ATPMAT: An Open Source Toolbox for Systematic Creation of EMTP Cases in ATP Using Matlab

Ahmad Abdullah
Texas A&M University
Department of Electrical and Computer Engineering
College Station, TX, 77840
E-mail: ahmad@tamu.edu

***Abstract*—This paper introduces ATPMAT which is an open source Matlab-based toolbox to automate many tasks done in ATP faced by researchers and engineers. The toolbox provides a rich set of M-files that is GPL based to change ATP files and run ATP from within Matlab without interacting with ATP or ATPDraw. The toolbox provides a systematic way not only to change one ATP file but to batch process ATP files to create a batch of faults, line switching or lightning strike cases for any line. The heart of the toolbox is a data translator that interprets the contents of the ATP file and inserts appropriate lines of code to accommodate changes requested by the user. The toolbox then commits changes to ATP file according to user input and then run ATP from within Matlab. The output of the toolbox is a set of mat files each of which contains the voltages and currents from both ends of the line under study. The mat files are saved in a folder specified by the user.**

Keywords— ATP, EMTP, Data Translator, MATLAB, Open Source Toolbox, Faults, Lightning, Line Switching.

I. INTRODUCTION

Generating hundreds or even thousands of simulations using ATP have become such a common task in most academic institutions, yet there is no free tool publicly available to generate such simulations.

The power system committee has seen an increase in application of artificial neural networks to power system problems notably in classification [1]. ANN and other methods have been successfully applied to classification of power quality disturbances [2]. Also transient based protection schemes have been proposed in literature [3]. In TBP schemes and all classification problems, a large data set has to exist for the evaluation of the method proposed. In case of fault classification [4], a large number of simulations has to be carried out in order to verify the method. In case of ANN [5] a large number of cases has to exist for ANN training, validation and testing. Even though such studies need a lot of cases for verification, yet there is no systematic way for creating such cases for research purposes. In [6] a Matlab algorithm is given for automatic creation of fault cases without releasing any source code.

In this paper, we provide ATPMAT®: a Matlab® toolbox to use ATP [7] in batch processing mode; that is to create many simulations with one click of a button after the user supplies parameters of simulations he wants to generate. The code that is freely available on a git repository [8] is used to automate creation of fault cases, line switching cases, line opening, and lightning strike simulations. It is assumed that the user has a network that is already built in ATP and he wants to make changes to it to create a batch of simulations. Thus the toolbox we are sharing is not used to create ATP files but to modify existing file to accomplish the studies mentioned above.

The user has three functions to choose from: CreateFault, SwitchLine, and LightStrike. The CreateFault function creates a fault on a line to be chosen by the user. The SwitchLine switches the line on or off based on the time the user specifies. The LightStrike function strikes a line determined by the user by a lightning strike with certain amplitude at certain time. The user supplies the above aforementioned functions with certain parameters for the case to run. If the user wishes to create a batch of simulations, of faults for example, then he needs to include such function in a for loop. Such a function has been also supplied and it is called BatchProcessATP and can be considered the central point of the toolbox. The toolbox provides a rich set of M-files to change ATP files. For example: the user can insert a switch at the beginning or the end of the line and set its opening and closing times; the user can specify the magnitude and the rise/fall time of the lightning surge and its location; the user can select measuring nodes as well. The user also has the option to convert PL4 files to mat files using PL42mat [9] utility. The user needs to adjust the settings of ATP before using the toolbox.

The code has been tested on Windows 7 running on an Intel core i7 processor. The paper builds upon a paper that we published before in IPST 2011 [10] which the user need to consult in case he needs to see how the toolbox is used to create a real case for relay testing. This is the first version of the toolbox and we will be increasing its capabilities over the course of next versions. We also share ATP model for IEEE 118 [11] case as a test system in our online git repo [8]. The model is shared mainly to provide an online tutorial on how to use the toolbox.

The paper is organized as follows: Section II discusses how to set up system environment for Windows. Section III gives an overview of the structure of the toolbox and the functions supported by it. Section IV contains information of the

flexibility of the toolbox and output naming convention. At last, Section V contains the conclusion and future work.

