oriented and accessible through any telnet-type connection to the host machine.

The interface shall have the following features:

- User input of scan data file names
- User assignment of detector type to input data files
- Ability to remove data sets that have already been selected
- Ability to list some or all of the data in the library.

In addition, the code shall check the validity of input filenames, and if a nonexistent file has been specified, the user shall be informed and given the option of entering a new name.

3.2.2 Hardware Interfaces

The TMAD system shall be loaded on one or more UNIX-based workstations. It should be accessible either directly from the workstation terminal or from remote terminals.

3.2.3 Software Interfaces

There are no software interfaces with the TMAD system.

3.2.4 Communication Interfaces

There are no communications interfaces with the TMAD system.

3.3 PERFORMANCE REQUIREMENTS

The TMAD system is intended to be a single-user system and need only accommodate one user at a time. The data library must be able to represent at least five anomaly types with at least ten anomaly sizes per type and at least ten moisture contents per size. The library must also be able to accept at least five different detector types. The TMAD code must be able to accept at least five different scans, with at least one thousand points each. The interactive portions of the code must be able to be completed in no more than 1% of the time required to perform the same calculations by hand, (i.e., approximately 1 hour).

3.4 DESIGN CONSTRAINTS

This section describes outside constraints that affect the design of the code.

3.4.1 Standards Limitations

No applicable standards or regulations have been identified for the TMAD system.

3.4.2 Resource Limitations

The maximum size of the code is limited by the amount of memory available on the UNIX workstations on which it is destined to run.

3.5 ATTRIBUTES

This section describes the desired attributes of the system.

3.5.1 Availability

The TMAD system should be available on a daily basis. Occasional periods of unavailability, consistent with normal operating experience of networked workstations, is acceptable.

3.5.2 Security

No security issues have been identified for the TMAD system.

3.5.3 Maintenance

It is anticipated that regular updates to the theory and functionality of the TMAD system may be desired. The code shall be written and implemented to accommodate this need.

3.5.4 Data Integrity

The benchmark data library files must be protected against inadvertent alteration. Write access to these files should be limited to the extent possible.

3.6 OTHER REQUIREMENTS

3.6.1 Data

See Sections 3.2.1.1 through 3.2.1.4 for a discussion of the data requirements.

3.6.2 Operations

There are no special operations modes required for the TMAD system.

3.6.3 Site Adaptation

There are no special site adaptations required for the TMAD system.

3.6.4 Options

The only alternative identified was to perform the calculations by hand. This approach had the advantage of allowing the engineer to immediately identify problem areas, but the time required to perform the calculations would be prohibitive.

3.6.5 Scheduling

TMAD is expected to be available whenever the host computer is available. There should be no regular or significant downtime associated with code maintenance.

3.6.6 Reliability and Recovery

The TMAD system should function normally every time it is executed. The code should fail to function normally only in the event of a host computer system failure or incorrect user input. In the latter case, a warning message should be generated informing the user of the problem. There are no special restart capabilities required.

3.6.7 Audits

No audit requirements have been identified for the TMAD system.

3.6.8 Priorities

The TMAD system cannot be released for use until a probe-modeling effort has been completed and probe implementation has been initiated. The TMAD system must be complete before tank moisture monitoring can be performed, specifically, before conclusions can be drawn from the data.

3.6.9 Transferability

No transferability issues have been identified for the TMAD system.

3.6.10 Conversion

There are no data conversion requirements for the TMAD system.

3.6.11 Testing and Acceptance Criteria

The TMAD system will be tested to ensure that it has met all of the design criteria. The testing will consist of executing the code on an artificial scan made up of several points from actual scans. These points should reflect a range of actual values encountered in tank scans. The same points will then be evaluated by hand, and the results of the two approaches will be compared. For acceptance, the TMAD predicted moisture content and anomaly size must match the results of the hand calculations to 0.1% and 0.1 cm, respectively. Any larger differences must be demonstrably the result of differences in the algorithm as it was implemented in TMAD, compared to that used in the hand calculations.

3.6.12 Documentation

The documentation required for the TMAD system includes the following:

- Project Management Plan
- System design description
- Verification and Validation Report
- User's Manual.

3.6.13 Training

No special training requirements have been identified for the TMAD system.

3.6.14 Security and Privacy

There are no special security or privacy issues associated with the TMAD system. The code and data must be protected against inadvertent damage. This should be accomplished by using standard backup procedures and by limiting access.