7. CODE AND DOCUMENTATION

This program is a C program that ran on a main frame computer or a workstation, and has been updated and translated into a Borland C program capable of running under Windows 3.1 or Windows 95 on a PC. Attention has been given to make the code self-documenting. The program was developed under a C++ environment; however, no object-oriented code is involved.

7.1 Purpose and General Description

The program is an implementation of the transfer function for an HF propagation model developed and described in Vogler and Hoffmeyer [3-5]. The program reads a data file containing parameters that are descriptive of the several paths or rays of an HF radio skywave channel. The program writes the complex Fourier coefficients that define the channel transfer function to a file that will be used in a wideband HF channel simulator in hardware. The model simulates the time-varying characteristics of the HF channel and, in particular, models delay spread, delay offset, Doppler frequency shift and spread, and the relationship between delay and Doppler. The transfer function simulates the influence of the HF channel medium upon the transmitted signal. In the future, the hardware simulator will also include models for noise and interference described in Lemmon and Behm [8,9].

The program is a windows executing code with command line

```
< LEWS FILE1 FILE2 FILE3 >
```

where LEWS.EXE is the executing code file, FILE1 is the input file, FILE 2 is the output file for input and computed parameters, and FILE3 is the output file for the complex Fourier coefficients defining the transfer function.

The source code for the program is in four files, which are compiled separately and linked together to form the executing code: LEW1.CPP, LEW2.CPP, LEW3.CPP, LEW4.CPP.

The documentation is organized as follows: The documentation for a function or a file's list of global variables, defines, and includes, etc. will be immediately followed by the listing of that function's code. When documentation refers to code, the following conventions are used: The names of constants and files are indicated by all caps as in MAXLAYERS; bold indicates both default and defined data types and functions, for example **comp_arrays**; and variables are in italics, e.g., *peak_amplitude*.

7.2. Project File LEW1.CPP

The file LEW1.CPP contains the main program.

Defines:

MAXLAYERS - the maximum number of reflecting layers (or reflected rays seen by the receiver) in the ionosphere that the program will handle.

Structures:

ray_path - structure that contains all input and computed variables characteristic of a path.

.path_Distance (D) - **float**, point-to-point distance, in kilometers, between the transmitter and the receiver, used to compute *tau_c*, read from input data.

.center_freq (f_c) - **float**, the center frequency, in Hertz, used to compute *tau_c*, read from input data.

.penetrate_freq (f_p) - **float**, the penetration frequency, in Hertz, used to compute *tau_c*, read from input data. The penetration frequency must be greater than the center frequency, since frequencies above this point will not be reflected. The penetration frequency is above the maximum useable frequency (MUF). If *penetrate_freq* is less than *center_freq*, an error message is generated.

.thick_scale (σ) - **float**, thickness scale factor, indicates the thickness of the reflecting layer in kilometers, used to compute *tau_c*, read from input data.

.maxD_hgt (h_0) - **float**, maximum electron density height in the layer, used to compute *tau_c*, read from input data.

.peak_amplitude (A) - **float**, the amplitude at the mean delay τ_c, read from input data.

.sigma_tau ($\sigma_\tau = \tau_U - \tau_L$) - **float**, the delay spread, read from input data.

.sigma_c ($\sigma_c = \tau_c - \tau_L$) - **float**, the partial delay spread, distance between τ_c, the point of peak amplitude and the lower value of the delay spread τ_L, read from input data. Essentially, σ_c is the rise time to τ_c of the impulse response with respect to τ_L.

.sigma_D (σ_D) - **float**, the Doppler spread half-width at τ_c, read from input data.

.fds (f_s) - **float**, the Doppler shift at τ_c, read from input data.

.fdl (f_{sL}) - **float**, the Doppler shift at τ_L, read from input data.

.tau_c (τ_c) - **double**, expected delay associated with the carrier frequency, calculated from input data in **big_c**.

