UFMG – FEDERAL UNIVERSITY OF MINAS GERAIS
LRC – LIGHTNING RESEARCH CENTER
BELO HORIZONTE – BRAZIL

USER MANUAL

ULM-ATP

VERSION 2.0

(July 2021)

Felipe O. S. Zanon[1] felipeos.zanon@gmail.com
Osis E. S. Leal[2] osisleal@utfpr.edu.br
Alberto De Conti[3] conti@cpdee.ufmg.br

1. Graduate Program on Electrical Engineering (PPGEE), Federal University of Minas Gerais (UFMG), Brazil.
2. UTFPR – Federal University of Technology – Paraná, Brazil.
3. Department of Electrical Engineering, Lightning Research Center (LRC), Federal University of Minas Gerais (UFMG), Brazil.

Revision history

Version	Date	Description of Revision
1.0	06-March-2021	Initial release.
2.0	24-July-2021	✓ Simplifications in the MODELS code. ✓ Modifications in the algorithm to increase efficiency and to allow simulations with multiple line sections.

Index

1 - Introduction

The implementation of ULM in ATP follows a strategy that combines the use of a external code, in the first stage, and ATP, in a second stage. The external code was entirely developed in MATLAB as shown in Fig 1.1. Initially, the user enters the transmission line data through a graphical interface developed in the GUIDE environment of MATLAB [1-2]. The associated code is responsible for calculating the line parameters, the time delays, the characteristic admittance $\boldsymbol{Y_c}$ and the propagation function $\boldsymbol{H}$ required in ULM [3], plus their fitting. In the end, a text file is generated containing the poles and residues of $\boldsymbol{Y_c}$ and $\boldsymbol{H}$, the elements of the conductance matrix $\boldsymbol{G}$ required in the implementation, and the minimum time delays associated with each mode. This file acts as a link between the external code and ATP, containing all information necessary to perform the transient simulation in ULM.

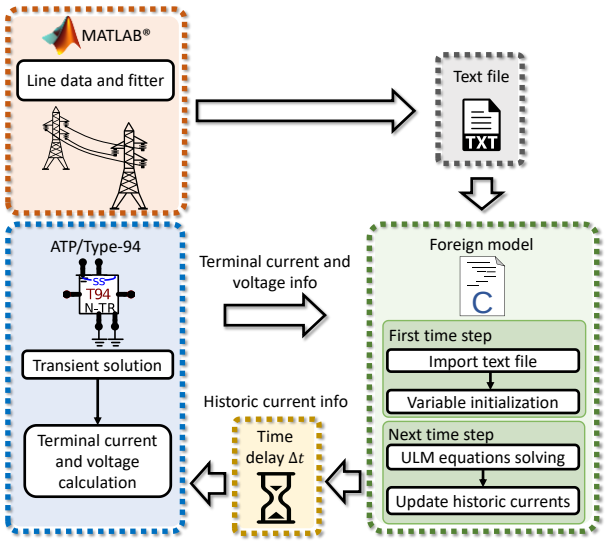


Fig. 1.1. Diagram of the strategy used in the implementation of ULM in ATP.

In the second stage, a foreign model implemented in ATP reads the text file generated by the external code. In this model, the ULM equations were implemented in ANSI C language. The program initializes the variables necessary for calculating the transient in the first time step. From the voltages and currents at the line terminals, the foreign model calculates the history terms, feeding back a type-94 component in ATP at each time step. Since the communication of the foreign model with ATP occurs with a delay of one time step [1], this effect is compensated in the calculation of the current sources containing the history terms. Finally, ATP returns the values of the terminal

voltages to the foreign model.

This document discusses the implementation of ULM as a foreign model in ATP, referred to as ULM-ATP. In the current version, it does not cover the preprocessing code written in MATLAB for performing the per-unit-length parameter calculation and fitting of model parameters. However, it provides detailed instructions on how to prepare the input data for use in the ULM-ATP model.

Users are free to use all support files and codes available on https://github.com/zanonfelipe/ULMAtp, but are kindly requested to include reference to the following paper:

Felipe O.S. Zanon, Osis E.S. Leal, Alberto De Conti, Implementation of the universal line model in the alternative transients program, Electric Power Systems Research, vol. 197, p. 107311, Aug. 2021, doi: 10.1016/j.epsr.2021.107311.

2 - Installation

The ATP software is executed using the file commonly called mytpbig.exe. New functions or new models can be added to the mytpbig.exe file and, consequently, to the ATP software. To incorporate new functions or new models, it is necessary to recompile ATP software. This chapter describes how to add the ulm.o model developed by the authors using the ATPLaucher1.16 tool, developed by Japanese ATP User Group (JAUG), together with the MinGW 5.1.0 compiler. In order to integrate the ulm.o model with ATP software, the user must strictly follow the steps described below.

1) Extract all files from "MinGW_5_1_0.zip" to "[dirInstall]\atpmingw\make\MinGW"[1], overwriting existing files and merging existing folders (this step is only necessary if the MinGW compiler is not up to date).
2) Copy "fgnmod.f" to "[dirInstall]\atpmingw\make" overwriting the existing file.
3) Copy "makefile_c" to "[dirInstall]\atpmingw\make\sample" overwriting the existing file.

If the root directory "[dirInstall]" of the ATP software installation is "C:\ATP" skip to step 4, else do:

3.1) open the makefile_c file.

3.2) correct the directories entered in the lines 10, 11 and 12 (for more details see chapter 5).

