NISTIR 89-3929

PLANAR NEAR-FIELD CODES FOR PERSONAL COMPUTERS

Lorant A. Muth
Richard L. Lewis

National Institute of Standards and Technology
U.S. Department of Commerce
Boulder, Colorado 80303-3328

October 1989

NISTIR 89-3929

PLANAR NEAR-FIELD CODES FOR PERSONAL COMPUTERS

Lorant A. Muth
Richard L. Lewis

Electromagnetic Fields Division
Center for Electronics and Electrical Engineering
National Engineering Laboratory
National Institute of Standards and Technology
Boulder, Colorado 80303-3328

October 1989

Sponsored, in part, by
Air Force Guidance and Metrology Center
Newark Air Force Base, Ohio 43057

CONTENTS

Planar Near-Field Codes for Personal Computers

by
Lorant A. Muth and Richard L. Lewis

We have developed planar near-field codes, written in Fortran, to serve as a research tool in antenna metrology. We describe some of the inner workings of the codes, the data management schemes, and the structure of the input/output sections to enable scientists and programmers to use these codes effectively. The structure of the codes is seen to be open, so that a user can incorporate a new application into the package for future use with relative ease. The large number of subroutines currently in existence are briefly described, and a table showing the interdependence among these subroutines is constructed. Some basic research problems, such as transformation of a near field to the far field and probe position error correction, are carried out from start to finish, to illustrate use and effectiveness of these codes. Sample outputs are shown. The advantage of a high degree of modularization is demonstrated by the use of DOS batch files to execute Fortran modules in a desired sequence.

Key words: antenna metrology; data management; planar near-field codes; research tool; subroutines

1. Introduction

Most research problems in antenna metrology are computationally intensive, and program development makes up a substantial part of the research effort. Hence, isolating frequent computational themes in this research area and developing *independent modules* that can perform any of these computational themes in any order independently of any previous computational step is very desirable. Improvements in both the quality and quantity of research can be a by-product of such a computational tool. Ideally, such a software package should be an open-ended system; that is, new modules can be added to it painlessly to increase the versatility of the package. It should also be easy to use and learn, and, therefore, adaptable to new areas of research. With the cooperative effort of the members of the Antenna Metrology Group such a software package could evolve into a comprehensive research tool over a short period of time.

With these thoughts in mind, we have taken the first steps to accomplish the goal of creating a comprehensive software package suitable for conducting state-of-the-art research on a personal computer.[1] We have achieved a very high level of modularity by creating a large number of subroutines (written in Fortran) that can

[1] Because most antenna metrology research problems are computationally intensive and usually have large memory requirements, the authors recommend that a personal computer equipped with the fastest available CPU and floating point processor be used and that at least 4 megabytes of RAM be made available.

be used in many different contexts, since the subroutines emphasize structure rather than content of small computational problems. By the same effort, we have made it relatively simple to create higher level subroutines, since such routines can rely heavily on the existing low level subroutines of general applicability. These higher level routines accomplish more complicated and complete computational tasks than the low level subroutines. In turn, they can be combined to form independent modules, which are the selected subtasks of a particular research effort. These subtasks usually will be subtasks in other research areas, too. Hence, the effort expended in creating them will be saved many times over in future endeavors.

Particular attention has been given to the way information flows to and from the modules and between modules. We have automated much of the data management needed to provide a smooth transition as one module finishes its task and another is executed to accomplish the next step of the research. A large number of small modules, playing a supportive role in data management, have been created to allow manipulation of datasets according to the needs of the current phase of the research project. For example, an existing dataset merely has to be *activated* to make it accessible to a module about to be executed. Thus, both the modules chosen to be executed and the datasets to be used can be controlled interactively by the scientist. This makes for a very flexible computational procedure, freeing one's time and energy to think about research procedure rather than computational detail.

In the next section, concentrating on the main features, we outline the structure of the Planar Near-Field Codes (PNFC), and in the subsequent sections we present essential details of the main features. It is our intention that any researcher, programmer or scientist, be able to use these codes effectively after familiarizing himself with the contents of this report.

2. General Features

The complete PNFC is structured into *modules*. To be able to determine the function of a module we merely have to decipher the acronym that was constructed to name the module. Once deciphered, the full function of the module should be self-evident. In Table 1 we have compiled the symbols used to construct module names and define the meaning of each symbol. In Table 2 and 3 we list the modules used to conduct research and the modules used to manage data access during the course of research, respectively. A brief descriptor of their function is also included.

All of the research modules listed in Table 2 perform some manipulation of an existing dataset, that is, they either numerically transform the dataset or perform some I/O operation on it. Each module was designed to perform a single computational task that is an important aspect of research in antenna metrology. Some of the modules are more specific to antenna metrology than others. For example, the module URDNFFF (Utility, ReaD a Near Field and transform it to the Far Field) is an ever present computational step in this research area, but UPRNCBD (Utility, PRiNt a Complex Binary Dataset) is obviously of more general applicability. How to execute these modules is demonstrated in Section 4.

The modules listed in Table 3 perform simple data management functions. For example, USWTOFF (Utility, SWitch TO Far Fields) activates the far field datasets that have been previously created and recorded within the data management part of the system. After USWTOFF has been executed any subsequent executions of modules that can use either far-field or near-field data will access the far-field datasets, unless this switch is overridden by a nonzero *active* field. How to *activate* a specific dataset to make it the dataset that any module will use will be covered in Section 3.

All these modules are constructed from a large set of independent subroutines that perform specific computational or I/O subtasks. They are used repeatedly in various sequences to produce the specific results of the module. These subroutines are compiled into a library, which is linked to a module at compilation time. All existing subroutines are listed in Table 7, along with a brief description of their function.

All modules use a general set of input parameters that belong to the original dataset. The original datasets are recorded as direct access binary files, so that specific records within them can be accessed at will. This might be useful if some selected records are found to be in error. (How to create the original direct access datasets from some ascii file that was created on some other computer or data acquisition system is explained in Appendix A.) The first 7 records in these datasets contain the essential parameters of the dataset. All modules access the original direct access file to input the original parameters of the dataset, although only a subset of these might actually be needed by the specific module in use. This procedure assures that the same parameter set will be used by all modules for a specific dataset. A list of these parameters is given in Appendix A.

Each module might also access a parameter (.PAR) file that is specific to it. For example, UMAKEDZ (Utility, MAKE DZ), which creates a probe displacement error function, reads the parameter file PERDZ.PAR if periodic error functions are requested, and UTS (Utility, Taylor Series) reads the parameter file SCALE.DZ to input the amplitude of the error function requested for the current execution. The parameter files currently in existence and the research modules that access them are listed in Table 4. The parameter files accessed by the data management modules are tabulated in Table 5. The contents of each parameter file will be defined in Section 3.

All necessary I/O procedures are handled within each module, but there are some specific modules that prepare the data and create ascii files that can be further processed for graphical output. Two such modules are UCBDGRD (Utility, Complex Binary Dataset to .GRD file) and UCBDDAT (Utility, Complex Binary Dataset to .DAT file), which create ascii datasets to be used for plotting 3-D and simple linear plots, respectively. These modules also rely on specific parameter files to perform their function as desired. These parameter files are listed in Table 4.

Finally, all modules have very similar structures and differ significantly only in their computational sections. The common structure is as follows:

a. Read all relevant switch settings and determine the unit numbers[2] of existing

[2] Here and throughout this report *unit numbers* refer to the file extensions *xx* in

datasets. Check to see whether there is room for more datasets on the disk and assign the new unit numbers.

b. Read all relevant parameters needed by the module.
c. Read all parameters describing the dataset to be used.
d. Read all datasets needed by the module.
e. Prepare for computations.
f. Perform the computations.
g. Output the results to the preassigned units.
h. Set the relevant switches and update the unit numbers of the new datasets.
i. Output an ascii file to record all parameters and I/O activity.
j. Update the history file to show which modules were executed.
k. Stop execution of the module with *'Successful termination'* message.

This structure seems to be very successful in that modules that are truly independent of each other have been constructed, which, therefore, can be executed in any order as long as the relevant datasets have been created. Under these conditions a research project can be implemented with relative ease either interactively or with the use of DOS batch files. (The use of DOS batch files to enhance research efficiency is discussed in Section 6.)

3. Data Management

In this section we present the details of unit or dataset management built into the system as a whole. Specific modules make use of this procedure according to their requirements. Here the terms *data management* and *unit management* have the same meaning, as datasets generated by PNFC for the purpose of computations reside on files with filenames FORT.*xx*, where *xx* is some integer refering to a Fortran *unit number* assigned internally by the module being executed. (The filename FORT is automatically assigned when a Fortran binary write is executed.)

a. Initialization of the system.

The system has to be initialized before starting any research project with a new dataset. Both the system parameters and the unit numbers where different datasets will reside are initialized in this procedure. Here we will describe how the unit numbers are set and manipulated at the start of the research project. In Appendix B the output of the initialization module is shown and an explanation of features not covered in this section is presented.

When the UINITUN (Utility, INITialize Unit Numbers) module is executed the initial unit numbers for the far-field and the near-field datasets are read from a parameter file (INIT.IUN) and entered into the unit number files named FF.IUN and NF.IUN. After initialization the modules URDFFNF or URDNFFF can be executed to read in the existing direct-access complex binary dataset containing the original data to be analyzed. (Subsequently, the same modules will access datasets according to the unit management switch settings. See Section 3b below.)

---

the filenames *fort.xx* that are automatically generated by the system when a binary dataset is written to the disk.

Both modules output both the far field and the near field to FORT.*xx* files; the filename extensions *xx* are obtaqined from the files FF.IUN and NF.IUN.

All far fields created after initialization will be assigned unit numbers one less than the previously assigned far field unit number, and all near fields created after initialization will be assigned unit numbers one higher than the previously assigned near field unit number. Hence, the far-field and near-field unit numbers will converge toward each other as datasets are created by executing module after module. Before any module proceeds with execution of its task it checks to see whether there is enough of a difference between the last far-field and the last near-field unit numbers to allow the creation of additional datasets. If the far-field and near-field unit numbers are adjacent to each other, no module that creates a new dataset is allowed to proceed, and an appropriate error message to that effect is displayed. In this manner, disk overload is prevented, since new datasets cannot be created indefinitely.

b. The Complex Binary Dataset (CBD) files.

