Direct Problem Program

Rafael E. Banchs

INTRODUCTION

This report presents a brief description of the computational implementation of the direct version of the time harmonic field electric logging problem. First, the structure of the program and its four modules are presented. Then, a discussion of each module and its most important related subroutines follows. Finally, a brief description of the program user's interface is presented. For more detailed and specific information about the program, subroutines and variables see [1].

GENERAL STRUCTURE OF THE PROGRAM

The "Direct_problem" program is composed by four principal modules that perform specific functions. They are the main_program module, which is composed basically by the main program and its function is to call the subroutines in the other modules; the i/o&control module that is in charge of the input/output operations and the validation of the data and the program results; the electromagnetics module, which is in charge of solving the electromagnetic equations for the basic current element [2]; and the linear_algebra module, which models the specific logging tool by using the method of moments [3].

The input data is provided by three input files to known; "Formation", "Tool" and "Experiment". The file "Formation" provides all the information related to the earthen formation to be considered in the simulation; number of zones, radii of the boundaries and electrical conductivities of the zones. The file "Tool" provides the information related to the logging tool; size of the segment length to be used in the tool representation [2], type of tool, number of electrodes and their sizes, tool radius, currents, potential conditions, measurement, etc... And the file "Experiment" contains the information related to the simulation; frequency of operation,

control flags, number of experiments. For a more detailed information about the input file parameters see [1].

The output data is given in six different types of output files depending on the experimental results and on the type of output data requested by the user in the input file “Experiment”. The output file “Electric Log” is always generated and it contains the logging tool readings resulting from the simulation; however, in the case of a simulation which results are going to be used as input data for inversion, the file “Inversion data” is generated instead. On the other hand, the output files “ΔR(n)”, “Tool_output” and “Circuit Model” are generated only upon request and they contain the basic current element response, the tool’s current and potential distributions and the tool’s circuit analog respectively. Finally, the file “Failure_report” is generated only when any error or warning occurs during the program execution.

Figure 1 presents a diagram that summarizes the basic structure of the program, the interactions among its modules and the input/output data flow.

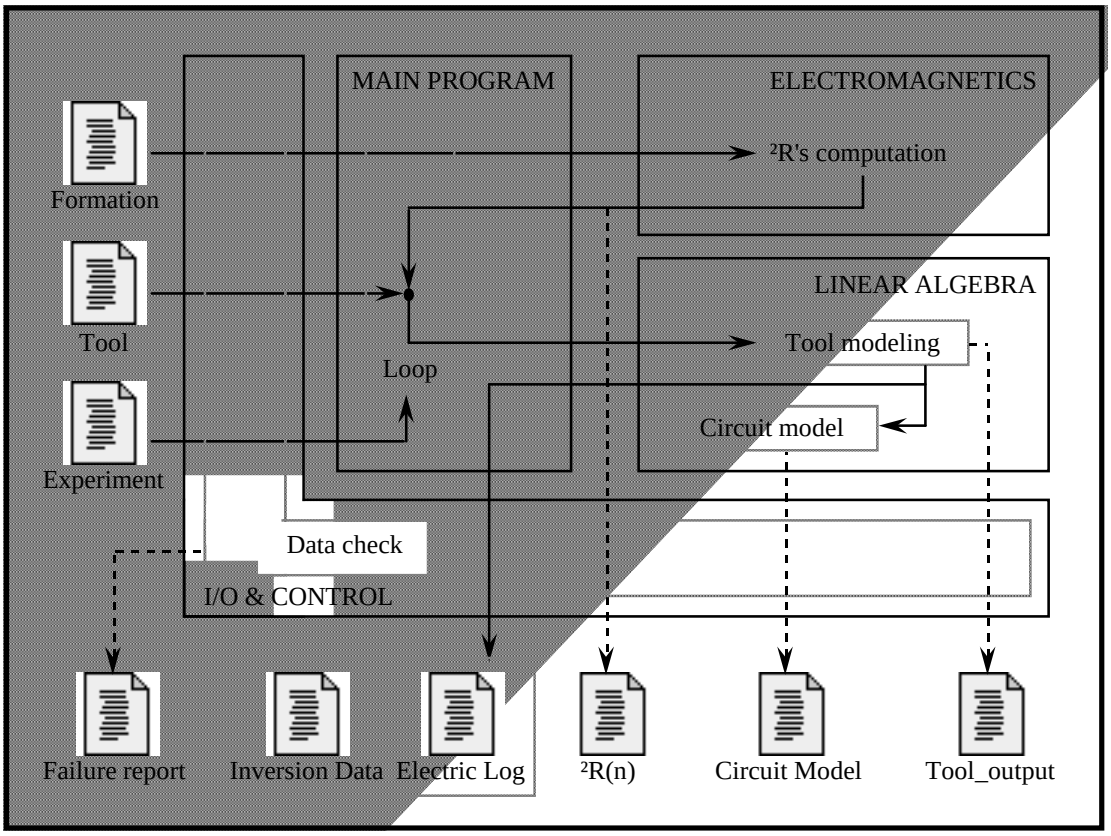

Figure 1: Basic structure of the program “Direct_problem”.