4) Copy "ulm.o" to "…[dirInstall]\atpmingw\make"
5) Copy "fitULM.txt" to "…\Documents\ATPdata\work"[2]
6) Open 'ULMtype-94.acp' and do some configurations:
7) Click on 'ATP' option on the menu bar
8) Select 'ATP Launcher'
9) In the appearing dialog, click on 'Tools'

[1] [dirInstall] represents the root installation directory of the user defined ATP at the time of installation. The default root directory is c:\ATP.

[2] ATPDraw software saves ATP cards in the directory called ATP folder. The ATP folder directory is set, by default in ATPDraw software, to the "work" folder (example: C:\ATP\ATPDraw\work). In this case, the ULM000000.in file should be copied to C:\ATP\ATPDraw\LCC. It is important that the work and LCC folders must be in the same directory. To correctly locate the directory that will be sent to ATP cards by ATPDraw software click on the "Tools" menu, then "Options". On the ATPDraw Options screen, click the File&Folders tab. ATP cards will be forwarded to the directory described in the textbox called "ATP folder".

10) Select “Make Tpbig.exe”

11) In the appearing dialog, set:

11.1) MyTpbig Directory: “[dirInstall]\mytpbig”

11.2) Make sure that “Make compiled TACS” and “Use Default Makefile” are unchecked.

11.3) Makefile: “[dirInstall]\atpmingw\make\sample\Makefile_c”.

11.4) MinGW Directory: “[dirInstall]\atpmingw\make\MinGW\bin” (an example for installing ATP software in the default directory is shown in Figure 2.1).

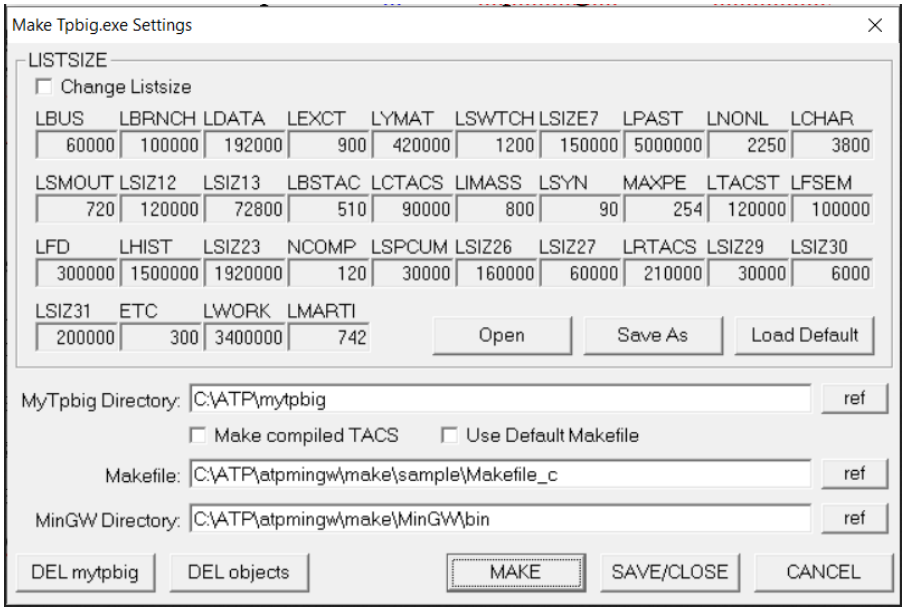


Fig. 2.1 – Make Tpbig.exe Settings.

12) Click on the “Make” button.

13) The massage “[dirInstall]\mytpbig\mytpbig.exe was successfully created!!” should appear.

14) Run the simulation.

3 - Type94 Configuration

The interface between the ATP software and ULM is performed by the Type 94 Norton TR model. This model allows the inclusion of new line models to ATP. In ATPDraw it is possible to access the Type 94 Norton TR model by right-clicking on the MODELS/TYPE94/NORT-TR menu. The Type 94 Norton TR model configuration screen is shown in Fig. 3.1. In order to assure that the model will work properly, select the "NORTON-transmission" and set the number of line phases in "NumPh" field, as shown in Fig. 3.1. After the initial configuration, it is necessary to edit the standard MODELS code so that it represents the ULM line model. To do this, click on the "Edit" button.

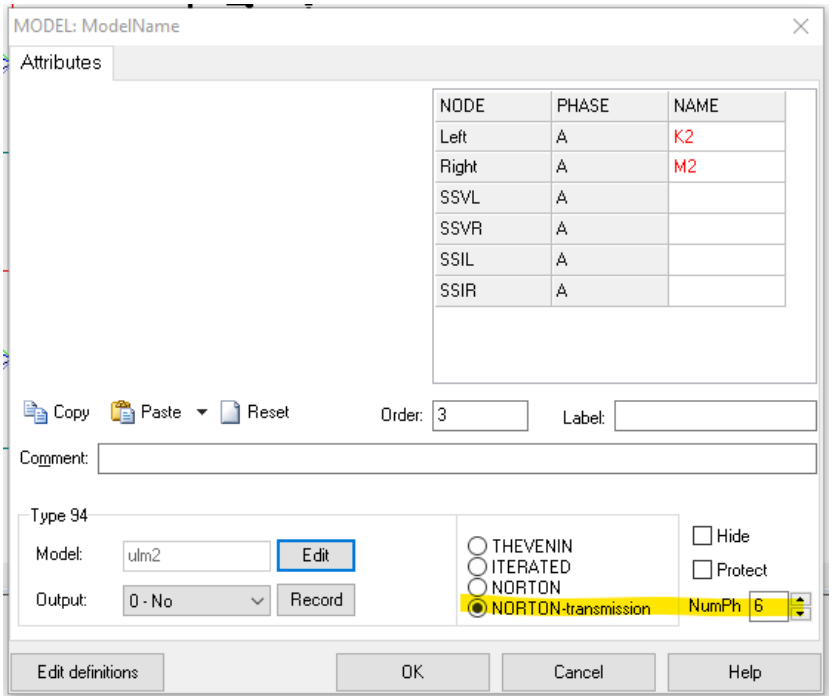


Fig. 3.1 – Type 94 Norton TR model configuration.

