UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ŞI ELECTRONICĂ

Departamentul de Calculatoare şi Tehnologia Informaţiei

***Computers in English***

# Software application for numerical analysis of analog circuits with advanced user interface

**BACHELOR DEGREE PROJECT**

Scientific coordinator,

*Professor Marius BREZOVAN, PhD, Eng*

Author,

*Diana MANDACHE*

**July 2016**

*„Any fool can know. The point is to understand"*

Albert Einstein

# CONTENT

# 1 INTRODUCTION

The rapid development of manufacturing technologies has allowed the increase in complexity of engineering applications that require advanced tools for development. Consequently there were developed integrated circuits that have hundreds or thousands of discrete elements and whose analysis would be impossible without electronic computing tools. As a result, specialized software for Computer Aided Design (CAD) of such systems emerged on the market, however, no commercial program is perfect in terms of covering all particular situations that arise in practice. Moreover, they are rigid, they don't allow interventions in the body of the program so the researchers cannot adapt them to their own needs.

In this context, through this project I seeked the development of a software application that was able to overcome these drawbacks: a modular, flexible, with future development opportunities, equipped with all the amenities of a commercial program.

The program is based on state-of-the-art analysis algorithms that incorporate both knowledge of electrical engineering and software engineering, such that optimum performances are achieved. It was developed using two of the most acclaimed technologies in the scientific community, Python and MATLAB.

The resulting application, named CIRCUS, which stands for CIRCUit Simulator, has a graphical user interface which facilitates the insertion of input data in form of electric diagrams, specifying the circuit parameters and working regimes along with the analysis parameters. Afterwards, this information is transmitted to the computational unit which performs the circuit analysis. Then the results are communicated to the user in a easy-to-read form (both numerical and graphical).

This paper is organized in seven sections which present the project requirements, the software tools used, the theoretical fundamentals the algorithms used rely on, the description of the software implementation, detailed running examples and concluding opinions.

# 2 APPLICATION REQUIREMENTS

CIRCUS is designed to be a user-friendly, simple and intuitive application for electrical circuit analysis that allows the user, whether he is experienced or not in the domain of electrical engineering, to draw the schematic of a circuit and specify the values of the input parameters. The system returns the output (values of current intensities and voltages) as a response. The user views these results and can choose to plot some of them.

The detailed list of functional requirements of the application are the following:

1. Add circuit elements to schematic (add resistor, add inductor, add capacitor, add voltage source, add current source, add ground).
2. Connect circuit elements by drawing wires (add wire).
3. Set the parameter values of the elements.
4. Move elements.
5. Rotate elements.
6. Delete elements.
7. Choose the working regime (direct current, alternating current).

8. Start a new drawing session.
9. Load the schematic from a file.
10. Save the schematic to a file.
11. Save the schematic as EPS.

12. Determine the nodes of the circuit and draw them on the schematic.
13. Error handling: determine if the topology of the circuit is correct.
14. Solve the circuit i.e. find the intensity of the currents and the voltages of each element and display the results.
15. Plot the amplitude-frequency characteristics for an element’s quantity (current or voltage) chosen by user.

# 3 TECHNOLOGIES USED

In the process of planning the project, for choosing the most suitable software resources to meet the objectives of the final program, I conducted a comparative study based on the knowledge acquired in the four years of bachelor studies, as well as on scientific literature. In the choice of the most appropriate tools, the following criteria were considered:

- advanced features for graphical interfaces
- comprehensive routines for matrix computations
- allowing effortless calculations by vectorization instead of repetitive statements
- compatible technologies that can interact fluently
- application portability
- allowing the graphical post-processing of the obtained results
- robustness and reliability
- intuitive ease of use
- extensive documentation and strong community

Subsequent to this analysis, I have chosen MATLAB for developing the computational modules destined for the systematic analysis of electrical circuits which involve vast matrix operations. Along with this, Python was an excellent match for building the graphical user interface and to acquire the data, interpret it and feed it to the algorithm.

## 3.1 MATLAB

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally as a wrapper on FORTRAN libraries for linear algebra. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and commercial enterprises around the world.

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has refined data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB a first-rate tool for teaching and research.

MATLAB has many advantages for solving technical problems in contrast with traditional coding languages (e.g., C, FORTRAN). MATLAB is an intuitive framework whose basic data element is an array that does not require dimensioning. It has powerful built-in procedures that allow a very wide range of computations. It also has easy to use graphics commands that make the visualization of results immediately available. Specific functions are gathered in packages specified as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

MATLAB also offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions, like sine, cosine, absolute value, phase angle or complex conjugate, as well as a number of predefined constant values, like Pi, Infinity, NaN, etc.

The fundamental component of MATLAB is the matrix (mxn), special cases of matrices are column vectors (mx1) and row vectors (1xn), even a single-valued element (number) is considered to be a one row and one column matrix (1x1). It provides functions that generate elementary matrices, for example, the matrix of zeros, the matrix of ones, and the identity matrix are returned by the functions zeros, ones, and eye, respectively. It also supplies special matrices, matrices that have unique properties which make them useful for testing.

MATLAB has two different types of arithmetic operations: matrix operations and array operations, the latter doing element-by-element calculations, such as multiplication, division or exponentiation.

One of the problems encountered most frequently in scientific computation is the solution of systems of simultaneous linear equations. With matrix notation, a system of simultaneous linear equations is written: Ax = b, where there are as many equations as unknown. A is a given square matrix of order n, b is a given column vector of n components, and x is an unknown column vector of n components. MATLAB offers two typical options to solve for x, one is by using matrix inverse, inv, so x = inv(A)*b and the other is to use the backlash operator, behind which stands the algorithm of Gaussian elimination.

Program files can be scripts that simply execute a series of MATLAB statements, or they can be functions that also accept input arguments and produce output. Both scripts and

functions contain MATLAB code, and both are stored in text files with a .m extension. While scripts store variables in a workspace that is shared with other scripts, a function's variables are internal for it. However, functions are more flexible and more easily extensible.

MATLAB is also a programming language, like other computer programming languages, it has some decision making structures for control of command execution. These decision making or control flow structures include for loops, while loops, and if-else-end constructions. By creating a file with the extension .m, the user can easily write and run programs which do not need to be compiled since MATLAB is an interpretative language. MATLAB has thousands of functions, and others can be added using m-files.

The problem formulation that is sought to be solved involves both many matrix operations and solving of linear equations systems (as explained in Chapter 4). Therefore, the most suitable programming environment for the computational unit of the application is MATLAB. Not only it is a world renowned choice for researchers, a reliable programming environment, but it also unburdens the programmer of writing redundant code by featuring extensive built-in procedures. The version used is R2015a.

## 3.2 Python

Python is a clear and powerful programming language created in the late 1980s by Guido van Rossum, and named after Monty Python. It is widely used for a variety of applications from testing microchips at Intel, to powering Instagram, to building video games with the PyGame library.

Python is an interpreted, high-level, general purpose, dynamic programming language. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It encourages Rapid Application Development (RAD) due to its dynamic typing and dynamic binding systems, automatic memory management and garbage collection along with high-level built-in data structures and large and extensive standard library. Python is also attractive for use as a scripting or glue language to connect existing components together; it supports modules and packages, which favors program modularity and code reuse.

Python's design philosophy prioritizes code readability, and its syntax allows programmers to express concepts in fewer lines of code and in a clearer manner. It closely

resembles the English language, using keywords like “not” and “in”, which makes it easy to discern; this is also stimulated by the lack of redundant punctuation , most importantly, the lack of curly brackets: it uses whitespace indentation to delimit blocks. Also, Python has a strict set of rules, known as PEP 8, that guide every Python developer into how to format their code. Therefore, any script, whether it was written by a novice or a professional, will look very similar and be just as easy to read.

Python interpreters are available for many operating systems, making Python code portable. With the help of third-party tools, such as Py2exe or Pyinstaller, Python code can be packaged into stand-alone executable programs for most of the operating systems, so Python-based software can be used on those environments with no need to install a Python interpreter.

Python is such a popular choice for programmers by virtue of the effortless and fast debugging it provides and¸ consequently, increased productivity, due to the absence of the compilation step. A segmentation fault will never be encountered, alternatively, when an error is detected, the interpreter raises an exception, if the program doesn't catch the exception, the interpreter prints a stack trace. The fast edit-test-debug cycle makes it very quick and effective to debug a program just by adding some print statements here and there in the code. Moreover, the debugger is also written in Python.

Some programming-language particularities of Python are:

- a variety of basic data types are available: numbers (floating point, complex, and unlimited-length long integers), strings (both ASCII and Unicode);
- three powerful data types: two sequence classes: lists and tuples, and a mapping class: dictionaries;
- it supports object-oriented programming with classes and multiple inheritance,
- code can be grouped into modules and packages;
- the language supports raising and catching exceptions, resulting in cleaner error handling;
- data types are strongly and dynamically typed, mixing incompatible types (e.g. attempting to add a string and a number) causes an exception to be raised, so errors are caught sooner;

- it contains advanced programming features such as generators and list comprehensions;
- automatic memory management: relieves the developer from having to manually allocate and free memory in your code.

One of Python's greatest strengths is its large standard library. The Python Package Index, the official repository of third-party software for Python, contains more than 72,000 packages offering a wide range of functionality, including: graphical user interfaces, web frameworks, multimedia, databases, networking and communications; test frameworks, automation and web scraping, documentation tools, system administration; scientific computing, text processing, image processing.

What is more, Python is free software, on one hand it is free to download, use and include it in applications and, on the other hand, it can be freely modified and distributed because the language is available under an open source license. It also has a strongly built and enthusiastic community which aims to continually expand the open source knowledgebase, as well as help the beginner users of Python.

Having gained fundamental knowledge of Python during my bachelor studies and having discovered all its advantages and rising acclaim among developers, I decided that developing the CIRCUS application using Python will give me the chance to improve my proficiency with this tool. Moreover, I exploited the features and strengths of Python programming language which proved to fit the project's requirements. I used version 2.7.10.

## 3.3 TkInter

The Tkinter module, which stands for "Tk interface", is the standard Python interface to the Tk GUI toolkit and is included with the standard Microsoft Windows and Mac OS X install of Python.

Tkinter is implemented as a Python wrapper around a Tcl interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

Tcl stands for Tool Command Language and it is a scripting language commonly used for rapid prototyping, scripted applications, GUIs and testing. The combination of Tcl and the Tk GUI toolkit is referred to as Tcl/Tk.

Some of the main components of TkInter are:

- Basic widgets: Toplevel, Frame, Button, Checkbutton, Entry, Label, Listbox, OptionMenu, Photoimage, Radiobutton, Scale;
- Complex widgets: Canvas, Text; these have taggable contents that act like objects;
- Menu, Menubutton, Scrollbar;
- Tk Variables: StringVar, IntVar, DoubleVar, BooleanVar; these are data containers needed by certain widgets, they can trigger callbacks when their data is changed;
- Extra modules: tkColorChooser, tkFileDialog, tkMessageBox and more;
- Geometry management: pack, grid, place; they are used to specify the relative positioning of the positioning of widgets within their container;
- Events: mouse events (e.g. ButtonPress), keyboard events (e.g. KeyRelease) and window events (e.g. Configure, FocusOut)

When developing the GUI of CIRCUS I used extensively the Canvas widget employing all of its features, along with other basic widgets, event handing, callback functions and tkFileDialog module. What TkInter gains in being simple, effective, powerful and stable lacks in visual attractiveness, but the final outcome proved to be successful.

## 3.4 Python Imaging Library (PIL) & Pillow

Python Imaging Library (abbreviated as PIL) is a free library for the Python programming language adds image processing capabilities to your Python interpreter. It is available for Windows, Mac OS X and Linux.

PIL supports many file formats including PPM, PNG, JPEG, GIF, TIFF, and BMP. It includes most of the standard procedures for image manipulation including: masking and transparency handling, image filtering (blurring, contouring, smoothing, edge finding), image enhancing (sharpening, adjusting brightness, contrast or color), per-pixel manipulations and more basic image operations like: resizing, cropping, rotating, color conversion and format conversion.

Since development stopped in 2011, another project called Pillow has forked the PIL repository, furthermore this fork has been adopted as a replacement for the original PIL.

I used PIL to maintain transparency and quality of the images, they were stored in memory as PNG files and TkInter does not support this file format. I also used this library for transforming PNG images, like rotating at a given angle. As told in §7.2, some problems were encountered during the development of CIRCUS due to this library.

## 3.5 MATLAB Engine API for Python

The MATLAB Engine API for Python provides a package for Python to call MATLAB as a computational engine. The engine supports the reference implementation (CPython) for Python versions 2.7, 3.3, and 3.4. After the installation of the package, the engine can be called during Python sessions.

The matlab package contains the MATLAB Engine API for Python and a set of MATLAB array classes in Python. The engine provides functions to call MATLAB, and the array classes provide functions to create MATLAB arrays as Python objects. You can create an engine and call MATLAB functions with matlab.engine. You can create MATLAB arrays in Python by calling constructors of an array type (for example, matlab.double to create an array of doubles). MATLAB arrays can be input arguments to MATLAB functions called with the engine.

This API provides the means to treat .m functions and scripts with the same ease as Python functions. You can call a MATLAB function from Python code with or without input arguments. When you want to retrieve the output you only have to tell the engine how many arguments the function returns; the number of arguments can be 0.

When working with MATLAB, it comes without saying that the main data type used is the matrix; since Python does not have an explicit data type for arrays (a matrix is treated as a list of lists) the engine provides constructors for MATLAB matrices which can receive a list of lists as input. Any other data conversion is done automatically by the engine, for example MATLAB's double and single are converted to Python's float and logical is transformed in bool.

Another interesting and extremely useful feature of this API is that the possible errors thrown by MATLAB can be caught by the Python application, thus avoiding it to crash and allowing further troubleshooting.

MATLAB Engine API for Python has some limitations: it cannot connect to MATLAB on a remote machine, it passes only positional arguments to MATLAB functions and it is not thread-safe.

I used this tool as a liaison between the GUI, data acquiring and data manipulation layers and the computational layer of the application which consists of .m functions with heavy matrix manipulations.

## 3.6 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like TkInter, wxPython or Qt.

You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code.

For simple plotting the pyplot interface provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

I used this tool to display some results after post-processing i.e. to plot amplitude-frequency characteristics of the behavior of certain circuit elements in alternate current. I integrated this with the TkInter toolkit.

# 4 FUNDAMENTALS ON NUMERICAL ANALYSIS OF ELECTRIC CIRCUITS

## 4.1 Modeling and analysis problem

The electric circuit is a physical object which accomplishes a certain function. It represents a practical implementation of a system (fig. 4.1) to convert an input signal $s(t)$ to a response $r(t)$ [3,4].

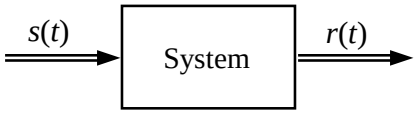



Fig. 4.1. Treatment of an electric circuit as a system

The signal can be any electrical quantity (voltage or current) while the response can be a voltage, a current or other quantity expressed in terms of voltages and currents (e.g. power). The correspondence between a circuit and a system is not unique, because the same system can be built as circuits with different structures.

The problems related to electric circuits are of two categories [2-6]:

- **Analysis** of a given circuit. The circuit structure is known, as well as the types and parameters of all circuit components. The input signal is also known, while the response is seeking. In common cases, the solution of an analysis problem is unique.

- **Synthesis** of a circuit which accomplishes a given function. If a synthesis problem has a solution, this is not unique.

The systematical analysis of circuits requires simplified models associated to the real circuit, named *circuit models* (fig. 4.2). They are concepts that use approximations and/or idealizations to simplify the study of real phenomena. The circuit models are represented with graphical symbols as *electric diagrams* together with the characteristics of the elements. The elements of the circuit model are, at their turn, idealizations of real elements, and they are named *ideal elements* (example: ideal capacitor, ideal resistor, etc.).

The ideal elements are described by extremely simple mathematical equations, so that the model circuit can be described by simple equations too. The assembly of these equations

is the mathematical model of the circuit (fig. 4.2).

By modeling, a problem of circuit analysis becomes a mathematical one, which requires the solving of an equations system. The solution of the equation system is accepted as solution of the circuit analysis problem.

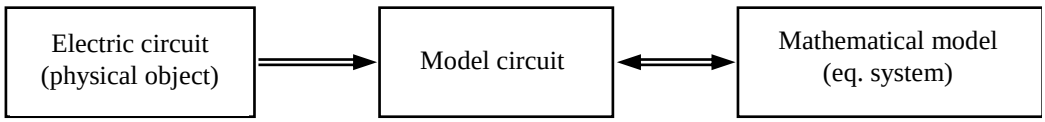



Fig. 4.2. Modeling of electric circuits

The analysis is acceptable if their results are found, with acceptable deviations, in the corresponding quantities (voltages and currents) of the real circuit.