II. Setting Up Sysmtem environment

Before using the toolbox, the user has to prepare his system for doing so. The user has to obtain the following files: runAF.exe, runATP.exe, makeATP.exe, PL42mat.exe and mytpbig.exe. All of those files have to be put on the same folder. The user will need to keep the path for this folder as this folder will be passed to the function BatchProcessATP. The author cannot provide those files as there are ATP licensing restrictions. However, it should be noted that all the above files can be obtained freely when the user first agrees to the terms and conditions of the ATP license [12]. Before using the toolbox, the user has to adjust listsize.big. The easiest way to adjust listsize.big is to open ATP Launcher then click on make Tpbig.exe from the tools menu. All the user needs to do is just enter the values given in a file uploaded to the bitbucket repository [8] and then click on make button. Besides obtaining the above files, the user must create a directory for his output files. The will need to enter the URI of this directory to BatchProcessATP. It is in this folder the user will find all output mat files of his simulations. This completes the setting up of environment in Windows.

III. Overview of Toolbox

The overall toolbox structure is shown in the flow chart in fig. 1. The starting function of the toolbox is CreateScenarios function. This is the function that is responsible for translating user input into a data structure that can be understood by the toolbox. After creating that structure, the user invokes the function BatchProcessATP which is the central point of the toolbox. To use BatchProcessATP, the user supplies the file path of the ATP file, the directory of all ATP related executables obtained in Section II, a user specified directory for the output of toolbox, the type of study and the structure that is created using CreateScenarios.

The type of the study has to be one of three options: “fault”, “lighting” or “energisation”. The information in the structure will vary according the study sought. If the user wishes to create a batch of faults, he needs to supply the following to CreateScenarios function: fault distance, fault resistance, fault type, line under consideration, start time, incipient angle and end time. If the user wants to create a lightning case, he needs to supply the line under consideration, the lightning amplitude, the phase being hit, the distance to be hit, the incipient angle and the start time. If the user wishes to energize a line, he needs to specify the line under consideration, which terminal to be energized and the angle at which the line will be energized. The inner working of the functions CreateFault, LightStrike, and SwitchLine will be described in the next subsections in more detail but in this section we focus on the overall picture and the systematic way of creating EMTP cases.

A fault or a lightning strike includes having the line divided into two sections and a generic block –which we will explain later- being inserted to the common point of both sections. In case of fault, we create a fault at the common point of both sections by inserting a generic fault block. In case of lightning we insert a lightning source at the common point of both sections. In case of line switching, we don’t divide the line but insert switches at the both ends of the. This is the first step in BatchProcessATP function. It makes generic changes to ATP file. These generic changes will be explained in the next subsections shortly. After these generic changes are made to the ATP file, a new file is saved in the toolbox

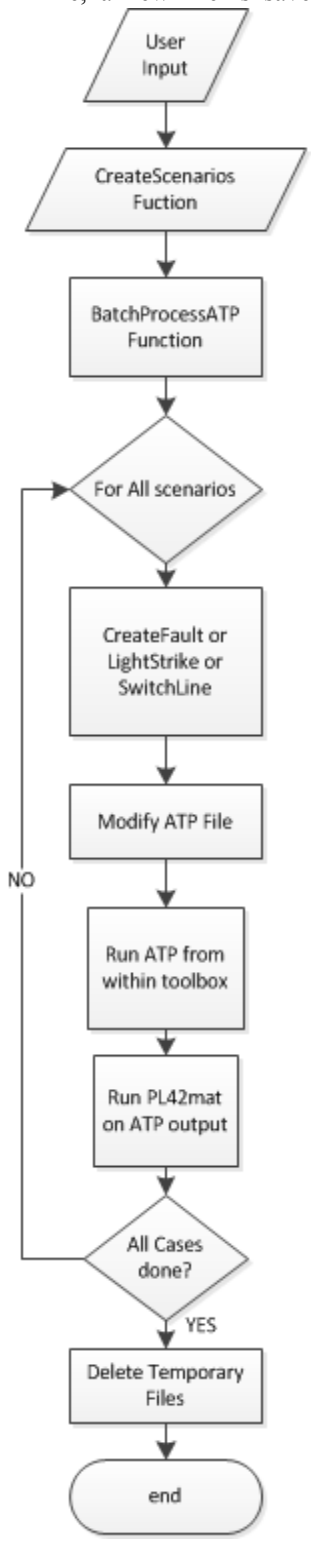

Fig. 1. Overall logic of toolbox

directory with a temporary name. It should be pointed out that these generic blocks are adjusted such that they are open circuit to the system.