The standard MODELS code for the Type 94 TR model is:

```
MODEL name            -- NB! 6 character name limit
-- Start header. Do not modify the type-94 header.
 comment-----------------------------------------------------------------
 | First, declarations required for any type 94 Norton TR model         |
 | - these data and input values are provided to the model by ATP       |
 | - these output values are used by ATP                                |
 | - these names can be changed, except 'n', but not their order        |
 ------------------------------------------------------------------endcomment
DATA  n                      -- number of phases
      ng {dflt: n*(n+1)/2}   -- number of conductances on each side

INPUT lv[1..n]    -- voltage(t) at each left node
      rv[1..n]    --    "       "   "   right node
      lv0[1..n]   -- voltage(t=0) at each left node
      rv0[1..n]   --    "         "   "   right node
      li0[1..n]   -- current(t=0) into each left node
      ri0[1..n]   --    "           "   "   right node
VAR   li[1..n]    -- current(t) into each left node (for plotting)
      ri[1..n]    --    "         "   "   right node  "     "
      lis[1..n]   -- Norton source(t+timestep) at each left node
      ris[1..n]   --    "      "                  "  "   right node
      lg[1..ng]   -- conductance(t+timestep) at each left node
      rg[1..ng]   --    "                     "  "   right node
                  -- sequence is 1-gr, 1-2, 1-3..1-n,2-gr,2-3..2-n,...n-gr
      flag        -- set to 1 whenever conductance value is modified
OUTPUT li[1..n],ri[1..n],lis[1..n],ris[1..n],lg[1..ng],rg[1..ng],flag
 comment-----------------------------------------------------------------
 | Next, declarations of user-defined data for this particular model    |
 | - their value is defined at the time of using the type-94 component  |
 ------------------------------------------------------------------endcomment
-- End header
DATA ...
 comment-----------------------------------------------------------------
 | Next, declarations provate to this model                             |
 ------------------------------------------------------------------endcomment
VAR ...
 ...
INIT
 ...
ENDINIT
EXEC
 ...
ENDEXEC
ENDMODEL -- End code
```

As described in the standard MODELS code, the user must make any desired changes only after the set of OUTPUT variables. The standard MODELS code already modified to use the ULM model is:

```
MODEL ulm2_0 -- NB! 6 character name limit
 DATA  n                      -- number of phases
       ng {dflt: n*(n+1/2}    -- number of conductances
       nLine                  -- number of line
       case                   -- number of case

 INPUT lv[1..n]    -- voltage(t) at left terminals
       rv[1..n]    --    "    "     right terminals
       lv0[1..n]   -- voltage(t=0) at left nodes
       rv0[1..n]   --    "    "    "  right nodes
       li0[1..n]   -- current(t=0) at left terminals
       ri0[1..n]   --    "    "    "  right terminals

 VAR   li[1..n]    -- current(t) into left terminals
       ri[1..n]    --    "    "    "  right terminals
       lis[1..n]   -- Norton source(t+timestep) at left terminals
       ris[1..n]   --    "    "    "    "    "     right terminals
       lg[1..ng]   -- conductance(t+timestep) at left terminals
       rg[1..ng]   --    "    "    "    "    "   right terminals
                   -- sequence is 1-gr, 1-2, 1-3..1-n, 2-gr, 2-3..2-n, ... n-gr
                   --  on each side separately
       flag        -- set to one whenever an admittance value is modified
 OUTPUT li[1..n], ri[1..n],lis[1..n], ris[1..n], lg[1..ng], rg[1..ng], flag
 comment ------------------------------------------------------------------
 | Next, declarations private to the operation of this model                |
  ------------------------------------------------------------------ endcomment
 VAR  Yc[1..n*n]
 MODEL osis FOREIGN ULM_LINE {ixdata:8, ixin:2*n, ixout:(2*n+ng), ixvar:2*n}
 INIT
  lg[1..ng] :=0
  rg[1..ng] :=lg[1..ng]
  li[1..n]  :=0     -- current for this step
  ri[1..n]  :=0
  lis[1..n] :=0
  ris[1..n] :=0
 ENDINIT
 EXEC
  li[1..n] :=0
  ri[1..n] :=0
  USE osis AS zanon2de_conti
    DATA xdata[1]:=timestep
         xdata[2]:=nLine -- direcionador para o arquivo conténdo os dados da linha
         xdata[3]:=case  -- direcionador para o aquivo padrão de leitura
    INPUT xin[1..n]        :=lv[1..n]
          xin[(n+1)..(2*n)]:=rv[1..n]
    HISTORY xvar[1..(2*n)]:=0
    OUTPUT lis[1..n]:=xout[1..n]
           ris[1..n]:=xout[(n+1)..(2*n)]
           lg[1..ng]:=xout[(2*n)+1..(2*n+ng)]
  ENDUSE
  IF t=0 THEN
   flag := 1          -- conductance values have been changed
   rg[1..ng] :=lg[1..ng]
  ELSE
   flag := 0          -- reset flag
  ENDIF
  ENDEXEC
ENDMODEL -- End code
```

After modifying the standard MODELS code, the dialog box shown in Fig. 3.2 appears on the screen, click on the OK button.