The choosing of idealized models depends on the structure and on the working regime of the circuit, so that the same circuit could accept many models, depending on some working parameters (as the working the frequency).

Although the real circuits have generally unique solutions (excepting particular situations), the solution may not be found because of wrong modeling. On the other hand, in practice, wrong solution obtained through analysis can be interpreted as correct.

## 4.2 Analysis of electric circuits through numerical computing techniques

The electric circuits used in practice have complex structures, so that it is difficult to build and solve the associated mathematical models. In most practical cases, this task could be impossible without using electronic computers. Due to the particularities of real circuits, they can be classified in categories which can be treated in unitary manner. The unitary treatment can be completed using dedicated software tools, which accomplish certain functions as follows:

- allow specifying the input data as a text or electric diagram;
- build automatically a mathematical model;
- solve the equations system;
- give the results as text files or graphics, depending on particularities.

The commercial computing programs from the market cannot substitute totally the

specialist in circuit analysis, they are complementary tools only. They cannot be used effectively without knowing the principles and methods of circuit analysis.

None of the dedicated programs can cover all situations possible in engineering practice. Another significant disadvantage of such programs is related to restrictions to intervene into their body to adapt it for certain particular applications. As consequence, it is totally explicable the necessity to develop new software tools for circuit analysis, more flexible and adaptable to particular applications.

## 4.3 Elements of topology of electric circuits

### 4.3.1. Circuit graphs

In order to treat the methods of analysis in a systematic manner, one needs to develop simple and robust algorithms intended for topology description, i.e. the interconnection of circuit elements. An algorithm for topological analysis must be able to convert any circuit into a unitary form, which requires a minimum amount of memory and holds the entire information needed for the analysis process. The technical literature presents several methods for topological analysis which use the same main concepts: *branch*, *node* (or *vertex*), *loop* of circuit.

The circuit *branch* is a part of a circuit with no ramifications; it is in fact one two-terminal element. The *node* is a point where one or many branches are convergent. The *grade* of a node is numerically equal to the number of branches connected together in this node. The *loop* is a polygon built from circuit branches.

For a given circuit it is useful indexing the branches and nodes with natural numbers, as it is shown in the example of fig. 4.3.

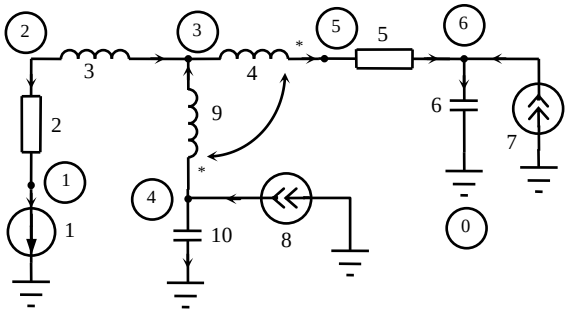



Fig. 4.3. Example circuit for topology study

A systematic and simple description of the circuit topology is possible by way of a

graph, called a *circuit graph*. It is obtained by replacing each two-terminal of the circuit (each circuit branch) with a line, called graph branch. If for each branch a certain sense of reference for the current flow and voltage is chosen, one obtains a *directed graph*, or simply *digraph* [2]. As an example, the directed graph of the circuit from fig. 4.3 is shown in fig. 4.4.

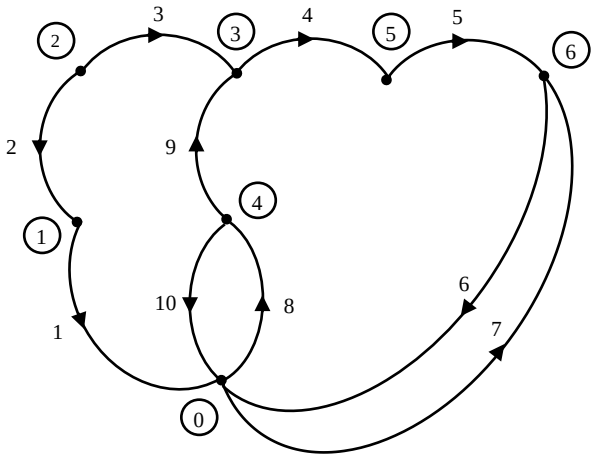



Fig. 4.4. Circuit direct graph of the example from fig. 4.3

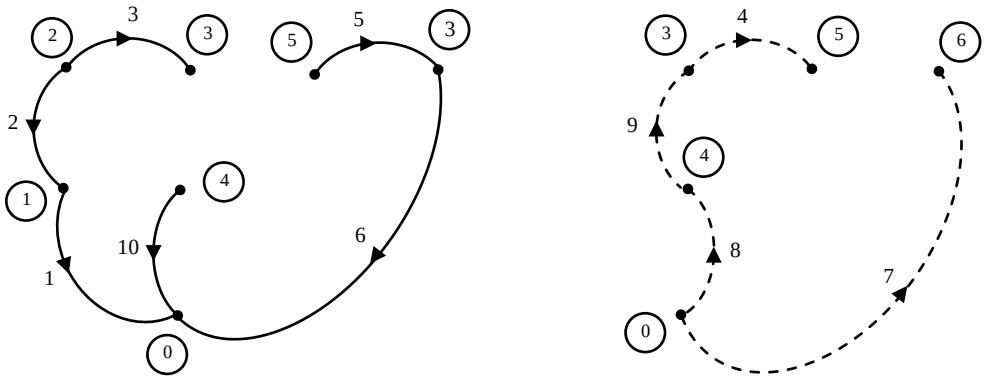
A part of a graph is called *subgraph*. Two subgraphs are complementary if they contain no common branch and if they contain together all branches and nodes of the circuit graph. The subgraph which contains two nodes of grade 1, all other nodes having the grade of 2, is called a *path*. If there is a path from any point to any other point in the graph, the graph is called *connected graph*.

The *tree* of a graph is a connected subgraph which accomplishes simultaneously the conditions as follows: it contains all nodes and it does not contain loops. As consequence, the number of tree branches is one unit less than the number of nodes. The *cotree* is a subgraph complementary to the tree. The cotree branches are called *links* as well.

The tree of a certain circuit (graph) is not unique. A particular tree (called normal tree) is useful for circuit analysis; it accomplishes the restrictions to contain [2-4]:

- all independent voltage sources;
- as many capacitors as possible;
- as few inductors as possible;
- no independent current source;
- any number of resistors.

In fig. 4.5 a normal tree is shown nearby its cotree, for the circuit of fig. 4.3.




(a) (b)

Fig. 4.5. A normal tree for the circuit of fig. 4.3 (a) and its cotree (b)

### 4.3.2. Description of topology matrix

Topology description through *incidence matrices* allows storing the information in a small amount of computer memory. It is also extremely useful for systematic algorithms to build mathematical models in circuit analysis. The incidence matrices are related to the circuit directed graph.

*Matrix of graph description*

To describe in a compact manner the circuit structure, one can assign a text line containing all information on the position of the element, its type and electrical parameters [3,4]. For a quick access to the information, it is useful to organize it as a matrix.

The matrix of graph description has a number of lines numerically equal to the circuit (graph) branches and four or more columns, depending on particular configuration of the circuit [3].

The first column contains codes assigned to the circuit elements from circuit branches. These codes are natural increasing numbers showing the priority of the circuit element to take part in a normal tree, according to the table 2.1. The second column contains the indexes of circuit branches. The 3$^{rd}$ and 4$^{th}$ columns contain the initial node and the final node respectively of each branch, according to the arbitrary chosen flowing direction of the current (shown on the directed graph).

Table 2.1

| Element code | Circuit element |
|---|---|
| 1 | Independent voltage source |
| 2 | Capacitor |
| 3 | Resistor |
| 4 | Inductor |
| 5 | Independent current source |

If the circuit contains magnetic couplings between coils, a 5th column is added, which contains the index of the pair inductor in the lines of coupled inductors, or "zero" in all other lines of the matrix. Additional columns are added to include the circuit element parameters (as capacities, resistances, electromotive forces, etc.)

*Node-branch incidence matrix*

The node-branch incidence matrix has a number of lines equal to the number of nodes and a number of columns equal to the number of circuit (graph) branches:

$$\boldsymbol{A}_0 = \left[ a_{ij} \right];\ i = \overline{1,\ n}\ ,\ \ j = \overline{1,\ l}\ . \tag{4.1}$$

The elements of the matrix are the coefficients of incidence of branches at nodes, defined as follows:

$$a_{ij} = \begin{cases} +1 & \text{if } i \text{ is the initial node of the branch } j; \\ 0 & \text{if the branch } j \text{ does not join the node } i; \\ -1 & \text{if } i \text{ is the final node of the branch } j. \end{cases}$$

This shows that the sum of all elements of each column is equal to zero, so that the lines are linearly dependent; any line is the sum of all other $n-1$ lines. As consequence, the rank of this matric is equal to $n-1$, and one keeps only $n-1$ lines for a complete description of the incidence of circuit branches to the nodes. The reduced form of the node- branch incidence matrix is obtained:

$$\boldsymbol{A} = \left[ a_{ij} \right];\ i = \overline{1,\ n-1}\ ,\ \ j = \overline{1,\ l}\ , \tag{4.2}$$

and the missing line corresponds to the node chosen as reference for node voltages. This is the form used in common practice. It is useful its partitioning by tree branches and cotree branches respectively:

$$\boldsymbol{A}' = \left[ \boldsymbol{A}_r \ \vdots\ \boldsymbol{A}_c \right]. \tag{4.3}$$

The partition $\boldsymbol{A}_r$ is always square and nonsingular.

It is extremely useful that the node-branch incidence matrix allows expressing the Kirchhoff's current low (KCL) for the whole circuit in the compact form:

$$\boldsymbol{A}\cdot\boldsymbol{i}=\boldsymbol{0}_{(n-1)\times 1}, \tag{4.4}$$

where $\boldsymbol{i}=\begin{bmatrix} i_1 & i_2 & \dots & i_l \end{bmatrix}^{\mathrm{t}}$ is the vector of branch currents.

By partitioning the vector of branch currents relative to the tree / cotree branches, the KCL (4.4) becomes:

$$\begin{bmatrix} \boldsymbol{A}_r & \boldsymbol{A}_c \end{bmatrix}\cdot\begin{bmatrix} \boldsymbol{i}_r \\ \boldsymbol{i}_c \end{bmatrix}=\boldsymbol{A}_r\boldsymbol{i}_r+\boldsymbol{A}_c\boldsymbol{i}_c=\boldsymbol{0}_{(n-1)\times 1}, \tag{4.5}$$

from where:

$$\boldsymbol{i}_r=-\left(\boldsymbol{A}_r\right)^{-1}\boldsymbol{A}_c\boldsymbol{i}_c\,. \tag{4.6}$$

which proves that only the cotree branch currents are independent variables. This is extremely useful to simplify the mathematical models.

Thus, the node-branch incidence matrix allows expressing the branch voltages in terms of node voltages:

$$\boldsymbol{u}=\boldsymbol{A}^{\mathrm{t}}\cdot\boldsymbol{v}, \tag{4.7}$$

where $\boldsymbol{u}=\begin{bmatrix} u_1 & u_2 & \dots & u_l \end{bmatrix}^{\mathrm{t}}$ is the vector of branch voltages, and $\boldsymbol{v}=\begin{bmatrix} v_1 & v_2 & \dots & v_{n-1} \end{bmatrix}^{\mathrm{t}}$ is the vector of node voltages. The voltage of the reference node is missing from the voltage vector.

*Branch-cutset incidence matrix*

A closed surface crossed by only one branch of a normal tree is called fundamental cutset. The sense of the tree branch which crosses the cutset imposes the reference sense of the cutset. There are $n-1$ fundamental cutsets for any given circuit. In fig. 4.6 the system of fundamental cutsets of the circuit directed graph from fig. 4.4 is shown. The fundamental cutsets are indexed as the corresponding tree branches.

The branch-cutset incidence matrix has $n-1$ lines and $l$ columns, where $l$ is the number of circuit (graph) branches:

$$\boldsymbol{Q}=\begin{bmatrix} q_{ij} \end{bmatrix};\ i=\overline{1,\,n-1},\ \ j=\overline{1,\,l}\,. \tag{4.8}$$

The elements of the matrix are the coefficients of incidence of branches to cutsets, defined as follows:

$$q_{ij} = \begin{cases} +1 & \text{if the branch } j \text{ crosses the cutset } i \text{ in the same direction as its reference;} \\ 0 & \text{if the branch } j \text{ does not cross the cutset } i; \\ -1 & \text{if the branch } j \text{ crosses the cutset } i \text{ in opposite direction compared to its reference.} \end{cases}$$

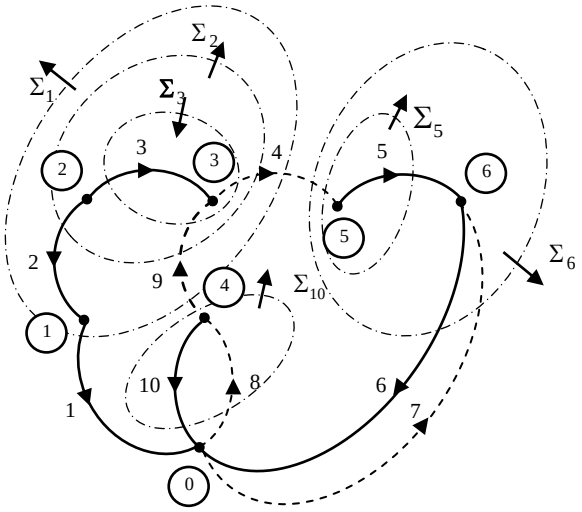



Fig. 4.6. System of fundamental cutsets of the graph from fig. 4.4

Since the partition corresponding to the tree is always equal to the unity square matrix of size $(n-1)\times(n-1)$, the complete description of branches to cutsets can be made the partition corresponding to the cotree. This latter is called *essential incidence matrix* or tree branch-links incidence matrix:

$$\boldsymbol{Q} = \left[ \mathbf{1}_{(n\text{-}1)\times(n\text{-}1)} \;\vdots\; \boldsymbol{D} \right]. \tag{4.9}$$

The size of this matrix is $(n-1)\times b$, where $b$ is the number of circuit links.

The branch-cutset incidence matrix allows expressing the KCL for the whole circuit under the compact form:

$$\boldsymbol{Q}\cdot\boldsymbol{i} = \mathbf{0}_{(n-1)\times 1}. \tag{4.10}$$

Using the partitioning (4.6), and with similar partitioning of the branch currents, the equation (4.7) becomes:

$$\left[ \mathbf{1}_{(n\text{-}1)\times(n\text{-}1)} \;\vdots\; \boldsymbol{D} \right] \cdot \begin{bmatrix} \boldsymbol{i}_r \\ \cdots \\ \boldsymbol{i}_c \end{bmatrix} = \boldsymbol{i}_r + \boldsymbol{D}\cdot\boldsymbol{i}_c = \mathbf{0}. \tag{4.11}$$

which allows expressing the tree-branch currents in terms of cotree-branch currents:

$$\boldsymbol{i}_r = -\boldsymbol{D} \cdot \boldsymbol{i}_c \,. \tag{4.12}$$

By comparing (4.6) and (4.12) one finds that the essential incidence matrix can be computed starting from the node-branch incidence matrix:

$$\boldsymbol{D} = \left(\boldsymbol{A}_r\right)^{-1} \boldsymbol{A}_c \,. \tag{4.13}$$

*Branch-loop incidence matrix*

The branch-loop incidence matrix is of size $b \times l$:

$$\boldsymbol{B} = \left[b_{ij}\right];\ i = \overline{1, b}\ ,\ \ j = \overline{1, l}\ . \tag{4.14}$$

The elements of the matrix are the coefficients of incidence of branches to loops, defined as follows:

$$b_{ij} = \begin{cases} +1 & \text{if the branch } j \text{ belongs to the loop } i \text{ and it has the same direction as the loop;} \\ 0 & \text{if the branch } j \text{ does not belong to the loop } i; \\ -1 & \text{if the branch } j \text{ belongs to the loop } i \text{ and it has an opposite direction as the loop.} \end{cases}$$

The matrix structure explains that its partition corresponding to the cotree branches is always square unity matrix of size $b \times b$. Its essential part corresponds to the tree branches, and this is the transpose of the essential incidence matrix with the sign minus:

$$\boldsymbol{B} = \left[-\boldsymbol{D}^{\mathrm{t}} \,\vdots\, \boldsymbol{1}_{b\times b}\right]\ . \tag{4.15}$$

The branch-loop incidence matrix allows expressing the Kirchhoff's voltage law (KVL) for the whole circuit, in the compact form:

$$\boldsymbol{B} \cdot \boldsymbol{u} = \boldsymbol{0}_{b\times 1}\,. \tag{4.16}$$

Using the partition (4.15) and by partitioning the vector of branch voltages related to tree / cotree branches, (4.16) becomes:

$$\left[-\boldsymbol{D}^{\mathrm{t}} \,\vdots\, \boldsymbol{1}_{b\times b}\right] \cdot \begin{bmatrix} \boldsymbol{u}_r \\ \cdots \\ \boldsymbol{u}_c \end{bmatrix} = -\boldsymbol{D}^{\mathrm{t}} \cdot \boldsymbol{u}_r + \boldsymbol{u}_c = \boldsymbol{0}\ , \tag{4.17}$$

which allows expressing the cotree-branch voltages in terms of tree-branch voltages:

$$\boldsymbol{u}_c = \boldsymbol{D}^{\mathrm{t}} \cdot \boldsymbol{u}_r\,. \tag{4.18}$$

The expression (4.18) proves that only the tree branches are independent variables.

