MATLAB Program for Systematic Simulation over a Transmission Line in Alternative Transients Program

G. D. Guidi-Venerdini, F. E. Pérez-Yauli

***Abstract*-- This paper presents a program developed in MATLAB® to manipulate four ATP base files in order to make systematic simulations of faults along an overhead transmission line. Faults are: single-phase to ground, two-phase to ground, isolated two-phase and three-phase fault, with the possibility of varying the fault resistance value in the first two cases and the fault inception angle in all cases.**

Taking one transmission line end as reference, the simulated fault is moved along the line with an Δx step, which is defined by the user, until reaching the other end of the line. For every Δx the case simulation is done. The program identifies the line stretch in which the fault has been inserted and modifies the lengths of its sections to insert the fault at the corresponding distance from the line end taking it as a reference, keeping both the stretch and the total line length constant. Also, the measuring elements are moved with the fault point for verifying the desired inception angle.

***Keywords*: ATP, EMTP, Faults, MATLAB, Systematic Simulation, Transmission Line, Transient Based Protection.**

I. Introduction

THE Alternative Transients Program (ATP) [1] is a powerful simulation tool based on the Electromagnetic Transients Program (EMTP) [2], and it has replaced the well-know Transient Network Analyzers (TNAs) today. The studies that employ the ATP/EMTP (henceforth only ATP) software have goals framed in two broad categories. One of these is the design, which includes the insulation coordination, sizing of equipment, protection system specification, control system design, etc. The other one is the operation troubleshooting such as faults in power systems and transient analyses.

ATP offers many benefits such as: the accurate simulation (through detailed model implementation of electric network elements) of high frequency transients that occur within short time intervals and the ease in handling simulation results (which can be plotted or printed as time functions and stored in files for later processing), to name a few. For that, the electric protection area has become one of the largest ATP consumers, in particular, for designing new protection schemes. Current bibliography [3]-[5] reveals that the design of such schemes requires databases built by a large number of fault signals simulated in various power system operating conditions.

The database generation implies the investment of large periods of time and often it represents a discouraging obstacle that prevents the completion or finalization of a particular project or study.

The algorithm proposed in this paper, is a valuable tool designed to facilitate the extensive fault simulation. Even though the algorithm was designed for fault simulation on a transmission line originally, it can be applied in a wide range of simulation possibilities due to its high versatility provided by the advantages offered by ATP and MATLAB® [6].

This paper is organized as follows: Section II briefly describes the signal set required to do studies with protection systems, the program developed is detailed in Section III, Section IV deals with the program application oriented to systematic fault simulation on a transmission line and presents the base cases implemented in ATPDraw [7]. Finally, the main conclusions of this work are summarized in Section V.

II. Signals Required by Protection Systems

In general, the traditional protection systems used in power systems, require a fault signal set for verifying the protection coordination and the performance of each of them and the whole, under different power system operating conditions.

On the other hand, for developing new protection schemes based on the principle known as Transient Based Protection (TBP) [8], having a fault signal set in the stages of: training, validation and test is fundamental.

The first stage consists in either: training the proposed protection algorithm [9]-[10], determining (or extract) characteristic patterns of fault in order to facilitate their classification [11], adjusting specific parameters for an appropriate algorithm operation [12], etc. whereas the validation stage makes use of a different fault signal set with the goal of tuning up the parameters required by the

This work was supported in part by the Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) and German Academic Exchange Service (DAAD).

G. D. Guidi-Venerdini is with the Instituto de Energía Eléctrica, Universidad Nacional de San Juan, San Juan, Argentina (e-mail: gguidi@iee.unsj.edu.ar).

F. E. Pérez-Yauli is with the Instituto de Energía Eléctrica, Universidad Nacional de San Juan, San Juan, Argentina (e-mail: fperez@iee.unsj.edu.ar).

