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

USER MANUAL

ULM-ATP

VERSION 3.2

(November 2023)

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

1. UTFPR – Federal University of Technology – Paraná, Brazil.
2. UFMG – Federal University of Minas Gerais, Department of Electrical Engineering, Brazil.
3. CEMIG – Companhia Energética de Minas Gerais, formerly with the Graduate Program on Electrical Engineering (PPGEE), Federal University of Minas Gerais (UFMG), Brazil.

Revision history

Version	Date	Description of Revision
1.0	06-March-2021	Initial release.
1.1	12-June-2021	Equations 3.2 and 3.3 revised.
2.0	08-August-2021	✓ Simplifications in the MODELS code. ✓ Modifications in the algorithm to increase efficiency and to allow simulations with multiple line sections.
3.0	09-March-2023	✓ Implementation the two-segment interpolation scheme. In this version, it is possible to use different sets of poles for each mode. ✓ Modifications in the MODELS code to allow simulations using statistic switch with the initial time different from zero. ✓ Modifications in the "fitULM" text file.
3.1	28-July-2023	✓ Modifications in the "fitULM" text file structure to increase the model efficiency. ✓ Modifications to ULM master file location. ✓ Implementation the strategy proposed in [6] to reduce the computational cost when there are complex poles.
3.2	20-Nov.-2023	✓ Modifications in the errors message. In this version some errors are indicated directly in the ATP lis file. ✓ Implementation of the possibility of integration with ATP through Dynamic Link Library (dll).

Index

1 – Introduction

The implementation of ULM in ATP follows a strategy that combines the use of an external code, in the first stage, and ATP, in the second stage. The external code was originally 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 [3-4], 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. Recently, ATPDraw was updated in its version 7.5 to perform the calculation and fitting of the parameters required by ULM, so that the user does not need to implement a code to perform this task.

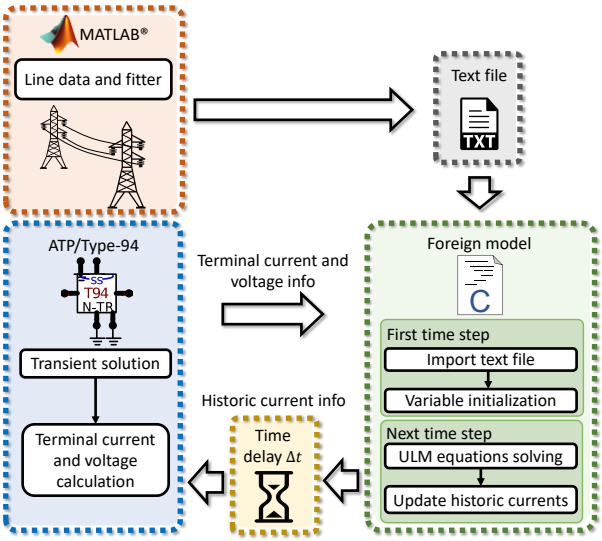


Fig. 1.1. Diagram of the strategy used in the implementation of ULM in ATP.

In the second stage, a foreign model implemented in ATP reads the text file generated by the external code. In this model, the original ULM equations modified to include the two-segment interpolation scheme proposed in [5] and the strategy proposed in [6] to reduce the computational cost when there are complex poles were implemented in ANSI C language. The program initializes the variables necessary for calculating the

transient at the first time step. From the voltages and currents at the line terminals, the foreign model calculates the history terms, feeding them back to 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, nor the equivalent procedure now available in ATPDraw, version 7.5. However, it provides detailed instructions on how to compile ATP files to include ULM-ATP. It also gives details on the input data required by ULM-ATP. It is to be noted that the ATP library files available from November 2023 already includes ULM-ATP. If the user is handling this version of ATP, installation compilation and details given in Sections 2 and 3 can be skipped. Older versions of ATP will require action from the user for installing and compiling ULM-ATP according to the instructions given in Sections 2 and 3.

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, which presents the original implementation of the ULM-ATP code (version 1.0):

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. This chapter describes how to add the ulmatp.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 32-bit compiler. To use the ULM-ATP model there are two installation possibilities. The first possibility, called static installation, is described in section 2.1. In this option, the installation is performed using the ulmmsg.o and ulmatp.o files and it is necessary to recompile the ATP software every time a new update of the ULM-ATP model is made available. The second possibility, called dynamic installation, is described in section 2.2. In this option, the installation is performed using the dllulm.o, ulmmsg.o and ulmatp.o files and a Dynamic Link Library (dll) is used as a link between ULM-ATP model and ATP. Once the compilation is performed, ULM-ATP can be updated just by replacing the old "dll" file by a new one. Therefore, by using the second option, inclusion of the ULM-ATP model and possible updates will require the user to recompile the ATP only once. It is expected that the ATP executable files distributed in the EEUG website will include the latest version of ULM-ATP already compiled into the mytpbig.exe files. In this case, the user will not need to perform any compilation to run the ULM-ATP code.

2.1 – Static Installation

In order to integrate the ulmatp.o model with ATP software using the static installation, the user must strictly follow the steps described below.

1) Extract all files from "MinGW.rar (version 5.1.0[1])" to "[dirInstall]\atpmingw\make\MinGW"[2], overwriting existing files and merging existing folders (this step is only necessary if the MinGW compiler is not up to date).

[1] In link: https://github.com/zanonfelipe/ULMAtp the user can also find ULM-ATP files for MinGW compiler version 3.4.5.

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

2) Copy “StaticPackage\fgnmod.f” to “[dirInstall]\atpmingw\make” overwriting the existing file.
3) Copy “StaticPackage\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) fix the directories entered in lines 10, 11 and 12 (for more details see chapter 5).

4) Copy “ulmmsg.o” and “ulmatp.o” to “…[dirInstall]\atpmingw\make”.
5) Copy “fitULM000001.txt” and “fitULM000002.txt” to “…\Documents\ATPdata\work”[3].
6) Copy ‘ULM000001.in’ to “…\Documents\ATPdata\work”.
7) Copy “ULM-ATP.acp” to “…\Documents\ATPdata\project”.
8) Open ‘ULM-ATP.acp’ and make the settings described in steps 9 to 16.
9) Click on ‘ATP’ option on the menu bar.
10) Select ‘ATP Launcher’.
11) In the appearing dialog, click on ‘Tools’.
12) Select “Make Tpbig.exe”.
13) In the appearing dialog, set:

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

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

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

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

[3] 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 master file should be copied to C:\ATP\ATPDraw\work. **Note that the “.atp” file and “master file ULM000001.in” 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”.