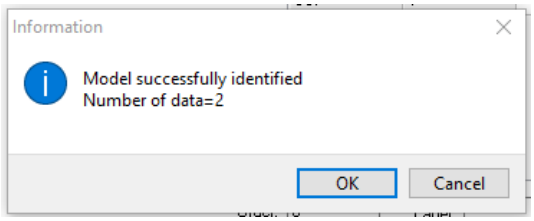


Fig. 3.2 – Dialog box.

Then, the Type 94 Norton TR model configuration screen is modified. The new configuration screen is shown in Fig. 3.3. In this new screen it is necessary to configure two new fields, namely “nLine” and “case”. The “nLine” field is used to connect the line to its respective fitted data written in a text file, whose path must be informed in the ULM000000.in file. On the other hand, the “case” field is used to select the default file called ULM000000.in.

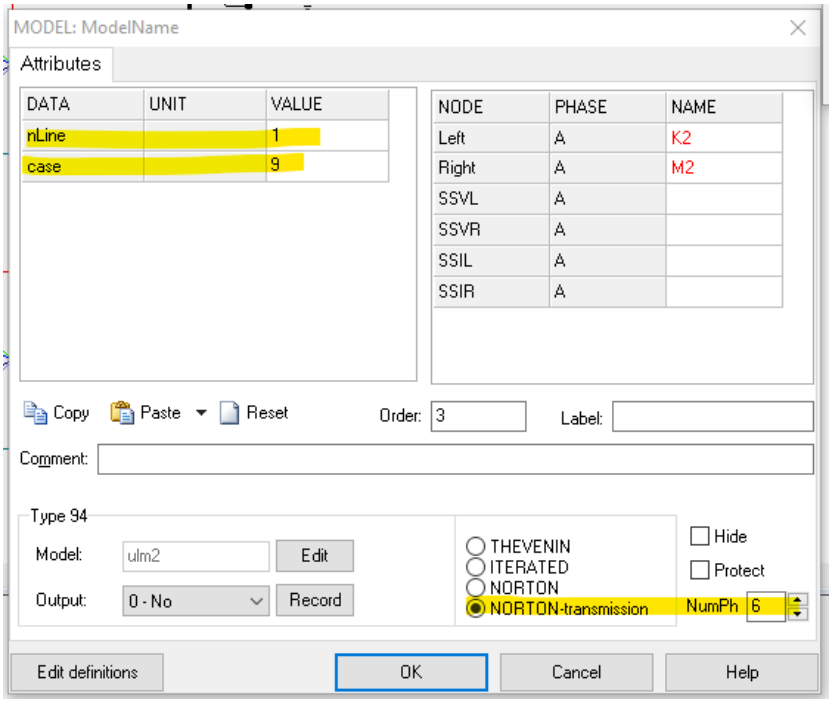


Fig. 3.3 – The new Type 94 Norton TR model configuration.

The link between the line and the file containing its parameters is made in the ULM000000.in file, where each line of the ULM000000.in file describes the path to a line file. The line file described in line 1 of ULM000000.in file represents the data of the line whose value in the “nLine” filed was set to 1. The line file described in line 2 of ULM000000.in file represents the data of the line whose value in the “nLine” field was set with 2 and so on. Each line of ULM000000.in file must be terminated by the special character “$”. In Fig. 3.4 an example of ULM000000.in file is shown.

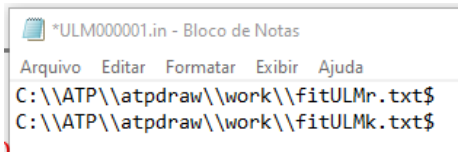


Fig. 3.4 – ULM000000.in file example.