The function BatchProcessATP then enters a for loop to create the cases the user requested. Having enetered the for loop, the fucntion consults the structured returned by the function CreateScenarios to make specific changes to the copy of the temporary file hitherto saved in the toolbox directory.

Having made the ATP file ready for the new file, the BatchProcessATP calls the runATP.exe from within Matlab to create an EMTP case. After than the utility PL42mat.exe is also called from Matlab to convert PL4 files to mat files. The function BatchProcessATP keeps executing the for loop till it reaches the last case after which is displays a message that all cases are created successfully if no errors occur during simulations. The last step is deleting any temporatry ATP files files that were created during simulations. In the coming subsections, we explain the inner working of the functions CreateFault, LightStrike, and SwitchLine.

A. Creation of fault cases (CreateFault function)

Although the function CreateFault is an internal function called by BatchProcessATP in its for loop, the user can use the function to create a single fault case. In this section we describe the logic implemented in this function.

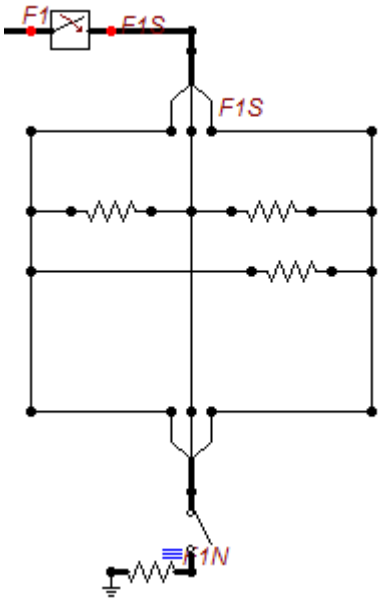

Fig. 2. Generic fault block being inserted in ATP file. All resistances are set to 1000 kΩ , switches are set to close at time = 5 seconds, making the block effectively open circuit during normal simulations.

As pointed out above the first case in creating a fault on a line is to divide the line into two sections and then inserting a generic fault block. The fault block inserted is given in fig. 2. The fault block simply consists of a phase splitter, two switches and resistances. The phase splitter is important to carry out different fault types. The block has two switches: one connecting the common point of the two line sections– called by default F1- to the phase splitter and the other provides connection to ground. The fault block is initially inserted with switch times and resistance values such that it is an open circuit to the simulation. When the user inputs certain type of fault, the function CreateFault changes those values. The function customizes the fault block to the fault type sought by adjusting the open and close times of the block switches as well as the value of resistors connecting various phases and the resistances to ground. The closing times of the switches are calculated according to equation (1)

$$Closing\ time = 2\pi \times f \times StartTime + IncipientAngle \quad (1)$$

where the start time (StartTime) and incipient angle (IncipientAngle) are all supplied by the user. Once the block fault is customized for the fault type requested by the user the function CreateFault proceeds to next step.

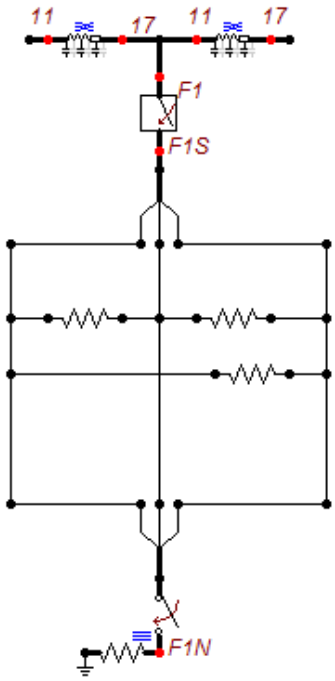

Fig. 3. Intermediate step in creating a fault where the fault block is customized for the fault type and line being divided into sections.