Except for the original datasets, which are stored as direct access binary files, the modules read and write complex binary datasets (CBD) during execution to store intermediate results in the course of the research project. These datasets are recorded with the filename FORT and with integer unit numbers for extensions. The unit numbers are automatically assigned, as described in the previous section. For example, FORT.40 would be the initial near field unformatted complex binary file, and FORT.60 would be the initial far field unformatted complex binary file.

Since all modules read and/or write one or more CBD files, we must keep track of these files and must be able to access a desired dateset with relative ease. For this purpose a unit number management support system has been constructed. This works as follows:

An *existing* dataset is identified by its *unit number*, which is the extension of the FORT file. An *existing* unit number is any unit number that has been created since initialization. An existing unit number, in general, has no special status and is not automatically accessed by any module until it is made *active*, *additional*, or *current*. A unit number is *active* if its value is recorded in the ACTIVE.IUN file, whereas a unit number is *additional* if its value is recorded in the ADD.IUN file. The *current* unit numbers are the last unit numbers recorded in the files FF.IUN and NF.IUN. In general, these are the unit numbers created by the most recently executed module, but can be altered according to the user's needs. A general purpose module can access either the *current near field* unit number or the *current far field* unit number, depending on the setting of the variable FFNF recorded in the file FFORNF.IUN. The variable FFNF can have the values 'ff' or 'nf'.

When modules access datasets a precedence rule is followed: the ACT*ive* file gets accessed first, and the ADD*itional* file gets accessed if the module requires 2 datasets. The *current* file gets accessed only if the ACT*ive* file is set to 0, and any *existing* file can be accessed only if it is made ACT*ive*, ADD*itional* or *current*. To access the desired *current* files with modules that process either far-field or near-field datasets the 'FFORNF' switch has to be set to tell the system that one is interested in far-field or near-field unit numbers.

A number of utilities have been written to define these file types easily. These utilities are listed in Table 3. To view the existing unit numbers we execute USHOWUN (Utility, SHOW Unit Numbers), which summarizes the existing files according to their type (as defined in FFORNF.IUN) and status (ACT, ADD, current, existing). USHOWUN will also identify the unit numbers of special datasets, such as the TS (Taylor Series) file, EC (error corrected) and DS (direct sum) files. To *activate* a dataset, execute one of the special utilities listed in Table 3. Similarly, we can *add* a dataset. To make a dataset *current*, one can execute the decrementing or incrementing modules (UDECFF, UDECNF, UINCFF, UINCNF) repeatedly until the desired unit number is the last unit number shown by USHOWUN. Two examples of the output of USHOWUN are given in Appendix C along with explanations.

c. Output Files.

As discussed above most modules read and write CBD files according to the unit management scheme built into every module. In addition, some of the modules create special ascii files to be used as input to graphics programs. The module UCBDGRD, for example, reads the ACT*ive* or *current* CBD file, with filename FORT and an extension defined by the *active* or *current* unit number. It then outputs ascii files, whose filenames are obtained by concatenating the setting of the switch FFORNF with the descriptors AMP or PHASE, and appending a filename extension .GRD. The structure of these files is determined by the requirement of the graphics package in use. Similary, the module UCBDDAT creates ascii files for simple xy-plots with filenames obtained the same way as for .GRD files, using .DAT as the filename extension. This module outputs a set of x-values and one, two or three y-values. The actual number of data columns output by UCBDDAT is determined by the ACT*ive*, ADD*itional* and *current* switch settings. The rules are as follows: to write only a single column of y-values, the *active* field must be non-zero and the *additional* field must 0. To write two sets of y-values, the *additional* field must also be non-zero. To write three sets of y-values, both the *active* and *additional* unit numbers must be 0, in which case the *current* unit number will be used to create the first column, and the next two adjacent *existing* unit numbers will be used to create columns 2 and 3 in the .DAT file. A simple module UACTADD0 (Utility, set ACTive and ADDitional to 0) will reinitialize the unit numbers so that up to three columns of data might be written.

All research modules create a .OUT file, with filenames identical to the module names, that contain information about the execution flow of the module. Parameters used and the unit numbers accessed or created are listed in these files, so that an orderly cross-referencing can be conducted if some of the results are brought into question. In addition, these modules record their activity in a history file (.HST) so that the sequence of executions can be checked at a later time.

4. Research Modules

In Table 2 we list the currently existing modules. These modules were designed in the course of a research project where the goal was to understand the propagation of errors in near-field data to the far field, and to develop techniques to remove the effects of these errors from the far field. Thus, some of these modules are very specific to this research projects; others, however, have more general applicability.

To illustrate the use of these modules in research, we provide first a simple, then a more elaborate example of computational sequence that delivers results required by two representative research problems.

Simple research problem.

Given a near field dataset, obtain a perspective plot of both the near field and the far field of the antenna.

Using 'x' to mean 'execute' a module, this simple task would be accomplished by entering the following batch commands at the DOS prompt:

```
x uinitun
x urdnfff
x ucbdgrd
plt ff
x uswtonf
x ucbdgrd
plt nf
```

Here *plt* is a DOS batch file that calls on the plot package on the system to process the graph data files output by UCBDGRD. Thus, this part of the procedure would vary from system to system, depending on the graphics package used.

From Table 2 we can easily ascertain that the above sequence of computational steps above will deliver the results required. First, by executing UINITUN we initialize the system variables and unit numbers to delete the results of all previous executions of modules. Next, we read in the original near-field dataset and transform it to the far field. At this point, the data management system sets the *ffornf* variable to *ff*, since the last field created was a far field. Now UCBDGRD will access the far-field dataset to create a plot file. To create a plot file using the current near field, we must set the system variable *ffornf* to *nf*. Hence, we execute USWTONF, and then UCBDGRD will access the near-field dataset to create a plot file for the near field.

A more complicated research problem.

Given a near-field dataset, introduce known errors into this near field. Use a known probe position error function and the Taylor series technique to generate error-contaminated near-field values. Then, remove these errors from the data using a well defined error correction technique, and compare the error-free, error-contaminated and error-corrected near and far fields by looking at the respective complex ratios of field values at each data point. Present the results in perspective plots and/or linear plots showing ratios of amplitudes and phase differences.

Using the existing set of research modules, this relatively involved research task can be brought to conclusion as follows:

```
x uinitun
x umakedz
x urdnfff
x uts
x ec
```

Executing this sequence, we have accomplished the first part of the research. Again, we started by initializing the system parameters and unit numbers. Then, a probe displacement error field has been created by executing UMAKEDZ. The function to be used is defined in the parameter file read by the module. This is listed in Table 4. A .GRD file to draw a perspective plot of the error function has also been created. Next, the near-field dataset is read in and the corresponding far field is calculated. Then errors are introduced into the original near field using the Taylor series technique by executing the module UTS, and the errors are removed by inverting the error operator when the module EC is executed. At this point each dataset has been recorded on the disk in *complex binary dataset* files with filenames *fort* and file extensions *.xx*, where *xx* is some unit number automatically assigned by the data management section of the system. We can now proceed to obtain the far field corresponding to each near field created up to now. We proceed as follows:

```
x udecnf
x urdnfff
x uincnf
x urdnfff
```

All far fields of interest have now been created. By executing UDECNF, the current near field unit number has been decremented by 1 (assuming that the unit increment/decrement parameter is 1, the default), thereby making the near field obtained prior to the last near field *current*. Then executing URDNFFF will transform this near field into a far field. Incrementing the near-field unit number will increase the *current* unit number by 1, which, in this case, is the last near field created. Again executing URDNFFF will create the corresponding far field.

Only plotting and comparing the various near fields and far fields is left. The module UDIVCBD can be used to form the complex ratio of two near-field or far-field datasets. As discussed above in the data management section, the two desired datasets are loaded by defining an *active* and *additional* unit numbers, or if these are set to 0, then the two most recently created fields (near or far) will be used. Thus, to take the ratio of the error-contaminated near field to the original near field, we execute the following:

```
x uswtonf
x uactts
x uaddnf0
x udivcbd
```

Similarly, to take the ratio of the error-corrected near field and of the original near field we execute the following:

```
x uactec
x uaddnf0
x udivcbd
```

In both of the above sequences of operations the complex ratio field is created, which is recorded sequentially using near-field unit numbers, since we executed USWTONF at the beginning of this sequence. Note that the second execution of UADDNF0 is really redundant, since it was already executed above.

To create far-fields ratios the procedure is somewhat different, since far fields have not been labeled by special identifiers, such as *ts* and *ec*. Any far field can be made *current* by incrementing or decrementing the far-field unit numbers an appropriate number of times, and can be selected by executing one of the modules UACTFF or UADDFF. Thus, to form all ratios we execute the following sequence:

```
x uswtoff
x uaddff0
x uincff
x uactff
x udecff
x udivcbd
x uincff
x uactff
x udecff
x udivcbd
```

All far-field ratios of interest have now been created and recorded on far-field unit numbers. This was accomplished by first switching to the far fields (USWTOFF), then making the original far field the *additional* field (UADDFF0), followed by making the far field created before the last one the *active* field (UINCFF, UACTFF and UDECFF) and taking the ratio (UDIVCBD). After the ratio was taken the *current* far field unit number was automatically increased. Next, the previously created far field was made *current* (UDECFF) and *active* (UACTFF), the *current* unit number reincremented (UINCFF) and then the ratio (UDIVCBD) was taken. Each ratio field was automatically recorded on next available far-field unit number.

At this point we can obtain a system status report, so that any problem with the sequence of operations could be detected. For this purpose we execute the module USHOWUN, whose output is presented in the second table in Appendix C, with a detailed discussion.

After examining the output of USHOWUN and ascertaining that no errors were made, we can proceed to plot any of the existing fields (*fort.xx* files). First, an ascii plot file (.GRD) needs to be created using the module UCBDGRD, after which plots can be created using the plot package. The module UCBDGRD will read the *current* far or near field depending on the setting of the switch *ffornf*. A setting can be selected by executing USWTOFF or USWTONF. Alternatively, a unit number can be loaded by making it *active* by executing one of the number of modules which have the phrase ACT embedded in their names.