## 4.4 Topological analysis of DC analog circuits

### 4.4.1 General mathematical model

The general form of a mathematical model of a linear lumped circuit in DC behavior is a linear algebraic equation system of size 2*l* which contains:

- $n-1$ equations given by KCL:

$$\boldsymbol{A}\cdot\boldsymbol{I}=\boldsymbol{0}_{(n\text{-}1)\times 1}; \tag{4.19}$$

- $b$ equations given by KVL:

$$\boldsymbol{B}\cdot\boldsymbol{U}=\boldsymbol{0}_{b\times 1}; \tag{4.20}$$

- $l$ characteristic equations of the ideal elements of the model circuit:

$$\boldsymbol{f}(\boldsymbol{U},\boldsymbol{I})=\boldsymbol{0}_{l\times 1}. \tag{4.21}$$

The components of the vector function (4.21) depend on the type of element, as follows:

- for linear resistors:

$$U_k-R_kI_k=0 \text{ sau } I_k-G_kU_k=0; \tag{4.22}$$

- for independent voltage sources:

$$U_k=-E_k; \tag{4.23}$$

- for independent current sources:

$$I_k=J_k; \tag{4.24}$$

The variables (unknowns) of the equations system (4.19)-(4.21) are the branch currents and voltages:

$$\begin{aligned}\boldsymbol{I}&=\begin{bmatrix}I_1 & I_2 & \dots & I_l\end{bmatrix}^{\mathrm{t}}\\ \boldsymbol{U}&=\begin{bmatrix}U_1 & U_2 & \dots & U_l\end{bmatrix}^{\mathrm{t}},\end{aligned} \tag{4.25}$$

and its solution defines an operation point.

A practical and effective manner to build a mathematical model is based on partitioning the circuit branches into $l_p$ passive branches, $l_E$ branches with independent voltage sources and $l_J$ branches with independent current sources. The incidence matrices and the vectors of branch currents and voltages will be partitioned similarly:

$$\boldsymbol{A}=\begin{bmatrix}\boldsymbol{A}_p & \boldsymbol{A}_E & \boldsymbol{A}_J\end{bmatrix}$$
$$\boldsymbol{B}=\begin{bmatrix}\boldsymbol{B}_p & \boldsymbol{B}_E & \boldsymbol{B}_J\end{bmatrix}, \tag{4.26}$$
$$\boldsymbol{I}=\begin{bmatrix}\boldsymbol{I}_p^{\mathrm{t}} & \boldsymbol{I}_E^{\mathrm{t}} & \boldsymbol{I}_J^{\mathrm{t}}\end{bmatrix}^{\mathrm{t}}$$
$$\boldsymbol{U}=\begin{bmatrix}\boldsymbol{U}_p^{\mathrm{t}} & \boldsymbol{U}_E^{\mathrm{t}} & \boldsymbol{U}_J^{\mathrm{t}}\end{bmatrix}^{\mathrm{t}}. \tag{4.27}$$

The matrix forms of the characteristic equations under one of the forms (4.22)-(4.24):

- for passive branches:

$$\boldsymbol{I}_p\cdot\boldsymbol{R}-\boldsymbol{U}_p=\boldsymbol{0}_{l_p\times 1}\ \text{ or }\ \boldsymbol{I}_p-\boldsymbol{G}\boldsymbol{U}_p=\boldsymbol{0}_{l_p\times 1}, \tag{4.28}$$

where the square diagonal matrices of resistances ( $\boldsymbol{R}_{l_p\times l_p}$ ) and conductances ( $\boldsymbol{G}_{l_p\times l_p}$ ) of passive branches were used;

- for independent voltage sources:

$$\boldsymbol{U}_E=-\boldsymbol{E}, \tag{4.29}$$

where the vector of electromotive forces of independent voltage sources ( $\boldsymbol{E}_{l_E\times 1}$ ) was used;

- for independent current sources:

$$\boldsymbol{I}_J=\boldsymbol{J}, \tag{4.30}$$

where the vector of currents of independent current sources ( $\boldsymbol{J}_{l_J\times 1}$ ) was used.

Therefore, the mathematical model (4.19)-(4.21) becomes:

$$\begin{cases}\begin{bmatrix}\boldsymbol{A}_p & \boldsymbol{A}_E & \boldsymbol{A}_J\end{bmatrix}\cdot\begin{bmatrix}\boldsymbol{I}_p\\ \boldsymbol{I}_E\\ \boldsymbol{I}_J\end{bmatrix}=\boldsymbol{0}_{(n-1)\times 1}\\ \begin{bmatrix}\boldsymbol{B}_p & \boldsymbol{B}_E & \boldsymbol{B}_J\end{bmatrix}\cdot\begin{bmatrix}\boldsymbol{U}_p\\ \boldsymbol{U}_E\\ \boldsymbol{U}_J\end{bmatrix}=\boldsymbol{0}_{b\times 1}\\ \boldsymbol{I}_p\cdot\boldsymbol{R}-\boldsymbol{U}_p=\boldsymbol{0}_{l_p\times 1}\ \text{ sau }\ \boldsymbol{I}_p-\boldsymbol{G}\boldsymbol{U}_p=\boldsymbol{0}_{l_p\times 1}\\ \boldsymbol{U}_E=-\boldsymbol{E}\\ \boldsymbol{I}_J=\boldsymbol{J}.\end{cases} \tag{4.31}$$

Since the equation system contains dependent variables, one can conceive reduced-order mathematical models equivalent with the extended form (4.31). Such models require a reduced amount of computation effort to provide the same solution as the general model [2-6]. Some of reduced-order mathematical models suitable for implementation in software applications are presented below.

### 4.4.2 Reduced-order mathematical models

*Kirchhoff's laws-based models*

By substituting the variables $\boldsymbol{U}_E$, $\boldsymbol{I}_J$ and $\boldsymbol{U}_p$ in (4.31), one obtains the equivalent mathematical model:

$$\begin{cases} \boldsymbol{A}_p\boldsymbol{I}_p + \boldsymbol{A}_E\boldsymbol{I}_E = -\boldsymbol{A}_J\boldsymbol{J} \\ \boldsymbol{B}_p\boldsymbol{R}\,\boldsymbol{I}_p + \boldsymbol{B}_J\boldsymbol{U}_J = \boldsymbol{B}_E\boldsymbol{E}, \end{cases} \tag{4.32}$$

whose compact matrix form is:

$$\left[\begin{array}{c|c|c} \boldsymbol{A}_p & \boldsymbol{A}_E & \boldsymbol{0}_{(n-1)\times l_J} \\ \hline \boldsymbol{B}_p\boldsymbol{R} & \boldsymbol{0}_{b\times l_E} & \boldsymbol{B}_J \end{array}\right] \cdot \begin{bmatrix} \boldsymbol{I}_p \\ \boldsymbol{I}_E \\ \boldsymbol{U}_J \end{bmatrix} = \begin{bmatrix} -\boldsymbol{A}_J\boldsymbol{J} \\ \boldsymbol{B}_E\boldsymbol{E} \end{bmatrix}. \tag{4.33}$$

The size of the equation system (4.33) is equal to the number of the circuit branches *l*. Its variables are the currents of passive branches, the currents of the independent voltage sources and the voltages of the independent current sources. To solve such an equation system, specific substituting methods are suitable [7,8].

A version of the method requires substituting the variables $\boldsymbol{U}_E$, $\boldsymbol{I}_J$ and $\boldsymbol{U}_p$ in (4.31), to obtain the equation system:

$$\begin{cases} \boldsymbol{A}_p\boldsymbol{G}\boldsymbol{U}_p + \boldsymbol{A}_E\boldsymbol{I}_E = -\boldsymbol{A}_J\boldsymbol{J} \\ \boldsymbol{B}_p\boldsymbol{U}_p + \boldsymbol{B}_J\boldsymbol{U}_J = \boldsymbol{B}_E\boldsymbol{E} \end{cases} \tag{4.34}$$

or

$$\left[\begin{array}{c|c|c} \boldsymbol{A}_p\boldsymbol{G} & \boldsymbol{0}_{(n-1)\times l_J} & \boldsymbol{A}_E \\ \hline \boldsymbol{B}_p & \boldsymbol{B}_J & \boldsymbol{0}_{b\times l_E} \end{array}\right] \cdot \begin{bmatrix} \boldsymbol{U}_p \\ \boldsymbol{U}_J \\ \boldsymbol{I}_E \end{bmatrix} = \begin{bmatrix} -\boldsymbol{A}_J\boldsymbol{J} \\ \boldsymbol{B}_E\boldsymbol{E} \end{bmatrix}. \tag{4.35}$$

wher voltages of passive branches were used as variables instead of the currents.

*Method of passive tree-branch voltages*

The mathematical model has a smaller size comparing to the previous method. One needs to build a normal tree and to perform a partitioning of circuit branches into $l_{pa}$ passive tree-branches, $l_{pc}$ passive cotree-branches (passive links), $l_E$ branches with independent voltage sources and $l_J$ branches with independent current sources. Therefore, the essential incidence matrix will be partitioned similarly:

$$\boldsymbol{D} = \begin{array}{c} a \setminus c \\ l_{pa} \\ l_E \end{array} \overset{\begin{array}{cc} l_{pc} & l_J \end{array}}{\left[\begin{array}{cc} \boldsymbol{D}_{pp} & \boldsymbol{D}_{pJ} \\ \boldsymbol{D}_{Ep} & \boldsymbol{D}_{EJ} \end{array}\right]}, \tag{4.36}$$

as well as the vectors of currents and voltages:

$$\begin{array}{c} \\ \boldsymbol{I} = \\ \boldsymbol{U} = \end{array} \overset{\begin{array}{cc} ramuri & coarde \end{array}}{\left[\begin{array}{cc|cc} \boldsymbol{I}_{pa}^{\mathrm{t}} & \boldsymbol{I}_{E}^{\mathrm{t}} & \boldsymbol{I}_{pc}^{\mathrm{t}} & \boldsymbol{I}_{J}^{\mathrm{t}} \end{array}\right]^{\mathrm{t}}} \\ \left[\begin{array}{cc|cc} \boldsymbol{U}_{pa}^{\mathrm{t}} & \boldsymbol{U}_{E}^{\mathrm{t}} & \boldsymbol{U}_{pc}^{\mathrm{t}} & \boldsymbol{U}_{J}^{\mathrm{t}} \end{array}\right]^{\mathrm{t}}. \tag{4.37}$$

The KCL becomes:

$$\left[\begin{array}{cc|cc} \mathbf{1} & \mathbf{0} & \boldsymbol{D}_{pp} & \boldsymbol{D}_{pJ} \\ \mathbf{0} & \mathbf{1} & \boldsymbol{D}_{Ep} & \boldsymbol{D}_{EJ} \end{array}\right] . \left[\begin{array}{c} \boldsymbol{I}_{pa} \\ \hdashline \boldsymbol{I}_{E} \\ \boldsymbol{I}_{pc} \\ \boldsymbol{I}_{J} \end{array}\right] = \left[\begin{array}{c} \mathbf{0}_{l_{pa}\times 1} \\ \mathbf{0}_{l_{E}\times 1} \end{array}\right], \tag{4.38}$$

from where the tree-branch currents are expressed in terms of cotree-branch currents:

$$\left[\begin{array}{c} \boldsymbol{I}_{pa} \\ \boldsymbol{I}_{E} \end{array}\right] = -\left[\begin{array}{cc} \boldsymbol{D}_{pp} & \boldsymbol{D}_{pJ} \\ \boldsymbol{D}_{Ep} & \boldsymbol{D}_{EJ} \end{array}\right] . \left[\begin{array}{c} \boldsymbol{I}_{pc} \\ \boldsymbol{I}_{J} \end{array}\right] \tag{4.39}$$

or

$$\begin{array}{l} \boldsymbol{I}_{pa} = -\boldsymbol{D}_{pp}\boldsymbol{I}_{pc} - \boldsymbol{D}_{pJ}\boldsymbol{J} \\ \boldsymbol{I}_{E} = -\boldsymbol{D}_{Ep}\boldsymbol{I}_{pc} - \boldsymbol{D}_{EJ}\boldsymbol{J}, \end{array} \tag{4.40}$$

From the first equation (4.40) it results:

$$\boldsymbol{D}_{pp}\boldsymbol{I}_{pc} = -\boldsymbol{I}_{pa} - \boldsymbol{D}_{pJ}\boldsymbol{J} . \tag{4.41}$$

The KVL, expressed according to (4.17), is:

$$\left[\begin{array}{cc|cc} -\boldsymbol{D}_{pp}^{\mathrm{t}} & -\boldsymbol{D}_{Ep}^{\mathrm{t}} & \mathbf{1} & \mathbf{0} \\ -\boldsymbol{D}_{pJ}^{\mathrm{t}} & -\boldsymbol{D}_{EJ}^{\mathrm{t}} & \mathbf{0} & \mathbf{1} \end{array}\right] . \left[\begin{array}{c} \boldsymbol{U}_{pa} \\ \hdashline \boldsymbol{U}_{E} \\ \boldsymbol{U}_{pc} \\ \boldsymbol{U}_{J} \end{array}\right] = \left[\begin{array}{c} \mathbf{0}_{l_{pc}\times 1} \\ \mathbf{0}_{l_{J}\times 1} \end{array}\right] \tag{4.42}$$

or

$$\begin{array}{l} -\boldsymbol{D}_{pp}^{\mathrm{t}}\boldsymbol{U}_{pa} + \boldsymbol{D}_{Ep}^{\mathrm{t}}\boldsymbol{E} + \boldsymbol{U}_{pc} = \mathbf{0}_{l_{pc}\times 1} \\ -\boldsymbol{D}_{pJ}^{\mathrm{t}}\boldsymbol{U}_{pa} + \boldsymbol{D}_{EJ}^{\mathrm{t}}\boldsymbol{E} + \boldsymbol{U}_{J} = \mathbf{0}_{l_{J}\times 1}, \end{array} \tag{4.43}$$

from where the cotree-branch voltages are expressed in terms of tree-branch voltages:

$$\begin{aligned} \boldsymbol{U}_{pc} &= \boldsymbol{D}_{pp}^{\mathrm{t}}\boldsymbol{U}_{pa} - \boldsymbol{D}_{Ep}^{\mathrm{t}}\boldsymbol{E} \\ \boldsymbol{U}_{J} &= \boldsymbol{D}_{pJ}^{\mathrm{t}}\boldsymbol{U}_{pa} - \boldsymbol{D}_{EJ}^{\mathrm{t}}\boldsymbol{E}. \end{aligned} \tag{4.44}$$

By multiplying the first equation (4.43), on the left side, with the diagonal matrix of passive cotree-branch conductances $\boldsymbol{G}_{pc}$, of size $l_{pc} \times l_{pc}$, one obtains:

$$-\boldsymbol{G}_{pc}\boldsymbol{D}_{pp}^{\mathrm{t}}\boldsymbol{U}_{pa} + \boldsymbol{G}_{pc}\boldsymbol{D}_{Ep}^{\mathrm{t}}\boldsymbol{E} + \boldsymbol{G}_{pc}\boldsymbol{U}_{pc} = \boldsymbol{0}_{l_{pc}\times 1} \tag{4.45}$$

The last term from the left-hand is equal to the vector of passive cotree-branches:

$$\boldsymbol{G}_{pc}\boldsymbol{U}_{pc} = \boldsymbol{I}_{pc} \tag{4.46}$$

By multiplying the last equation, on the left side, with the partition $\boldsymbol{D}_{pp}$, of size $l_{pa} \times l_{pc}$, one obtains:

$$-\boldsymbol{D}_{pp}\boldsymbol{G}_{pc}\boldsymbol{D}_{pp}^{\mathrm{t}}\boldsymbol{U}_{pa} + \boldsymbol{D}_{pp}\boldsymbol{G}_{pc}\boldsymbol{D}_{Ep}^{\mathrm{t}}\boldsymbol{E} + \boldsymbol{D}_{pp}\boldsymbol{I}_{pc} = \boldsymbol{0}_{l_{pa}\times 1}. \tag{4.47}$$

The last term form the left-hand is replaced according to (4.41), then the passive tree-branch currents are expressed in terms of the voltages using the passive tree-branch conductance matrix $\boldsymbol{G}_{pa}$, of size $l_{pa} \times l_{pa}$

$$\boldsymbol{G}_{pa}\boldsymbol{U}_{pa} = \boldsymbol{I}_{pa}. \tag{4.48}$$

Finally, the mathematical model of the method is obtained:

$$\left(\boldsymbol{D}_{pp}\boldsymbol{G}_{pc}\boldsymbol{D}_{pp}^{\mathrm{t}} + \boldsymbol{G}_{pa}\right)\boldsymbol{U}_{pa} = \boldsymbol{D}_{pp}\boldsymbol{G}_{pc}\boldsymbol{D}_{Ep}^{\mathrm{t}}\boldsymbol{E} - \boldsymbol{D}_{pJ}\boldsymbol{J}. \tag{4.49}$$

This is an linear algebraic equation system of size $l_{pa}$, whose variables are the voltages of the passive tree-branches. After solving it, the voltages of cotree-branches are computed with (4.44), then the currents of passive cotree-branches with (4.46) and the tree-branch currents with (4.40).