Paper submitted to the International Conference on Power Systems Transients (IPST2013) in Vancouver, Canada July 18-20, 2013.

algorithms. Finally, the test stage allows for verifying that the proposed protection schemes satisfy all imposed conditions for ensuring their correct operation and the desirable performance.

Thereby, for the TBP protection case, the larger the databases used in the three stages described above, the better adjustment and test of the proposed schemes. Therefore, the algorithm detailed in this paper constitutes a useful tool that aids development of new TBP schemes and it also involves a significant amount of time saving and reduces human errors; and so, it can be practically employed in studies with traditional protection systems.

III. Program Developed

Fig. 1 shows the program general flowchart.

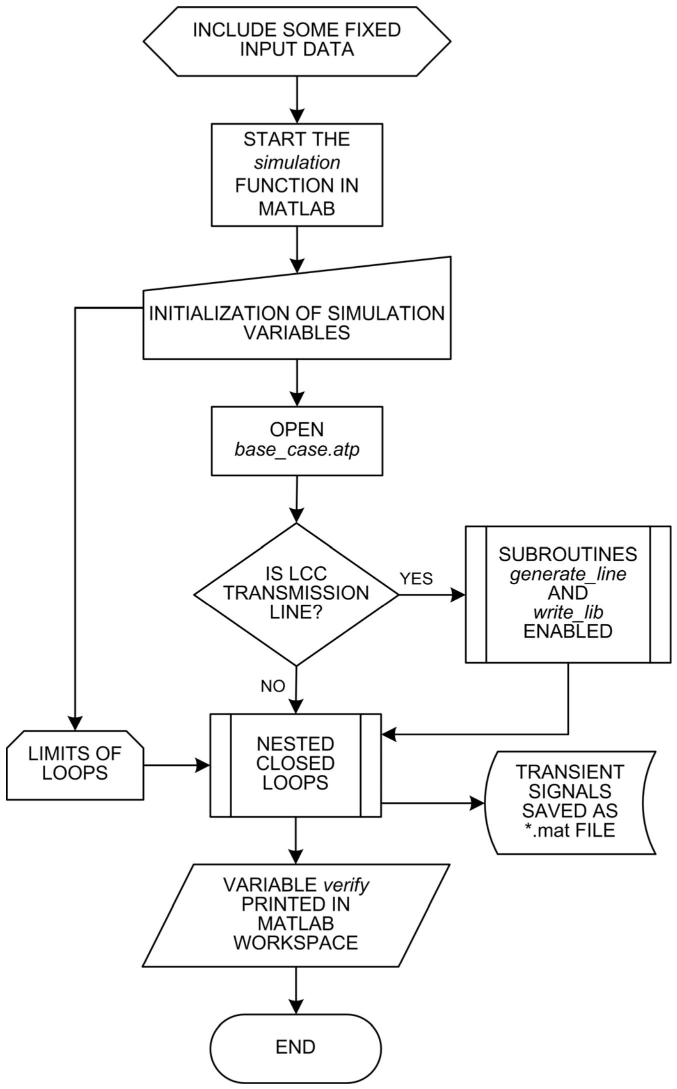
INCLUDE SOME FIXED INPUT DATA
START THE *simulation* FUNCTION IN MATLAB
INITIALIZATION OF SIMULATION VARIABLES
OPEN *base_case.atp*
IS LCC TRANSMISSION LINE?
YES
SUBROUTINES *generate_line* AND *write_lib* ENABLED
NO
LIMITS OF LOOPS
NESTED CLOSED LOOPS
TRANSIENT SIGNALS SAVED AS *.mat FILE
VARIABLE *verify* PRINTED IN MATLAB WORKSPACE
END

Fig. 1. Program general flowchart.

Before starting the simulation process, some data must be taken from the power system modeled in ATP and they are included into the *simulation* function as fixed input data; for instance: the power frequency, the transmission line length, the zero-crossing voltage time in both line ends, among others.