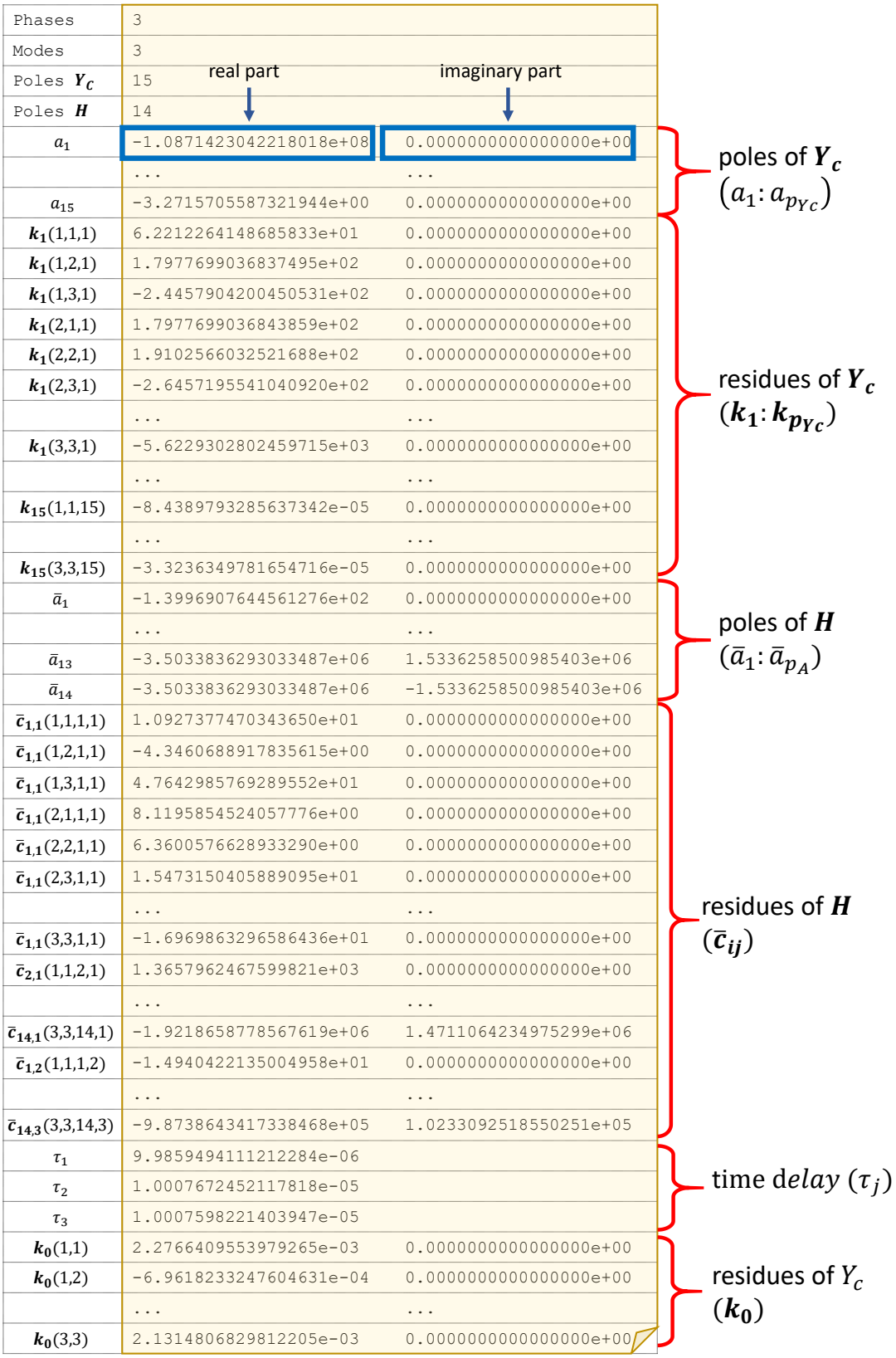
The last 6 characters of name ULM000000.in file are used to differentiate simulations or case studies and must be numerical values. Therefore, the value entered in the case field must match the value entered in the last 6 characters of ULM000000.in file. Then the user can use these characters to differentiate their simulations. For example, in Fig. 3.3 the "case" field is equal 9. In this condition, the ULM code will search for the ULM000009.in file.

4 - fitULM Structure

The line information calculated by the external code is recorded in a text file that stores the necessary information for the transient simulation in ATP [1]. The file follows the structure shown in Figure 4.1, which refers to a 3-phase line. The default for the text file is to insert one element per line, whether it comes from a vector or a matrix. If the element is complex, both the real part and the imaginary part are inserted on the same line. The real part is written at the beginning and the imaginary part is written next. Both are separated by a tab in the text file. The sequence of recording the information is described below:

- Number of phases (n);
- Number of modes (m);
- Number of poles p_{Yc} resulting from the fitting of $\boldsymbol{Y}_c$
- Number of poles p_A resulting from the fitting of $\boldsymbol{H}$;
- Vector of poles resulting from the fitting of $\boldsymbol{Y}_c$, that is, $a_1 : a_{p_{Yc}}$;
- Matrices of residues $\boldsymbol{k}_1 : \boldsymbol{k}_{p_{Yc}}$ resulting from the fitting of $\boldsymbol{Y}_c$, included element by element. The $\boldsymbol{k}_{n_{th}}(n, n, p_{Yc})$ matrix is three-dimensional.
    - First enter the elements from the first row of $\boldsymbol{k}_1$, that is $\boldsymbol{k}_1(1,1,1)$, $\boldsymbol{k}_1(1,2,1)$, $\boldsymbol{k}_1(1,3,1)$, for this example. Then, enter the elements from second row of $\boldsymbol{k}_1$, and finally the elements from the last row of $\boldsymbol{k}_1$, that is, $\boldsymbol{k}_1(3,1,1)$, $\boldsymbol{k}_1(3,2,1)$, $\boldsymbol{k}_1(3,3,1)$.
    - Next, enter the elements of $\boldsymbol{k}_2$, following the procedures described above, and repeat for all $\boldsymbol{k}_{n_{th}}$ matrices.
    - Last matrix is $\boldsymbol{k}_{15}$, in this example. So, enter the elements $\boldsymbol{k}_{15}(1,1,15)$, $\boldsymbol{k}_{15}(1,2,15)$, $\boldsymbol{k}_{15}(1,3,15)$, firstly. Then enter the elements $\boldsymbol{k}_{15}(2,1,15)$, $\boldsymbol{k}_{15}(2,2,15)$, $\boldsymbol{k}_{15}(2,3,15)$, and finally the elements $\boldsymbol{k}_{15}(3,1,15)$, $\boldsymbol{k}_{15}(3,2,15)$, $\boldsymbol{k}_{15}(3,3,15)$.
- Vector of poles resulting from the fitting of $\boldsymbol{H}$, that is, $\bar{a}_1 : \bar{a}_{p_A}$;
- Matrices of residues $\bar{\boldsymbol{c}}_{\boldsymbol{ij}}$ resulting from the fitting of $\boldsymbol{H}$, included element by element. The matrix $\bar{\boldsymbol{c}}_{\boldsymbol{ij}}(n, n, p_A, m)$ has four dimensions.
    - First enter the elements from the first row of $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}$, that is $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(1,1,1,1)$, $\bar{\boldsymbol{c}}_{\boldsymbol{11,}}(1,2,1,1)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(1,3,1,1)$, for this example. Then, enter the elements from second row of $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}$, and finally the elements from the last row of $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}$, that is, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(3,1,1,1)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(3,2,1,1)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(3,3,1,1)$.
    - Next, enter the elements of $\bar{\boldsymbol{c}}_{\boldsymbol{2,1}}$, following the procedures described above, and repeat for all $\bar{\boldsymbol{c}}_{\boldsymbol{i_{th},1}}$ matrices.
    - After, enter the elements from the first row of $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}$, that is $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(1,1,1,2)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(1,2,1,2)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(1,3,1,2)$. Then, enter the elements from second row of $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}$, and finally the elements from the last row of $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}$, that is, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(3,1,1,2)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(3,2,1,2)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(3,3,1,2)$.