Sample plotting procedures would be as follows:

```
x uswtonf
x unorm1
x ucbdgrd
plt nf
```

or

```
x uswtonf
x unorm0
x uactts
x ucbdgrd
plt nf
```

or

```
x uswtoff
x uact0
x ucbdgrd
plt ff
```

In all these examples we first specify the type of fields we want to access. Then, in the first example, we set the normalization constant to 1, since we are plotting a ratio field, which does not have to be normalized when it is converted to decibels. Next, a plot file is created by UCBDGRD. In the second example, the normalization constants are restored to their proper values (UNORM0), then the error-contaminated near field created by the Taylor series method is *activated* (UACTTS), and a plot file is created. In the third example, we switch to the far field, zero out the *active* unit number so that the *current* far field is accessed by UCBDGRD to create the plot file. In all three cases, we use the DOS batch command *plt* to plot either the far field (*ff*) or the near field (*nf*).

5. Output Files.

All research modules have been constructed to write an output file where the parameters and data files used during execution are clearly listed. This way the settings of input/output parameters can be cross-referenced, and the correctness of the computational sequence and numerical inputs can be ascertained. These output files have the name of the modules as their filenames and .OUT for the file extension.

Certain modules write ascii datasets to be used by the graphics package on the system. The module UCBDGRD creates two-dimensional ascii datasets for perspective and contour plots, and the module UCBDDAT creates ascii datasets (.DAT) for simple xy-plots. The module URMSCBD creates a .DAT file to plot the rms distribution of the power radiated in a far field. These .GRD and .DAT ascii files may also be used to examine the data for any features we might be interested in.

Finally, the module UPRNCBD prints the rows and/or columns of any far- or near-field CBD file, according the switch setting of *ffornf* and the settings of the *current* and *active* unit numbers. If the *active* unit number is 0, then the *current* file will be printed. The particular rows and/or columns to be printed and the respective ranges of data are set by the parameter file *sub.prn*.

6. DOS Batch Files

DOS batch files can be used to advantage to save time and effort when performing step-by-step computations to obtain a result. We can write batch files merely as abbreviations of longer commands, or to collect a set of executable steps that will be used many times over. The complexity of the batch files and their usefulness are limited only by the programmer's knowledge of the DOS operating system and the programmer's imagination.

The use of the *plt.bat* file has been illustrated in the previous section a number of times. Another example of a batch file is the abbreviation of the execution of the first simple research problem discussed above. Thus, the batch file *pltnfff* would look like this:

```
command /c x uinitun
command /c x urdnfff
command /c x ucbdgrd
command /c x uswtonf
command /c x ucbdgrd
command /c plt nf
command /c plt ff
```

Simply typing *pltnfff* at the DOS prompt would execute all the steps in this batch file. We now have a very easily usable, high level program that will produce plots of the near and far fields of the current dataset. The DOS expression *command /c* is used here to continue execution within the batch file to the last line. Without *command /c* execution would not return to the next step, but exit to the DOS prompt.

The second research problem is the implementation of the error-correction problem using a specific error-creation and error-correction technique. What might change from one implementation to the next is the original dataset to be used, the form of the error function and the magnitude of the error function. These are all inputs to the complete procedure; that is, the program execution steps are the same, independent of these parameters. Therefore, a DOS batch file is appropriate for recording the steps of this relatively complicated research project. This batch file could be appropriately called *error.bat* (error correction), and would look like this:

```
command /c x uinitun
command /c x umakedz
command /c x urdnfff
command /c x uts
command /c x ec
command /c x udecnf
command /c x urdnfff
command /c x uincnf
command /c x urdnfff
command /c x uswtonf
command /c x uactts
command /c x uaddnf0
command /c x udivcbd
command /c x uactec
command /c x udivcbd
command /c x uswtoff
command /c x uaddff0
command /c x uincff
command /c x uactff
command /c x udecff
command /c x udivcbd
command /c x uincff
command /c x uactff
command /c x udecff
command /c x udivcbd
```

This batch file goes as far as creating all the required near and far fields of the research project, as well as the ratio fields. It stops short of plotting any of the existing fields. A separate batch file would be appropriate for creating a desired set of plots.

The batch files using the executable modules of the PNFC allows one to create and save complicated research procedures in a straightforward and efficient manner. A collection of such batch files over a period of time would greatly enhance the computational ability and efficiency of any research group.

7. Symbol Definitions

To help the user understand the names of the modules and subroutines used in the PNFC, we compiled a table of symbols with their most commonly used definitions. This list is presented in Table 6. This table should make reading the source codes easier. We hope that authors of new code will use existing symbols as far as possible to contribute to the coherence of the full package.

8. Subroutine Descriptors

In Table 7 we compiled a list of subroutines along with brief descriptors of their functions. This can be helpful when creating new modules or when planning to write new subroutines to perform computational tasks not yet addressed in the package.

9. Table of Dependencies

In Appendix D a table of dependencies showing the interrelationship between the various subroutines is presented. Primarily this table can serve as an index of subroutines file, since all existing subroutines are included alphabetically in the leftmost column. The subroutines called by the routine on the left are listed following the colon in the order in which they are called. In this manner we can get an overview of both the contents and structure of the complete code. Such a file can be used to advantage when developing new code, or when improvements in the existing code are contemplated.

10. Summary

In this report we have outlined and documented the computational structrures and procedures of a newly created software package named Planar Near Field Codes (PNFC) for personal computers. This package supports the computational effort needed to solve research problems in antenna metrology.

The PNFC can be used to address diverse research problems because of its highly modular structure wherein independent subroutines have been combined to create independent research modules. These modules have been constructed to provide the computational procedure for recurring research themes in antenna metrology as well as for research problems that arose in connection with the specific task of correcting for probe position errors in planar near field data. A data management procedure has been implemented that automatically keeps track of the various datasets being created and stored during the course of research. Because of the highly modular nature of the PNFC new research modules can be easily constructed and incorporated into the total system. A large number of independent subroutines are available to support new efforts, and new subroutines can be added without any difficulty.

Streamlining computational procedures along the lines built into this software package can result in significant reduction of time needed to obtain answers to complicated research problems. Additions to the current version of the package over an extended period of time would create a truly comprehensive computer package capable of dealing with most computational needs of antenna metrology easily. For this reason all users are encouraged to add to the effort as they see appropriate.

Table 1.
Definition of Symbols Used in Naming Modules

SYMBOL	MEANING
0	initial
	set to 0
1	version 1
2	squared quantity
act	active, activate
act0	set ACTive switch to 0
add	additional
add0	set ADDitional switch to 0
amp	amplitude
ap	amplitude, phase
cbd	complex binary dataset
cor	correct, corrected, correction
db	in dBs
dif	difference
div	divide, divided (ratio)
drv	derivative
ds	direct sum
dacb	direct access complex binary (file)
dat	.DAT (file)
dbp	dB, phase complex storage
dc	decrement
dec	decrement
deriv	derivative
dif2	difference between squared amplitudes
difa	difference in amplitude
dz	function dz
ec	error correction
err	error
ff	far field
ff0	original far field
grd	.GRD (DOS file extension)
hst	history
inc	increment
init	initialize
laplcn	Laplacian
make	
nf	near field
nf0	original near field

nc	increment
norm0	normalization of original datasets
norm1	normalization with 1
op	operator
prn	print
rbd	real binary dataset
rd	read
rms	root mean square
show	
sw	switch
to	to
ts	Taylor series
u	utility
un	unit number

Table 2
List of Modules That Perform Basic Computational Tasks

UAMP2CBD	read a near-field or a far-field dataset and write its squared amplitude to a complex binary data file
UAPDACB	read an amplitude, phase ascii file and write a direct access complex binary file
UCBDDAT	read a complex binary data file and create a .DAT file for x-y plots
UCBDGRD	read a complex binary data file and create a .GRD file for two dimensional contour or surface plotting
UDBPDACB	read a dB,phase ascii file and write a direct access complex binary file
UDERIV	read a near-field dataset and write the derivative of some specified order
UDIF2CBD	read two far-field or near-field datasets and write the difference of the squared amplitudes to a CBD file
UDIFACBD	read two far-field or near-field datasets and write the difference of the amplitudes to a CBD file
UDIFCBD	read two far-field or near-field datasets and write the complex difference to a CBD file
UDIFDB	read two far-field or near-field datasets and write the difference of amplitudes in dBs and the phase difference to a CBD file
UDIVCBD	read two far-field or near-field datasets and write the complex ratio to a CBD file
UDIVRBD	read two real binary data files and write the ratio to a RBD file
UDS	create a near-field dataset with errors in it using the direct sum algorithm
UERR	create a near-field dataset with errors in it using the Taylor series method
UERRCOR2	read a near-field dataset with errors in it and do a second order error correction
ULAPLCN	read a near-field dataset and form the Laplacian and check that it satisfies the scalar wave equation
UMAKEDZ	create an array DZ using a specified error function and write a GRD file to plot the error function
UOPNORM	calculate the norm of the error operator
UPRNCBD	print specified rows and columns of a complex binary data file
URBDGRD	read a real binary dataset and create a grid file for plotting
URDDZ	read the error function file dz and create a GRD file for plotting
URDFFNF	read a far field and transform it to the near field
URDNFFF	read a near field and transform it to the far field
URMSCBD	sum the rms values at grid points of a CBD file, and create a .DAT file for plotting

USUBGRD	convert a specified part (SUB) of a CBD file to a GRD file for plotting
UTS	introduce errors into a near-field dataset using the Taylor series method

Table 3.
List of Modules That Perform Basic Data Management Functions.