The program starts by running the *simulation* function created in MATLAB®, for which some input variables must be initialized. This function can have as many input variables as needed. It is recommendable that the first input variable will be the *base_case*. The *base_case* variable represents the .atp file (created previously in ATPDraw) on which the simulation will be done. In this manner, there are as many base cases as are required. For instance, for the program application described in Section IV there are four base cases, which are the fault types considered on an overhead transmission line.

The remaining input variables are the parameters to be modified along simulation within the .atp file selected.

Once the base case has been chosen, the program uses closed loops through the command “for” in order to go on to modifying the simulation parameters. That is, there are as many nested closed loops as parameters to be modified, depending on the disturbance to be simulated. For the example detailed in next section, three nested closed loops are required with the purpose of changing: the fault point along the line, the fault inception angle and the fault resistance value (if admitted, according to the fault type).

The limits of simulation are established in the variable initialization.

Special attention must be given to the disturbance point variation. ATP offers a vast assortment of simulation options and it allows for choosing among a great variety of power system component models. If the Line Constant (LCC) model [1] is selected for the transmission line on which the disturbance will be simulated, two additional subroutines called *generate_line* and *write_lib* are enabled in the *simulation* function.

The *generate_line* subroutine generates a variable that contains the values needed to create a .lib file belonging to a LCC transmission line of length *x*. In this case, *x* is the variable that goes changing the disturbance point, which is performed by modifying the corresponding data in the template file *base_line.dat*.

For its part, the *write_lib* subroutine is used to write to disk all variables that contain the information of the line sections (see Section IV). For each variable a .lib file is created. This subroutine calls another one named *save_file*, which is briefly described below.

On the other hand, if another transmission line model (different from LCC) is chosen, then the changes for simulation are made only within the .atp file.

The program is complemented by a subroutine called *save_file* located within the innermost loop and it aims to save to disk the changes in the .dat and .lib files generated by the subroutines *generate_line* and *write_lib*; and also to write to disk the *name_case.atp* file before starting each simulation.

The link between MATLAB® and ATP is made through the MATLAB command “dos” to run the file *name_case.atp* taking into account the modifications done for each simulation.

The output data (transient signals) are saved as MATLAB data files (*.mat) by using the freeware routine Pl42mat [13] and they are called *name_case.mat* in accordance to all simulation parameters. For instance, the file named *90_r5_k_230_2ground.mat* corresponds to a two-phase to ground fault simulated on a selected transmission line at kilometer 230 with 5 Ω as fault resistance and an inception angle of 90° (taking the voltage signal of phase A as reference). The phases involved in the fault, are those established in the corresponding .acp file, as shown in Fig. 3.

The *simulation* function has just one output variable, which is called *verify* and it serves as a control variable for checking a correct simulation. Within this output variable, the program records all data needed by the user for their examination.

More details about each program part are given in the next section for a particular application.

IV. Program Application for Extensive Fault Simulation

The program presented in this paper was employed for building a database made up by transient signals rising from a total of 5,328 faults simulated on the Argentinean power system modeled in ATP. Such a database was created with the intention of proposing a new TBP protection scheme for transmission lines [14]. That scheme uses current measurements of only one phase of the three-phase systems for determining the fault direction, the faulted line and the fault type. The discrete wavelet transform is used to extract information concerning the high-frequency components of transient current signals. An adaptive wavelet, which has been specifically designed for relaying purposes, it was used. For signal classification, the protection scheme at hand employs Bayesian linear discriminant analysis.

In [14], a 396 km transmission line was used for analyses. Fig. 2 shows the aforementioned line divided into four stretches of 1/6, 1/3, 1/3 and 1/6 of total line length by virtue of three existing transpositions. As can be observed, each stretch was subdivided in two sections, and four internal nodes denominated NF1, NF2, NF3 and NF4 are used for connecting the fault models illustrated in Fig. 3.

Table I listed the stretch lengths and their corresponding sections, taking into consideration a total transmission line length of *L*.