The next step is actually dividing the line under study into two sections. For this, we created a search function that goes in the ATP file and find the specific line to be studied and then duplicate that line. An example will make this clearer. If we are running fault cases on line connecting buses 11 and 17, then after the above step the line after being divided into two sections and the fault block inserted will look like what is shown in fig. 3.

After the generic block has been inserted and customized, the line divided into two sections, what is remaining is to insert probes at both sides. We have another function that inserts such current and voltage probes at both ends of the line. One has to note that duplicating lines and adding probes will cause runtime errors if we didn't change the names of the nodes of both lines. For this, we have another function that

changes the names of the nodes once the line is duplicated. Completing the above example, after the probes are inserted, we have what is shown in fig. 4.

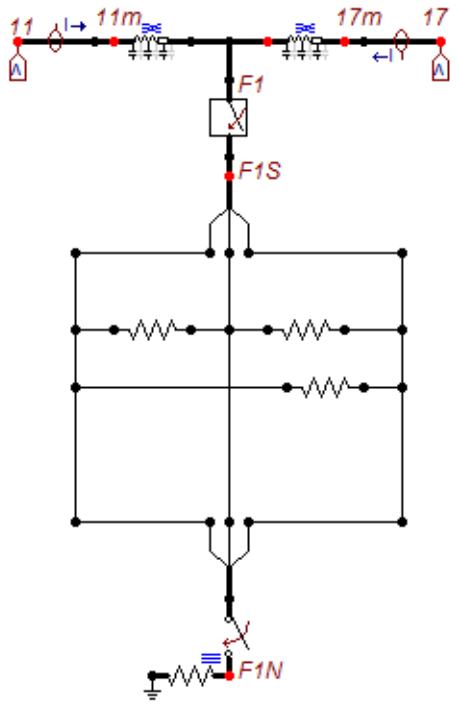

Fig. 4. Addition of voltage and current probes is done along with renaming the nodes. Note that we keep the names of both ends of the line we only change the internal names suffixing the letter 'm' to each section to avoid name collision with other nodes in the system. Note also the common point is always called F1.

The last part of this function is adjusting the lengths of both sections such that the summation of both lengths equal to the original line length. Another function is written for this purpose. The function takes its input from the data structure created by CreateScenarios function.

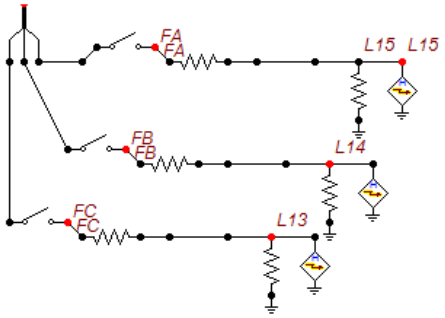

Fig. 5. Generic fault block being inserted to create a lightning case. All lightning sources are Heidler type and switches are set to close at time= 5 seconds making it effectively open circuit during normal simulation

Having changed both line sections lengths, the modification of ATP file is now complete. The new file is saved under another temporary name in the toolbox directory. After we exit this function, the toolbox calls ATP from within Matlab, waits for simulations to be done, calls PL42mat outputting the mat file to output directory specified by the

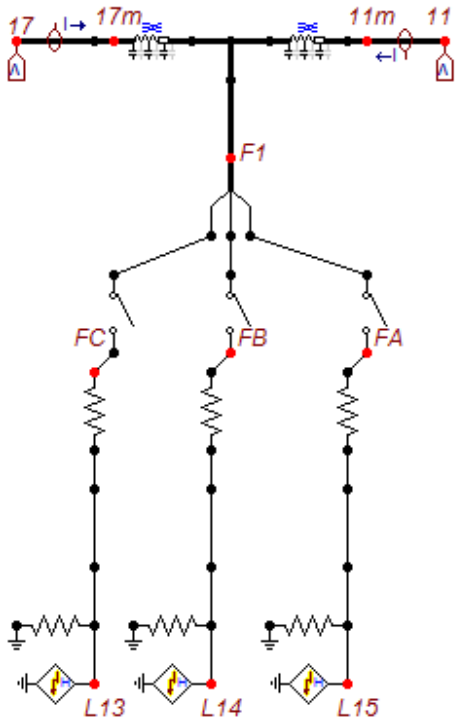

Fig. 6. Line division, insertion of probes and lightning source is complete. At this step, block parameters and sections' lengths is complete.

user. Giving the call back to the original for loop in the BatchProcessATP function, the function starts over till all cases are simulated. In summary, the function inserts a generic fault block, adjusts its parameters to correspond to the fault case, splits the line into two sections, renames the nodes of those sections, adjusts the length of the same two sections, calls runATP.exe and finally calls Pl42mat.exe to convert the pl4 output simulation file to mat files to be used with Matlab.