*Method of passive cotree-branch currents*

The method of passive cotree-branch currents is an improved method of the cotree-branch currents method [3,4]. The size of its corresponding mathematical model is equal to the number of passive cotree-branches $l_{pc}$.

As the previously presented method, this needs to build a normal tree and to perform

similar partitioning of the circuit branches and essential incidence matrix. The KCL and KVL are expressed as (4.38) and (4.42). By substituting the variables $\boldsymbol{I}_{pa}, \boldsymbol{I}_{E}, \boldsymbol{U}_{pa}, \boldsymbol{U}_{pc}, \boldsymbol{U}_{J}$, one obtains the reduced-order mathematical model:

$$\left(\boldsymbol{R}_{pc} + \boldsymbol{D}_{pp}^{\mathrm{t}} \boldsymbol{R}_{pa} \boldsymbol{D}_{pp}\right) \boldsymbol{I}_{pc} = -\boldsymbol{D}_{Ep}^{\mathrm{t}} \boldsymbol{E} - \boldsymbol{D}_{pp}^{\mathrm{t}} \boldsymbol{R}_{pa} \boldsymbol{D}_{pJ} \boldsymbol{J} \ . \tag{4.50}$$

The model variables are the passive cotree-branch currents.

*Hybrid method*

The hybrid method uses both currents and voltages as variables: the voltages of passive tree-branches and the currents of the cotree-branches respectively. It has some significant advantages: the mathematical model is easy to build; the size of the mathematical model is relatively small; the post-processing of the results is easier than for other methods.

The method needs a normal tree and similar partitioning of the circuit branches and essential incidence matrix as the two methods described above. One substitutes the passive tree-branch currents in (4.41) using (4.48):

$$\boldsymbol{D}_{pp} \boldsymbol{I}_{pc} + \boldsymbol{G}_{pa} \boldsymbol{U}_{pa} = -\boldsymbol{D}_{pJ} \boldsymbol{J} \ . \tag{4.51}$$

One substitutes the passive cotree-branch voltages with the currents in (4.43), according to the obvious equation:

$$\boldsymbol{U}_{pc} = \boldsymbol{R}_{pc} \boldsymbol{I}_{pc} \tag{4.52}$$

to obtain:

$$-\boldsymbol{D}_{pp}^{\mathrm{t}} \boldsymbol{U}_{pa} + \boldsymbol{R}_{pc} \boldsymbol{I}_{pc} = -\boldsymbol{D}_{Ep}^{\mathrm{t}} \boldsymbol{E} \tag{4.53}$$

The equations system containing (4.51) and (4.53) represents the mathematical model of the hybrid method, with the compact form:

$$\begin{bmatrix} \boldsymbol{D}_{pp} & \boldsymbol{G}_{pa} \\ \boldsymbol{R}_{pc} & -\boldsymbol{D}_{pp}^{\mathrm{t}} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{I}_{pc} \\ \boldsymbol{U}_{pa} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{D}_{pJ} & \boldsymbol{0}_{l_{pa} \times l_E} \\ \boldsymbol{0}_{l_{pc} \times l_J} & -\boldsymbol{D}_{Ep}^{\mathrm{t}} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{J} \\ \boldsymbol{E} \end{bmatrix} . \tag{4.54}$$

*It is noticeable that in order to exploit the advantages exposed before, I implemented this hybrid method in the software application developed within this project.*

## 4.5 Topological analysis of AC analog circuits

### 4.5.1 Symbolic representation of harmonic quantities

If a linear circuit is excited by harmonic independent voltage and current sources with the same frequency, then an AC steady state is established. All currents and voltages are sinusoidal quantities with the same frequency as the sources. The power networks intended for generation, transport and distribution of the electric energy almost everywhere in the world are AC networks.

The AC steady state is specific to linear circuits only [5], and it is a particular case of variable working regime. The analysis through time-domain mathematical models requires laborious computation, and to avoid this issue it is more convenient to use a mathematical transformation, by replacing time-domain quantities with complex numbers (phasors) with no physical sense. As consequence, mathematical models with differential equations are replaced by algebraic equations with complex coefficients [2,5].

The common phasor method requires the transformation of sinusoidal quantities such:

$$x(t) = \sqrt{2}X\sin(\omega t + \gamma) \tag{4.55}$$

into complex numbers:

$$\underline{X} = X\,e^{j\gamma} = X\cos\gamma + jX\sin\gamma = \mathrm{Re}(\underline{X}) + j\,\mathrm{Im}(\underline{X}) \tag{4.56}$$

which contain no information on the working frequency. I used common notations for the angular frequency ( $\omega$ ) and the phase ( $\gamma$ ) of the sinusoidal quantity, capital letter for the RMS value ( $X$ ), and underlined capital for the complex quantity ( $\underline{X}$ ). For this kind of analysis, the concept of complex impedance associated to a passive two-pole is important [5]:

$$\underline{Z} \overset{D}{=} \frac{\underline{U}}{\underline{I}} = \frac{Ue^{j\gamma_U}}{Ie^{j\gamma_I}} = \frac{U}{I}e^{j(\gamma_U-\gamma_I)} = \frac{U}{I}\cos\varphi + j\frac{U}{I}\sin\varphi = R + jX = \mathrm{Re}(\underline{Z}) + j\,\mathrm{Im}(\underline{Z})\,. \tag{4.57}$$

I denoted $\varphi$ the phase difference between the voltage and the current flowing the two-pole, $R$ the real part (called resistance) and $X$ the imaginary part (called reactance) of the two-pole. The complex admittance is also defined as:

$$\underline{Y} \overset{D}{=} \frac{\underline{I}}{\underline{U}} = \frac{1}{\underline{Z}} = \frac{I}{U}\cos\varphi - j\frac{I}{U}\sin\varphi = G - jB\,, \tag{4.58}$$

where $G$ is called conductance and $B$ is susceptance.

According to the theorems of complex-domain representation [2,5], the characteristic

equations of the circuit elements in the complex domain are obtained:

- for resistors

$$\underline{U}_k = R_k \underline{I}_k \text{ or } \underline{I}_k = G_k \underline{U}_k ; \tag{4.59}$$

- for inductors

$$\underline{\Phi}_k = L_k \underline{I}_k , \tag{4.60}$$

where $\underline{\Phi}_k$ is the phasor associated to the magnetic flux of the inductor of the *k*-th circuit branch;

- for capacitors

$$\underline{q}_k = C_k \underline{U}_k , \tag{4.61}$$

where $\underline{q}_k$ is the phasor associated to the electric charge of the capacitor of the *k*-th circuit branch;

- for independent voltage sources

$$\underline{U}_k = -\underline{E}_k ; \tag{4.62}$$

- for independent current sources

$$\underline{I}_k = \underline{J}_k ; \tag{4.63}$$

The equations describing the reactive elements become:

- for magnetically insulated inductors:

$$\underline{U}_k = j\omega L_k \underline{I}_k ; \tag{4.64}$$

- for pairs of magnetically coupled inductors:

$$\begin{aligned} \underline{U}_k &= j\omega L_k \underline{I}_k + j\omega L_{kj} \underline{I}_j \\ \underline{U}_j &= j\omega L_{jk} \underline{I}_k + j\omega L_j \underline{I}_j , \end{aligned} \tag{4.65}$$

where the coupling inductances $L_{kj}$ and $L_{jk}$ could be positive or negative, depending on the type of the magnetic coupling. By solving the equation system (4.65) by respect to the currents, one obtains:

$$\begin{aligned} \underline{I}_k &= -\frac{j}{\omega(L_k L_j - L_{kj} L_{jk})} \cdot \left( L_j \underline{U}_k - L_{kj} \underline{U}_j \right) \\ \underline{I}_j &= -\frac{j}{\omega(L_k L_j - L_{kj} L_{jk})} \cdot \left( -L_{jk} \underline{U}_k + L_k \underline{U}_j \right); \end{aligned} \tag{4.66}$$

- for capacitors:

$$\underline{U}_k = -j\frac{1}{\omega C_k}\underline{I}_k \text{ sau } \underline{I}_k = j\omega C_k \underline{U}_k . \tag{4.67}$$

The general form of a mathematical model in the complex domain is an algebraic equation system of size $2l + l_L + l_C$ which contains:

- $(n-1)$ equations given by the KCL:

$$\boldsymbol{A}\cdot\underline{\boldsymbol{I}} = \boldsymbol{0}_{(n-1)\times 1}; \tag{4.68}$$

- $b$ equations given by the KVL:

$$\boldsymbol{B}\cdot\underline{\boldsymbol{U}} = \boldsymbol{0}_{b\times 1}; \tag{4.69}$$

- $l$ equations of the circuit elements with the particular forms (4.59)-(4.63):

$$\boldsymbol{f}(\underline{\boldsymbol{U}},\underline{\boldsymbol{I}},\underline{\boldsymbol{\Phi}},\underline{\boldsymbol{Q}}) = \boldsymbol{0}_{l\times 1}; \tag{4.70}$$

- $l_L$ equations of the form (4.64) for magnetically insulated inductors, and (4.65) for magnetically coupled inductors:

$$\underline{\boldsymbol{U}}_L = j\omega \boldsymbol{L}\underline{\boldsymbol{I}}_L , \tag{4.71}$$

where $l_L$ is the number of circuit branches with inductors, and $\boldsymbol{L}_{l_L\times l_L}$ is the square matrix of the inductances. This latter contains the self-inductances on the diagonal, and mutual inductances as non-diagonal elements. $\underline{\boldsymbol{U}}_L, \underline{\boldsymbol{I}}_L$ are the vectors of phasors of voltages and currents of circuit branches with inductors.

- $l_C$ equations of the form (4.67) for the capacitors:

$$\underline{\boldsymbol{I}}_C = j\omega \boldsymbol{C}\underline{\boldsymbol{U}}_C , \tag{4.72}$$

where $l_C$ is the number of circuit branches with capacitors, and $\boldsymbol{C}_{l_C\times l_C}$ is the square and diagonal matrix of the capacitances; $\underline{\boldsymbol{U}}_C, \underline{\boldsymbol{I}}_C$ are the vectors of phasors of voltages and currents of circuit branches with capacitors.

The variables of the so built system are the phasors of the voltages and currents of the circuit branches, as well as the magnetic fluxes of inductors and electric charges of capacitors. Once the solutions of the system are obtained, the time-domain quantities are computed using the inverse transform formula:

$$x(t) = \sqrt{2}\,|\underline{X}|\sin\left(\omega t + \operatorname{arctg}\frac{\operatorname{Im}(\underline{X})}{\operatorname{Re}(\underline{X})}\right). \tag{4.73}$$

### 4.5.2 General mathematical model

An effective method to build a generally valid mathematical model requires partitioning the circuit branches into $l_p$ passive branches, $l_E$ branches with independent voltage sources and $l_J$ branches with independent current sources. The incidence matrices, as well as the vectors of branch currents and voltages are partitioned similarly:

$$\begin{aligned} \underline{\boldsymbol{I}} &= \left[\underline{\boldsymbol{I}}_p^{\mathrm{t}} \quad \underline{\boldsymbol{I}}_E^{\mathrm{t}} \quad \underline{\boldsymbol{I}}_J^{\mathrm{t}}\right]^{\mathrm{t}} \\ \underline{\boldsymbol{U}} &= \left[\underline{\boldsymbol{U}}_p^{\mathrm{t}} \quad \underline{\boldsymbol{U}}_E^{\mathrm{t}} \quad \underline{\boldsymbol{U}}_J^{\mathrm{t}}\right]^{\mathrm{t}}. \end{aligned} \tag{4.74}$$

The characteristic equations of passive branches have the general form:

$$\underline{U}_k - \underline{Z}_k \underline{I}_k = 0 \text{ sau } \underline{Y}_k \underline{U}_k - \underline{I}_k = 0, \tag{4.75}$$

where the complex impedances $\underline{Z}_k$ and the complex admittances $\underline{Y}_k$ are:

- for resistors:

$$\underline{Z}_k = R_k \,;\; \underline{Y}_k = G_k \,; \tag{4.76}$$

- for magnetically insulated inductors:

$$\underline{Z}_k = j\omega L_k \,;\; \underline{Y}_k = -j\frac{1}{\omega L_k} \,; \tag{4.77}$$

- for magnetically coupled inductors, from (4.65) and (4.66) the impedance matrix and the admittance matrix are obtained:

$$\underline{\boldsymbol{Z}} = j\omega \begin{bmatrix} L_k & L_{kj} \\ L_{jk} & L_j \end{bmatrix}; \; \underline{\boldsymbol{Y}} = -j\frac{1}{\omega(L_k L_j - L_{kj} L_{jk})} \begin{bmatrix} L_j & -L_{kj} \\ -L_{jk} & L_k \end{bmatrix}; \tag{4.78}$$

- for capacitors:

$$\underline{Z}_k = -j\frac{1}{\omega C_k} \,;\; \underline{Y}_k = j\omega C_k \,. \tag{4.79}$$

In order to gain in terms of computational effort, the equations of the type (4.60), (4.61) may be missed, so that the mathematical model becomes:

$$\begin{cases} \begin{bmatrix} \boldsymbol{A}_p & \boldsymbol{A}_E & \boldsymbol{A}_J \end{bmatrix} \cdot \begin{bmatrix} \underline{\boldsymbol{I}}_p \\ \underline{\boldsymbol{I}}_E \\ \underline{\boldsymbol{I}}_J \end{bmatrix} = \boldsymbol{0}_{(n-1)\times 1} \\ \begin{bmatrix} \boldsymbol{B}_p & \boldsymbol{B}_E & \boldsymbol{B}_J \end{bmatrix} \cdot \begin{bmatrix} \underline{\boldsymbol{U}}_p \\ \underline{\boldsymbol{U}}_E \\ \underline{\boldsymbol{U}}_J \end{bmatrix} = \boldsymbol{0}_{b\times 1} \\ \underline{\boldsymbol{I}}_p \cdot \underline{\boldsymbol{Z}} - \underline{\boldsymbol{U}}_p = \boldsymbol{0}_{l_p\times 1} \text{ sau } \underline{\boldsymbol{I}}_p - \underline{\boldsymbol{Y}}\underline{\boldsymbol{U}}_p = \boldsymbol{0}_{l_p\times 1} \\ \underline{\boldsymbol{U}}_E = -\underline{\boldsymbol{E}} \\ \underline{\boldsymbol{I}}_J = \underline{\boldsymbol{J}}, \end{cases} \tag{4.80}$$

It is an algebraic equation system of size $2l$ .

Since the equation system (4.80) contains dependent variables, one can conceive reduced-order mathematical models to reduce the amount of computation effort [1-6,9].