All line sections were modeled by using the LCC model that generates a .lib file, which is read and included by the ATP when running the simulation case. The LCC routine is included in the ATP and it calculates the transmission line primary constants from its geometry and the line conductors. Several LCC models are available in ATP: Jmarti, Pi, Bergeron, etc. The Jmarti model was chosen to create the line depicted in Fig. 2.

The LCC routine takes a .dat file as input, with all line data (geometry, conductors, earth resistivity, line length, model type, etc.). Particularly, in the case of Jmarti line models, running with ATP the .dat file, generate a .pch file. The latter contains the information of frequency dependent line model such as: the distortion function that waves experienced while they traveled along the line, the characteristic impedance function, the wave propagation speed and the modal transformation matrix. With the .pch file, a MATLAB subroutine *generate_line* generates a .lib file, which is read and included in the complete model to run the transient by the ATP for a particular fault point on the line.

The geometry and the conductors of the line are considered equal throughout its length. This allows for using a single data line template (*base_line.dat* file) to generate all sections of Fig. 2 through MATLAB, and obtaining all .lib files according to the section length of each stretch. Also, diverse templates can be used for each line section if their geometry or conductors are different.

TABLE I
Stretch and Section Length of Transmission line of Fig. 2

Stretch	Sections	Length
1	S1+S2	1/6*L
2	S3+ S4	1/3*L
3	S5+ S6	1/3*L
4	S7+ S8	1/6*L

The .lib file names that have been given to the 8 line sections of Fig. 2 in all base cases are: S1.lib, S2.lib, S3.lib, S4.lib, S5.lib, S6.lib, S7.lib and S8.lib.

From a first base model built with ATPDraw corresponding to the desired power flow setting, four base cases were created, which are: single-phase to ground fault, two-phase to ground fault, isolated two-phase fault and three-phase fault. Each of these is obtained by inserting in the NF*i* node of Fig. 2 the schemes illustrated in Fig. 3. In this manner, the ATPDraw generates the ATP code for each fault type, which is subsequently modified by MATLAB.

A. Methodology for moving the fault along line

By taking advantage that each stretch is divided in two sections, the faults are located between them. In this way, the section lengths are adjusted consecutively and the stretch total

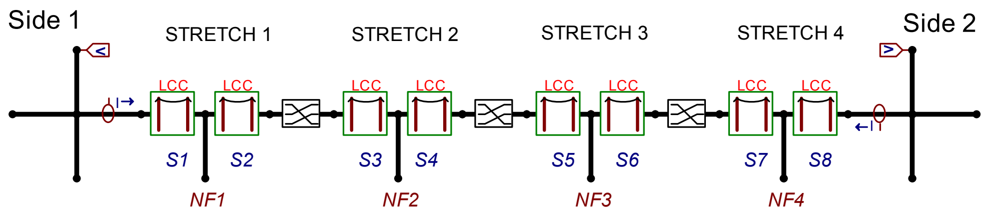

Fig. 2. ATPDraw line model scheme.

length is kept constant.

The .atp base file is kept invariable, however, it reads to disk the .lib files that are manipulated by the program developed for moving the fault point. Thus, each time a simulation case is run, 8 new .lib files corresponding to the 8 line sections are generated and read by the .atp base file.

B. Initial Arguments

Fig. 4 shows the program flowchart for this particular application.

Before starting the program some arguments must be included within the *simulation* function, some of them are obtained from a previous ATP run, and most of them are defined by the user.

1) Data obtained from the model:

L → Total line length.

2) Data obtained from an ATP run:

angS1 → Phase A angle of side 1 of the line in degrees.

angS2 → Phase A angle of side 2 of the line in degrees.

t0 → Reference time. It is the time at which the voltage signal at side 1 passes through zero.

3) Other arguments defined by user:

directory → Simulation output directory.

fault_type → Flag used to identifies the fault type: 1 for single-phase to ground, 2 for isolated two-phase, 3 for two-phase to ground, and 4 if it is three-phase fault.