UACT0	set the *active* unit number to 0
UACTADD0	set the *active* and *additional* unit numbers to 0
UACTDDB	set the *active* unit number to the value in *difdb.iun*
UACTDIF	set the *active* unit number to the value in *dif.iun*
UACTDIV	set the *active* unit number to the value in *div.iun*
UACTDRV	set the *active* unit number to the value in *drv.iun*
UACTDS	set the *active* unit number to the value 0
UACTEC	set the *active* unit number to the value in *ec.iun*
UACTFF	set the *active* unit number to the final value in *ff.iun*
UACTFF0	set the *active* unit number to the initial value in *ff.iun*
UACTNF	set the *active* unit number to the final value in *nf.iun*
UACTNF0	set the *active* unit number to the initial value in *nf.iun*
UACTTS	set the *active* unit number to the value in *ts.iun*
UADD0	set the *additional* unit number 0
UADDDRV	set the *additional* unit number to the value in *drv.iun*
UADDDS	set the *additional* unit number to 0
UADDEC	set the *additional* unit number to the value in *ec.iun*
UADDFF	set the *additional* unit number to the final value in *ff.iun*
UADDFF0	set the *additional* unit number to the initial value in *ff.iun*
UADDNF	set the *additional* unit number to the final value in *nf.iun*
UADDNF0	set the *additional* unit number to the initial value in *nf.iun*
UADDTS	set the *additional* unit number to the value in *ts.iun*
UDCDFDV	decrement the unit number recorded in *difdiv.iun*
UDECFF	decrement the value of the *current* far-field unit number
UDECNF	decrement the value of the *current* near-field unit number
UINCFF	increment the unit number recorded in *ff.iun*
UINCNF	increment the unit number recorded in *nf.iun*
UINITUN	initialize the system parameters and unit numbers
UNCDFDV	increment the unit number recorded in *difdiv.iun*
UNORM0	set the far-field and near-field normalization constants to their initial values
UNORM1	set the far-field and near-field normalization constants to unity
URESTFF	restore the unit numbers in *ff.iun* to the values saved in *save.ffs*
URESTNF	restore the unit numbers in *nf.iun* to the values saved in *save.nfs*
USAVEFF	save the unit numbers recorded in *ff.iun* in *save.ffs*
USAVENF	save the unit numbers recorded in *nf.iun* in *save.nfs*
USETDB1	copy the first set of values of *dbctoff,dbfloor* in *dbmins.set* to *dbmin.db*
USETDB2	copy the second set of values of *dbctoff,dbfloor* in *dbmins.set* to *dbmin.db*

USETDB3	copy the third set of values of *dbctoff,dbfloor* in *dbmins.set* to *db-min.db*
USHOWUN	display on the screen the current system parameter settings and the current unit settings
USWBOTH	record the value *ffnf* into *ffornf.iun*
USWFFNF	toggle the value recorded in *ffornf.iun* between *ff* and *nf*
USWTOAMP	record the value *amp* in *ampordb.grd*
USWTODB	record the value *dB* in *ampordb.grd*
USWTODS	record the value *ds* in *dsorts.iun*
USWTOFF	record the value *ff* in *ffornf.iun*
USWTONF	record the value *nf* in *ffornf.iun*
USWTONON	record the value *none* in *ampordb.grd*
USWTOTS	record the value *ts* in *dsorts.iun*

Table 4

List of Parameter Files Used by the Research Modules and the Data Files They Create

module	parameter files	data files
UAMP2CBD	dabd.iof	uamp2cbd.out, fort.xx[1]
UAPDACB	adab.iof	[adab.iof][2]
UCBDDAT	dabd.iof, ampordb.grd ampnorn.nf, ampnorm.ff difdiv.iun, dbmin.db dbloss.grd, iregion.nf iregion.ff	nfyamp.dat, nfyphase.dat nfxamp.dat, nfxphase.dat ffyamp.dat, ffyphase.dat ffxamp.dat, ffxphase.dat ucbddat.out
UCBDGRD	ampordb.grd, ampnorm.nf ampnorm.ff, dbloss.grd dbmin.db , dabd.iof	nfamp.grd, nfphase.grd ffamp.grd, ffphase.grd ucbdgrd.out
UDBPDACB	adab.iof	[adab.iof][2]
UDERIV	order.drv, dabd.iof, sub.grd dbmin.db	order.drv, uderiv.out, fort.xx[1] drvamp.grd, drvphase.grd
UDIF2CBD	difdiv.iun, dabd.iof	udif2cbd.out, fort.xx[1]
UDIFACBD	difdiv.iun, dabd.iof	udifacbd.out, fort.xx[1]
UDIFCBD	difdiv.iun, dabd.iof	udifcbd.out, fort.xx[1]
UDIFDB	difdiv.iun, dabd.iof, ampnorm.ff ampnorm.nf, dbmin.db	udifdb.out, ddbamp.grd, fort.xx[1] ddbphase.grd
UDIVCBD	difdiv.iun, dabd.iof, iregion.ff iregion.nf	udivcbd.out, fort.xx[1]
UDIVRBD	difdiv.iun, dabd.iof, iregion.ff iregion.nf	udivrbd.out, fort.xx[1]
UDS	sub.ds, filter.ff, scale.dz, dabd.iof	scale.dz, uds.out, fort.ud$x$[1]
UERR	dabd.iof, scale.dz	uerr.out, scale.dz, fort.xx[1]
UERRCOR	dabd.iof, scale.dz	scale.dz, uerrcor.out, fort.xx[1]
ULAPLCN	dabd.iof, dbmin.db	ulaplcn.out, lnamp.grd, lnphase.grd amp0.grd, phase0.grd
UMAKEDZ	dabd.iof, fun.dz, polydz.par perdz.par, randz.par	iregion.ff, iregion.nf, umakedz.out [dz.grd][2], fort.xx[1]
UOPNORM	difdiv.iun, dabd.iof, iregion.ff iregion.nf	uopnorm.out
UPRNCBD	dabd.iof, sub.prn	uprncbd.out
URBDGRD	dabd.iof	ffamp.grd, nfamp.grd, urbdgrd.out
URDDZ	dabd.iof, makedz.par, scale.dz	scale.dz, urdz.out, [dz.grd][2]
URDFFNF	filter.ff, dabd.iof, db.ff, db.nf	iregion.ff, iregion.nf, fort.xx[1] urdffnf.out, db.ff, fort.xx[1]

		ampnorm.ff, db.nf, ampnorm.nf
URDNFFF	filter.ff, dabd.iof, db.ff, db.nf	iregion.ff, iregion.nf, fort.xx[1]
		urdnfff.out, db.ff, fort.xx[1]
		ampnorm.ff, db.nf, ampnorm.nf
URMSCBD	dabd.iof, ampnorm.ff	db.ff, db.nf, urmscbd.out, fort.xx[1]
		ampnorm.ff, rms.dat
USUBGRD	dbloss.grd, dabd.iof, sub.grd	ffamp.grd, ffphase.grd, nfamp.grd
	ampnorm.nf, ampordb.grd	nfphase.grd, usubgrd.out
	ampnorm.ff, dbmin.db	
UTS	dabd.iof, scale.dz	uts.out, scale.dz, fort.xx[1]

[1] The DOS extension number xx added to the filename FORT is recorded in the appropriate .IUN file
[2] The brackets [*filename*] is to be understood as the contents of the *filename*. For example, the output file name is read in as a parameter from file adab.iof

Table 5

List of Parameter Files Used by the Data Management Modules

module	input file	output file
UACT0		active.iun
UACTADD0		active.iun, add.iun
UACTDDB	difdb.iun	active.iun
UACTDIF	dif.iun	active.iun
UACTDIV	div.iun	active.iun
UACTDRV	drv.iun	active.iun
UACTDS	- ERROR[1] -	active.iun
UACTEC	ec.iun	active.iun
UACTFF	ff.iun	active.iun
UACTFF0	ff.iun	active.iun
UACTNF	nf.iun	active.iun
UACTNF0	nf.iun	active.iun
UACTTS	ts.iun	active.iun
UADD0		add.iun
UADDDRV	drv.iun	add.iun
UADDDS	- ERROR[1] -	add.iun
UADDEC	ec.iun	add.iun
UADDFF	ff.iun	add.iun
UADDFF0	ff.iun	add.iun
UADDNF	nf.iun	add.iun
UADDNF0	nf.iun	add.iun
UADDTS	ts.iun	add.iun
UDCDFDV	difdiv.iun, ffornf.iun ff.iun, nf.iun	difdiv.iun
UDECFF	ff.iun	ff.iun
UDECNF	nf.iun	nf.iun
UINCFF	ff.iun	ff.iun
UINCNF	nf.iun	nf.iun
UINITUN	init.iun, data.dir	ampordb.grd, filter.ff, order.drv scale.dz, fun.dz, active.iun, add.iun amp2.iun, asci.iun, difdiv.iun dif2.iun, dif.iun, difdb.iun, div.iun drv.iun, ds.iun, dz.iun, ec.iun, err.iun ff.iun, nf.iun, rdiv.iun, ts.iun
UNCDFDV	difdiv.iun, ffornf.iun ff.iun, nf.iun	difdiv.iun
UNORM0	tempnorm.nf, tempnorm.ff ampnorm.nf, ampnorm.ff	tempnorm.nf, ampnorm.nf tempnorm.ff, ampnorm.ff

UNORM1	ampnorm.nf, ampnorm.ff tempnorm.nf, tempnorm.ff	tempnorm.nf, ampnorm.nf tempnorm.ff, ampnorm.ff
URESTFF	save.ffs	ff.iun
URESTNF	save.nfs	nf.iun
USAVEFF	ff.iun	save.ffs
USAVENF	nf.iun	save.nfs
USETDB1	dbmins.set	dbmin.db
USETDB2	dbmins.set	dbmin.db
USETDB3	dbmins.set	dbmin.db
USHOWUN	ampordb.grd, filter.ff, fun.dz ampnorm.ff, ampnorm.nf ffornf.iun, active.iun difdiv.iun, dif2.iun, dif.iun div.iun, drv.iun, ds.iun err.iun, ff.iun, nf.iun, ts.iun dz.iun, order.drv, amp2.iun rdiv.iun, difdb.iun, asci.iun scale.dz, add.iun, ec.iun	
USWBOTH		ffornf.iun
USWFFNF	ffornf.iun	ffornf.iun
USWTOAMP		ampordb.grd
USWTODB		ampordb.grd
USWTODS		dsorts.iun
USWTOFF		ffornf.iun
USWTONF		ffornf.iun
USWTONON		ampordb.grd
USWTOTS		dsorts.iun

[1] Output from module UDS is written to file FORT.DSx, where x denotes a single digit. Consequently these unit numbers do not fit into a purely integer unit numbering scheme.