MODULE 1: MAIN_PROGRAM

The main_program module is specific of the “Direct_problem” program. It is composed by the main program, an initialization subroutine and a finishing subroutine. The main program procedure can be described as follows. First, it calls the initialization subroutine, which calls subroutines in the i/o&control module that are responsible for reading the input files, verifying the validity of the input data, initializing the control flags and experiment definitions.

Once the initialization subroutine have been executed, the main program enters the experiment loop where the simulations are performed. At the beginning of every cycle of the loop the subroutine *EXPSET* of the i/o&control module is called. This subroutine is responsible of

updating the input parameters for the simulation to be performed in that cycle. Then, the execution is transferred to the electromagnetics module that returns the array *dR*, which contains the basic current element's response. Then, the execution is passed along with *dR* to the linear_algebra module, which is responsible for the computation of the specified logging tool's response by using a linear combination of the basic current element's response. This module returns the tool reading and other output parameters in the array *result*.. Finally, the cycle ends by passing this array to the i/o&control module that stores it for future generation of the output file.

If it was requested by the user, after the loop's execution has been completed, a circuit analog for the tool and formation is computed. This is done by calling the subroutine *Impedance_Model* in the linear_algebra module.

Finally, the main program calls the finishing subroutine, which calls subroutines in the i/o&control module that are responsible for writing the output files and ending the program execution.

MODULE 2: I/O&CONTROL

The i/o&control module is also specific of the "Direct_problem" program. Its principal objective is to provide an error free program execution and data handling. The basic functions are performed by this module; loading of input data, execution failure control, and output data formatting and saving.

LOADING OF INPUT DATA

The subroutines in this category are responsible for reading the three input files "Formation", "Tool" and "Experiment", and for verifying the validity of the data contained in them. At this first level verification, the parameters are checked to be in the appropriate ranges and to be free of any kind of incompatibilities. After the first checking is performed, the experiment

parameters, control flags and common memory regions containing the model parameters are initialized. At this point a second level verification of the data is performed. In this second verification, inappropriate settings than can lead to an algorithm failure or to an invalid model representation are looked for.

EXECUTION FAILURE CONTROL

The subroutines in this category are responsible for handling errors and warning reports generated by any of the subroutines in the program. This is mainly done by the subroutine *Failure_report* that is called when a problem is detected. The calling subroutine provides and error code or a warning code that is then matched with a message number, displayed in the screen and stored in a failure log. At the end of the program execution, if any warning or error was reported, the failure log is saved in the output file “Failure_report”.

Errors are generally reported when the execution of the program cannot be physically o logically continued; for example, an overflow condition occurred, a matrix inversion could not be completed, invalid input data was provided, etc... The occurrence of an error always results in the immediate termination of the program execution. On the other hand, warnings are reported when certain conditions that can lead to an algorithm failure or to an invalid model representation occur; for example, an extremely high frequency such that the wave lengths are comparable to the segment length size, invalid measurement or tool definitions that can lead to invalid result or to a zero reading, etc...

OUTPUT DATA FORMATTING AND SAVING

The subroutines in this category are responsible for formatting the output data and saving it into the appropriate output files. As it was explained before, six different types of output files exist. Most of them are only generated upon request by using the control flags in the input file “Experiment”. On the other hand, the tool readings are always saved into either the output file “Electric Log” or “Inversion Data”. This last is only generated when input data for the inverse problem is desired; such a file also includes information about the tool and the earth formation.

MODULE 3: ELECTROMAGNETICS

The electromagnetics module is common for both the “Direct_problem” and the “Inverse_problem” programs. It constitutes the computational implementation of the procedure described in [2] by using the methodology presented in [4] and [5]. Basically, it is responsible for calculating the set of coefficients $\Delta R(n)$ that represents the solution of the time harmonic electric logging problem for the basic current element described in [2].

The principal subroutine in this module is the subroutine *dRCOMP*. All other subroutines are transparent to the main program in the sense that they are only called by subroutines inside the same module. Figure 2 presents a diagram that summarizes the procedure in *dRCOMP* and its interaction with the most important subroutines in the module.