Rrfault → Fault resistance value.

phi → Fault inception angle in degrees.

x0 → Initial fault distance.

step_x → Displacement length along the fault line.

C. Start Program: Initialize variables

The program initializes the following variables with the input arguments:

x → Fault distance measured from side 1 of the line. This is the most important variable and will be increased in *step_x* each time.

RT → Total row of the *base_case.atp* file.

$$delay = 0.02 \cdot (phi / 360) \qquad (1)$$

where delay is the inception angle expressed in seconds and the constant 0.02 is the time corresponding to one cycle of power frequency (50 Hz Argentinean power system).

$$Dtime = 0.02 \cdot \left(\frac{angS1 - angS2}{360} \right) \qquad (2)$$

where *Dtime* is the diphase angle between the voltage signals at both line ends expressed in seconds.

$$Dtx = Dtime / L \qquad (3)$$

where *Dtx* is the diphase angle (*Dtime*) per unit length expressed in seconds.

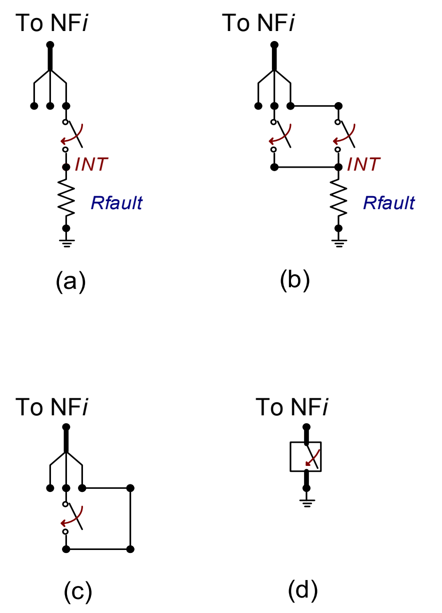

Fig. 3. Fault schemes modeled in ATPDraw. (a) Single-phase to ground fault. (b) Two-phase to ground fault. (c) Isolated two-phase fault. (d) Three-phase fault.

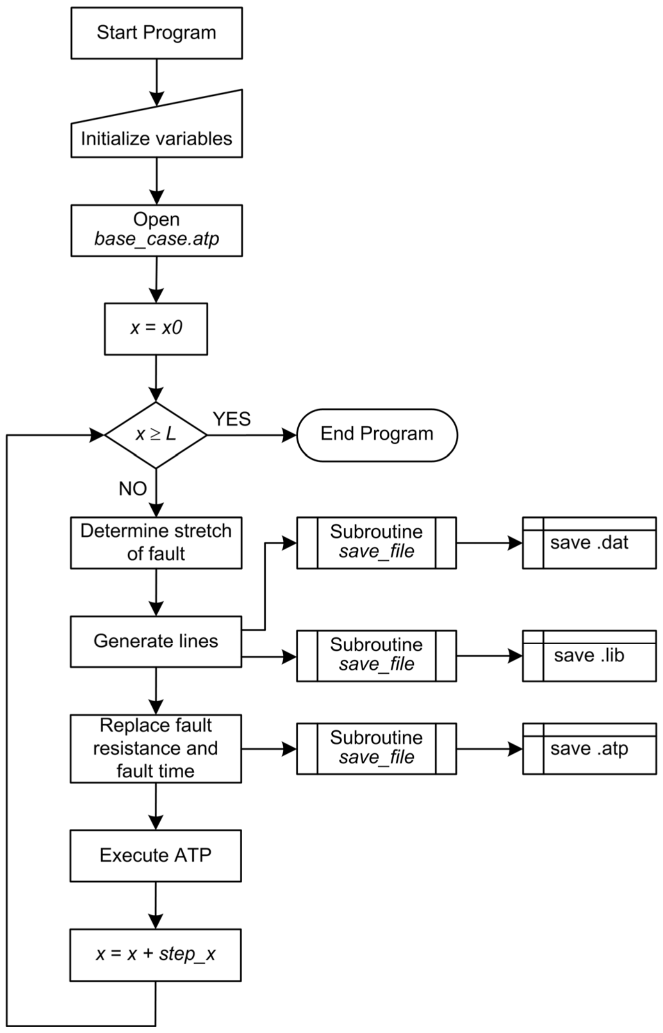

Fig. 4. General flowchart of the developed program for extensive fault simulation.