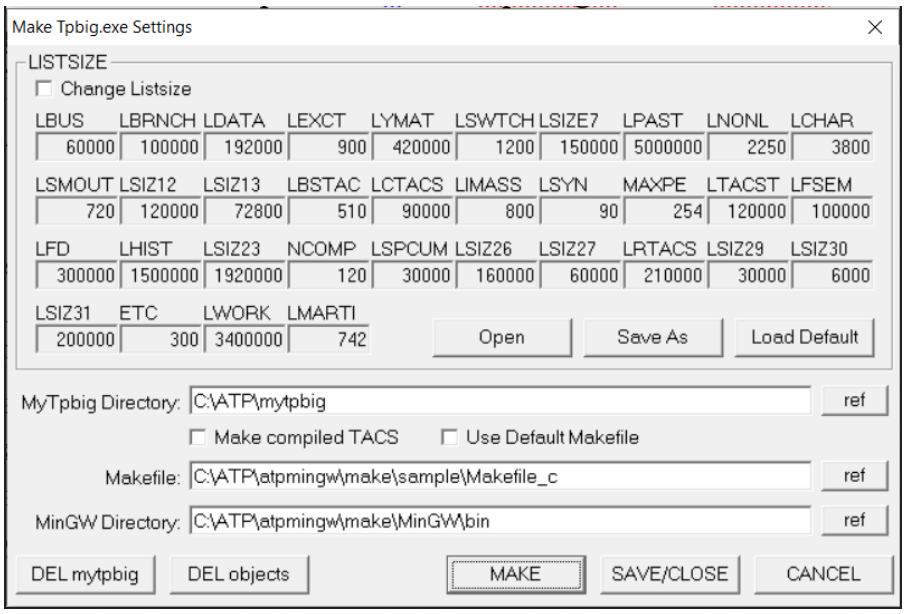


Fig. 2.1 – Make Tpbig.exe Settings.

14) Click on the "Make" button.
15) The massage "[dirInstall]\mytpbig\mytpbig.exe was successfully created!!" should appear.
16) Run the simulation.

2.2 – Dynamic Installation

In order to integrate the ulmatp.o model with ATP software using the dynamic (dll-based) approach, the user must strictly follow the steps described below.

1) Extract all files from "MinGW.rar (version 5.1.0[4])" to "[dirInstall]\atpmingw\make\MinGW"[5], overwriting existing files and merging existing folders (this step is only necessary if the MinGW compiler is not up to date).
2) Copy "DynamicPackage\fgnmod.f" to "[dirInstall]\atpmingw\make" overwriting the existing file.
3) Copy "DynamicPackage\makefile_dll" to "[dirInstall]\atpmingw\make\sample" overwriting the existing file.

[4] In link: https://github.com/zanonfelipe/ULMAtp the user can also find ULM-ATP files for MinGW compiler version 3.4.5.

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

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

3.3) open the Makefile_dll file.

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

4) Copy "DynamicPackage\dllULM.o", "DynamicPackage\ulmmsg.o" and "DynamicPackage\ulmatp.o" to "…[dirInstall]\atpmingw\make".
5) Copy "fitULM000001.txt" and "fitULM000002.txt" to "…\Documents\ATPdata\work"[6].
6) Copy 'ULM000001.in' to "…\Documents\ATPdata\work".
7) Copy "ULM-ATP.acp" to "…\Documents\ATPdata\project".
8) Open 'ULM-ATP.acp' and make the settings described in steps 9 to 16.
9) Click on 'ATP' option on the menu bar.
10) Select 'ATP Launcher'.
11) In the appearing dialog, click on 'Tools'.
12) Select "Make Tpbig.exe".
13) In the appearing dialog, set:

13.1) MyTpbig Directory: "[dirInstall]\mytpbig"

13.2) Make sure that "Make compiled TACS" and "Use Default Makefile" are unchecked.

13.3) Makefile: "[dirInstall]\atpmingw\make\sample\Makefile_dll".

13.4) MinGW Directory: "[dirInstall]\atpmingw\make\MinGW\bin" (an example for installing ATP software in the default directory is shown in Figure 2.2).

[6] 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 master file should be copied to C:\ATP\ATPDraw\work. **Note that the ".atp" file and "master file ULM000001.in" 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".

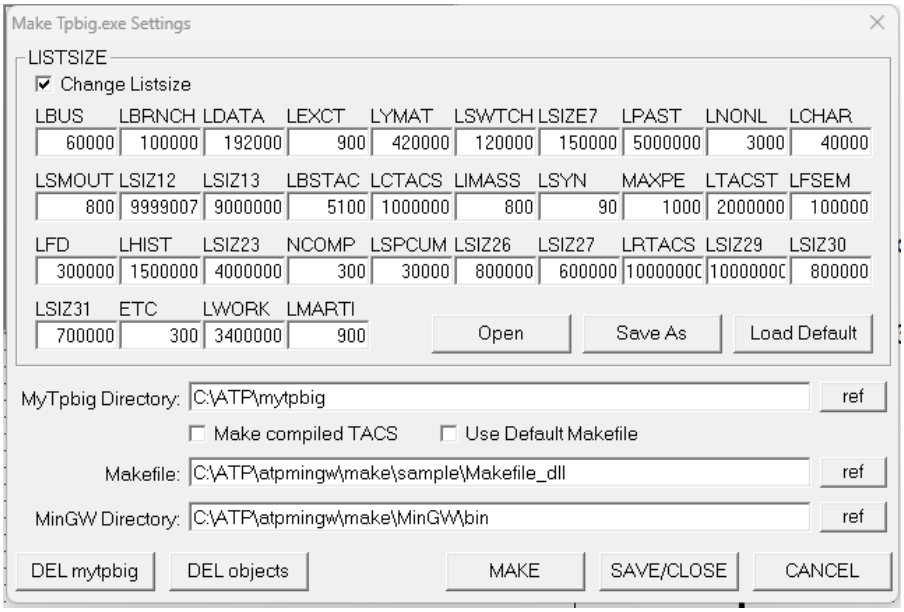


Fig. 2.2 – Make Tpbig.exe Settings.

14) Click on the “Make” button.

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

16) Run the simulation.

3 – Type94 Configuration

The interface between ATP and ULM-ATP model is performed by the Type 94 Norton TR model[7]. This model allows the inclusion of new line models into ATP. In ATPDraw it is possible to access the Type 94 Norton TR model by right-clicking on the MODELS/TYPE94/NORT-TR menu [7-8]. The Type 94 Norton TR model configuration screen is shown in Fig. 3.1. 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-ATP model. To do this, click on the "Edit" button.