### 4.5.3 Reduced-order mathematical models

*Kirchhoff's laws-based models*

By substituting the variables $\underline{\boldsymbol{U}}_E$, $\underline{\boldsymbol{I}}_J$ and $\underline{\boldsymbol{U}}_p$ in (4.80), one obtains the equivalent mathematical model:

$$\begin{cases} \boldsymbol{A}_p \underline{\boldsymbol{I}}_p + \boldsymbol{A}_E \underline{\boldsymbol{I}}_E = -\boldsymbol{A}_J \underline{\boldsymbol{J}} \\ \boldsymbol{B}_p \underline{\boldsymbol{Z}}\, \underline{\boldsymbol{I}}_p + \boldsymbol{B}_J \underline{\boldsymbol{U}}_J = \boldsymbol{B}_E \underline{\boldsymbol{E}}\,, \end{cases} \tag{4.81}$$

whose compact matrix form is:

$$\left[\begin{array}{c|c|c} \boldsymbol{A}_p & \boldsymbol{A}_E & \boldsymbol{0}_{(n-1)\times l_J} \\ \hline \boldsymbol{B}_p \underline{\boldsymbol{Z}} & \boldsymbol{0}_{b\times l_E} & \boldsymbol{B}_J \end{array}\right] \cdot \begin{bmatrix} \underline{\boldsymbol{I}}_p \\ \underline{\boldsymbol{I}}_E \\ \underline{\boldsymbol{U}}_J \end{bmatrix} = \begin{bmatrix} -\boldsymbol{A}_J \underline{\boldsymbol{J}} \\ \boldsymbol{B}_E \underline{\boldsymbol{E}} \end{bmatrix}. \tag{4.82}$$

The size of this equation system is equal to the number of the circuit branches *l*. Its variables are the currents of passive branches, the currents of the independent voltage sources and the voltages of the independent current sources. To solve such an equation system, specific substituting methods are suitable [7,8].

A version of the method requires substituting the variables $\underline{\boldsymbol{U}}_E$, $\underline{\boldsymbol{I}}_J$ şi $\underline{\boldsymbol{I}}_p$ from (4.80), to obtain the equation system:

$$\begin{cases} \boldsymbol{A}_p \underline{\boldsymbol{Y}}\underline{\boldsymbol{U}}_p + \boldsymbol{A}_E \underline{\boldsymbol{I}}_E = -\boldsymbol{A}_J \underline{\boldsymbol{J}}, \\ \boldsymbol{B}_p \underline{\boldsymbol{U}}_p + \boldsymbol{B}_J \underline{\boldsymbol{U}}_J = \boldsymbol{B}_E \underline{\boldsymbol{E}}, \end{cases} \tag{4.83}$$

with the compact matrix form:

$$\left[\begin{array}{c|c|c} \boldsymbol{A}_p \underline{\boldsymbol{Y}} & \boldsymbol{0}_{(n-1)\times l_J} & \boldsymbol{A}_E \\ \hline \boldsymbol{B}_p & \boldsymbol{B}_J & \boldsymbol{0}_{b\times l_E} \end{array}\right] \cdot \begin{bmatrix} \underline{\boldsymbol{U}}_p \\ \underline{\boldsymbol{U}}_J \\ \underline{\boldsymbol{I}}_E \end{bmatrix} = \begin{bmatrix} -\boldsymbol{A}_J \underline{\boldsymbol{J}} \\ \boldsymbol{B}_E \underline{\boldsymbol{E}} \end{bmatrix}. \tag{4.84}$$

*Method of passive tree-branch voltages*

The method of passive tree-branch voltages used similar principles as for DC circuits, and it requires a smaller number of variables comparing to the previous method. The obtained mathematical model is:

$$\left(\boldsymbol{D}_{pp} \underline{\boldsymbol{Y}}_c \boldsymbol{D}_{pp}^{\mathrm{t}} + \underline{\boldsymbol{Y}}_a\right) \underline{\boldsymbol{U}}_{pa} = \boldsymbol{D}_{pp} \underline{\boldsymbol{Y}}_c \boldsymbol{D}_{Ep}^{\mathrm{t}} \underline{\boldsymbol{E}} - \boldsymbol{D}_{pJ} \underline{\boldsymbol{J}}\,, \tag{4.85}$$

where:

$\underline{\boldsymbol{Y}}_c$ – complex admittance matrix of passive cotree branches;

$\underline{\boldsymbol{Y}}_a$ – complex admittance matrix of passive tree branches;

$\underline{\boldsymbol{U}}_{pa}$ – vector of voltage phasors of passive tree branches.

The admittance matrices are non-diagonal for circuits with magnetically coupled inductors only.

*Method of passive cotree-branch currents*

By extrapolating the result obtained for DC circuits, the mathematical model of the method is obtained:

$$\left(\underline{\boldsymbol{Z}}_c + \boldsymbol{D}_{pp}^{\mathrm{t}} \underline{\boldsymbol{Z}}_a \boldsymbol{D}_{pp}\right) \underline{\boldsymbol{I}}_{pc} = -\boldsymbol{D}_{Ep}^{\mathrm{t}} \underline{\boldsymbol{E}} - \boldsymbol{D}_{pp}^{\mathrm{t}} \underline{\boldsymbol{Z}}_a \boldsymbol{D}_{pJ} \underline{\boldsymbol{J}}\,, \tag{4.86}$$

where impedance matrices of passive branches occur.

*Hybrid method*

The mathematical model of the method is built using a similar procedure as for DC circuits:

$$\begin{bmatrix} \boldsymbol{D}_{pp} & \underline{\boldsymbol{Y}}_a \\ \underline{\boldsymbol{Z}}_c & -\boldsymbol{D}_{pp}^{\mathrm{t}} \end{bmatrix} \cdot \begin{bmatrix} \underline{\boldsymbol{I}}_{pc} \\ \underline{\boldsymbol{U}}_{pa} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{D}_{pJ} & \boldsymbol{0}_{l_{pa}\times l_E} \\ \boldsymbol{0}_{l_{pc}\times l_J} & -\boldsymbol{D}_{Ep}^{\mathrm{t}} \end{bmatrix} \cdot \begin{bmatrix} \underline{\boldsymbol{J}} \\ \underline{\boldsymbol{E}} \end{bmatrix}. \tag{4.87}$$

*I implemented the hybrid method for AC networks in the software application developed within this project.*

# 5 APPLICATION IMPLEMENTATION

## 5.1 Overview

The CIRCUS application is structured on three levels based with different functionality: the presentation layer, the data translation layer and the logic layer. They communicate between each other and each one performs several related functions executed by their sub-blocks as follows:

- Presentation Layer (TkInter GUI)

  Two blocks for providing data to the system:

  - **Draw Schema:** this block facilitates the creation of data to fuel the system; the user draws a schematic for the circuit that will be analyzed by the system specifying the values of the necessary parameters.
  - **Load/Save Schema:** allows the user to save a schematic at an external location or load a previously saved schematic.

  Two blocks for presenting the results obtained by the system:

  - **View Results:** this block displays the results received form the data translation layer in tabular form.
  - **Plot Results:** this block communicates with the result interpreter block of the middle layer if the user wishes a graphical representation of some of the results present in tabular form; this feature is available only for AC analysis with multiple frequencies.

- Data Translation Layer (Python scripts)
  - **Interpret Schema:** this block receives the information from one of the blocks of the GUI that is in charge of data acquisition, information concerning the electrical schematic drawn or loaded by the user; its role is to interpret the electrical diagram by establishing dependencies between the circuit elements and the circuit nodes.

- **Interpret Results:** this block communicates with the logic layer and maps the results received in unstructured form to the inherent quantities of the elements of the electrical circuit (currents and voltages); then sends the results to the presentation layer.

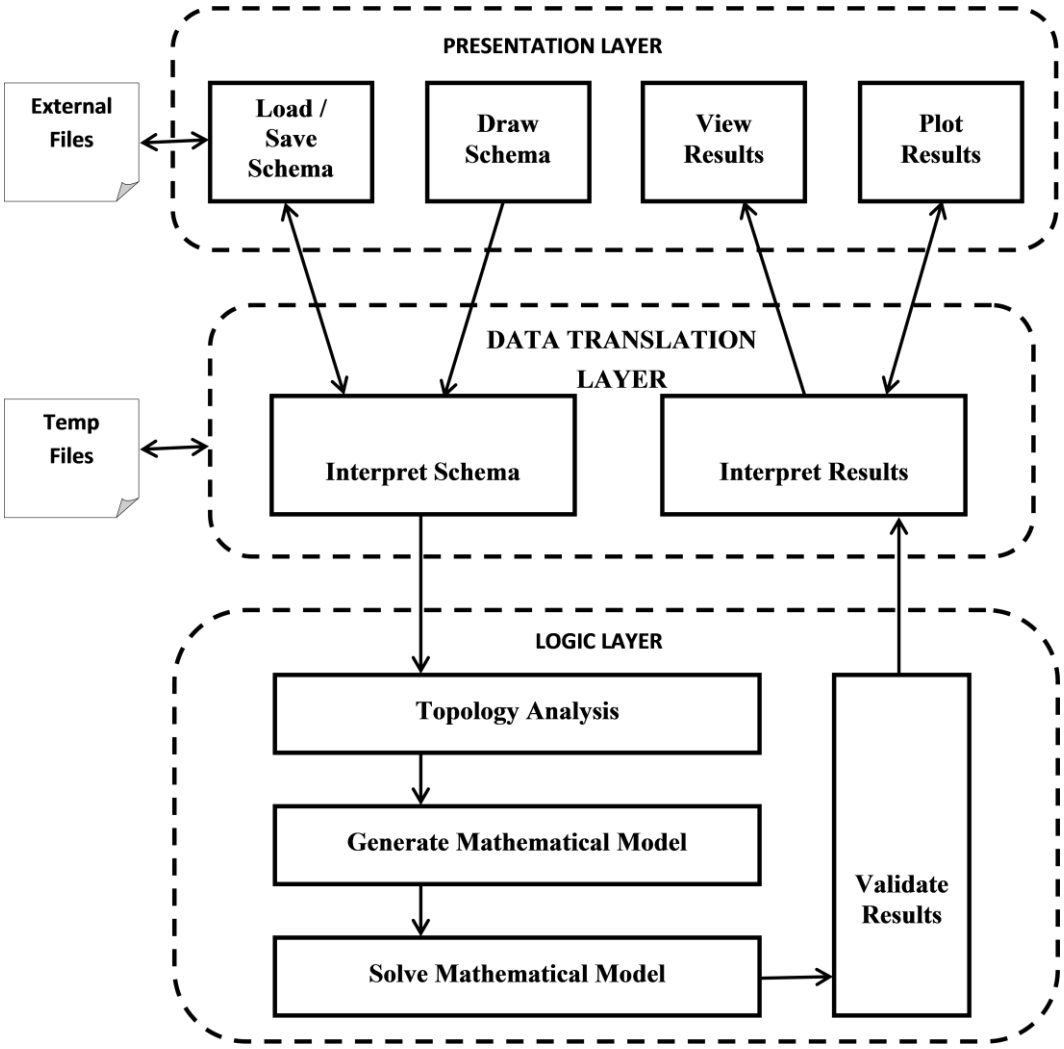



Fig 5.1 Application Design

- Logic Layer (MATLAB scripts)
  - **Analyze Topology:** this block converts the information received from the schema interpreter into a connectivity matrix which describes in a complete manner the topology of the given electrical circuit as depicted in §4.3.
  - **Generate Mathematical Model:** this block is comprised of two complementary blocks with the purpose of generating the mathematical model by applying the hybrid method presented in §4.4.2 and §4.5.3, although they have many similarities, they are customized for AC and DC working regimes.

- **Solve Mathematical Model:** this block actually consists of two related blocks; both are responsible for solving the systems of linear equations that represent the mathematical model. One block is in charge of DC analysis which is represented by a system of linear equations with real coefficients and the other is responsible of AC analysis i.e. solving a system of linear equations with complex coefficients.
- **Validate Results:** this block verifies the correctness of the results based on the well-known Power Theorem.

## 5.2 Presentation Layer

### 5.2.1 Draw Schematic

The focal point of the GUI is the gridded canvas which is extensively used when drawing and editing the schematic, along with its callback functions triggered by mouse events on it, like click, release and motion.

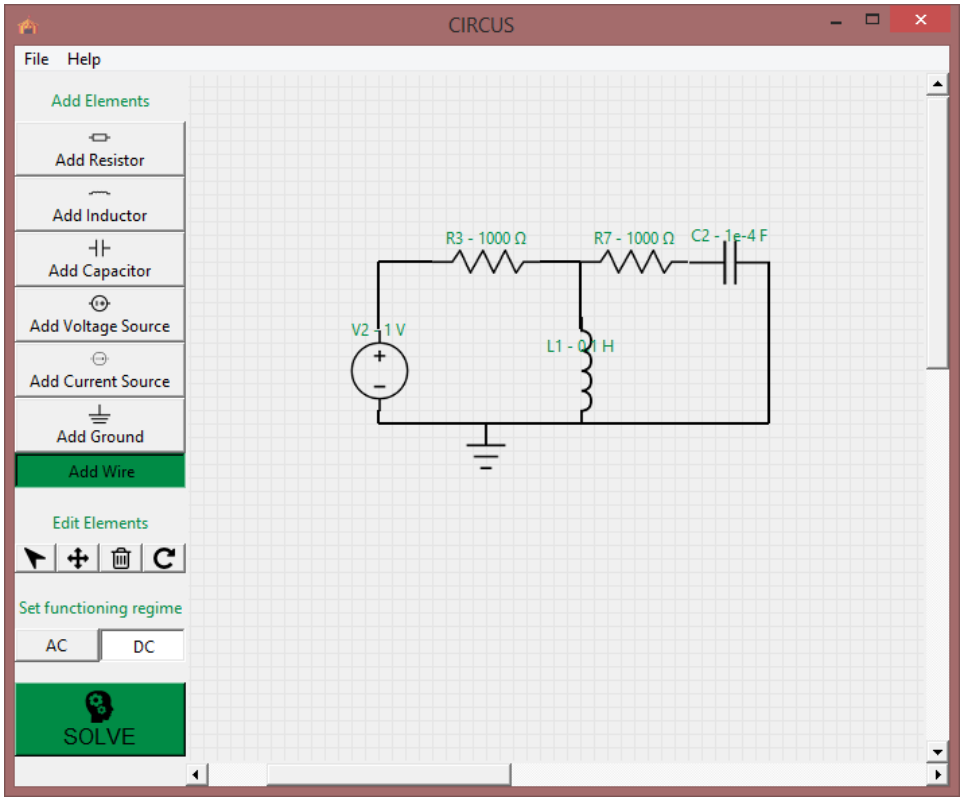



Fig 5.2 GUI presentation

Beside it comes a set of action buttons which are categorized according to their functionality:

- drawing buttons : Add Resistor, Add Inductor, Add Capacitor, Add Voltage Source, Add Current Source, Add Ground, Add Wire.
- editing buttons : cursor button for editing an element's parameters , move button, delete button, rotate clockwise button;
- resolve buttons : radio buttons for setting the working regime, for AC there are extra entry text fields for specifying the frequency or a range of frequencies and finally there is the solving button which "sets the gears of the application in motion".

The GUI keeps track of the user's actions by raising flags at the push of the buttons, monitoring the mouse events and treating them according to the flags that are raised at a given moment.

The TkInter Canvas comes with a very useful function i.e. find_withtag( ) that gets the handles for all items having a given tag; the widget also provides two predefined tags: ALL which matches all items on the canvas and CURRENT which matches the item under the mouse pointer, if any. Therefore, when editing the circuit elements, this function was used to determine which object was selected by the user and then perform the necessary modifications on it.

The piece of code below is extracted from the callback function that is triggered when clicking on the canvas after having pressed the delete button and depicts the way an element is removed from the circuit.

```
'''
        Delete element
'''
if self.deleteElement is True:
    if self.canvas.find_withtag(CURRENT) and 'image' in self.canvas.gettags(CURRENT):   # Delete Element
        e = self.getCurrentElement()
        self.canvas.delete(e.getId())          #delete image
        if e.type is not ElementsEnum.Ground:
            self.canvas.delete(e.getLabelId())     #delete label
        self.circuit.remove_element(e)
    else:                                                  # Delete Wire
        id = self.canvas.find_closest(x, y, halo=5)[0]
        if 'wire' in self.canvas.gettags(id):
            w = self.circuit.get_wire_by_id(id)
            self.circuit.remove_wire(w)
            self.canvas.delete(id)
```

Fig 5.3 Source code for deleting an element - extracted from gui.py

Another widget used is designing the graphical interface is TkInter Toplevel which is a pop-up window that behaves like any other window; it was used it for displaying the

results, announcing eventual errors and allowing the user to edit the parameters of the elements (as seen below).