Table 6

List of Symbols Used by PNFC Subroutines

0	initialization designator
1	one dimensional, subsequent operation designator, alternate procedure designator
2	two dimensional
a	"a", access, array, ascii, amplitude
b	"b", backward, binary
c	change, convert to, column, complex, constant, copy
d	data, derivative, difference, dimensional, direct (access), disk (as a storage location), double precision
e	exponential, even
f	far, field, file, forward, formated, function
g	gamma, generate
i	imaginary, imaginary part, integer
k	k (integer constant), wave number, spectrum space k
l	l (integer constant)
m	m (integer constant), maximum, minimum, minus, multiple
n	near (fresnel region), negative quantity
o	odd, or
p	parameter(s), phase, plot, plus, power, print, product, pseudo
r	read, real (single precision), real part, row
s	shift, shifted, single precision, store, sum, plural designation
t	taylor (series), times, transform
w	weight, weighted, write
x	coordinate (distance along x axis), general variable designation name change letter (to avoid conflicts)
y	coordinate (distance along y axis)
z	coordinate (distance along z axis)
as	ascii
bd	binary data
ca	complex array "a"
cb	complex array "b"
cc	complex constant
ch	character variable
cm	centimeter
cr	create
da	direct access
db	decibel
df	difference
ds	direct sum
dx	derivative with respect to x, increment in x direction

dy	derivative with respect to y, increment in y direction
dz	derivative with respect to z, increment in z direction, z error terms
ec	error correction
eq	equality
ff	far field
fp	floating point
fs	files
hd	header
hi	high
im	imaginary
ik	third index of three dimensional array
iy	first (column) index of an array
jx	second (row) index of an array
ln	logarithm
mk	make
mm	maximum/minimum
mx	maximum
nf	near field
or	or
pf	plot file
ph	phase
rc	real constant
rd	read
re	real
rz	real dz array
sm	sum
sq	square, squared
to	to
ts	taylor series
ud	update
un	unit number, "unorm"
wl	wave length
wn	wave number
xp	exponential
amp	amplitude
add	add
ary	array
asc	ascii
box	box
cos	cosine
chk	check
cnt	center
dat	file extension designation for two-dimensional plot files
dif	difference

div	divide
dnf	derivative of near field
dot	dot product (of two vectors)
drv	derivative
end	end
err	error, error field
exp	exponent
fbt	forward/backward transform
fft	fast fourier transform
fil	file
flt	filter
fun	function
get	get
grd	file extension designation for perspective-plot files
hst	history
iof	input/output file
img	imaginary
inp	input
ins	insertion
int	integrate
iun	integer unit number
leq	less than or equal
log	logarithm
mak	make
mul	multiply
mod	modulate, modulated by
out	output
par	parameter
per	periodic
pff	psuedo far field
plt	plot
ply	polynomial
prg	program
prn	print
pws	plane-wave spectrum
scl	scale
set	set, setup
sft	shift
sin	sine
str	store
aray	array
bndr	boundry
char	character
gama	gamma

gamm	gamma
file	file
fils	files
find	find
fltr	filter
func	function
grid	grid (coordinate grid)
init	initalize
limt	limit
loss	loss
make	make
mult	multiply
mess	message
prnt	print
rndm	random
swap	swap
unit	unit
gamma	gamma
polyn	polynomial
ratio	ratio
const	constant
lngth	length
range	range
timer	
gather	
laplcan	Laplacian

TABLE 7

List of Subroutines of the PNFC

ACPCFFD	introduce Amplitude Change and Phase Change to Far-Field Data
ACPCNFD	introduce Amplitute Change and Phase Change to Near-Field Data
ADABIOF	get the Input Output File for Ascii to Direct Access Binary routines
ADABPAR	read the PARameters for the Ascii to Direct Access Binary conversion
ADDBOX	ADD a BOX function of amplitude EPS to the complex CDATA
AMPDIF2	form a real array equal to the difference between the absolute values of two complex arrays
APDSET1	convert to A-P prior to taking Difference between SElected columns or rows from Two 2-dimensional arrays created from 3-dimensional array
CABD	Convert two Ascii data sets to self-documented complex Binary Data set
CABDIOF	get the Input Output File for the Conversion of Ascii to Binary Data
CABDPAR	read PARameters for the Conversion of Ascii to Binary Data routines
CADACB	Convert 2 Ascii data sets to self-document Direct-Access Complex Binary
CADD1	Complex ADDition of 1 dimensional complex arrays
CADD2	Complex ADDition of 2 dimensional complex arrays
CAEIPH2	Complex Array times E to power I times Phase, phase 2 dim. real array
CAEIPHC	Complex Array times Exponential I times Phase Constant
CAPCCD1	Complex Array Plus Complex Constant, 1 Dimensional
CAPRI	Convert Amplitude-Phase format complex numbers to Real-Imaginary format
CAPRI1	Convert A-P format complex numbers to R-I format, 1 dimensional
CAPRNT	find max and min values of a Complex Amplitude-Phase array and pRiNT
CARAYMX2	find the Complex ARrAY MaXimum, for a 2 dimensional array
CARCBD2	Complex Array plus Real weight times Complex array B, dbl. precision
CATOCB2	copy a Complex array CA TO CB, for 2 dimensional arrays
CBDTODB	read a Complex Binary data set, converting from R-I TO A-P, amp in dB
CCA2B1	copy a Complex Column of data from array A 2 dim. to array B 1 dim.
CDFSET1	form Complex DiFerence of SElected columns or rows of Two arrays formed from a 3-dimensional's sub-arrays, then convert from R-I to A-P
CDIF1	Complex DIFerence of 1 dimensional complex arrays
CDIF2	Complex DIFerence of 2 dimensional complex arrays
CDIVDS2	Complex DIVision of single precision array by Double precision array
CDOT	Complex Dot product returned in complex ANSwer
CGATHER	sequentially copy regularly spaced elements of one array to another
CHKEQFP	CHecK for EQuality between two Floating Point numbers: stop if unequal
CHKEQI	CHecK for EQuality between two Integer numbers: stop if unequal
CHKLEQI	CHecK if Less than or EQual relates two Integer numbers: stop otherwise
CHKPAR0	CHecK if one integer is less than or equal to another: stop otherwise
CHKPAR1	CHecK if two integers are less than or equal to two others: else stop
CHKPAR2	CHecK if two integers are less than or equal to two others: else stop

CHLNGTH	determine number of CHaracters up to a blank in a character variable
CIMGSTR	a Complex array's IMaGinary part is SToRed in a real array
CINIT1	Complex INITialization of 1 dimenstional array with a complex constant
CINIT2	Complex INITialization of 2 dimenstional array with a complex constant
CIRCFLT	CIRCular FiLTer of CDATA
CMULDS2	Complex MULtiplication of a Double precision complex array by a Single precision complex array in 2 dimensions
CMULRD2	Complex MULtiplication of Real Double precision array by complex array
CMULT1	Complex MULTiplication of complex arrays in 1 dimension
CMULT2	Complex MULTiplication of complex arrays in 2 dimensions
CMULTR2	Complex MULTiplication of Real array by complex array in 2 dimensions
CNIMCC1	add a Complex Constant to Negative Imaginary part of Complex array
CNTCACB	CeNTer the data in a zero padded array by copying it from CA to CB
CONST	return the constant unity for the value of a function
CONSTAX	return the constant unity for the value of a function
COS2	calculate the function COSine squared of x
COS3	calculate the function COSine cubed of x
COS4	calculate the function COSine of x raised to the fourth power
COSAX	calculate the function COS(A*X)
COSAX2	calculate the function COSine squared of A*X
COSX	calculate the function COS(X)
CRA2B1	copy a Complex Row of data from array A 2 dim. to array B 1 dim.
CRATIO2	Complex RATIO of two arrays in 2 dimensions
CRIAP	Complex Real-Imaginary format numbers to Amplitude-Phase format, 2 dim.
CRIAP1	Complex Real-Imaginary format numbers to Amplitude-Phase format, 1 dim.
CRNFERR	CReate ERRor Near Field via multiple Fourier transforms for all z dist.
CSMWCP1	Complex SuM of Weighted Complex Product real and complex arrays, 1 dim.
CSMWCP2	Complex SuM of Weighted Complex Product real and complex arrays, 2 dim.
CSUM1	Complex SUM in 1 dimension
CSUM2	Complex SUM in 2 dimensions
CSUMCP1	Complex SUM of Complex and real array Products, 1 dimensional result
CSUMCP2	Complex SUM of Complex and real array Products, 2 dimensional result
CSUMIK2	Complex SUM of complex array over 3rd dimension, 2 dimensional result
CSUMRW1	Complex SUM of a Real Weight times a complex array in 1 dimension
CSUMRW2	Complex SUM of a Real Weight times a complex array in 2 dimensions
CSWCPE2	Weighted Complex Sum of ComPlex times real to Exponent in 2 dimensions
CXPCLOG	add a Complex EXPonent array to the Complex LOGarithm of array CDATA
DABDIOF	get the Direct Access Binary Data Input Output File
DABDPAR	obtain Direct Access Binary Data processing PARameters
DATORB1	copy a Double precision Array TO a Real array in 1 dimension
DB1	convert real part of amplitude-phase-format complex array to dB, 1 dim.
DB2	convert real part of amplitude-phase-format complex array to dB, 2 dim.
DCLN2	Double-precision Complex Logarithm of complex array in 2 dimensions
DNFDX	Derivative of Near Field with respect to X, a scan plane coordinate