  - Next, enter the elements of $\bar{\boldsymbol{c}}_{\boldsymbol{2,1}}$, following the procedures described above, and repeat for all $\bar{\boldsymbol{c}}_{\boldsymbol{i_{th},2}}$ matrices.
  - Repeat the procedures above for all $\bar{\boldsymbol{c}}_{\boldsymbol{i_{th},j_{th}}}$ matrices.
  - Last matrix is $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}$ in this example. So enter the elements from the first row of $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}$, that is $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}(1,1,14,3)$, $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}(1,2,14,3)$, $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}(1,3,14,3)$, for this example. Then, enter the elements from second row of $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}$, and finally the elements from the last row of $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}$, that is, $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}(3,1,14,3)$, $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}(3,2,14,3)$, $\bar{\boldsymbol{c}}_{\boldsymbol{14,3}}(3,3,14,3)$.
- Time delays;
- Matrix containing the elements of the independent term $\boldsymbol{k}_0$ resulting from the fitting of $\boldsymbol{Y}_c$, included element by element. The matrix $\boldsymbol{k}_0(n,n)$ is two-dimensional.



Phases	3	
Modes	3	
Poles Y_C	15	
Poles H	14	
a_1	-1.0871423042218018e+08	0.0000000000000000e+00
	...	...
a_{15}	-3.2715705587321944e+00	0.0000000000000000e+00
$k_1(1,1,1)$	6.2212264148685833e+01	0.0000000000000000e+00
$k_1(1,2,1)$	1.7977699036837495e+02	0.0000000000000000e+00
$k_1(1,3,1)$	-2.4457904200450531e+02	0.0000000000000000e+00
$k_1(2,1,1)$	1.7977699036843859e+02	0.0000000000000000e+00
$k_1(2,2,1)$	1.9102566032521688e+02	0.0000000000000000e+00
$k_1(2,3,1)$	-2.6457195541040920e+02	0.0000000000000000e+00
	...	...
$k_1(3,3,1)$	-5.6229302802459715e+03	0.0000000000000000e+00
	...	...
$k_{15}(1,1,15)$	-8.4389793285637342e-05	0.0000000000000000e+00
	...	...
$k_{15}(3,3,15)$	-3.3236349781654716e-05	0.0000000000000000e+00
$\bar{a}_1$	-1.3996907644561276e+02	0.0000000000000000e+00
	...	...
$\bar{a}_{13}$	-3.5033836293033487e+06	1.5336258500985403e+06
$\bar{a}_{14}$	-3.5033836293033487e+06	-1.5336258500985403e+06
$\bar{c}_{1,1}(1,1,1,1)$	1.0927377470343650e+01	0.0000000000000000e+00
$\bar{c}_{1,1}(1,2,1,1)$	-4.3460688917835615e+00	0.0000000000000000e+00
$\bar{c}_{1,1}(1,3,1,1)$	4.7642985769289552e+01	0.0000000000000000e+00
$\bar{c}_{1,1}(2,1,1,1)$	8.1195854524057776e+00	0.0000000000000000e+00
$\bar{c}_{1,1}(2,2,1,1)$	6.3600576628933290e+00	0.0000000000000000e+00
$\bar{c}_{1,1}(2,3,1,1)$	1.5473150405889095e+01	0.0000000000000000e+00
	...	...
$\bar{c}_{1,1}(3,3,1,1)$	-1.6969863296586436e+01	0.0000000000000000e+00
$\bar{c}_{2,1}(1,1,2,1)$	1.3657962467599821e+03	0.0000000000000000e+00
	...	...
$\bar{c}_{14,1}(3,3,14,1)$	-1.9218658778567619e+06	1.4711064234975299e+06
$\bar{c}_{1,2}(1,1,1,2)$	-1.4940422135004958e+01	0.0000000000000000e+00
	...	...
$\bar{c}_{14,3}(3,3,14,3)$	-9.8738643417338468e+05	1.0233092518550251e+05
τ_1	9.9859494111212284e-06	
τ_2	1.0007672452117818e-05	
τ_3	1.0007598221403947e-05	
$k_0(1,1)$	2.2766409553979265e-03	0.0000000000000000e+00
$k_0(1,2)$	-6.9618233247604631e-04	0.0000000000000000e+00
	...	...
$k_0(3,3)$	2.1314806829812205e-03	0.0000000000000000e+00

Fig. 4.1 – "fitULM" text file structure.