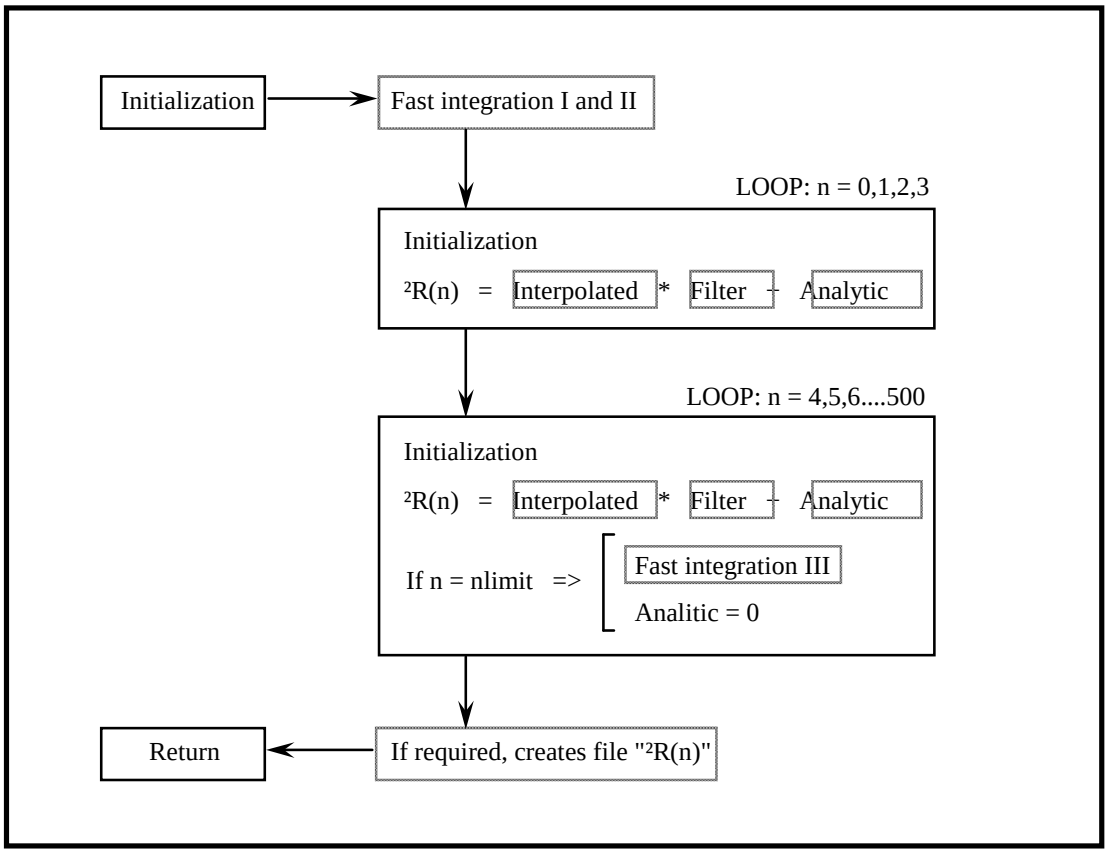

Figure 2: Main procedure in the electromagnetics module.

As it can be seen from figure 2, after the initialization of parameters, the fast integration routines are called to compute integrals I and II (dotted boxes in figure 2 represent calls to other subroutines or functions). I and II are a cosine and a sine transform integrals respectively; for a more detailed description refer to [4]. The fast integration algorithms use the function *integrand* which, accord-ing to the information contained in the common memory regions of the module, provides the appropriate values for the integration. Those values are computed by recursively using the set of functions *gkk(m,n)* , *gii(m,n)* , *fki(m,n)* and *dki(m,n)* that are described in [6].

Next, in figure 2, the computation of the values ΔR(0), ΔR(1), ΔR(2), ΔR(3) is performed. As it can be seen from [4], the ΔR(n) values are approximated by a discrete convolution between the results of the integrals and two filters. The samples to be convolved are obtained by calling the subroutines *INTERP*, which interpolates the fast integration subroutine outputs; and *FILTER*,

which provides the appropriated filter samples. The function *ANALIT* provides the analytically computed offset value resulting from the way II is computed (see details in [4]).

The initialization step in each cycle of the loop is responsible for setting the appropriated sampling period to be used in the convolutions. That is necessary because the functions to be convolved present rapid variations close to the origin; so, for those ΔR(n)'s with small values of n, very small sampling periods are required; while for larger values of n, more larger sampling periods can be used without compromising the results.