DNFDY	Derivative of Near Field with respect to Y, a scan plane coordinate
DNFDZ	multiple Derivatives of Near Field with respect to Z, coordinate variable orthogonal to the scan plane
DNFDZE	Derivatives of Near Field with respect to Z, Even orders
DNFDZO	Derivatives of Near Field with respect to Z, Odd orders
DSPWS	Direct Sum of Plane Wave Spectrum with log of spectrum as input data
ECEXP	compute E to a Complex EXPonential
EIGAMAZ	create the array E raised to the power i times GAMma times Z
ERRMESS	print a set of ERRor MESSages in accord with the origin of call
FAXSBYS	generate an array equal to a Function of A times Shifted coordinate X multiplied by a function of B times Shifted coordinate Y
FBTIOF	get Input Output File for real to complex binary data read routines
FBTPAR	read PARameters for real to complex binary data read routines
FFNF	given a Far-Field, compute the corresponding Near-Field using the FFT
FFNFZXY	given Far-Field, compute variable z-distance Near-Field via direct sum
FFPFF	given a Far-Field, obtain the Pseudo Far-Field
FFTFFT	Fast Fourier Transform followed by inverse Fast Fourier Transform
FILSIOF	get the Input Output File for reading designated data output file names
FILSPAR	read the names of designated data-output files
FINDEND	skip to the end of a file
FLTEIGZ	filter an array according to criteria regarding array GAMMA2
FLTGAMA	set array GAMMA2 to zero whenever its value is less than some constant
FLTLIMT	obtain data-point-spacing criteria for limiting plane-wave spectrum
FLTPWSG	FiLTer sum of logarithm of Plane-Wave Spectrum plus i Gamma z
FLTRHIK	FiLTeR HIgh far-field frequencies in K-space in near-field calculation; change from near-field plane at Z0 to near-field plane at Z1
FRBW	perform a Formated Read and a Binary Write of a Real DATAset
FRGRD	perform a Formated Read of a .GRD DATAset with conversion from decibels to amplitude
FRRAD	perform a FoRmated Read of a Real Ascii Dataset
FRRADHD	perform a FoRmated Read of a Real Ascii Dataset with HeaDer information
FUNAXBY	generate an array equal to a FUNction of A times coordinate variable X multiplied by a function of B times coordinate variable Y
FUNCSCL	calculate a SCaLed grid-increment
FUNCXY	generate an array equal to a FUNCtion in X times a function in Y
FWDCRA1	Formated Write of Double precision array Converted to Real Array, 1 dim.
FWRAD1	Formated Write of Real Ascii Data in 1 dimension, self documented
FWRAD2	Formated Write of Real Ascii Data in 2 dimensions, self documented
GAMMASQ	calculate real double precision array GAMMA**2
GETFILE	obtain the next file name from a file-name array
GETWN	given a frequency, calculate a wavenumber
GRDDACB	read amp and phase ,GRD files and write out a Direct-Access Complex Binary Data file
GRID	set up a GRID using single precision along a single axis

GRIDD	set up a GRID using Double precision along a single axis
HSTDFDB	append file information to HiSTory file from program UDIFDB
HSTDIF	append file information to HiSTory file from program UDIFCBD
HSTDIV	append file information to HiSTory file from program UDIVCBD
HSTDRV	append file information to HiSTory file from program UDERIV
HSTDS	append file information to HiSTory file from program UDS
HSTEC	append file information to HiSTory file from program UERRCOR
HSTFFNF	append file information to HiSTory file from program URDFFNF
HSTMKDZ	append file information to HiSTory file from program UMAKEDZ
HSTNFFF	append file information to HiSTory file from program URDNFFF
HSTTS	append file information to HiSTory file from program UTS
HSTUN0	append file information to HiSTory file from program UNORM0
HSTUN1	append file information to HiSTory file from program UNORM1
INPDABP	INPut Direct-Access Binary data-processing Parameters from a file
INPDACB	INPut Direct-Access Complex Binary data and parameters from a file
INPDRVP	INPut a subset of the direct-access pinary file for program UDERIV
INPDZP	INPut a subset of the direct-access binary file for program UMAKEDZ
INPFBT	INPut complex binary data by reading two real ascii files
INPFFP	INPut Far-Field Parameter subset of direct-access binary data file
INPFFP0	INPut Far-Field Parameters and data from direct-access binary file
INPFILS	get the file containing a list of file names and INPut the FILe nameS
INPGRDP	INPut a subset of the direct-access binary file for program UCBDGRD
INPNFP	INPut Near-Field Parameter subset of direct-access binary data file
INPNFP0	INPut Near-Field Parameters and data from direct-access binary file
INPRCBD	INPut two Real data sets into a Complex Binary Data array
INPTSP	INPut a subset of the direct-access binary file for program UTS
INSLOSS	convert INSertion LOSS from decibels to amplitude and scale data array
INTNF3	INTegral of Near Field with respect to Z
IUNIT	function call to increment by 1 the current Integer UNIT number
IYJXCNT	determine the distance between the CeNTers of a line and a line segment along X or Y axes
LAPLCAN	calculate the LAPLaCiAN of a near field; store in adjacent location
LNPPWS	add i gamma times z to Logarithm of Plane-Wave Spectrum and filter
MAKEDZ	MAKE a function DZ, which is a function of X and Y
MAKPFF	MAKe a Pseudo Far Field equal to the Fourier transform of box function
MDARB1	copy two Double precision Arrays to Real arrays in 1 dimension
MDNFDX	Multiple Derivatives of the Near Field with respect to X
MDNFDY	Multiple Derivatives of the Near Field with respect to Y
MDNFDZ	Multiple Derivatives of the Near Field with respect to Z
MDNFDZE	Multiple derivatives of the Near Field with respect to Z, Even orders
MDNFDZO	Multiple derivatives of the Near Field with respect to Z, Odd orders
MIGAMMZ	calculate the array Minus i times GAMMa times Z
MKGAMMA	MaKe (create) the arrays GAMMA**2 and KY, KX
MKPERDZ	MaKe a function DZ, which is a PERmutated function of X and Y

MKPLYDZ	MaKe a PoLYnomial function DZ of X and Y
MMICA	get the Minimum and Maximum of the Imaginary part of a Complex Array
MMRCA	get the Minimum and Maximum of the Real part of a Complex Array
NF	create a complex array equal to a FUNCtion in X times a function in Y
NFFF	given a Near-Field, compute the corresponding Far-Field
NFMODX	MODulate a complex array with a function of X
NFMODY	MODulate a complex array with a function of Y
NFPFF	given a Near-Field, obtain the corresponding Psudo Far-Field
OUTASC	convert a complex array of R-I format to A-P format and OUTput the result as two ASCii data files
OUTDACB	OUTput a Direct Access Complex Binary data file
OUTDPS	OUTput a complex array to Disk storage, the Printout file, and/or a Storage array
OUTGRD	setup to OUTput amplitude and phase of a complex array to a .GRD file
OUTPFS0	get 4 file names and OUTput 4 arrays as .PLT files
OUTPFS1	OUTput 4 amplitude and phase arrays formed from a column and a row of a complex array as .PLT Files
OUTRGRD	OUTput a Real array to a .GRD file for input to 'surfer' graphics codes
PCCRGRD	for a Complex array, Print a Column and/or Row of amplitude and phase values and then output the amplitude and pahse data to two .GRD files
PERFUNC	dummy routine to select the name of a FUNCtion to be evaluated
PFCORR	obtain a Column OR Row of data and store the real or imaginary part for submital to a Plot File
PFCRAP	obtain one dim. Amp. and Phase arrays from a Column and a Row of a complex array for submital to a Plot File
PFFFF	given a Psudo Far-Field, obtain the corresponding Far-Field
PFFNF	given a Psudo Far-Field, obtain the corresponding Near-Field
PFREIM	obtain Real or Imaginary part of 1 dim. complex array and copy to 2nd column of a real 2 dim. array
PFSET	Plot File SETup: collect column or row data from a complex array and convert to R-I format (or A-P format)
PLTFILE	output a real .PLT file for PLoTing multiple two-dimensional curves
POLYN	function call to sum a POLYNomial of a single variable
POLYNXY	calculate an array equal to a POLYNomial function of X added to a polynomial function of Y
PPWSNF	given log of Plane-Wave Spectrum, calculate Near Field via direct sum
PRDCTC2	raise to a Power a Real Double precision Column array; then Times a Complex array: 2 dimensional result
PRDRTC2	raise to a Power a Real Double precision Row array; then Times a Complex array: 2 dimensional result
PRDTC2	raise to a Power a Real 2 dimensional Double precision array: then Times a Complex array: 2 dimensional result
PRNCORR	PRiNt out the amplitude and phase of a Column OR a Row of complex array

PRNPLT	PRiNt and output to 4 .PLT files the amp. and phase of a column and a row of a complex array
PRNRCOR	PRiNt out a Real-array's Column OR Row
PRNTC1D	PRiNT the maximum amplitude of a Complex array and one column of Data
PRNTR1D	PRiNT maximum and minimum values of a real array and one column of Data
RADDRC2	Real array ADDed to a Real Constant, 2 dimensional
RANGED	lower and upper RANGE values of a Double precision 1 dimensional array
RANGES	lower and upper RANGE values of a Single precision 1 dimensional array
RARYMM2	get a Real ARraY's Maximum and Minimum values, 2 dimensional array
RCA2B1	copy a Real Column of data from array A 2 dim. to array B 1 dim.
RCBD2	Read in a Real binary dataset and store as a Complex Binary Dataset in a 2 dimensional array
RCBDIOF	get the Input Output File name for inputing 2 Real Binary data sets into a Complex array
RCBDPAR	Read PARameters and file names for inputing 2 Real Binary data sets into a Complex array
RCBDSET	Set up to read two Real Binary data sets into a Complex array
RDCBD1	Read in a Real Binary data set into a Complex 1 dimensional array
RDCBD2	ReaD a Complex Binary data set into a 2 dimensional array
RDDABP	ReaD a Direct Access Binary data set for the set Parameters
RDDACBD	ReaD a Direct Access Complex Binary Data set
RDIF2	get the Real Difference of two dimensional real arrays
RDOT	for two Real arrays form their DOT product
RDRBD2	ReaD a Real Binary data set into a 2 dimensional array
REARANG	amplitude, phase, distance correction and swap to obtain far-field data
RINIT1	Real INITialization of 1 dimenstional array with a real constant
RMULT2	Real MULTiplication of real arrays in 2 dimensions
RNDM	function call to return a RANDoM number
RNDMDZ	calculate a real array DZ all of whose elements are RANDoMly obtained
RRA2B1	copy a Real Row of data from array A 2 dim. to array B 1 dim.
RZTORC1	Raise a 1 dimensional array TO a Real Constant power
RZTORC2	Raise a 2 dimensional array TO a Real Constant power
SCLCC1	SCaLe by a Complex Constant a Complex array in 1 dimension
SCLCC2	SCaLe by a Complex Constant a Complex array in 2 dimension
SCLRC1	SCaLe by a Real Constant a Complex array in 1 dimension
SCLRR2	SCaLe by a Real Constant a Real array in 1 dimension
SETBNDR	SET the BouNDaRy regions of complex array CDATA to complex constant CC
SETFILS	SET up to read a list of FILES containing formated-output file names
SETFIOF	get the Input Output File for reading formated-data-output file names
SETFPAR	read the names of designated formated-data-output files
SETTSZ	SET up the necessary arrays for TayLoR series in Z calculations
SFTCACB	ShiFT the location of Complex data in zero padded array to array center