5 - ATP Compilation Files

5.1 - Makefile_c

A makefile gathers the set of instructions necessary for the automatic compilation of a software. For the compilation of ATP software including new functions programmed in C language in mytpbig.exe, the makefile is called “Makefile_c”. This file is responsible for telling the compiler which files and libraries are linked to ATP software, including saying which compiler will do that task. To include the ULM model, the ATP software standard “Makefile_c” file must be modified as described on the next page. The changes were made in line 23.

In the “Makefile_c” it is necessary to set some directories, see lines 32, 34 and 46 on the next page. These directories are used to inform the compiler in which folder the libraries, functions and new models are stored therefore they will be incorporated into mytpbig.exe. In addition, the directory where the new executable mytpbig.exe will be stored is also informed (line 12). In cases where the user does not choose installing ATP software and the MINGW compiler in the default directory “C:\ATP”, that is, customized installation, special attention should be given to lines 10, 11 and 12 because it will be necessary to correct the directories so that they correspond to the customization carried out by the user.

```
# ------------------------------------------------------
#            By Ms. Felipe O. S. Zanon - UFMG
#            Prof. Osis E. S. Leal - UTFPR
#            Prof. Alberto De Conti - UFMG
# Federal University of Minas Gerais - (UFMG)
# Federal University of Technology – Paraná – (UTFPR)
# ------------------------------------------------------
CC=gcc
FOR=g77
GWDIR=C:\ATP\atpmingw\make
GWDIRL=C:\ATP\atpmingw\make\MinGW\lib\gcc\mingw32\5.1.0
GWMY= C:\ATP\mytpbig
OBJECTS =dimdef.o \
          newmods.o \
          comtac.o \
          fgnmod.o \
          usernl.o \
          analyt.o \
          devt69.o \
          usrfun.o \
          hopcod.o \
          user10.o \
          $(GWDIR)\ulm.o

INSFILE =blkcom.ins \
          comta1.ins \
          comta2.ins \
          comta3.ins \
          dekspy.ins \
          tacsar.ins \
          space2.ins

CFLAGS = -DUNDERSCORE -O2
FFLAGS = -O2
          IMAGE= $(GWMY)\mytpbig.exe

LIBRARY = $(GWDIR)\tpbig.a $(GWDIR)\dismg7.a -lgdi32 $(GWDIRL)\libgcc.a

all: $(IMAGE)

.f.o:
          $(FOR) -c $(FFLAGS) $<
.c.o:
          $(CC) -c $(CFLAGS) $<

$(IMAGE) : $(OBJECTS) $(INSFILE)
            $(FOR) -s -o $(IMAGE) $(OBJECTS) $(LIBRARY)

```

5.2 - fgnmod.f file

In the fgnmod.f file, described in the Fortran programming language, all foreign models are registered. In the subroutine "FGNMOD" it is possible to register foreign models as they are declared in the models present in ATP. In other words, in the fgnmod.f file, the name of the new foreign model is declared, correlating to its respective source code.

The ulm.o file follows all the syntax and programming of a common ANSI C code. However, its structure must contain the functions “ULM_i” (function executed in the first time step) and “ULM_m” (executed in the next time steps). These functions must be registered in the file “fgnmod.f” (see lines 63 and 65 of the code described on the next page, taken from the fgnmod.f file used). The variables xdata, xin, xout, xvar are ATP standards and must be present in the C program because they link ATP to MODELS.

In the MODELS code (as in the examples shown in chapter 3) the foreign model is called by the name assigned to its respective line of the variable “refname”. ULM was registered on refname line 2 as ULM_LINE (see line 19 on the next page).

```
1         SUBROUTINE FGNMOD ( name, namlen, xdata, xin, xout, xvar,
2        1                    iniflg, ierflg)
3         IMPLICIT REAL*8 (A-H, O-Z),  INTEGER*4 (I-N)
4         DIMENSION xdata(*), xin(*), xout(*), xvar(*)
5         CHARACTER*1 name(*)
6         PARAMETER ( namcnt = 8 )
7         CHARACTER*32 refnam(namcnt)
8         CONTINUE !  ------------------------------------------------
9         CONTINUE !  You may increase namcnt above to allow more names:
10        CONTINUE !  ------------------------------------------------
11        CONTINUE !  In the following lines, register your foreign model
12        CONTINUE !  names as they are declared in your models:
13        CONTINUE !   - use only uppercase characters for the name here
14        CONTINUE !   - you can use any case for the name in the models
15        CONTINUE !   - make a copy of the modifications you make to this
16        CONTINUE !     file so that you don't lose them when installing
17        CONTINUE !     a newer version of ATP later
18        DATA refnam(1) / 'SAMPLE_MODEL' /  ! Do not modify this line
19        DATA refnam(2) / 'ULM_LINE' /
20        DATA refnam(3) / ' ' /
21        DATA refnam(4) / ' ' /
                  (⋮)
47        CONTINUE !  ------------------------------------------------
48        CONTINUE !  In the following lines, this is where you call the
49        CONTINUE !  actual foreign subroutines/procedures:
50        CONTINUE !   - actual names may be different from the foreign
51        CONTINUE !     names used in the models
52        CONTINUE !   - notice how each one uses both an
53        CONTINUE !     initialization routine and an execution routine
54        IF ( iname.EQ.1 ) THEN
55         IF (iniflg.EQ.1) THEN
56          CALL sampli(xdata, xin, xout, xvar)
57         ELSE
58          CALL samplm(xdata, xin, xout, xvar)
59         ENDIF
60        CONTINUE !      ----------------------------------------
61        ELSE IF ( iname.EQ.2 ) THEN
62         IF (iniflg.EQ.1) THEN
63          CALL ulm_i(xdata, xin, xout, xvar)
64         ELSE
65          CALL ulm_m(xdata, xin, xout, xvar)
66         ENDIF
67        CONTINUE !      ----------------------------------------
                  (⋮)
```