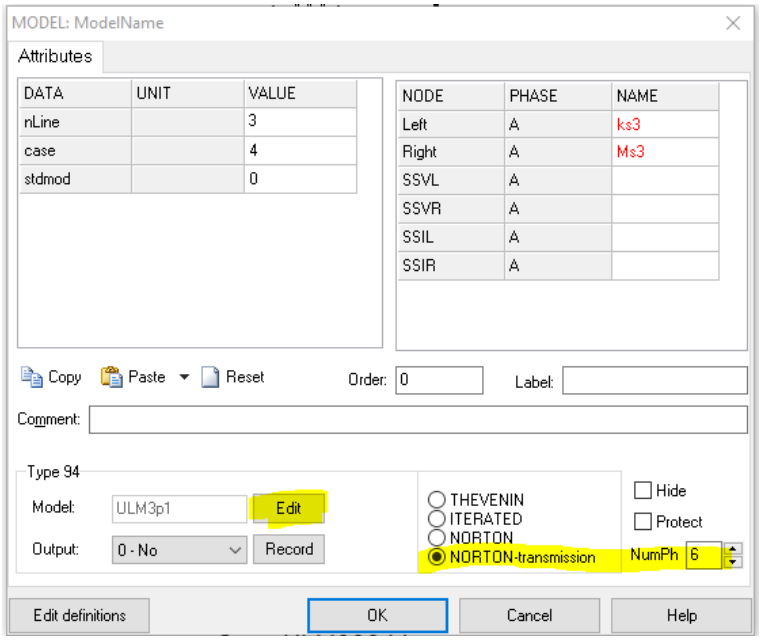


Fig. 3.1 – Type 94 Norton TR model configuration [7].

[7] From ATPDraw version 7.5, option is given to the user to select ULM-ATP as a foreign model. In this case, the setting of the type-94 MODEL is performed automatically and internally by ATPDraw, becoming transparent to the user.