In the second loop of figure 2, computation of ΔR(n) for n = 4,5,6... 500 is performed. Here, the procedure is exactly the same, except that when certain limit value of n is reached the fast integration subroutine is called again in order to compute a new cosine transform (integral III). The new integral values, along with a different filter, are used instead of II for obtaining the remaining ΔR's. This is done to improve the performance of the original algorithm; for more details see [5].

Finally, if it was requested by the user, the computed results are saved into the output file "ΔR(n)" by calling the subroutine *dR_OUTPUT* in the i/o&control module.

MODULE 4: LINEAR_ALGEBRA

The linear_algebra module is also common for both the "Direct_problem" and the "Inverse_problem" programs. It constitutes the computational implementation of the procedure described in [3]. Basically, it uses the method of moments to approximate the logging device's response as a linear combination of the basic current element's response. So, the results generated in the electromagne-tics module are used to compute the logging device's response.

The principal subroutine in this module is *laCOMP*. All other subroutines, except *Impedance_Mo-del* that will be discussed later, are transparent to the main program. *laCOMP* is responsible for calling the appropriate tool's initialization and simulation subroutines. Five different tool simulation subroutines exist, they are the generic tool, the 2-electrodes tool (symmetric case and general case) and the laterolog 9 or LLS tool (symmetric case and general case). The generic tool subroutine is capable of solving the linear algebra problem for almost any tool configuration; but, for the same reason, it lacks of the speed and efficiency of an specific tool's algorithm. On the other hand, the other four subroutines are customized for handling specific type of tools and constitute more efficient and faster algorithms.

Figure 3 summarizes the procedure of the generic tool algorithm. The basics of the other four algorithms are mainly the same, with small variations and simplifications according to the specific tool characteristics. For a more detailed information on each of the tool algorithms, refer to [1], [2], [7] and [8].

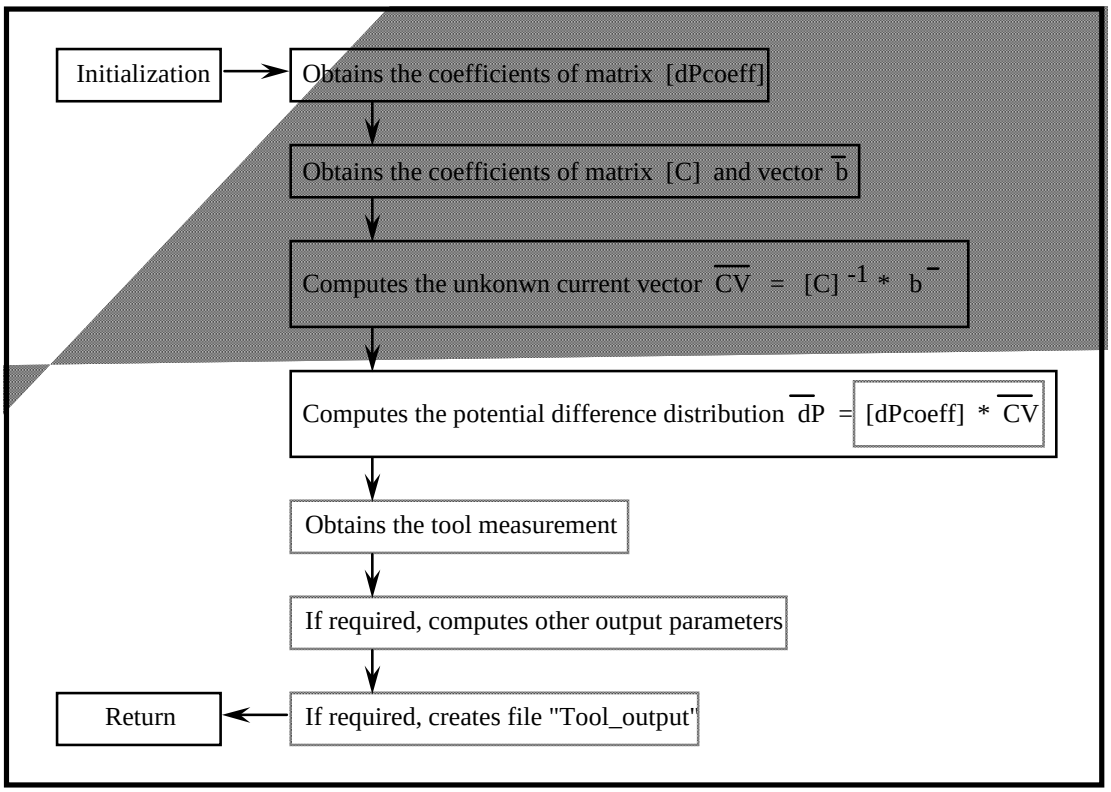

Figure 3: Linear algebra problem algorithm for the generic logging tool.