D. *Open base_case*

By using the command "switch and case" the program compares the variable *fault_type* and so, the fault type defined by the user is recognized. Afterwards the corresponding .atp file is loaded into the variable *base_case*.

Then, the number of atp code row containing the close time of circuit breaker used to perform the fault maneuvers is identified and recorded into a variable *Rint*. In addition, the row number related to the fault resistance value is also identified and stored into *Rrfault* variable as the total row number of the atp base code in *RT*.

The MATLAB command used for these tasks is “strmatch”, which finds matches for strings [6].

Fig. 5 shows the corresponding code for single-phase to ground fault case.

```
switch fault_type
case 1 %single-phase to ground fault
base_case=char(textread('base_casesingle.atp',...
'%s','delimiter','\n','whitespace',''))
Rint = strmatch('  NF1A  INT',base_case)
Rrfault = strmatch ...
('  INT                ',base_case)
end
RT=length(base_case) %total rows of base_case file
```

Fig. 5. Fault type selection and atp *base_case* variable.

E. *Variable x initialization and main loop start*

By following the flowchart of Fig. 4, the variable *x* is initialized to the value *x0* and the main loop for fault displacement is started by using a “for” statement from *x0* to *L* in *step_x* intervals.

F. *Determine stretch of fault*

By moving the fault point from one side to another, it must pass for the four line stretches.

The stretch to insert the fault is determined by the value of variable *x* and the node name (NF*i*) to connect the fault circuit breaker depends on this determination.

This procedure has been solved by an “if” statement as shown in Fig. 6. The variable *stretch* is used to identify the stretch where the fault is simulated.

```
if x > L
     warning('x out of range (x>L)')
     stop
     elseif x > L*5/6
          stretch=4          %stretch
          Lstretch =1/6*L    %stretch length
          Lini=5/6*L         %stretch initial length
     elseif x > L/2
          stretch =3
          Lstretch=1/3*L
          Lini=L/2
     elseif x > L/6
          stretch=2
          Lstretch=1/3*L
          Lini=L/6
     elseif x > 0
          stretch=1
          Lstretch=1/6*L
          Lini=0
     else
     warning('x out of range (x<0)')
     stop
end
```

Fig. 6. Stretch determination.

G. *Generate lines*

The two line sections corresponding to the stretch in which the fault is placed are named section A and B. The lengths of these sections are calculated by:

$$LA = x - Lini \tag{4}$$

$$LB = Lstretch - LA \tag{5}$$

where:
LA is the length of section A.
LB is the length of section B.
Lini is the length in which the stretch begins; it is measured respect to the line side 1.
Lstretch is the length of stretch corresponding to the fault position.

As previously mentioned, the program has a subroutine called *generate_line* which creates the code .lib taking into account the length of the line. This subroutine opens the .dat line template, replaces the required length, runs ATP to generate the .pch file and finally, from it, generates the corresponding .lib code in a variable called *LineXlib*. This subroutine does not save that variable to a file disk, it is just within a variable. The save to disk task is done by another subroutine called *save_file*, which is explained in paper Section IV.J.

The length of sections S1, S2, S7 and S8 of the default base_case files are *L*/12, and the remaining *L*/6. Therefore, in addition to sections A and B (that are loaded in every ATP run), a .lib code for lines of length *L*/12 and *L*/6 is necessary. These .lib codes are generated once and they are stored with variable names *SL12* and *SL6*, respectively.

Once the section A and B .lib codes have been generated (sections where the fault lies) in variables *SA* and *SB*, all the 8 .lib variables corresponding to the 8 sections are sent to write to disk files by overwriting the previous .lib files in the work directory.