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
                   --  on each side separately
      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 ulm3p2 -- NB! 6 character name limit
 DATA  n                        -- number of phases
       ng {dflt: n*(n+1)/2}   -- number of conductances
 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
       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
 DATA nLine                -- number of line
      case                 -- number of case
      stdmod               -- set simulation type (default stdmod=0)
 VAR flagmemoryulm
  comment ------------------------------------------------------------------------
  |       {ULM-ATP}                Universal Line Model - ATP                   |
  |       {ULM-ATP}                Version 3.1 July 28, 2023                    |
  |       {ULM-ATP}  Developed by:                                               |
  |       {ULM-ATP}              Osis E. S. Leal    - UTFPR                      |
  |       {ULM-ATP}              Alberto De Conti   - UFMG                       |
  |       {ULM-ATP}              Felipe O. S. Zanon - UFMG                       |
  |       {ULM-ATP}  UTFPR - Federal University of Technology - Paraná, Brazil  |
  |       {ULM-ATP}  UFMG  - Federal University of Minas Gerais, Brazil          |
  |       {ULM-ATP}  Reference paper:                                            |
  |       {ULM-ATP}  Felipe O.S. Zanon, Osis E.S. Leal, Alberto De Conti,        |
  |       {ULM-ATP}  Implementation of the universal line model in the           |
  |       {ULM-ATP}  Alternative Transients Program, Electric Power Systems      |
  |       {ULM-ATP}  Research, vol. 197, p. 107311, Aug. 2021.                   |
  ------------------------------------------------------------------- endcomment
MODEL ULM3p2 FOREIGN ULM_LINE  {ixdata:3, ixin:2*n+1, ixout:2*n+ng, ixvar:1}
  INIT
   lg[1..ng] :=0,  rg[1..ng] :=lg[1..ng]
   li[1..n]  :=0,  ri[1..n]  :=0
   lis[1..n] :=0,  ris[1..n] :=0
   flagmemoryulm:=0
  ENDINIT
  EXEC
   li[1..n] :=0
   ri[1..n] :=0
   IF t=stoptime and stdmod=1 THEN
     IF abs(atp(NENERG)-atp(KNT))=0 THEN
       flagmemoryulm:=-1
     ELSE
       flagmemoryulm:=1
     ENDIF
   ELSIF t=stoptime and stdmod<>1 THEN
       flagmemoryulm:=-1
   ENDIF
  USE ULM3p2 AS ULM3p2
     DATA xdata[1]:=timestep
          xdata[2]:=nLine
          xdata[3]:=case
     INPUT xin[1..n]:=lv[1..n]
           xin[(n+1)..(2*n)]:=rv[1..n]
           xin[(2*n)+1]:=flagmemoryulm
     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
```

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

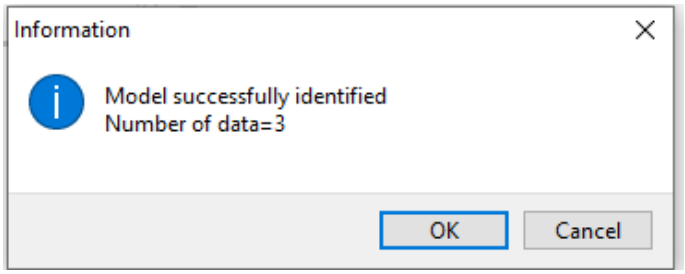


Fig. 3.2 – Dialog box [7].

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 three new fields, namely "nLine", "case" and "stdmod". 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 master file. On the other hand, the "case" field is used to select the default master file called ULM000000.in. The "stdmod" field is used enable/disable the ULM-ATP statistic mode. When the simulation include the ATP Statistic Switch, it is required to set "stdmod" equal to one.

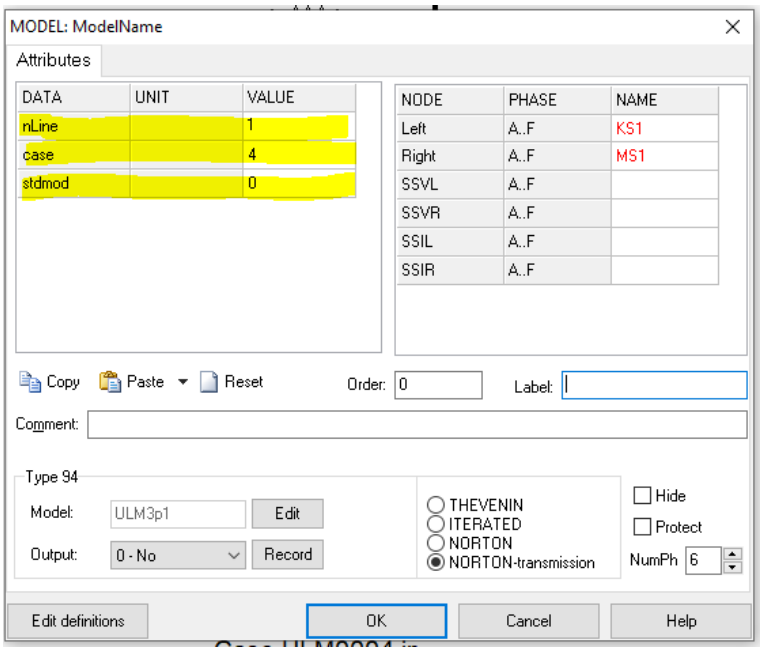


Fig. 3.3 – The new Type 94 Norton TR model configuration [7].

The link between the line and the file containing its parameters is made in the ULM000000.in master file, where each line of the ULM000000.in master 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.

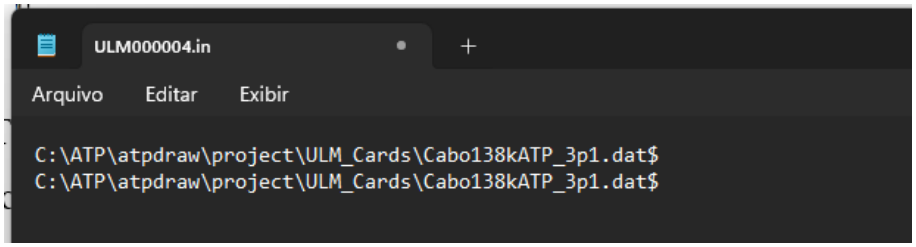


Fig. 3.4 – ULM000000.in file example.

The last 6 characters of name ULM000000.in master 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 master file. Then the user can use these characters to differentiate their simulations. For example, in Fig. 3.3 the “case” field is equal **4**. In this condition, the ULM code will search for the ULM000004.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 and Figure 4.2, which refers to a n_f-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, the real part is inserted first and then the imaginary part is inserted in the next line. For stable simulations with ULM-ATP the real part of the poles must always be negative. However, a positive real part is used in the text file as a flag to identify the complex poles. To reduce redundancy, only one of the poles pertaining to the complex conjugate pair is written together with the associated residues matrix. Therefore, when writing the total number of poles in the file heading, only one pole out of each complex conjugate pair should be counted. Furthermore, for the complex poles, the user must enter the imaginary parts of all elements of the associated residues matrix, even if these are so small they can be considered null. On the other hand, for real poles, only the real part of the associated matrix of residues must be inserted. To exploit the symmetry of the matrices associated with the fitting of $\boldsymbol{Y}_c$ ($\boldsymbol{k}_{p_{Yc}}$ and $\boldsymbol{k}_0$), only their upper triangulars are inserted. The sequence of recording the information is described below:

- Number of phases (n_f);
- Number of modes (m);
- Number of poles p_{Yc} resulting from the fitting of $\boldsymbol{Y}_c$ (each complex conjugate pair count as one);
- Number of poles $p_{H(1)}$ resulting from the fitting of the first mode of $\boldsymbol{H}$ (each complex conjugate pair count as one);
- Number of poles $p_{H(2)}$ resulting from the fitting of the second mode of $\boldsymbol{H}$ (each complex conjugate pair count as one);

  ⋮
- Number of poles $p_{H(m)}$ resulting from the fitting of the *m-th* mode of $\boldsymbol{H}$ (each complex conjugate pair count as one);
- First time delay (associated with mode 1);
- Second time delay (associated with mode 2);

  ⋮
- *m-th* time delay (associated with mode *m*);
- First pole of $a_{Y_c(1)}$ resulting from the fitting of $\boldsymbol{Y}_c$ (a negative number, if purely real; a positive number containing the absolute value of the real part of the pole, if complex); in the next line, the imaginary part of the pole $a_{Y_c(1)}$ must be inserted, if it

exists; **only one pole pertaining to a complex conjugate pair and its corresponding set of residues must be considered**. The missing information is automatically generated by the ULM-ATP code.

- First matrix of residues $\boldsymbol{k}_1$ resulting from the fitting of $\boldsymbol{Y}_c$, included element by element.
    - First enter the elements from the first row of $\boldsymbol{k}_1$, that is, for complex pole, $real[\boldsymbol{k}_1(1,1,1)]$, next line: $imag[\boldsymbol{k}_1(1,1,1)]$, next line: $real[\boldsymbol{k}_1(1,2,1)]$, next line: $imag[\boldsymbol{k}_1(1,2,1)]$ (…), next line: $real\left[\boldsymbol{k}_1\left(1,n_f,1\right)\right]$, next line: $imag\left[\boldsymbol{k}_1(1,n_f,1)\right]$, for this example. Then, enter the elements from second row of $\boldsymbol{k}_1$, starting from element $real[\boldsymbol{k}_1(2,2,1)]$, and finally the one element from the last row of $\boldsymbol{k}_1$, that is, $imag\left[\boldsymbol{k}_1\left(n_f,n_f,1\right)\right]$ (for real poles, only the real part of the residue must be written).
- Second pole $a_{Y_c(2)}$ resulting from the fitting of $\boldsymbol{Y}_c$;
- Upper triangular of the matrix of residues $\boldsymbol{k}_2$ resulting from the fitting of $\boldsymbol{Y}_c$, included element by element.
- Repeat the procedure described above, until all the poles and respective matrices of residues, resulting from the fitting of $\boldsymbol{Y}_c$, be written in the text file.
- First pole resulting from the fitting of the first mode of $\boldsymbol{H}$ (a negative number, if purely real; a positive number containing the absolute value of the real part of the pole, if complex); in the next line, the imaginary part of the pole $a_{H(1,1)}$, if it exists.
- First matrix of residues $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}$ resulting from the fitting of first mode of $\boldsymbol{H}$, included element by element. The matrix $\bar{\boldsymbol{c}}_{\boldsymbol{i,j}}(n_f,n_f,m,p_{H(m)})$ has four dimensions.
    - First enter the elements from the first row of $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}$, that is $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(1,1,1,1)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(1,2,1,1)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(1,n_f,1,1)$, 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, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(n_f,1,1,1)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(n_f,2,1,1)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(n_f,n_f,1,1)$.
- Second pole $a_{H(2,1)}$ resulting from the fitting of first mode of $\boldsymbol{H}$;
- Second matrix of residues $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}$ resulting from the fitting of first mode of $\boldsymbol{H}$, included element by element.
    - First enter the elements from the first row of $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}$, that is $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(1,1,1,2)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(1,2,1,2)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(1,n_f,1,2)$, for this example. 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, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(n_f,1,1,2)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(n_f,2,1,2)$, $\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(n_f,n_f,1,2)$.
- Next, enter the pole $a_{H(1,3)}$ and elements of $\bar{\boldsymbol{c}}_{\boldsymbol{1,3}}$, following the procedures described above, and repeat for all pole $a_{H(1,i_{th})}$ and $\bar{\boldsymbol{c}}_{\boldsymbol{1,j_{th}}}$ matrices.