B. Creation of lightning case (LightStrike function)

Once again the LightStrike function can be used outside the BatchProcessATP function to create one lightning case. We now proceed to describe the logic implemented in this function.

As has been stated above, creating a lightning case involved dividing the line being hit into two sections and inserting a generic lightning block. The generic block being inserted is shown in fig. 5. The block when inserted function as open circuit by opening the switches F1-FA, F1-FB, F1-FC. The block as shown consists of three switches and three Heidler type lightning sources. Once the block is inserted, the function reads the data structure created by CreateScenarios function. The switches opening and closing times, the rising

and the tailing times, and the amplitude of the strike are all adjusted according to the data structure being created by CreateScenarios function.

Following the same lines of thought as we did with creating faults, we divide the lines into two sections. After dividing the lines into two sections, we insert probes at both ends, rename all the nodes and at the end modify the length of both sections to complete modifying the ATP file. As an example, if we want to create a lightning strike on line 11-17 then fig. 6 shows the last form of the line after we inserted the lightning block, probes and renamed all nodes.

C. *Creating Line Switching cases (SwitchLine function)*

Creation of line switching cases is the easiest and the most straightforward task. What is needed is not insertion of generic fault and lightning blocks or dividing the line into sections but only insertion of two switches at the both terminals, current and voltage probes at both ends.

First, a special function inserts both switches at both ends of the line. After that the call is passed to another function that inserts current and voltage probes at both ends. After that both switches' opening and closing times are adjusted according the data structure created by CreateScenarios function. At last, nodes are being renames not to cause error when running

Fig. 7. Addition of switches at both ends of the line, changing closing and opening times of the switches according to user input and finally addition of voltage and current probes at both ends.

runATP.exe while ATP is building the connectivity matrix. An example is when we try to energize line 11- 17. The block corresponding to the above operation is given in fig. 7 after all operations are complete.

IV. Toolbox Flexibility and Output File Names

In this section we highlight some features of the toolbox that emphasizes its flexibility. We also show how the toolbox gives names to its output mat files.

Up till now very little has been said regarding how generic fault and lightning models are inserted. The lightning and fault generic blocks are actually copied from a text file shipped with the toolbox to the temporary ATP file created in the course of execution of CreateFault and LightStrike functions. This gives the user the flexibility in inserting whatever he wants in the ATP file. The flexibility comes from the fact that those generic blocks are not hard coded in the toolbox. If the user for example wants instead of using a Heidler type lightning source to use another source, he can modify the text file corresponding to the generic lightning block accordingly. However, the user still needs to modify the function that changes the parameters of the lightning strike.

Toolbox flexibility stems from the search functions it contains. The toolbox has built in search functions that can locate certain nodes, lines or sections in the ATP file. The search functions are used to locate any component but the user has to supply the names of the two terminals of the device. The functions can locate lines, switches and probes based only the names of the nodes of those components.

The flexibility also comes from the function that inserts the probes. Current and voltage probes are actually copied from a text file included with the toolbox to the temporary ATP file being created by BatchProcessATP. This function pastes two current probes and two voltage probes to measure all terminal currents and voltages of the line under study. If the user wants to have other measuring devices at other nodes, he needs to add those devices in that text file. The function that inserts the probes into the temporary ATP file employs special search function to find the generic probes that have been inserted to rename the nodes. If the user wishes to add more probes, he needs to adapt this function to such task which is a straight forward task.

A systematic way to rename the output mat files is also offered. For faults, any mat file is of this form "Case_LineUnderStudy_FaultType_FaultResistance_FaultIncipientAngle_FaultDistance". For example an AG fault on line 11-17 at a distance of 60% from node 11 and an incipient angle of 30 degrees and fault resistance of 10 Ohms will have the following name: Case_11-17_AG_10Ohms_30d_60percent.mat.