For this purpose the subroutine *write_lib* is used, which has 8 input arguments corresponding to the codes of 8 line sections. This subroutine saves the contents of each argument in .lib files with names according to Table II.

Thus, by calling the subroutine *write_lib*, variables that contain the .lib code are passed as arguments sorted by the stretch in which the fault lies. This task is done by a sentence “switch and case”. The stretch in which the fault lies must be previously identified by the variable *stretch* (see Section IV.F).

The .lib file names in table II match the names of the base cases as can be seen in Fig. 2.

Finally, the arguments used to call the subroutine *write_lib* are cited in Table III.

TABLE II
NAMES USED BY WRITE_LIB SUBROUTINE
TO STORE VARIABLES IN DISK

Argument	File name
1	S1.lib
2	S2.lib
3	S3.lib
...	...
8	S8.lib

TABLE III
SUBROUTINE *WRITE_LIB* ARGUMENTS

Value of variable *stretch*	Order of arguments
1	*write_lib* (SA, SB, SL6, SL6, SL6, SL6, SL12, SL12)
2	*write_lib* (SL12, SL12, SA, SB, SL6, SL6, SL12, SL12)
3	*write_lib* (SL12, SL12, SL6, SL6, SA, SB, SL12, SL12)
4	*write_lib* (SL12, SL12, SL6, SL6, SL6, SL6, SA, SB)

H. Replace fault resistance and fault time

Depending on the fault type chosen to be simulated (see Section IV.B), different fault resistance values are employed for fault simulation according to the arguments provided by the user. For that, the program automatically replaces the fault resistance information in *Rrfault* row of *base_case* file that contains all atp code.

To obtain the desired fault inception angle (*phi*), the program computes with (2) the diphase angle between the phase A (if chosen as reference) voltage signals of both line ends in the power system steady state. After that, the angle diphase per unit length is calculated through (3) and with this information, the close time of circuit breaker that inserts the fault is regulated as the fault is moved. Thereby, the circuit breaker close time *tf* (fault inception angle) is always desirable, as follows:

$$tf = t0 + Dtx \cdot x \tag{6}$$

I. Execute ATP

Once the modifications required have been made, the *base_case*.atp file is saved with subroutine *save_file* and it is named respecting the code given in Section III.

Then the .atp file saved is run with ATP and its output file is saved as .mat file.

Finally, *step_x* is added to *x* and all actions within main closed loop are run over and over until satisfying the imposed condition.

J. Subruoutine save_file

To write to disk the new files, a subroutine named *save_file* was developed in MATLAB environment. This subroutine is responsible for saving the modified files .dat and .atp and the generated files .lib. To do that, the MATLAB Escape Characters [6] are firstly replaced by the *save_file* subroutine. These characters are:

''	Single quotation mark
%%	Percent character
\\	Backslash

Later a new file is generated to disk, the file in which the variable that contains the data to be saved is written.

Fig. 7 shows the complete code of this subroutine. The "*fopen*" command creates a new file to disk and the *wt* option is for opening or creating a new text file to be written, discarding existing content.

The "*fprintf*" command writes data to a text file. *V1* and *V2* are auxiliary variables and *directory* is the path in which the new file is going to be placed.

```
function[]=writetodisk(var,name,ext)
V1=[]
V2=[]
for fila=1:size(var)
    V1=strrep(var(fila,:),'\','\\');
    V1=strrep(V1,'''','''''');
    V1=strrep(V1,'%','%%');
    V2=strvcat(V1,V2);
end

fid = fopen([directory,...
       '\',strcat(name,ext)], 'wt');

for fila=1:size(V2);
    fprintf(fid,[(V2(fila,:)),'\n']);
end

fclose(fid)

end
```

Fig. 7. MATLAB code of subroutine *save_file*.