- After, enter the first pole $a_{H(2,1)}$ resulting from the fitting of second mode of $\boldsymbol{H}$;
- First matrix of residues $\bar{\boldsymbol{c}}_{\boldsymbol{2,1}}$ resulting from the fitting of second mode of $\boldsymbol{H}$, included element by element, following the procedures described for the first mode.
- Second pole $a_{H(2,2)}$ resulting from the fitting of second mode of $\boldsymbol{H}$;
- Second matrix of residues $\bar{\boldsymbol{c}}_{\boldsymbol{2,2}}$ resulting from the fitting of second mode of $\boldsymbol{H}$, included element by element.
- Repeat the procedures above for all pole $a_{H(i_{th},j_{th})}$ and respective $\bar{\boldsymbol{c}}_{i_{th},j_{th}}$ matrix.
- Matrix containing the elements of the independent term $\boldsymbol{k}_0$ resulting from the fitting of $\boldsymbol{Y}_c$, included element by element (real part). The matrix $\boldsymbol{k}_0(n_f,n_f)$ is two-dimensional.

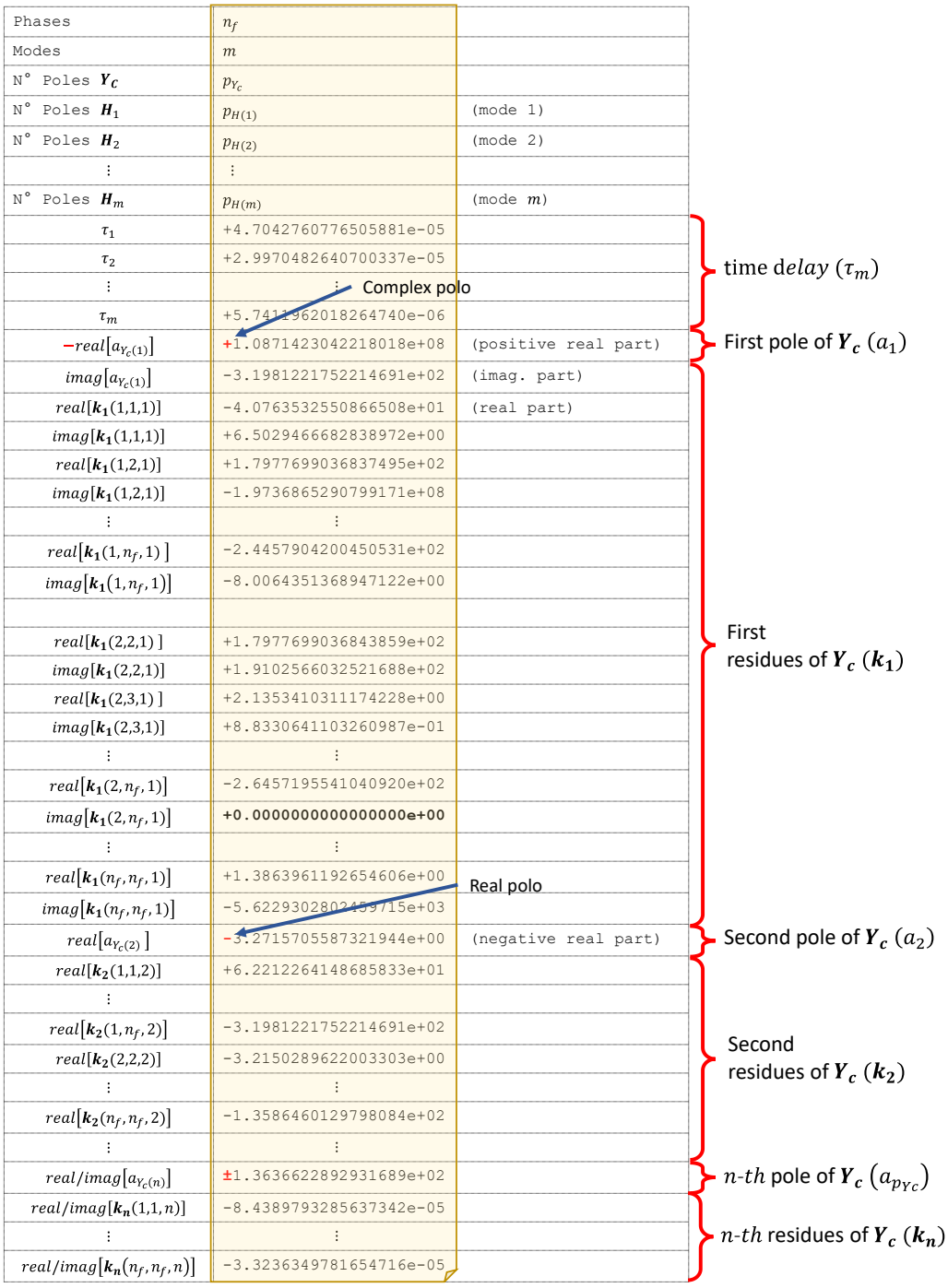


Fig. 4.1 – "fitULM" text file structure.

$real/imag[a_{H(1,1)}]$	±1.3996907644561276e+02	First pole of $\boldsymbol{H}$ $(a_{H(1,1)})$ to first mode
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(1,1,1,1)]$	1.0927377470343650e+01	First residues of $\boldsymbol{H}$ $(\bar{\boldsymbol{c}}_{\boldsymbol{1,1}})$
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(1,2,1,1)]$	-4.3460688917835615e+00	
⋮	⋮	
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(1,n_f,1,1)]$	4.7642985769289552e+01	
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(2,1,1,1)]$	8.1195854524057776e+00	
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(2,2,1,1)]$	6.3600576628933290e+00	
⋮	⋮	
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(2,n_f,1,1)]$	1.5473150405889095e+01	
⋮	⋮	
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,1}}(n_f,n_f,1,1)]$	-1.6969863296586436e+01	
$real/imag[a_{H(2,1)}]$	±1.3996907644561276e+02	Second pole of $\boldsymbol{H}$ $(a_{H(1,2)})$
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(1,1,1,2)]$	+1.0927377470343650e+01	Second residues of $\boldsymbol{H}$ $(\bar{\boldsymbol{c}}_{\boldsymbol{1,2}})$
⋮	⋮	
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{1,2}}(n_f,n_f,1,2)]$	-1.6969863296586436e+01	
⋮	⋮	
$real/imag[a_{H(1,i_{th})}]$	±1.3996907644561276e+02	*i-th* pole of $\boldsymbol{H}$ $(a_{H(1,i_{th})})$ to first mode
$real/imag\left[\bar{\boldsymbol{c}}_{\boldsymbol{1},p_{H(1)}}(1,1,1,p_{H(1)})\right]$	+1.0927377470343650e+01	*i-th* residues of $\boldsymbol{H}$ $\left(\bar{\boldsymbol{c}}_{\boldsymbol{1},p_{H(1)}}\right)$
⋮	⋮	
$real/imag\left[\bar{\boldsymbol{c}}_{\boldsymbol{1},p_{H(1)}}(n_f,n_f,1,p_{H(1)})\right]$	-1.6969863296586436e+01	
⋮	⋮	
$real/imag[a_{H(i_{th},j_{th})}]$	±1.3996907644561276e+02	*i-th* pole of $\boldsymbol{H}$ $(a_{H(i_{th},j_{th})})$ to *j-th* mode
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{2,1}}(1,1,2,1)]$	+1.0927377470343650e+01	*i-th* residues of $\boldsymbol{H}$ $(a_{H(i_{th},j_{th})})$
⋮	⋮	
$real/imag[\bar{\boldsymbol{c}}_{\boldsymbol{2,1}}(n_f,n_f,2,1)]$	-1.6969863296586436e+01	
⋮	⋮	
$real/imag\left[a_{m,p_{H(m)}}\right]$	±1.3996907644561276e+02	$p_{H(m)}$-th pole of $\boldsymbol{H}$ $\left(a_{H(p_{H(m)},m)}\right)$ to m-th mode
$real/imag\left[\bar{\boldsymbol{c}}_{m,p_{H(m)}}(1,1,m,p_{H(m)})\right]$	+1.0927377470343650e+01	$p_{H(m)}$-th residues of $\boldsymbol{H}$ $\left(a_{H(p_{H(m)},m)}\right)$
⋮	⋮	
$real/imag\left[\bar{\boldsymbol{c}}_{m,p_{H(m)}}(n_f,n_f,m,p_{H(m)})\right]$	-1.6969863296586436e+01	
$real[\boldsymbol{k_0}(1,1)]$	+2.2766409553979265e-03	residues of Y_c $(\boldsymbol{k_0})$
$real[\boldsymbol{k_0}(1,2)]$	-6.9618233247604631e-04	
⋮	⋮	
$real[\boldsymbol{k_0}(1,n_f)]$	-2.5346035896258579e-04	
$real[\boldsymbol{k_0}(2,2)]$	+3.5008982945592621e-03	
⋮	⋮	
$real[\boldsymbol{k_0}(n_f,n_f)]$	+2.1314806829812205e-03	

Fig. 4.2 – "fitULM" text file structure (continuation).

5 – ATP Compilation Files

5.1 – Makefile

Makefile[8] gathers the set of instructions necessary for the automatic compilation of a software. For the compilation of ATP including new functions programmed in ANSI C language in mytpbig.exe, the makefile is called “Makefile_c” for the static installation and Makefile_dll for dynamic installation. These files are responsible for telling the compiler which files and libraries are linked to ATP, including saying which compiler will do that task. To include the ULM-ATP model, the ATP standard “Makefile_c” or Makefile_dll file must be modified as described on the next page.

In the “Makefile” it is necessary to set some directories, see lines 10, 11 and 12 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 change the directories so that they correspond to the customization carried out by the user.

[8] The makefile example was created based on the 2008 ATP library. Therefore, small modifications may be necessary if more recent libraries are used.

Makefile example