For lightning on the other hand the following naming conversion is offered: Case_LineUnderStudy_PhaseBeingStroke_LightningAmplitude_IncipientAngle_Distance. For example a lightning striking phase A of line 11-17 having amplitude of 5000 Amperes at a distance of 60% of length from bus 11 at a 30 degrees incipient will have the following name: Case_11-17_5000A_30d_60percent.mat.

For line switching the following naming conversion is being offered: Case_LineUnderStudy_Angle_Distance, where distance is either 0 or 100 based on which end being switched. For example if we energize line 11-17 using the end 11 then distance is zero else it is 100.

The toolbox has been tested on a Windows 7 running on a 3GHz Intel core i7 processor with 8 GB of ram. The toolbox made changes to IEEE 118 bus ATP file before running ATP in less than 0.24 seconds in any case whether it was a fault, a line switching or lightning strike. This a tiny overhead compared to the running time of ATP program itself.

V. Conclusions

The paper presented a systemic way for creating a batch of EMTP studies. The algorithm given in the function BatchProcessATP is used to automate creation of fault, lightning strike and line switching cases. The toolbox provides a rich set of m files that can be adapted to search and manipulate ATP files. The toolbox features a function that converts user input to a data structure that can be understood by the toolbox. All functions make use of search subroutines that is aimed to search ATP files effectively. The toolbox is

very flexible in terms of changing its code to accommodate new studies or needs of the users. We provide the toolbox as open source in a git repository for all engineers, researchers and students. Users can change or modify the code for their needs and can contact the author to push their contributions to the main git repository online. The logic implemented in this paper can be used to build similar toolboxes targeting other EMTP programs as well. This is ongoing effort that will be expanded and enhanced over the course of the next release versions. In the end, a tutorial on how to use the toolbox is given online in the git repository. The tutorial includes generation fault, lightning and line switching cases.

ACKNOWLEDGMENT

The author would like to acknowledge the help of Po-Chen Chen of Texas A&M University for his help solving many ATP related problems.

REFERENCES

[1] M. Kezunović, Igor Rikalo, D.J. Šobajić, High-speed fault detection and classification with neural nets, *Electric Power Systems Research*, Volume 34, Issue 2, August 1995, Pages 109-116

[2] Gaouda, A.M.; Kanoun, S.H.; Salama, M.M.A.; Chikhani, A.Y., "Wavelet-based signal processing for disturbance classification and measurement," *Generation, Transmission and Distribution, IEE Proceedings-* , vol.149, no.3, pp.310, 318, May 2002

[3] Bo, Z.Q.; Jiang, F.; Chen, Z.; Dong, X.Z.; Weller, G.; Redfern, M.A., "Transient based protection for power transmission systems," Power Engineering Society Winter Meeting, 2000. IEEE , vol.3, no., pp.1832,1837 vol.3, 23-27 Jan 2000

[4] Martin, F.; Aguado, J.A., "Wavelet-based ANN approach for transmission line protection," *Power Delivery, IEEE Transactions on* , vol.18, no.4, pp.1572,1574, Oct. 2003

[5] Ibrahim, W.R.A.; Morcos, M.M., "Artificial intelligence and advanced mathematical tools for power quality applications: a survey," Power Delivery, IEEE Transactions on , vol.17, no.2, pp.668,673, Apr 2002

[6] G. D. Guidi-Venerdini; F. E. Pérez-Yauli, "MATLAB Program for Systematic Simulation over a Transmission Line in Alternative Transients Program, " International Conference on Power System Transients IPST 2013, Vancouver, BC, Canada, July 18-20, 2013

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

[8] [online] https://bitbucket.org/ahmadmabdullah/atpmat

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

[10] A Abdullah, A Esmaeilian, G Gurrala, P Dutta, T Popovic, M Kezunovic, "Test Bed for Cascading Failure Scenarios Evaluation", International Conference on Power System Transients IPST 2013, Vancouver, BC, Canada, July 18-20, 2013

[11] Power Systems Test Case Archive, [online] https://www.ee.washington.edu/research/pstca/pf118/pg_tca118bus.htm

[12] http://www.emtp.org/