SFTRARB	ShiFT the location of Real data in zero padded array to array center
SIN4X	calculate the function SINe of X raised to the fourth power
SWAP	SWitch begining to end Array-element Positions of both rows and columns
TIMER	store system TIME on first call, return time difference on second call
TIMERS	multiple TIME initilizations, time differences returned on second call
TODAY	write current date to screen
TSZK	add TayLoR Series (in Z) term of order K to existing CDATA array
TSZK1	obtain sum of Kth Taylor Series term with CDATA then output partial sum
TSZSLM	Taylor Series in Z Summation from Low order to Max order added to CDATA
TSZSLM0	initialize Taylor Series in Z with terms Summed from Low to Max order
TSZSLM1	Taylor Series in Z Sum from Low to Max order, each partial sum output
UDASCUN	UpDate file record of ASCii output-file UNits
UDDSIUN	UpDate Direct-Sum Integer-UNit-number file of output-name extensions
UDDZIUN	UpDate file record of DZ file-name extensions that have been created
UDFFIUN	UpDate file record of existing Far-Field file-name extensions
UDFIUN	UpDate File-Index Unit Number file
UDNFIUN	UpDate file record of existing Near-Field file-name extensions
WCBD1	Write an unformatted Complex Binary Data set, which is 1 dimensional
WCBD2	Write an unformatted Complex Binary Data set, which is 2 dimensional
WDACBD	Write a Direct Access Complex Binary Dataset which is self-documented
WLTOCM	given a frequency, convert from Wave-Lengths to CentiMeters
WRBD2	Write an unformatted Real Binary Data set, which is 2 dimensional
WRCHKF	WRite a CHecK list of parameters to the standard print file
XCHAR	eXpress an integer modulus 100 as a CHARacter variable
XSCHAR	eXpress an integer modulus 10 as a Single CHARacter variable
XYGRIDS	set up both X and Y GRIDs using Single precision

Appendix A

Creating the Original Direct Access Binary Dataset

Two modules are provided for inputing ascii data files to create direct-access complex binary datasets. The module UAPDACB reads in two ascii files, one containing amplitude data and one containing phase data. For these files, the data in each column precedes the data in each succeeding column. The required structure of these ascii files is seen by inspection of subroutine FRRAD or subroutine FRRADHD (the latter assumes that a 120 character HeaDer preceedes the ascii data). Alternately, module UDBPDACB is used to read two ascii files, one containing amplitude data expressed in decibels and one containing phase data. Both files are assumed to have been set up as .GRD files suitable for input to the system plot package. For these files, the data in each row preceeds the data in each succeeding row. The structure of these ascii files is seen by inspection of subroutine FRGRD. The parameters associated with the binary datasets created by these modules are obtained from a user-supplied parameter file, which also contains the names of the two ascii input files, a format specification for the ascii input data, and the name of the binary output file. This user-supplied parameter file is specified in the file ADAB.IOF. The parameter file entries are seen by inspection of subroutine ADABPAR, which reads the file. The parameters occupying the first seven records of the direct-access binary file are defined in the following table:

List of Parameters Included in the Direct Access Binary Datasets

LENGTH	the LENGTH of each record in the file, nominally equal to 8*NY (required for Direct Access files)
FFORNF	data type specifier distinguishes between Far-Field OR Near-Field data
LABEL	character variable of up to 120 characters identifying the file
NY, NX	the number of respective columns and rows in the complex data array
DY, DX	incremental data point spacing in near-field plane for Y and X axes
FREQ	operating frequency expressed in gigahertz
Z0	z-axis distance in centimeters to the near-field measurement plane

Appendix B

System Initialization

At the beginning of any research project the system has to be initialized to properly set the the system parameters and the far-field and near-field unit numbers. This is accomplished by executing the module UINITUN, which will write the following table to the screen:

```
THE INITIAL SETTINGS are:

      ampordb.grd: ampordb=:              dB
            filter.ff: cksqrd=:   0.0000000E+00
     order.drv: idrvinc,iorder=:               1                 0
   scale.dz: scalinc,dzscale=:   0.0000000E+00         0.1000000
             fun.dz: funtype=:             per
          active.iun: iactive=:               0
              add.iun: iadd=:               0
         amp2.iun: iunamp2=:               0
            asci.iun: iunasci=:               7
          difdiv.iun: idifdiv=:               1
            dif2.iun: iundif2=:               0
              dif.iun: iundif=:               0
         difdb.iun: iundfdb=:               0
              div.iun: iundiv=:               0
              drv.iun: iundrv=:               0
      ds.iun: iunds0,iundsl=:              -1                 0
                dz.iun: iundz=:              61
               ec.iun: iunec=:               0
              err.iun: iunerr=:               0
                 ff.iun: iunff=:              60
                nf.iun: iunnf=:              40
          rdiv.iun: iunrdiv=:               0
                ts.iun: iunts=:               0

STOP: UINITUN: normal termination
```

In the above table the first entries on each line give the name of the file where the information is recorded, the second entries give the name of the fortran variable(s) in the modules that contain the value(s), which are shown last. The key abbreviations in the file names and variable names can be deciphered by consulting Table 1. For example, *iunasci* specifies the *current* setting of the ascii output unit number, and *iundz* specifies the unit number of the *dz* dataset. Many of the unit numbers are set to 0, simply signifying that no data has yet been created for these fields. There are a few remaining variables included in the table that have special meanings. These are defined below:

ampordb	.GRD files will be created in dB; it can also be set to *amp*
cksqrd	filter limit for truncating plane-wave spectrum
idrivinc	increment by which *iorder* is increased whenever *order.drv* is accessed by module UDERIV
iorder	order of derivative calculated by module UDERIV
funtype	TYPE of FUNction used by module UMAKEDZ to create incremental scalar field. It may have the value *per* (periodic), *poly* (polynomial), or *ran* (random) function
iunds0	initial value of *iundsl*: set to -1 when first initialized, thereafter equal to 0
iundsl	single digit used as the last character in the filename extension, exclusively used by module UDS

Appendix C

System Status Reports

After the execution of any module one can request a system status report to examine the system parameter settings and the unit number settings. This is accomplished by executing USHOWUN. One might do this to check the sequence of executions for correctness and to decide what data management steps one needs to take to access the next dataset needed to continue the research correctly. When USHOWUN is executed after URDNFFF and UMAKEDZ have been executed only once the following table is displayed:

```
THE CURRENT SETTINGS are:

      ampordb.grd:               dB
         filter.ff:   0.0000000E+00
         order.drv:               1              0
          scale.dz:   0.0000000E+00      0.2000000
      ampff,invff:         1987.822  5.0306314E-04
      ampnf,invnf:         1.059250  9.4406420E-01
           ffornf:               ff
           fun.dz:              per
       active.iun:                0
          add.iun:                0
         amp2.iun:                0
         asci.iun:                7              9
       inc difdiv:                1
         dif2.iun:                0
          dif.iun:                0
        difdb.iun:                0
          div.iun:                0
          drv.iun:                0
           ds.iun:               -1              0
           dz.iun:               61             61
           ec.iun:                0
          err.iun:                0
           ff.iun:               60             60
           nf.iun:               40             40
         rdiv.iun:                0
           ts.iun:                0

STOP: USHOWUN: unit status report complete
```

Most of features and entries in the above table have been explained in Appendix A. Here, however, some of the entries show 2 unit numbers. The combinations of 2 equal unit numbers signifies that the modules writing these unit numbers have only been executed once, thereby making the initial unit numbers, as defined in Appendix A, the *current* unit numbers.

After creating all the datasets required by the error correction research problem (see Section 4), USHOWUN can be executed to get an overview of the system status. The output table appears as below:

```
THE CURRENT SETTINGS are:

      ampordb.grd:              dB
         filter.ff:  0.0000000E+00
         order.drv:              1               0
          scale.dz:  0.0000000E+00       0.2000000
       ampff,invff:       1987.822  5.0306314E-04
       ampnf,invnf:       1.059250  9.4406420E-01
           ffornf:              ff
           fun.dz:             per
       active.iun:              59
          add.iun:              60
         amp2.iun:               0
         asci.iun:               7              17
       inc difdiv:               1
         dif2.iun:               0
          dif.iun:               0
        difdb.iun:               0
          div.iun:               0
          drv.iun:               0
           ds.iun:              -1               0
           dz.iun:              61              61
           ec.iun:              42
          err.iun:               0
           ff.iun:              60              56
           nf.iun:              40              44
         rdiv.iun:               0
           ts.iun:              41

STOP: USHOWUN: unit status report complete
```

Now we see that two unequal unit numbers appear in some of the entries. These indicate the range of unit numbers for the particular type of field, (*ff* or *nf*), that *exist* after repeated executions of the various modules. The first unit number indicates the initial unit number created and the last number indicates the *current* value of the unit number. The dataset referred to by the *current* value of the unit number will be automatically accessed if the value in *active.iun* is 0. In addition, all special types of near fields that have been created during the course of the research are recorded in their respective unit number files. For example, the entry under *ts.iun* is 41, meaning that the dataset with filename *fort.41* contains the error-contaminated near field that was created using the Taylor series method.