```
# ------------------------------------------------------------
#        By Osis E. S. Leal - UTFPR
#           Alberto De Conti - UFMG
#           Felipe O. S. Zanon - UFMG
# UTFPR - Federal University of Technology – Paraná, Brazil
# UFMG - Federal University of Minas Gerais, Brazil
# ------------------------------------------------------------
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)\ulmatp.o\
         $(GWDIR)\ulmmsg.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)

```

Makefile_dll example

```
# ------------------------------------------------------------
```

```
#         By Osis E. S. Leal - UTFPR
#            Alberto De Conti - UFMG
#            Felipe O. S. Zanon - UFMG
# UTFPR - Federal University of Technology – Paraná, Brazil
# UFMG - Federal University of Minas Gerais, Brazil
# ----------------------------------------------------------
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)\dllULM.o\
          $(GWDIR)\ulmmsg.o

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

DLL_OBJS =dllulm.o ulmatp.o
DLL_NAME = $(GWMY)\ulmatp.dll

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)

$(DLL_NAME): $(DLL_OBJS)
       $(CC) -shared -o $(DLL_NAME) $(DLL_OBJS) \
              -Wl,--export-all-symbols

```

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 ulmatp.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). For the static installation these functions must be registered in the “fgnmod.f” file (see lines 63 and 66 of the code described on the next page, taken from the fgnmod.f file used for the **static** installation). The variables xdata, xin, xout, xvar are ATP standards and must be present in the C program because they link ATP to MODELS. In addition, there is the ulmerror function responsible for printing identified errors in the “lis” file. On the other hand, for the dynamic installation there is a small difference in the “fgnmod.f” file. Now the ulm_i and ulm_m functions are replaced by ulmdll_i and ulmdll_m functions file (see lines 63 and 66 of the code described on the next page, taken from the fgnmod.f file used for the **dynamic** installation).

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).

Fgnmod.f for static installation

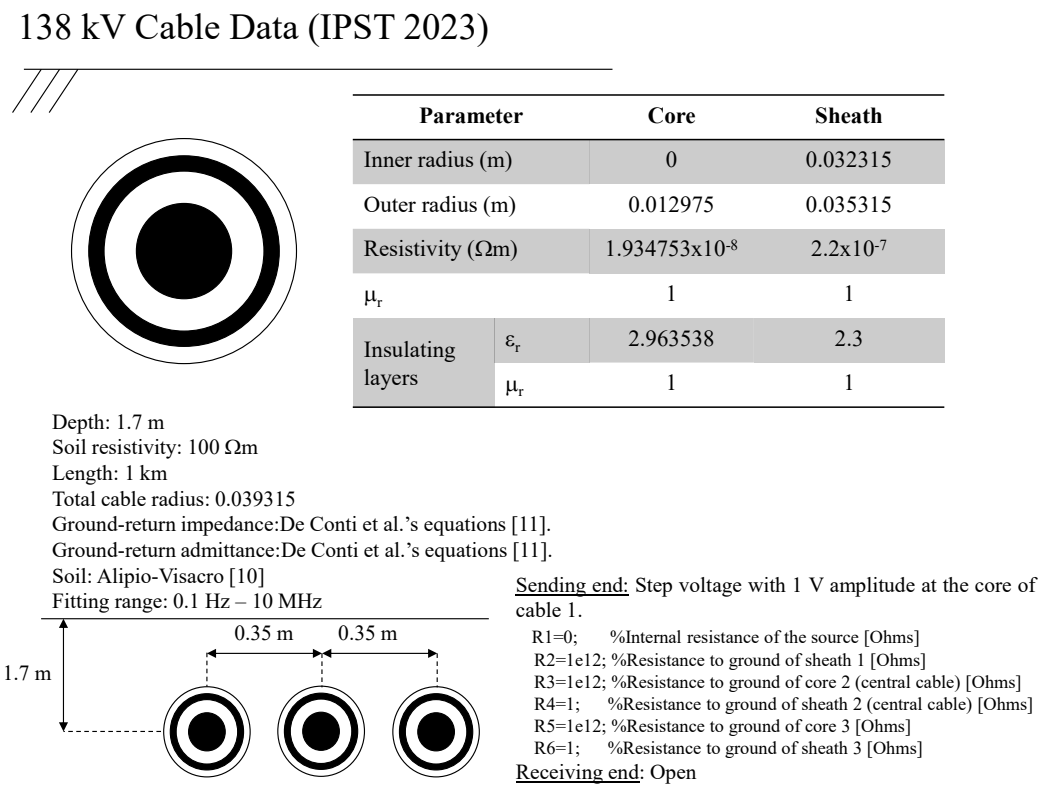
```
1         SUBROUTINE FGNMOD ( name, namlen, xdata, xin, xout, xvar,
2        1                    iniflg, ierflg)
3         IMPLICIT REAL*8 (A-H, O-Z),  INTEGER*4 (I-N)
4         DIMENSION xdata(*), xin(*), xout(*), xvar(*)
5         CHARACTER*1 name(*)
6         PARAMETER ( namcnt = 8 )
7         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          CALL ulmerror(xdata, xvar)
65         ELSE
66          CALL ulm_m(xdata, xin, xout, xvar)
67         ENDIF
68        CONTINUE !      ----------------------------------------
                  (⋮)
```

Fgnmod.f for dynamic installation