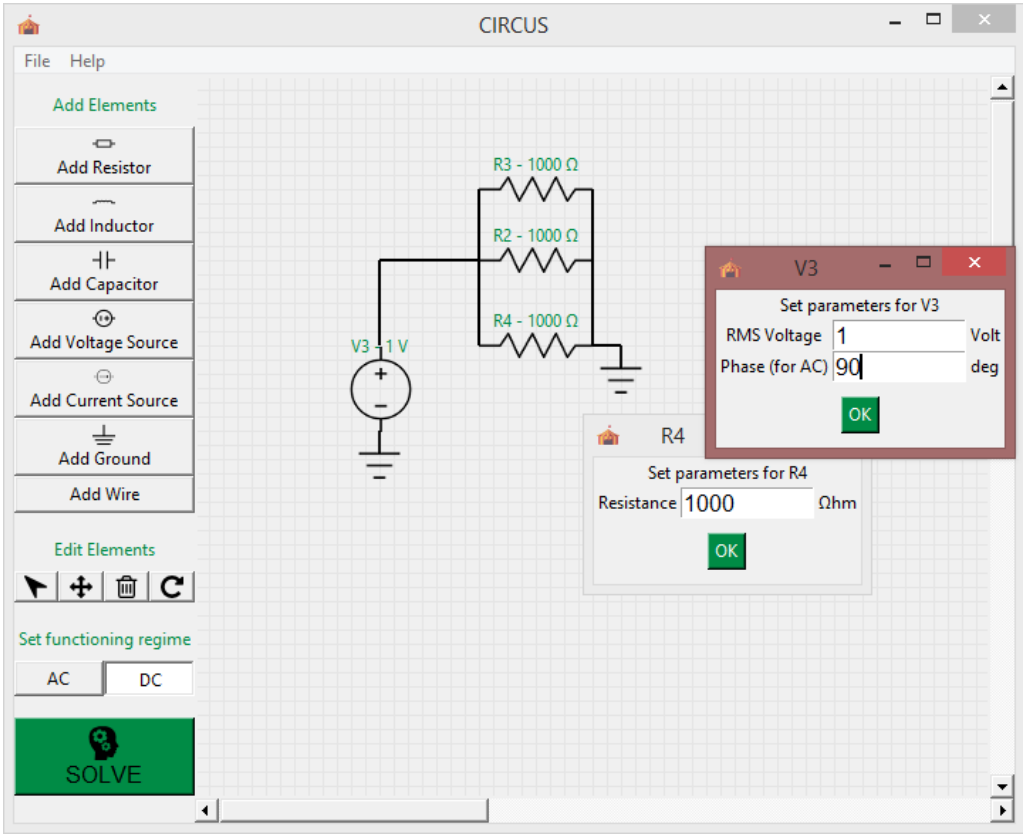



Fig 5.4 Editing parameter values in GUI

### 5.2.2 Load/Save Schema

Another purpose that the GUI serves is loading schematics from an external file and saving them for later uses; this actions are carried out by accessing the File tab from the app's menu that presents the user with the options to: start a new session, load a schematic from an external file, save the current schematic to an external file or save the drawing as an EPS picture.

The codification of the external files which are in charge with storing the information about the diagram of an electrical circuit is the following: for every element is reserved a line and the various information on a line is separated by a whitespace. The first token in a line is a character telling the type of the element (e.g. R for resistor, G for ground, etc), the next two tokens are the x and y canvas coordinates of the central placing point of the element, followed by an integer ranging from 0 to 3 which informs if the element is rotated (e.g. 1 means a clockwise rotation of 90 degrees), the next token is the value of its parameter and the

last one is the element's name. For encoding a wire segment the structure is a little different; the first token represents the type codification, then the next four token represent the x and y canvas coordinates of the starting point of the segment and the ending point, respectively.

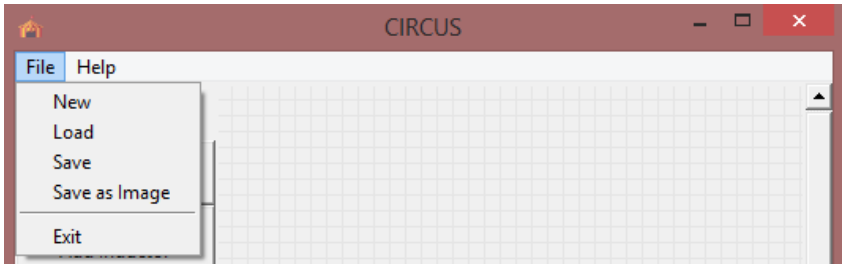



Fig 5.5 GUI menu presentation

### 5.2.3 View Results

After clicking the Solve button, the information on the canvas is being passed on the rest of the components of the system and the app responds to the user's command with the solution of the electrical circuit, i.e. current and voltage values. These results are displayed on a Toplevel window in a tabular form, along with the potential parameters of the simulation (frequencies in case of AC working regime).

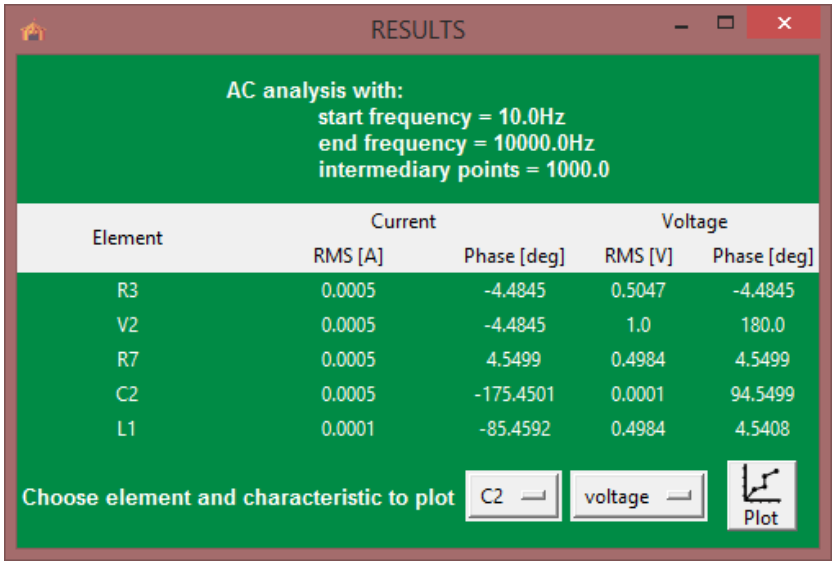



Fig 5.6 Result window

However, the system can throw some errors when trying to solve the circuit, these are usually caused by the faulty schematic so the user needs to be informed of these faults and their causes (if possible) in order to fix the problem and rerun the solving process.

CIRCUS informs the user about exceptions encountered in its attempt to obtain the results by showing a pop up window like one of the following:

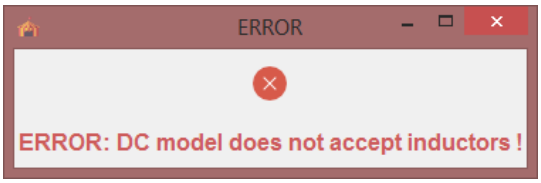



Fig 5.7 Error message example


Fig 5.8 Error message example

### 5.2.4 Plot Results

As seen in Fig 5.6 the user has the possibility to examine the graphical representations of the analysis results by choosing from two dropdown lists the element and its electrical measurement (current or voltage) that will be plotted. The user visualizes the behavior of the element in question when functioning in alternating current at the specified range of frequencies.

In Fig 5.10 is presented the source code for the callback function of the button “Plot” which uses the library matplotlib to produce MATLAB-like graphics with Python, which can also be integrated with TkInter widgets. The function plot(X,Y) creates a 2D plot of the data in Y versus the corresponding values in X; both X and Y are vectors of the same length. In our case, on x-axis there is the frequencies vector and on y-axis the vector with the value of the current or voltage at every frequency in the aforementioned vector. The current and voltages are being treated both in terms of absolute value and phase shift.

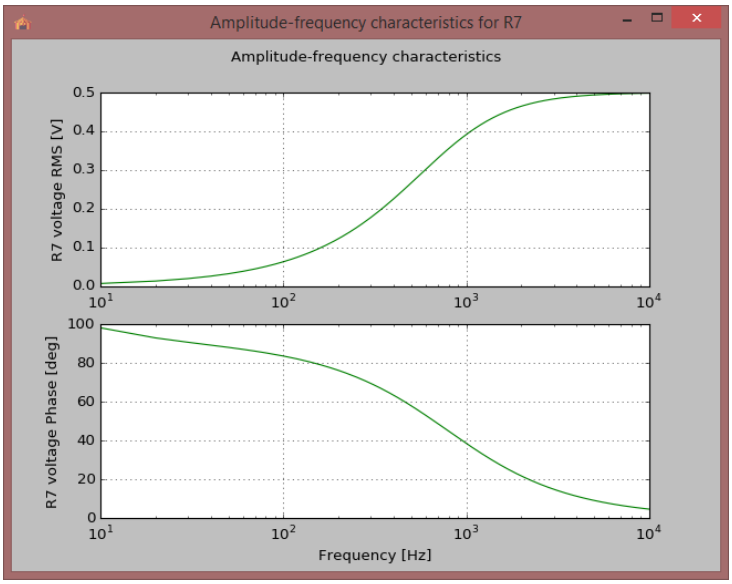



Fig 5.9 Plot example

```
def plot_CB(self):
    element_to_plot = self.circuit.get_element_by_name(self.plot_element_name.get())
    measurement_to_plot = self.plot_characteristic_name.get()

    plotWindow = Toplevel(bg=self.myColor)
    plotWindow.title("Amplitude-frequency characteristics for " + element_to_plot.name)
    plotWindow.iconbitmap(self.iconFile)

    x = self.circuit.freq_plot
    if measurement_to_plot == "current":
        y1 = self.circuit.I_ef_plot[element_to_plot.branch-1]
        y2 = self.circuit.I_faza_plot[element_to_plot.branch-1]
        unit = " [A]"
    elif measurement_to_plot == "voltage":
        y1 = self.circuit.U_ef_plot[element_to_plot.branch-1]
        y2 = self.circuit.U_faza_plot[element_to_plot.branch-1]
        unit = " [V]"

    f = Figure()
    f.suptitle('Amplitude-frequency characteristics')
    a = f.add_subplot(211)
    b = f.add_subplot(212)

    a.semilogx(x,y1,'g-')
    a.grid(True)
    a.set_ylabel(element_to_plot.name + " " + measurement_to_plot + " RMS" + unit)

    b.semilogx(x, y2, 'g-')
    b.grid(True)
    b.set_xlabel('Frequency [Hz]')
    b.set_ylabel(element_to_plot.name + " " + measurement_to_plot + " Phase [deg]")

    canvas = FigureCanvasTkAgg(f, master=plotWindow)
    canvas.get_tk_widget().pack()
```

Fig 5.10 Source code for plotting callback function - extracted from gui.py

## 5.3 Data Translation Layer

### 5.3.1 Interpret Schema

This application unit is in charge with interpreting the schematic drawn by the user into data that can be understood by the computational unit of the program i.e. unify the canvas objects in form of a circuit by establishing the relationships between its components.

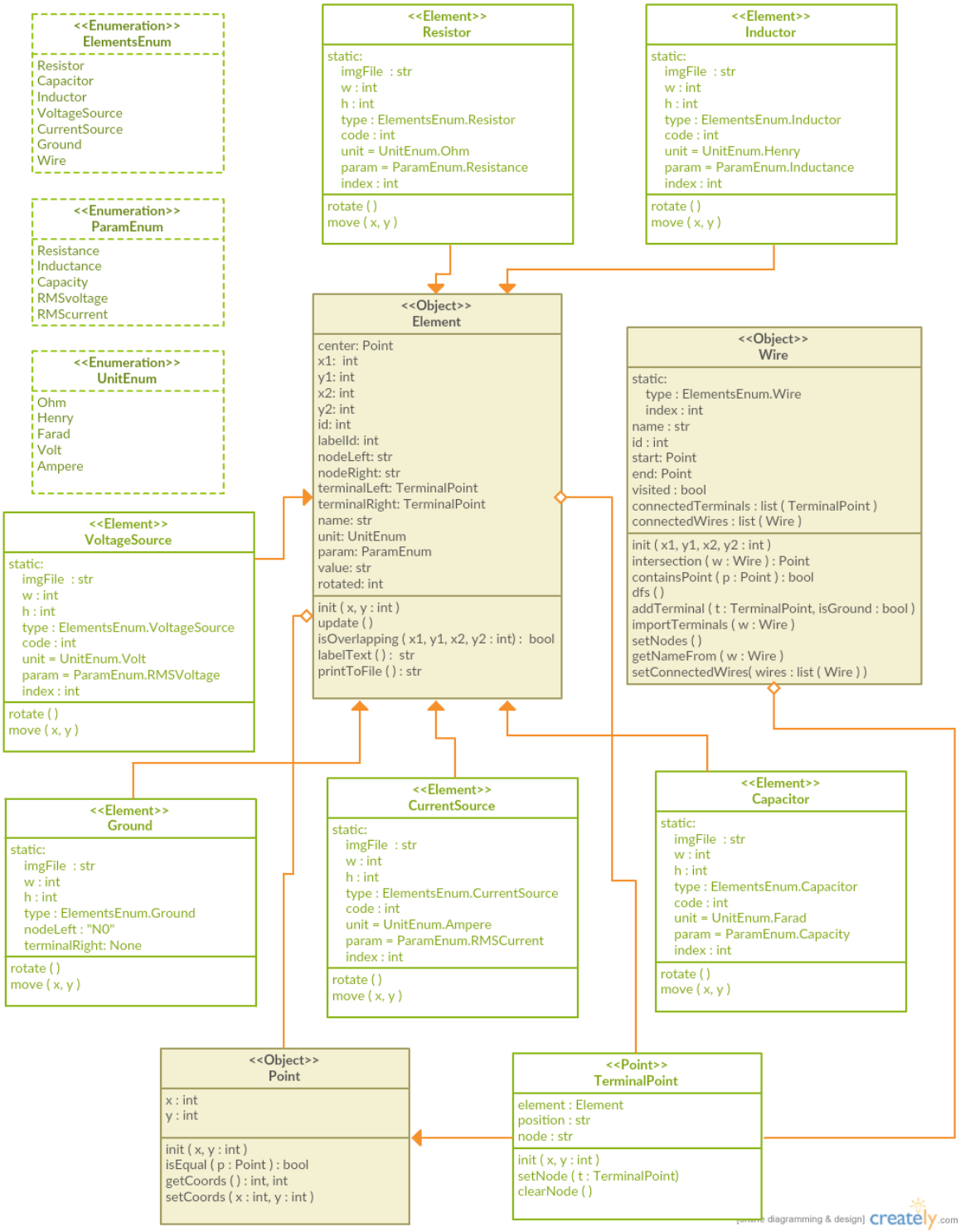



Fig 5.11 Class Diagram of elements.py module

An object oriented approach is employed to serve this purpose; in Fig 5.11 it is depicted the abstraction of the circuit elements using inheritance: the superclass Element holds the common attributes of electrical components which are then particularized in subclasses (Resistor, Inductor, Capacitor, VoltageSouce, CurrentSource and Ground). The wires are treated in an independent class because they don't share enough similarities with the other elements to be constructed through inheritance.

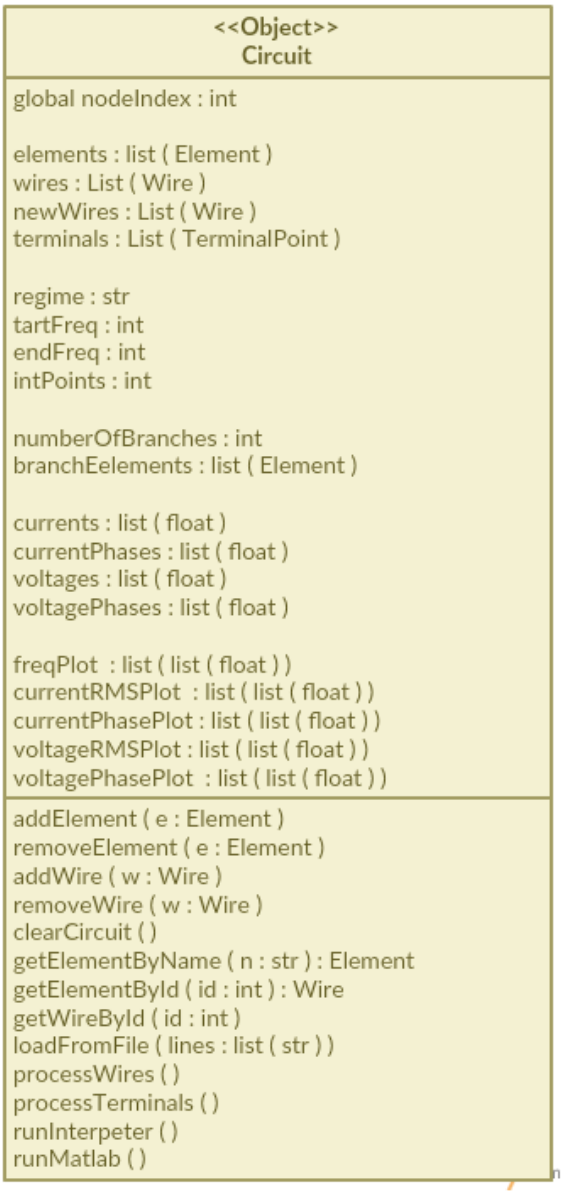



Fig 5.12 Circuit Class Diagram - circuit.py

Enumeration classes are made use of to keep track of the element's type, their parameter's names and their measurement units in a clean an organized manner; these classes are: ElementsEnum, ParamEnum and UnitEnum.