As it can be seen from figure 3, the algorithm starts by obtaining the coefficients for two matrices [dPcoeff] and [C]. The coefficients of [dPcoeff] are computed by using the tool's configuration and the method of moments' current equations. The coefficients of [C] are obtained from the coefficients of [dPcoeff] by using the method of moments' potential equations, as well as the system equations. For details refer to [8].

Then, by inverting matrix [C], all the unknown currents in the tool model are computed, which allows the subsequent computation of the potential differences between the centers of all consecutive segments in the model. This is achieved by multiplying the matrix [dPcoeff] and the computed current vector $\overline{CV}$, as it is shown in figure 3. Once the potential differences are known, the tool measurement and any other required output parameter can be easily obtained by discrete integration. Finally, if it was requested by the user, the output file "Tool_output" containing the tool's potential and current distributions is created by calling the i/o&control module.

The other important procedure in the linear_algebra module is the subroutine *Impedance_Model* . This subroutine is directly called by the main program when the computation of a circuit analog has been requested by the user. What this subroutine basically does is to call repeatedly the generic tool algorithm while setting different configurations of the given logging device. This is done in such a way that the mutual impedances for the circuit analog can be computed from the obtained measurements. After all the impedances have been obtained, their values are saved into the output file "Circuit Model".

USER'S INTERFACE

In order to simplify the process of generating the input files and to make the using of the program easier, an interface program has been developed. The main functions of this program can be summarized as follows:

A.- To provide a friendly environment for the setting and definition of the input parameters, making data verification and corrections more simple.

B.- To offer a set of utilities for performing convenient tasks as creating backups files, changing the format of the data, printing and plotting the output data, etc...

C.- To provide a common platform for the use of the “Direct_problem” and “Inverse_problem” programs.

D.- To detect and notify the existence of any invalid input data in order to avoid launching the programs under conditions that will result in error reports.

TECHNICAL SPECIFICATIONS

All the “Direct_problem” program subroutines where coded in FORTRAN 77 and compiled with the ABSOFT FORTRAN 77 For Power Macintosh compiler. Also, some preexistent subroutines that where available in the literature have been used. That is the case of the fast integration functions and the matrix inversion subroutine.

The Macintosh Runtime Window Environment, which is included in the compiler, was employed. It consists of a collection of subroutines that provide a Macintosh-type interface by using the Macintosh Toolbox routines and the Apple user-interface guidelines. For this reason, the code of the “Direct_problem” program is not directly portable to other platforms different

than Macintosh (68xxx) or Power Macintosh (PowerPC 60x). Some modifications are required in order to add portability to the current code.

In the actual setting of the program, it requires about 5 Mbytes of memory to run. This requirement is related to the maximum size of logging tool that the program can handled; it is basically determined by the parameters *NUMAX* and *NVMAX* , which are defined in the header file “Dimensions.inc”. By reducing these parameters and recompiling the program the run time memory requirements can be dramatically reduced; however the maximum size of logging device allowed will be reduced too. For more information about these and other dimensional parameters see [1].

The program’s interface has been developed by using Hypercard 2.3.5, which provides a very convenient and flexible way to perform such a task. The interface can be either run from Hypercard as a Stack or can be compiled into a stand alone application. It is important to mention that the program’s interface is not required for the “Direct_problem” program execution; it just provides a friendly environment for handling the program in a very simple and easy way. Indeed, the input files can be alternatively created by using any editor and the “Direct_problem” can be directly launched by opening it or double-clicking on its icon.

FINAL NOTES

In the present report, a brief discussion about the computational implementation of the direct version of the time harmonic field electric logging problem have been presented. For a more detailed an precise information please refer to [1] or to the code files directly, which are fully commented and will provide the most updated information about the program.

REFERENCES

[1] Help & Information Stack.

[2] Bostick, F.; Smith, H. (1994), Propagation Effects in Electric Logging.
University of Texas at Austin.

[3] Update Report #1: The Direct Problem.

[4] Update Report #5: Solution of the Potential Difference Integral by using Exponential Windowing.

[5] Update Report #6: Enhanced Method for the Solution of the Potential Difference Integral.

[6] Update Report #8: Recursive Computation of the Derivatives.

[7] Update Report #9: Modeling of Laterolog Systems.

[8] Update Report #10: Modeling of Generic Logging Devices.