V. CONCLUSIONS

A valuable tool for extensive simulation in ATP/EMTP has been presented in this paper. It is a linking program between MATLAB® and ATP that thanks its high versatility, has a wide range of simulation possibilities of power system disturbances, for example: faults or lightning strokes.

The main advantage of this technique is the time savings on building databases for systematic studies along a transmission line. Once the program is adjusted for the case under study, the results are obtained automatically. Little adjustment modifications can be done to develop a sensitivity analysis, without having to repeat all the cases one by one.

The program application in some previous research work has evidenced the simulation time reduction and the human mistake elimination along simulation process. Its use is highly recommended in those cases that require large databases built with transient signals; databases that are specifically useful in diverse power system areas, like the example detailed in Section IV that relates to protection systems. However, the program can be also applied on traditional analyses that do not require large databases.

In addition, the ease in resulting data handling within a MATLAB environment adds another advantage and increases the program potential to be widely used by the scientific community.

VI. References

[1] Bonneville Power Administration, "Alternative Transients Program (ATP)," ed. Portland, Oregon, U.S.

[2] H. W. Dommel, "Digital Computer Solution of Electromagnetic Transients in Single-and Multiphase Networks," *IEEE Trans. Power Apparatus and Systems*, vol. PAS-88, pp. 388-399, Apr.1969.

[3] J. A. Jiang, Ching-Shan Chen, and Chih-Wen Liu, "A new protection scheme for fault detection, direction discrimination, classification, and location in transmission lines," *IEEE Trans. Power Delivery*, vol. 18, pp. 34-42, Jan. 2003.

[4] E. M. Aboul-Zahab, E.-S.T. Eldin, D. K. Ibrahim and S. M. Saleh, "High impedance fault detection in mutually coupled double-ended transmission lines using high frequency disturbances," in *MEPCON 2008 12th International Middle-East Power System Conf.*, pp. 412-419.

[5] G. Mahmoud, I. Doaa khalil, and T. El Sayed, "Traveling-Wave-Based Fault-Location Scheme for Multiend-Aged Underground Cable System," *IEEE Trans. Power Delivery*, vol. 22, pp. 82-89, Jan. 2007.

[6] MATLAB, User's Guides. Natick, USA: The MathWorks Inc.

[7] ATPDraw, Dr. Hans Kr. Høidalen, SINTEF Energy Reseach - Norwegian University of Science and Technology.

[8] Z. Q. Bo, F. Jiang, Z. Chen, X. Z. Dong, G. Weller and M. A. Redfern, "Transient based protection for power transmission systems," *in 2000 IEEE Power Engineering Society Winter Meeting*, pp. 1832-1837 vol.3.

[9] G. Chawla, M. S. Sachdev and G. Ramakrishna, "Artificial neural network applications for power system protection," in *2005 Electrical and Computer Engineering Canadian Conf.*, pp. 1954-1957.

[10] M. A. Figueroa and E. Orduna, "Ultra-high-speed protection for medium voltage distribution networks with distributed generation," in *2008 IEEE/PES Latin America Transmission and Distribution Conf. and Exp.*, pp. 1-8.

[11] C. Aguilera, E. Orduña, and G. Rattá, "Fault detection, classification and faulted phase selection approach based on high-frequency voltage signals applied to a series-compensated line," *IEE Proceedings Generation, Transmission & Distribution*, vol. 153, pp. 469 - 475, July 2006.

[12] R. Aguilar, F. Pérez, and E. Orduña, "High-speed transmission line protection using principal component analysis, a deterministic algorithm," *IET Generation, Transmission & Distribution*, vol. 5, pp. 712-719, July 2011.

[13] M. Ceraolo and R. Salutari, "PL42mat," ed. Pisa, 2009.

[14] F. E. Perez, R. Aguilar, E. Orduña, J. Jäger, and G. Guidi, "High-speed non-unit transmission line protection using single-phase measurements and an adaptive wavelet: zone detection and fault classification," *IET Generation, Transmission & Distribution*, vol. 6, pp. 593-604, July 2012.