Another key data structure of the application is TerminalPoint; it is a subclass of the basic class Point defined by two parameters which represent the x and y coordinates of a point on the canvas. Each element has two terminal points and keeping track of overlapping terminals is a crucial step in the program.

The class that gathers together the aforementioned structures and represents the core of this block is the Circuit class, presented in Fig 5.12. Its attributes and methods can be split in two categories: one that takes care of the circuit's construction and keeps track of its components and another that is responsible with the circuit functioning (working regime, results).

An instance of the Circuit class is present in the GUI, so they communicate dynamically, when changes are being made on the canvas (add/remove/edit element) the circuit instance is updated. Information is being stored in lists of Element objects and Wire objects. After the Solve button is pressed the application processes the information stored in these lists.

One of the challenges is to determine the circuit nodes that serve in describing the circuit in terms of topology. The algorithm for finding the nodes has as input a list of elements and a list of wire segments, its steps are:

1. For every wire segment a list of directly connected (intersecting) wires is computed and stored.
2. For every wire segment a list of directly connected element terminals is computed and stored.
3. The groups of connecting wires are determined; the affiliation to a group is marked by sharing the same name between all connected wires. To achieve this, a Depth First Search - like approach is chosen. Wire segments represent the nodes of the undirected graph and their connections represent, intuitively, the vertexes. Having made these connections, it is straightforward that the connected components (from graph theory) need to be computed in order to achieve that the algorithm aims for i.e. find all groups of connected wire segments.
4. With the indirectly connected wires, all indirectly connected terminals can also be resolved for every wire segment.
5. As a result of these steps, there are duplicate wires, now only one instance of each wire is kept.

6. At this step we have a list of wires and for every wire a list on connected terminals; all connected terminals are assigned the same node, starting from the node N0 which signifies connection to ground.

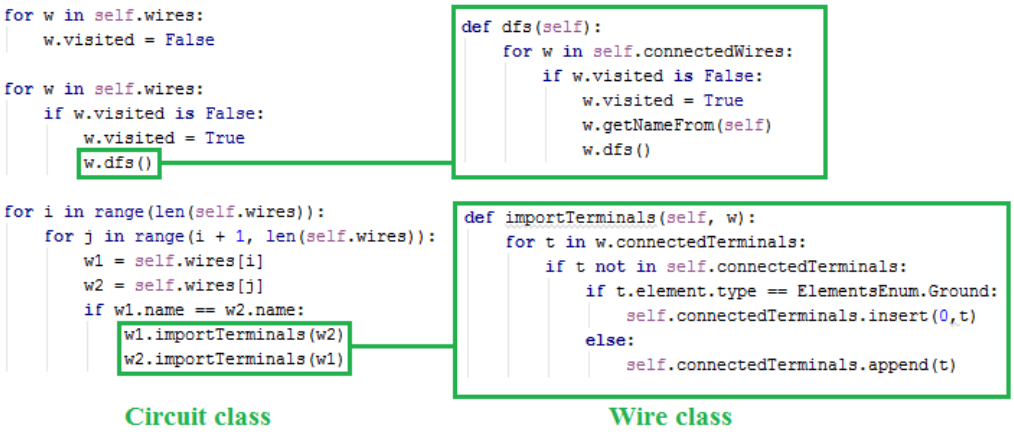



Fig 5.13 Source code for steps 3 and 4 of the finding nodes algorithm

Having computed the nodes of the circuit, it is effortlessly generated a circuit description matrix (as seen in Fig 5.14) with the structure presented in §4.3 and pass it as input to the logic unit.

```
branch = 0
input_matrix = []
for e in self.elements:
    if e.type is not ElementsEnum.Ground:
        if e.terminalLeft.node is None or e.terminalRight.node is None:
            raise Exception('\n\nERROR: Be aware of floating terminals!')
        branch += 1
        e.branch = branch
        N_initial = float(e.terminalLeft.node[1])
        N_final = float(e.terminalRight.node[1])
        param1 = float(e.value)
        if self.regime == "ac" and (e.type == ElementsEnum.VoltageSource
                                    or e.type == ElementsEnum.CurrentSource):
            param2 = float(e.phase)
        else:
            param2 = 0
        line = [e.code, branch, N_initial, N_final, param1, param2]
        input_matrix.append(line)
self.number_of_branches = branch

if self.regime == "ac":
    self.startFreq = float(self.startFreq)
    self.endFreq = float(self.endFreq)
    self.intPoints = float(self.intPoints)
    input_matrix.append([11, self.startFreq, self.endFreq, self.intPoints, 0, 0])
else:
    input_matrix.append([0, 0, 0, 0, 0, 0])      # no frequencies in DC

input_matrix_matlab = matlab.double(input_matrix)
```

Fig 5.14 Source code for generating the circuit description matrix

### 5.3.2 Interpret Results

The purpose of this block is to receive the results obtained from running the MATLAB scripts, results which are presented in form of vectors or matrices, and to map them to the corresponding circuit elements in order to be correctly displayed to the user.

```
if self.regime == "dc":                                                    # DC
    I_lat, U_lat, P_ced, P_cons = matlab_engine.Analysis_DC(input_matrix_matlab, nargout=4)
    self.currents = I_lat
    self.voltages = U_lat
elif self.regime == "ac":
    if self.endFreq == 0 and self.intPoints == 0:                          # AC with one frequency
        I_ef, I_faza, U_ef, U_faza = matlab_engine.Analysis_AC_1f(input_matrix_matlab, nargout=4)
        self.currents = I_ef[0]
        self.currents_phases = I_faza[0]
        self.voltages = U_ef[0]
        self.voltages_phases = U_faza[0]
    else:                                                                  # AC with range of frequencies
        I_ef, I_faza, U_ef, U_faza, freq_plot, I_ef_plot, I_faza_plot, U_ef_plot, U_faza_plot \
            = matlab_engine.Analysis_AC_3f(input_matrix_matlab, nargout=9)
        self.currents = I_ef[0]
        self.currents_phases = I_faza[0]
        self.voltages = U_ef[0]
        self.voltages_phases = U_faza[0]
        #info for plotting
        self.freq_plot = freq_plot
        self.I_ef_plot = I_ef_plot
        self.I_faza_plot = I_faza_plot
        self.U_ef_plot = U_ef_plot
        self.U_faza_plot = U_faza_plot
```

Fig 5.15 Source code for calling the MATLAB Engine

## 5.4 Logic Layer

### 5.4.1 Topology Analysis

This is conceivably the most important unit of the application, it takes as input the description matrix delivered by the data interpreter and generates a node-branch incidence matrix and its partitions, a normal tree and its corresponding cotree, an essential incidence matrix and its partitions, whose structure is described in Chapter 4.

For obtaining the above-mentioned matrices the next steps are followed:

1. Identify ideal circuit elements by investigating the first column of the description matrix; Types vector is generated that contains alphanumeric characters identifying the elements in ascending order by branch index.

```
branch =

1x5 struct array with fields:

    element
    cod
    nodin
    nodfin
    valoare
```

```
branch(3)
    element: 'C'
        cod: 2
      nodin: 4
     nodfin: 1
    valoare: 5.0000e-06
```

Fig 5.16 Branch data structure exemplification

2. Create a data structure array called Branch (Fig 5.16) that contains fields holding information about the topology and parameter value of the branch element (type, code, initial node, final node, value); the array size is equal to the number of branches in the circuit.

3. Build node-branch incidence matrix according to §4.3.2.
4. Construct a normal tree:
   - The purpose of this step is to investigate whether the circuit topology can lead to results (is valid); if this is not the case then errors are thrown.
   - Building the tree is based on an algorithm described in [1]
   - The principle on which is based the generation of a normal tree is identifying the first n-1 linearly independent columns of the node-branch incidence matrix. These columns correspond to the branches of the normal tree. The function used for this step is Reduced Row Echelon Form (rref) which is found in the MATLAB library.
   - Partition the node-branch incidence matrix into tree matrix and cotree matrix.
   - Build essential incidence matrix (tree branches - cotree branches incidence matrix) and its partitions, these are incidence matrices for: passive tree branches - passive cotree branches, passive tree branches - cotree branches with current sources, tree branches with voltage sources - passive cotree branches, tree branches with voltage sources - cotree branches with current sources.

### 5.4.2 Generate Mathematical Model

The equations system is built according to the hybrid method (§4.4.2, §4.5.3) as $\boldsymbol{M} \cdot \boldsymbol{X} = \boldsymbol{N} \cdot \boldsymbol{S}$ , where:

- $\boldsymbol{M}$ is the system matrix of size equal to the number of passive branches of the circuit, built by assembling the partitions returned by topology analysis;

```
M=[d21, Ya;...
   Zc, -d21'];

N=[-d22, zeros(length(lat_Za),length(lat_E));...
   zeros(length(lat_Zc),length(lat_J)), d11'];
```

Fig 5.17 Source code for generating the mathematical model

- $\boldsymbol{X}$ is the vector of unknowns, containing the currents of passive cotree branches and the voltages of the passive tree branches;
- $\boldsymbol{S}$ is the vector of the sources;
- $\boldsymbol{N}$ is a matrix built by assembling the partitions returned by topology analysis.

The matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ have real elements for DC analysis and complex elements for AC analysis.

### 5.4.3 Solve Mathematical Model

The equation system built before is solved with the help of backslash operator which involves a Gauss elimination method. Then we extract from the solution vector the currents of the passive cotree branches and the voltages of the pasive tree branches, respectively (Fig 5.18).

```
x=M\(N*[J;E]);
Ic=x(1:length(lat_Zc));
Ua=x(length(lat_Zc)+1:length(lat_Zc)+length(lat_Za));
```

Fig 5.18 Source code for solving mathematical model

### 5.4.4 Validate Results

This block ensures the correctness of the results by applying the Power Conservation Theorem; in order to validate the results, the power delivered by the sources is compared with the power consumed by the passive circuit elements [5].

```
% Bilant puteri
% Curentii, tensiunile surselor independente si puterea compexa cedata
sind=[lat_E,lat_J];
I_ind=I_lat(sind);
U_ind=U_lat(sind);
S_ced=sum(-U_ind.*conj(I_ind));
% Curentii, tensiunile laturilor pasive si puterea compexa consumata
I_pas=I_lat; I_pas(sind)=[];
U_pas=U_lat; U_pas(sind)=[];
S_cons=sum(U_pas.*conj(I_pas));
```

Fig 5.19 Source code for computing power balance for AC

# 6 TESTING

## 6.1 DC Running Example

A complete example is treated to prove the capabilities of CIRCUS for DC networks. The chosen circuit is of medium level of difficulty (fig. 6.1). It contains 12 branches, so it requires solving a 12-equation system using the classical method of Kirchhoff's laws. The goal is to compute the branch currents and voltages for the parameters specified below:

$R_2 = 15\Omega;\ R_3 = 10\Omega;\ R_4 = 5\Omega;\ R_5 = 20\ \Omega;\ R_6 = 10\ \Omega;$
$J_4 = 1\text{A};\ J_3 = 4\,\text{A};\ J_4 = 2\,\text{A};$
$V_3 = 30\,\text{V};\ V_4 = 20\,\text{V};\ V_5 = 20\,\text{V};\ V_6 = 50\,\text{V}.$

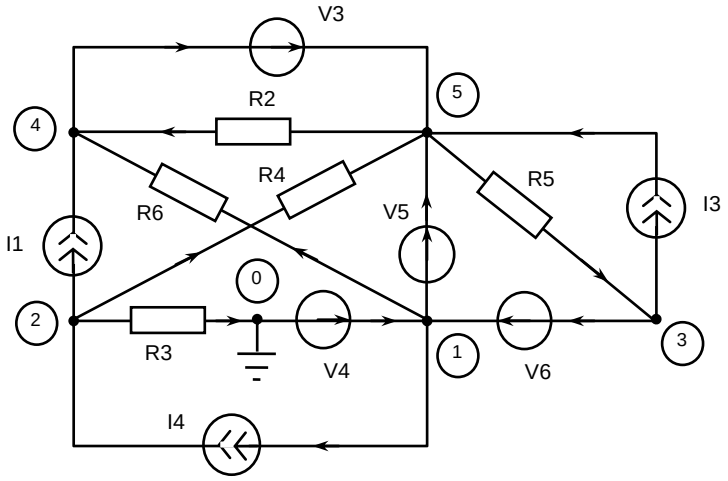



Fig. 6.1 Schematic of DC circuit example

I will firstly solve the problem through a CIRCUS analysis, and then the results will be compared with those given by a SPICE simulation [9]. The main stages of the CIRCUS simulation, as well as intermediary results will be pointed. The electric diagram drawn in CIRCUS main window is shown in fig. 6.2.

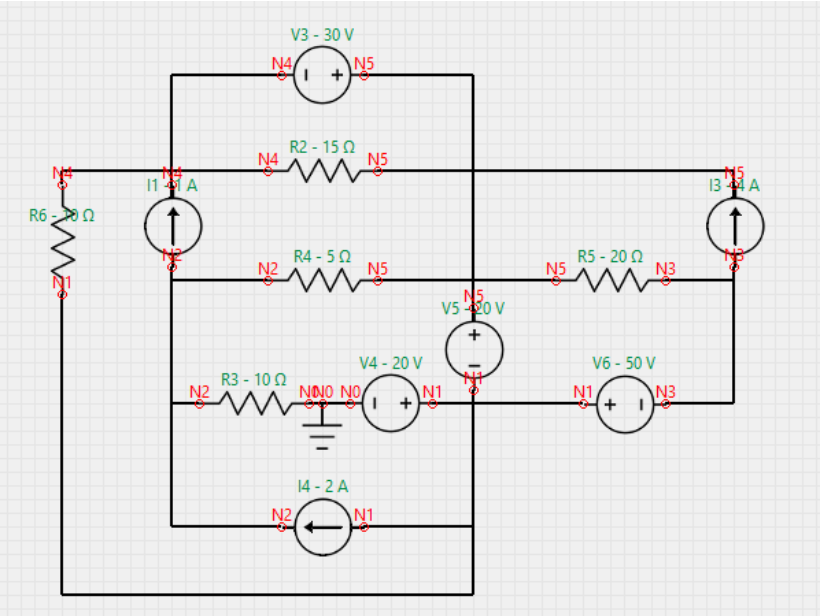



Fig. 6.2 Electric diagram built with CIRCUS GUI

Starting from the diagram, the matrix of graph description is automatically built as in Fig. 6.3. It has the structure as described in §4.3.2.

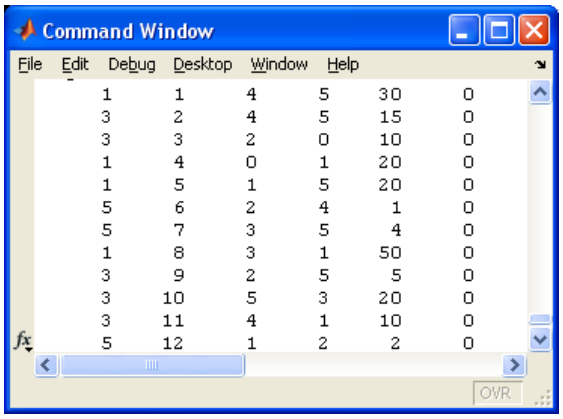



Fig. 6.3 Description matrix for the chosen example

The node-branch incidence matrix is then built as in fig. 6.4 (§4.3.2).

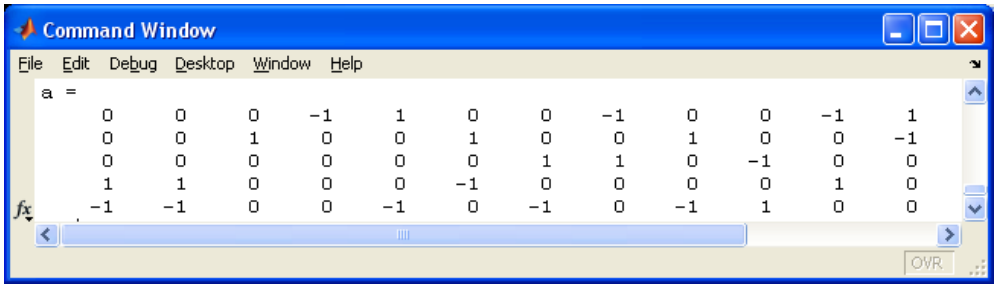



Fig. 6.4 Node-branch incidence matrix for the chosen example

This allows finding a normal tree and the corresponding cotree of the network [3] shown in fig. 6.5 nearby the corresponding partitions of the matrix.