6 - Application example

The example considered in this section is taken from [1]. It deals with a 10-km long 230-kV three-phase line with two shield wires and a 115-kV three-phase horizontal line running in parallel (see [1] for configuration details). A step voltage was applied at the sending end of the 230-kV line, on phase *a*, assuming all remaining line terminals to be grounded on terminal *k* (sending end) and open-ended on terminal *m* (receiving end). The voltages induced on the 115-kV line were calculated considering terminal *k* grounded and terminal *m* open-ended. In all simulations, the fitting was performed with complex poles and residues from 10^{-1} to 10^{8} Hz considering a shunt conductance of 0.2×10^{-9} S/m. The fitting process considered 20 poles for $\boldsymbol{Y_C}$ and $\boldsymbol{H}$, and the ground return impedance was calculated using Carson's integral equations [5].

Fig. 6.1 shows voltages calculated at the receiving end of the 115-kV line considering the ULM component implemented in ATP (ULM-ATP) and ULM available in EMTP-RV (ULM-RV) for a soil resistivity of 100 Ωm. It is observed that the voltage waveforms calculated with ULM-ATP and ULM-RV are coincident.

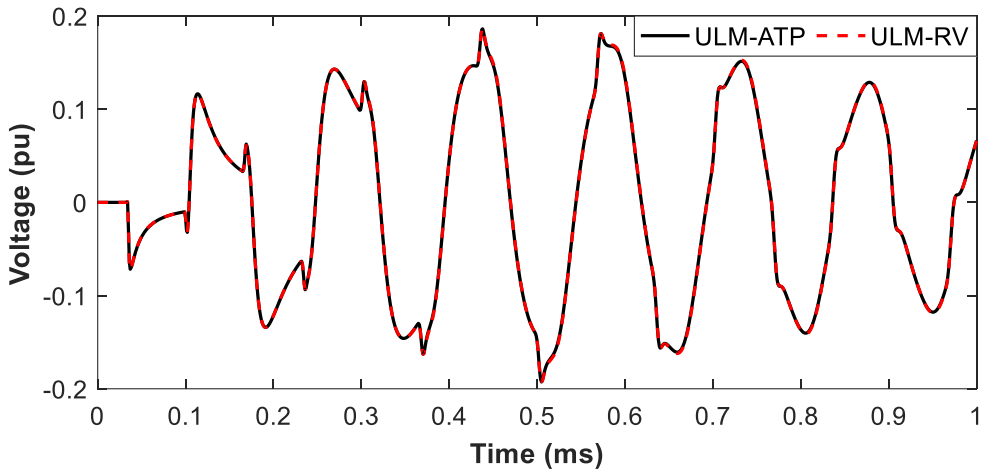


Fig. 6.1 – Voltage at the receiving end of the 115-kV line (phase *d*). Simulation for 100 Ω-m soil assuming constant soil parameters.

The .acp and fitULM.txt files for the simulation of this example are available on the following link: https://github.com/zanonfelipe/ULMAtp

7 - Summary

These documents present instructions and guidelines for implementing ULM in ATP. All files are available at the link https://github.com/zanonfelipe/ULMAtp. Users are free to use all support files and codes available for non-commercial activities, but are kindly requested to include reference to the following paper:

Felipe O.S. Zanon, Osis E.S. Leal, Alberto De Conti, Implementation of the universal line model in the alternative transients program, Electric Power Systems Research, vol. 197, p. 107311, Aug. 2021, doi: 10.1016/j.epsr.2021.107311.

8 - Bibliography

[1] Felipe O.S. Zanon, Osis E.S. Leal, Alberto De Conti, Implementation of the universal line model in the alternative transients program, Electric Power Systems Research, vol. 197, p. 107311, Aug. 2021, doi: 10.1016/j.epsr.2021.107311.

[2] Felipe O. S., Implementação do modelo ULM na plataforma ATP para o estudo de transitórios em linhas de transmissão aéreas com configuração assimétrica. Belo Horizonte, MG, Brasil: Dissertação de Mestrado, Programa de Pós Graduação em Engenharia Elétrica, Universidade Federal de Minas Gerais, December, 2019. Available in: https://www.ppgee.ufmg.br/diss_defesas_detalhesi.php?aluno=1657

[3] A. Morched, B. Gustavsen, M. Tartibi, A universal model for accurate calculation of electromagnetic transients on overhead lines and underground cables, IEEE Trans. Power Deliv., vol . 14, n° 3, pp. 1032–1038, 1999.

[4] B. Gustavsen, Modal Domain-Based Modeling of Parallel Transmission Lines With Emphasis on Accurate Representation of Mutual Coupling Effects, IEEE Trans. Power Deliv., vol. 27, pp. 2159–2167, 2012.

[5] J.R. Carson, Wave propagation in overhead wires with ground return, Bell Syst. Tech. J., vol. 5, n° 4, pp. 539–554, 1926.