.sigma_f (σ_f) - **double**, used to determine the Doppler shape, computed in **comp_arrays**.

.slp (b) - **double**, the slope or "slant" of the linear relationship between delay and Doppler, computed in **comp_arrays**.

.tau_L ($\tau_L = \tau_c - \sigma_c$) - **double**, left end of the delay spread at the half power point A_{fl}, computed in **comp_arrays**.

.tau_U ($\tau_U = \tau_L + \sigma_\tau$) - **double**, right end of the delay spread, at the half power point A_{fl}, computed in **comp_arrays**.

.tau_l (τ_l), **double**, location parameter for the delay function, computed in **little_el**. This is not necessarily the same for every layer nor is it necessarily at zero delay for this model.

.alpha (α) - **double**, shape parameter for the delay function, computed in **comp_arrays**.

.sigma_l ($\sigma_l = \tau_c - \tau_l$) - **double**, the rise time of the impulse response with respect to τ_l.

.lambda (λ) - **double**, exponential autocorrelation factor through time for random input streams.

compute - structure that contains all the variables specific to the computations or not specific to an individual path.

.layers - **integer**, the number of reflective ionospheric layers or the number of reflected rays, read from input data.

.slices - **integer**, number of time slices, read from input data.

.seed - **integer**, primary seed for random number generation, between 0 and 30,268 inclusive, read from input data file.

.delta_t (Δt)- **float**, the sampling interval or the real time step, read from input data.

.afl (A_{fl}) - **float**, the receiver threshold from input data.

.delta_tau ($\Delta\tau$) - **double**, delay step, computed in **comp_arrays**.

.big_el - **double**, the least or earliest of the *tau_l* values for the layers, determined in **comp_arrays**.

String type:

STRING - used for handling file names of input and output files.

```
#include <stdio.h>

#define MAXLAYERS 3

typedef struct ray_path
{
        float path_Distance, center_freq, penetrate_freq, thick_scale, maxD_hgt;
        float peak_amplitude, sigma_tau, sigma_c, sigma_D, fds, fdl;
        double tau_c, sigma_f, slp, tau_L, tau_U, tau_l, alpha, sigma_l, lambda;
};

typedef struct compute
{
        int layers, slices, seed;
        float delta_t, afl;
        double delta_tau, big_el;
};

typedef char *STRING;
```

7.2.1. Function **void main**

Description:

The **main** function calls functions **init** and **doit** and also handles file names input from the command line. **Main** is in the file LEW1.CPP.

Parameters:

argc - **integer**, indicates the number of parameters on the command line; for this program there are four.

argv - array of **pointers** to the string arrays of the parameters, *argv*[1] points to the input file, *argv*[2] points to the first output file, *argv*[3] points to the second output file *argv*[0] points to the file with the executable code. Example command line for execution:

```
LEWS FILE1.DAT FILE2.DAT FILE3.DAT
```

Structures:

p - an array of size MAXLAYERS of **ray_path**.
c - a one element array of **compute**.

Functions called:

init - type **void**, initialization procedure, reads and checks data from input file, writes parameter output file. **Main** passes *argc* by value, and **STRING** *argv*[1], *c* - an array of **compute**, and *p* - an array of **ray_path** by reference. These arrays are initialized here. **Init** is in LEW2.CPP.

doit - type **void**, the procedure that accomplishes everything but input/output. **Main** passes *c*, the single element array of **compute**, and *p*, an array of **ray_path**, and **STRING**s *argv*[2] and *argv*[3], by reference. **Doit** is in LEW3.CPP.

```
void main(int argc, char *argv[])
{
                /* Function prototypes */

        extern void init(int, STRING, compute[], ray_path[]);
        extern void doit(compute[], ray_path[], STRING, STRING);

                /* Structures */

        struct ray_path p[MAXLAYERS];
        struct compute c[1];

                /* Code */

        init(argc, argv[1], c, p);

        doit(c, p, argv[2], argv[3]);

}  /* End of main */
```