```
      SUBROUTINE FGNMOD ( name, namlen, xdata, xin, xout, xvar,
     1                    iniflg, ierflg)
      IMPLICIT REAL*8 (A-H, O-Z),  INTEGER*4 (I-N)
      DIMENSION xdata(*), xin(*), xout(*), xvar(*)
      CHARACTER*1 name(*)
      PARAMETER ( namcnt = 8 )
      CHARACTER*32 refnam(namcnt)
      CONTINUE !  ------------------------------------------------
      CONTINUE !  You may increase namcnt above to allow more names:
      CONTINUE !  ------------------------------------------------
      CONTINUE !  In the following lines, register your foreign model
      CONTINUE !  names as they are declared in your models:
      CONTINUE !   - use only uppercase characters for the name here
      CONTINUE !   - you can use any case for the name in the models
      CONTINUE !   - make a copy of the modifications you make to this
      CONTINUE !     file so that you don't lose them when installing
      CONTINUE !     a newer version of ATP later
      DATA refnam(1) / 'SAMPLE_MODEL' /  ! Do not modify this line
      DATA refnam(2) / 'ULM_LINE' /
      DATA refnam(3) / ' ' /
      DATA refnam(4) / ' ' /
                (⋮)
      CONTINUE !  ------------------------------------------------
      CONTINUE !  In the following lines, this is where you call the
      CONTINUE !  actual foreign subroutines/procedures:
      CONTINUE !   - actual names may be different from the foreign
      CONTINUE !     names used in the models
      CONTINUE !   - notice how each one uses both an
      CONTINUE !     initialization routine and an execution routine
      IF ( iname.EQ.1 ) THEN
       IF (iniflg.EQ.1) THEN
        CALL sampli(xdata, xin, xout, xvar)
       ELSE
        CALL samplm(xdata, xin, xout, xvar)
       ENDIF
      CONTINUE !      ----------------------------------------
      ELSE IF ( iname.EQ.2 ) THEN
       IF (iniflg.EQ.1) THEN
        CALL ulmdll_i(xdata, xin, xout, xvar)
        CALL ulmerror(xdata, xvar)
       ELSE
        CALL ulmdll_m(xdata, xin, xout, xvar)
       ENDIF
      CONTINUE !      ----------------------------------------
                (⋮)
```

6– Application Example

The example considered in this section is taken from [9]. It deals with the 1-km long 138-kV cable system shown in Figure 6.1 (see [9] for configuration details). The tested cases considered the core excitation shown in Figure 6.2. To excite a wide frequency range, a step voltage with 1 V amplitude was applied. The calculation of the cable parameters including core losses, sheath losses, internal insulation and external jacket was performed in Matlab following the approach described in [9]. Ground resistivity values of 100 Ωm were considered according to the Alipio-Visacro model [10].



Parameter		Core	Sheath
Inner radius (m)		0	0.032315
Outer radius (m)		0.012975	0.035315
Resistivity (Ωm)		1.934753×10^{-8}	2.2×10^{-7}
μ_r		1	1
Insulating layers	ε_r	2.963538	2.3
	μ_r	1	1

Fig. 6.1 – Voltage at the receiving end of the 138-kV line (phase *d*). Simulation for 100 Ω-m soil

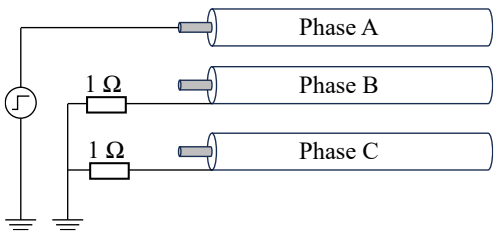


Fig. 6.2 – Configurations for the transient simulations.

Figure 6.3 shows voltages calculated at the receiving end core of phases A and C. The results reproduce exactly those shown in Figure 7(b) in [9].

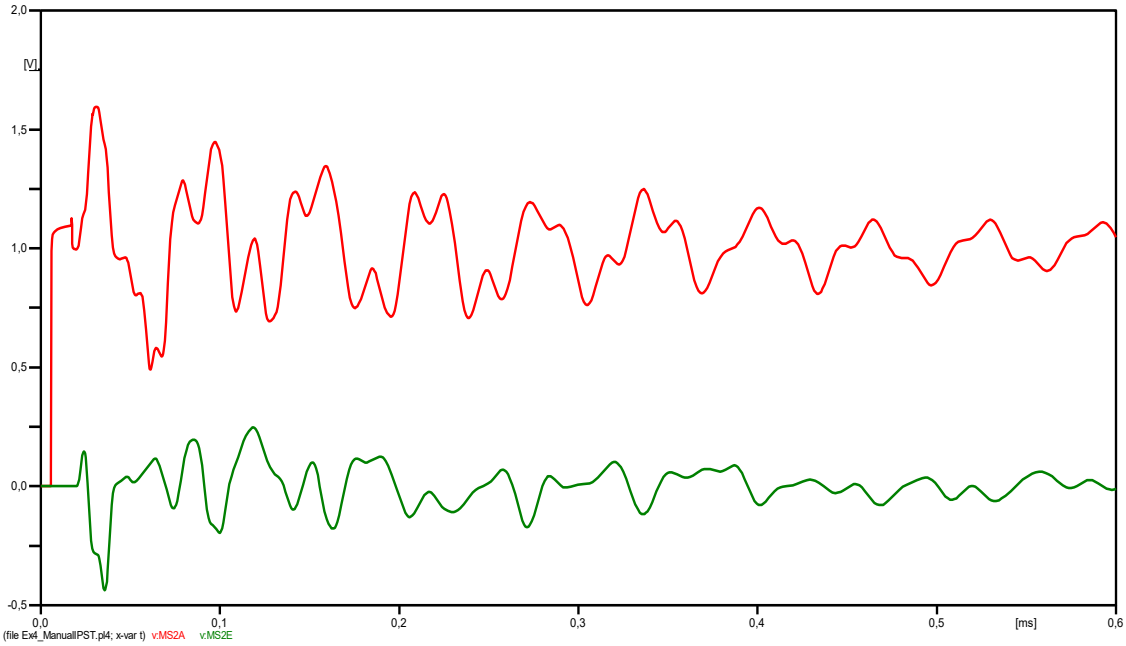


Fig. 6.3 – Receiving-end voltages at the core of phases A and C [9].

The Ex4_ManualIPST.acp, ULM008003.in, Cabo138kIPST2023_3p1.dat and Ex4_ResultIPST2023.pl4 files for the simulation and validation of this example are available on the following link: https://github.com/zanonfelipe/ULMAtp.

7– Tips and Comments

ULM-ATP was originally developed in ANSI C language and was compiled with ATP files dated 2008 using the MinGW 5.1.0 32-bit compiler. So far, we have not experienced failures when using this specific pairing. However, when it was decided to move to newest versions of ATP and to test different compilers, in some cases it was found that the ATP simulation would crash for specific timesteps while it worked for others. For example, for a given data set the simulation would run perfectly for an extremely small timestep of 1 ns but would arbitrarily fail for 1.25 ns. Amazingly enough, the same case would run perfectly for, e.g., 1.24 ns or 1.26 ns. The reasons for this erratic behavior are presently unknown, although they are under investigation. Strong suspicion lays upon the incompatibility of long deprecated Fortran77 functions in ATP with a FOREIGN MODEL written in C and compilation performed with more up-to-date 64-bit compilers. A test to identify if the simulation has crashed is to check whether the .lis file was written at all. If it was not, then action will be needed from the user as discussed below.

In the ATP version released in November 2023, ULM-ATP was compiled using the dll approach considering a 32-bit MinGW 3.4.5 compiler. In thousands of systematic tests performed with this version of the program, the simulation crashed due to the erratic timestep dependency in less than 0.55% of the cases. However, the failure rate is expected to be greater if different compilers are used, especially 64-bit ones.

If the user is faced with simulation crashes caused by the timestep dependency, the following suggestions are given:

1. Perform a slight change in the timestep and try again; if this does not give a satisfactory response, then:
2. Select the mytpbig.exe or equivalent file and put it into compatibility mode with an older version of Windows (e.g., Windows 7). To put the mytpbig.exe file in compatibility mode, right-click on it, select ‘Properties’, then select ‘Compatibility’ and finally choose the desired option. If the simulation does not run satisfactorily, give another try on step 1.

Another limitation of the current version of ULM-ATP is that it only allows transient simulations starting with zero initial conditions. This happens because no MODELS block is recognized in the steady-state initialization in ATP and an error

message is expected whenever the user attempts to run this feature. Therefore, if steady-state initialization is needed in a case study including ULM-ATP, a suggestion is to start the simulation with zero initial conditions, wait for sufficient time for the transient response to accommodate and then perform the desired transient simulation. Work on incorporating steady-state initialization into ULM-ATP is expected to be soon underway.

The developers of ULM-ATP are continuously updating the code. If any bug or issue is identified with the model, please send a message to osisleal@utfpr.edu.br or conti@cpdee.ufmg.br.

8– 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.

9– 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, july 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, oct. 2012.

[5] B. Gustavsen, Avoiding numerical instabilities in the universal line model by a two-segment interpolation scheme, IEEE Trans. Power Deliv., vol. 28, no. 3 pp. 1643-1651, July 2013, doi: 10.1109/TPWRD.2013.2257878.

[6] B. Gustavsen and H. M. J. De Silva, "Inclusion of Rational Models in an Electromagnetic Transients Program: Y-Parameters, Z-Parameters, S-Parameters, Transfer Functions," in IEEE Transactions on Power Delivery, vol. 28, no. 2, pp. 1164-1174, April 2013, doi: 10.1109/TPWRD.2013.2247067.

[7] Alternative Transients Program (ATP): Rule Book. Leuven EMTP Center, 1992.

[8] Hans Kr. Høidalen, ATPDraw for Windows Users' Manual Version 7.3, Norwegian University of Technology Trondheim, Norway, May, 2021.

[9] Alberto De Conti, Naiara Duarte, Rafael Alipio, Osis E. S. Leal, Small-Argument Analytical Expressions for the Calculation of the Ground-Return Impedance and Admittance of Underground Cables, Electric Power Systems Research, vol. 220, p. 10200, doi.org/10.1016/j.epsr.2023.109299.

[10] R. Alipio and S. Visacro, "Modeling the Frequency Dependence of Electrical Parameters of Soil," in IEEE Transactions on Electromagnetic Compatibility, vol. 56, no. 5, pp. 1163-1171, Oct. 2014, doi: 10.1109/TEMC.2014.2313977.

[11] A. De Conti, N. Duarte and R. Alipio, "Closed-Form Expressions for the Calculation of the Ground-Return Impedance and Admittance of Underground

Cables," in IEEE Transactions on Power Delivery, vol. 38, no. 4, pp. 2891-2900, Aug. 2023, doi: 10.1109/TPWRD.2023.3264614.