Appendix D

An index of PNFC subroutines and the calling sequences in these routines.

ACPCFFD	
ACPCNFD	
ADABIOF	: chlngth
ADABPAR	: errmess
ADDBOX	
AMPDIF2	
APDSET1	: cca2b1 cra2b1 criap1 capri1 cdif1
CABD	: cabdiof cabdpar frbw
CABDIOF	: chlngth
CABDPAR	: errmess
CADACB	: adabiof adabpar frradhd capri wdacbd
CADD1	
CADD2	
CAEIPH2	
CAEIPHC	
CAPCCD1	
CAPRI	
CAPRI1	
CAPRNT	: mmrca prntr1d mmica
CARAYMX2	
CARCBD2	
CATOCB2	
CBDTODB	: rdcbd2 criap1 db1
CCA2B1	
CDFSET1	: cca2b1 cra2b1 cdif1 criap1 capri1
CDIF1	
CDIF2	
CDIVDS2	
CDOT	
CGATHER	
CHKEQFP	
CHKEQI	
CHKLEQI	
CHKPAR0	
CHKPAR1	
CHKPAR2	
CHLNGTH	
CIMGSTR	
CINIT1	
CINIT2	

CIRCFLT	
CMULDS2	
CMULRD2	
CMULT1	
CMULT2	
CMULTR2	
CNIMCC1	
CNTCACB	: iyjxcnt sftcacb
CONST	
CONSTAX	
COS2	
COS3	
COS4	
COSAX	
COSAX2	
COSX	
CRA2B1	
CRATIO2	
CRIAP	
CRIAP1	
CRNFERR	: nfpff catocb2 ffnf
CSMWCP1	
CSMWCP2	
CSUM1	
CSUM2	
CSUMCP1	
CSUMCP2	
CSUMIK2	
CSUMRW1	
CSUMRW2	
CSWCPE2	
CXPCLOG	
DABDIOF	: chlngth
DABDPAR	: chkpar0 chkpar1
DATORB1	
DB1	
DB2	
DCLN2	
DNFDX	: catocb2 prdrtc2 sclcc1 fourt acpcnfd swap
DNFDY	: catocb2 prdctc2 sclcc1 fourt acpcnfd swap
DNFDZ	: dnfdze cmulds2
DNFDZE	: catocb2 prdrtc2 fourt acpcnfd swap sclrc1
DNFDZO	: migammz cmulds2 dnfdze

DSPWS	
ECEXP	
EIGAMAZ	
ERRMESS	
FAXSBYS	: fx fy (unspecified functions)
FBTIOF	: chlngth
FBTPAR	
FFNF	: ffpff pffnf
FFNFZXY	: dcln2 migammz ppwsnf sclcc2
FFPFF	: eigamaz fltlimt flteigz cmulds2
FFTFFT	: sclrr2 fourt
FILSIOF	
FILSPAR	
FINDEND	
FLTEIGZ	
FLTGAMA	
FLTLIMT	
FLTPWSG	: fltlimt
FLTRHIK	: nfff ffnf
FRBW	
FRGRD	
FRRAD	
FRRADHD	
FUNAXBY	: fx fy (unspecified functions)
FUNCSCL	
FUNCXY	: fx fy (unspecified functions)
FWDCRA1	: datorb1 fwrad1
FWRAD1	
FWRAD2	
GAMMASQ	
GETFILE	
GETWN	
GRDDACB	: adabiof adabpar frgrd capri wdacbd
GRID	
GRIDD	
HSTDFDB	: findend
HSTDIF	: findend
HSTDIV	: findend
HSTDRV	: findend
HSTDS	: findend

HSTEC	: findend
HSTFFNF	: findend
HSTMKDZ	: findend
HSTNFFF	: findend
HSTTS	: findend
HSTUN0	: findend
HSTUN1	: findend
INPDABP	: dabdiof dabdpar rddabp getwn wrchkf
INPDACB	: dabdiof dabdpar rddacbd getwn wrchkf
INPDRVP	: dabdiof dabdpar rddabp
INPDZP	: dabdiof dabdpar rddabp
INPFBT	: fbtiof fbtpar rcbdset
INPFFP	: dabdiof dabdpar rddabp
INPFFP0	: dabdiof dabdpar rddacbd
INPFILS	: filsiof filspar
INPGRDP	: dabdiof dabdpar rddabp
INPNFP	: dabdiof dabdpar rddabp
INPNFP0	: dabdiof dabdpar rddacbd
INPRCBD	: rcbdiof rcbdpar rcbdset
INPTSP	: dabdiof dabdpar rddabp
INSLOSS	: sclcc1
INTNF3	: migammz fourt swap cdivds2 acpcffd acpcnfd
IUNIT	
IYJXCNT	
LAPLCAN	: nfpff dnfdze dnfdx cadd2 dnfdy pffnf
LNPPWS	: carcbd2 fltpwsg
MAKEDZ	: grid funaxby sclrr2
MAKPFF	: cinit1 addbox nfpff
MDARB1	: datorb1
MDNFDX	: dnfdx
MDNFDY	: dnfdy
MDNFDZ	: mdnfdze mdnfdzo
MDNFDZE	: dnfdze
MDNFDZO	: cmulds2 mdnfdze
MIGAMMZ	
MKGAMMA	: gridd gammasq
MKPERDZ	: grid faxsbys rarymm2 sclrr2
MKPLYDZ	: grid polynxy rarymm2 sclrr2
MMICA	
MMRCA	

NF	
NFFF	: nfpff pffff
NFMODX	
NFMODY	
NFPFF	: setbndr fourt swap acpcffd
OUTASC	: fwrad2
OUTDACB	: setfils wdacbd
OUTDPS	: prncorr catocb2 wcbd2
OUTGRD	: criap db1 cnimcc1 mmrca getfile outrgrd mmica
OUTPFS0	: getfile pltfile
OUTPFS1	: pfcrap outpfs0
OUTRGRD	
PCCRGRD	: caraymx2 prncorr outgrd
PERFUNC	: cosax2, etc.
PFCORR	: pfset pfreim
PFCRAP	: pfcorr pfreim
PFFFF	: migammz cxpclog ecexp
PFFNF	: fourt acpcnfd swap
PFREIM	
PFSET	: cca2b1 cra2b1 criap1 capri1
PLTFILE	
POLYN	
POLYNXY	: polyn
PPWSNF	: lnppws dspws
PRDCTC2	
PRDRTC2	
PRDTC2	
PRGTSZ	: catocb2 ffnf tszSLM0
PRNCORR	: cca2b1 criap1 cra2b1
PRNPLT	: prncorr outpfs1
PRNRCOR	: rca2b1 rra2b1
PRNTC1D	: caraymx2
PRNTR1D	
RADDRC2	
RANGED	
RANGES	
RARYMM2	
RCA2B1	
RCBD2	: errmess
RCBDIOF	: chlngth
RCBDPAR	

RCBDSET	: rcbd2
RDCBD1	
RDCBD2	
RDDABP	: chkpar2 errmess
RDDACBD	: chkpar2 errmess
RDIF2	
RDOT	
RDRBD2	
REARANG	: swap
RINIT1	
RMULT2	
RNDM	
RNDMDZ	: rndm
RRA2B1	
RZTORC1	
RZTORC2	

SCLCC1	
SCLCC2	
SCLRC1	
SCLRR2	
SETBNDR	: cinit1
SETFILS	: setfiof setfpar
SETFIOF	
SETFPAR	
SETTSZ	: mkgamma fltgamma migammz catocb2 nfpff
SFTCACB	
SFTRARB	
SIN4X	
SWAP	

TIMER	: sec_100()
TIMERS	: sec_100()
TSZK	: dnfdzo dnfdze cswcpe2
TSZK1	: tszK outdps
TSZS03	: catocb2 nfpff dnfdzo dnfdze cswcpe2
TSZSLM	: dnfdzo dnfdze cswcpe2
TSZSLM0	: settsz tszSLM
TSZSLM1	: dnfdz cswcpe2 criap caprnt catocb2 wcbd2

UDASCUN	
UDDSIUN	
UDDZIUN	
UDFFIUN	

UDFIUN	
UDNFIUN	
WCBD1	
WCBD2	
WDACBD	
WLTOCM	
WRBD2	
WRCHKF	: wltocm
XCHAR	
XSCHAR	
XYGRIDS	: grid

NIST-114A
(REV. 3-89)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 89-3929

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
October 1989

4. TITLE AND SUBTITLE

Planar Near-Field Codes for Personal Computers

5. AUTHOR(S)
Lorant A. Muth and Richard L. Lewis

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)
U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)
Sponsored, in part, by
Air Force Guidance and Metrology Center
Newark Air Force Base, Ohio 43057
(Contract SD10-WPD-B238)

10. SUPPLEMENTARY NOTES

☐ DOCUMENT DESCRIBES A COMPUTER PROGRAM; SF-185, FIPS SOFTWARE SUMMARY, IS ATTACHED.

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

We have developed planar near-field codes, written in Fortran, to serve as a research tool in antenna metrology. We describe some of the inner workings of the codes, the data management schemes, and the structure of the input/output sections to enable scientists and programmers to use these codes effectively. The structure of the codes is seen to be open, so that a user can incorporate a new application into the package for future use with relative ease. The large number of subroutines currently in existence are briefly described, and a table showing the interdependence among these subroutines is constructed. Some basic research problems, such as transformation of a near field to the far field and probe position error correction, are carried out from start to finish, to illustrate use and effectiveness of these codes. Sample outputs are shown. The advantage of a high degree of modularization is demonstrated by the use of DOS batch files to execute Fortran modules in a desired sequence.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

antenna metrology; data management; planar near-field codes; research tool; subroutines

13. AVAILABILITY

[X] UNLIMITED
[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
[ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
[X] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
52

15. PRICE

ELECTRONIC FORM

IR 89-3930

CANCELLED

UNAVAILABLE FOR BINDING