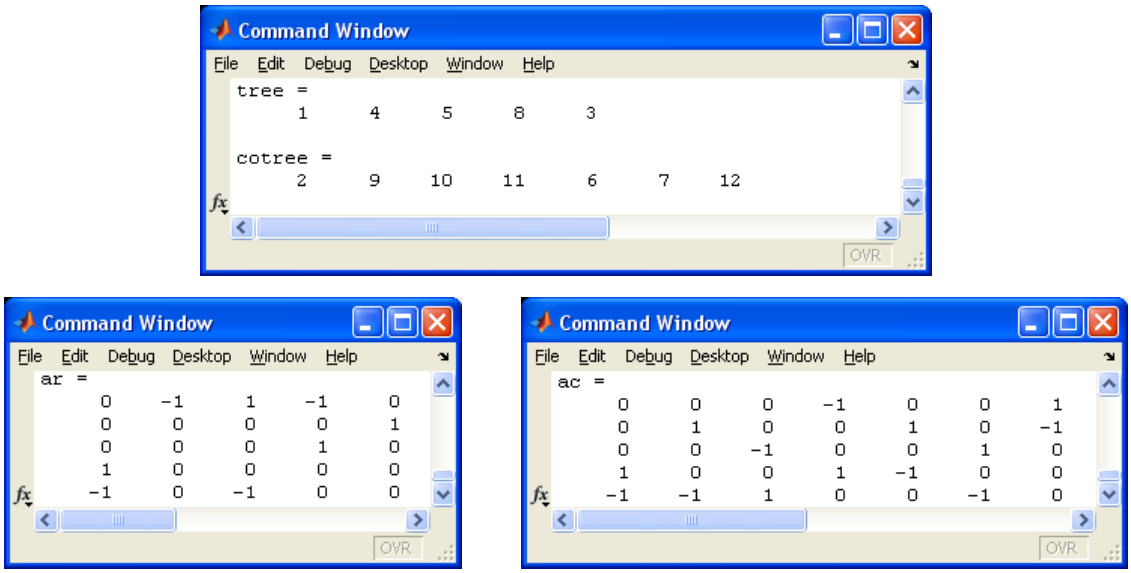



Fig. 6.5 Tree / cotree branches and the partitions of the node-branch incidence matrix

The branch-loop (tree-cotree) incidence matrix is then computed according to equation (4.13), as in fig. 6.6.

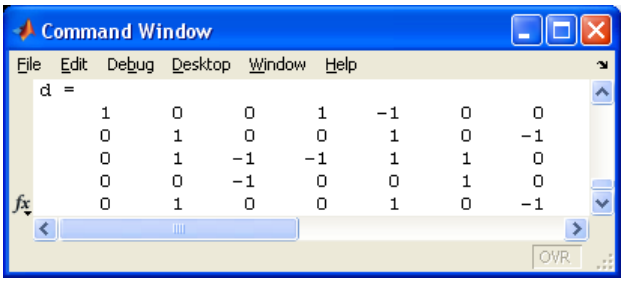



Fig. 6.6 Branch-loop (or tree-cotree) incidence matrix.

The partitions of the branch-loop incidence matrix required by the hybrid analysis method (§4.4.2) are shown in fig. 6.7.

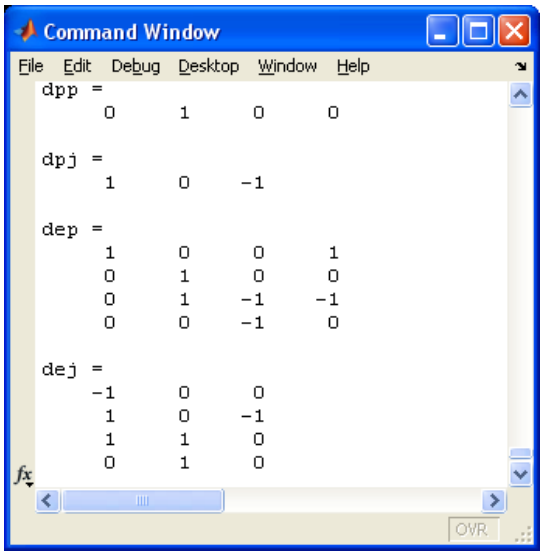



Fig. 6.7 Partitions of branch-loop incidence matrix required by the hybrid analysis method

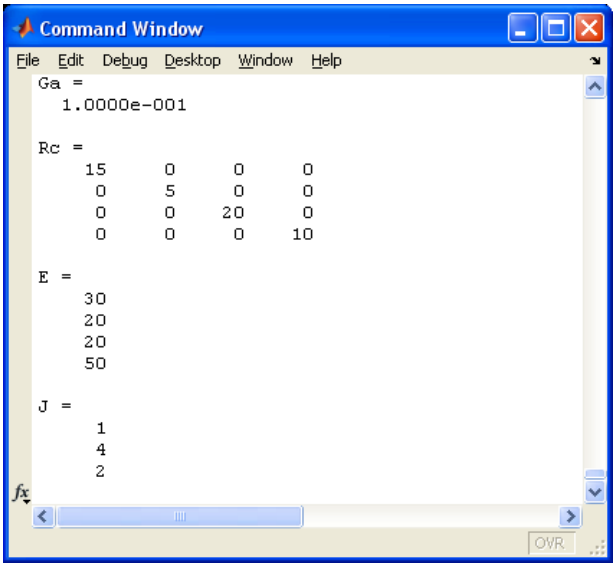



Fig. 6.8 Matrices of circuit parameters

The matrices of circuit parameters are built and shown in fig. 6.8: passive tree-branch conductance matrix (*Ga*), passive cotree-branch resistance matrix (*Rc*), vector of electromotive forces of voltage sources (*E*), and vector of currents of independent current sources (*J*).

The next step allows assembling the mathematical model of the hybrid method, as an equation system of size equal to the number of passive branches of the network (§4.4.2). Therefore, only 5 equations are necessary to solve this 12-branch network (fig. 6.9).

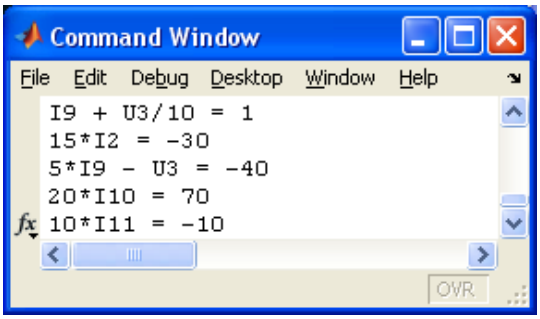



Fig. 6.9 Mathematical model of the hybrid method

The solution of this equation system are found through a Gauss elimination algorithm, then all other currents and voltages of the circuit are computed with simple explicit equations as described in §4.4.2. In order to validate the results, the power balance is performed (fig. 6.10) [5].

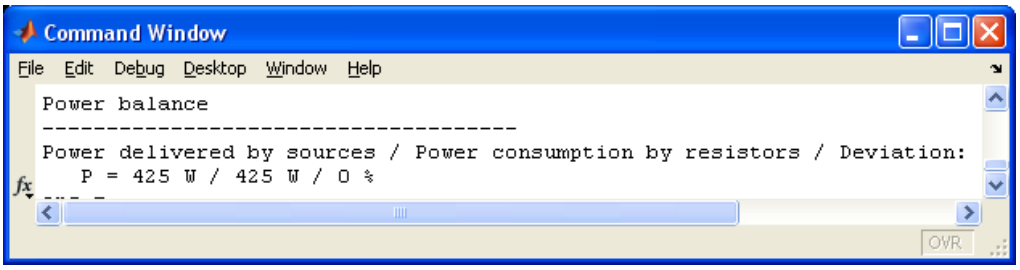



Fig. 6.10 Power balance to validate the simulation results

The final results are transferred to the user in a separate window which is placed on the screen nearby the circuit diagram to make the results interpretation easier (Fig. 6.11). It is noticeable that CIRCUS gives complete information regarding the simulation results: the currents and voltages of all circuit branches.

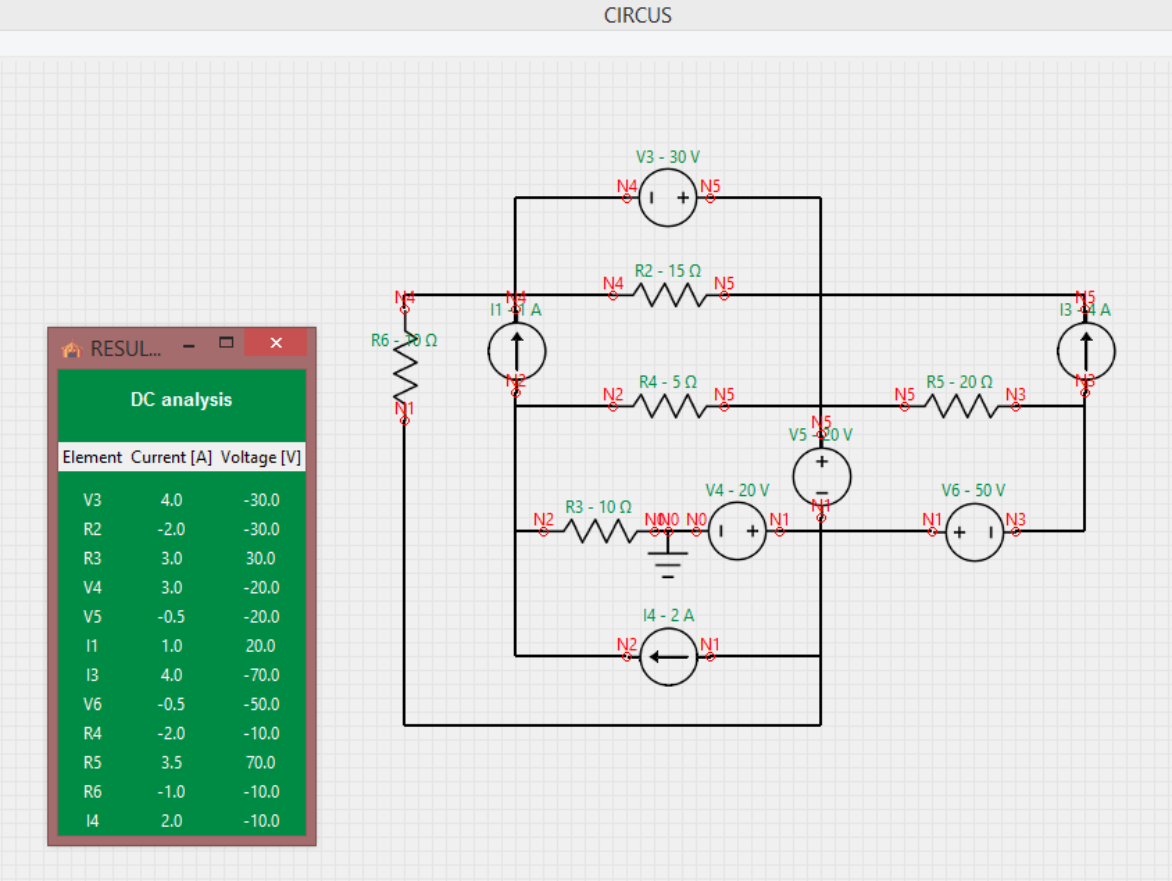



Fig. 6.11 CIRCUS window with simulation results

To prove the performance of CIRCUS, I completed a witness SPICE simulation. The SPICE input diagram is shown in fig. 6.12 and the result file in fig. 6.13.

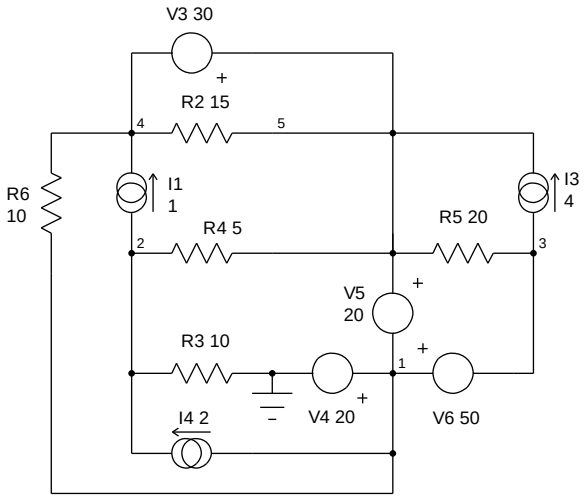



Fig. 6.12. SPICE input diagram

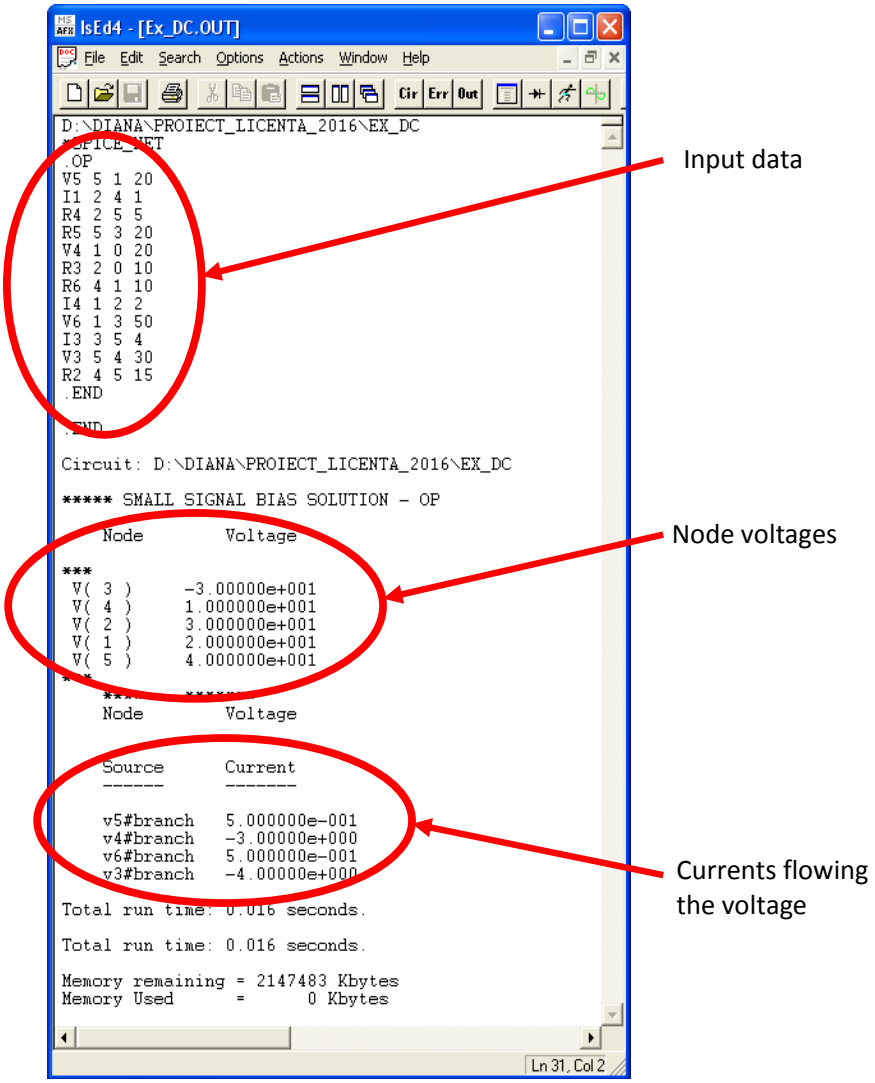



Fig. 6.13 SPICE output file

SPICE returns few results, but the correspondence with the results given by CIRCUS is obvious. SPICE displays only the currents of the voltage sources, assumed as positive values if they flow from the terminal – toward + inside the source. It also displays the node voltages with regard to the reference (ground), so that to compute the branch voltages one needs additional arithmetic operations. SPICE does not perform a power balance [9].

## 6.2 AC Running Example

A common RLC series circuit is treated as an AC example. A frequency analysis will be performed, and frequency characteristics for the output quantities will be built. The frequency domain, as well as the number of intermediary computation points are specified in the main window of CIRCUS (fig. 6.14). The circuit parameters and the frequency domain are viewed on the GUI.

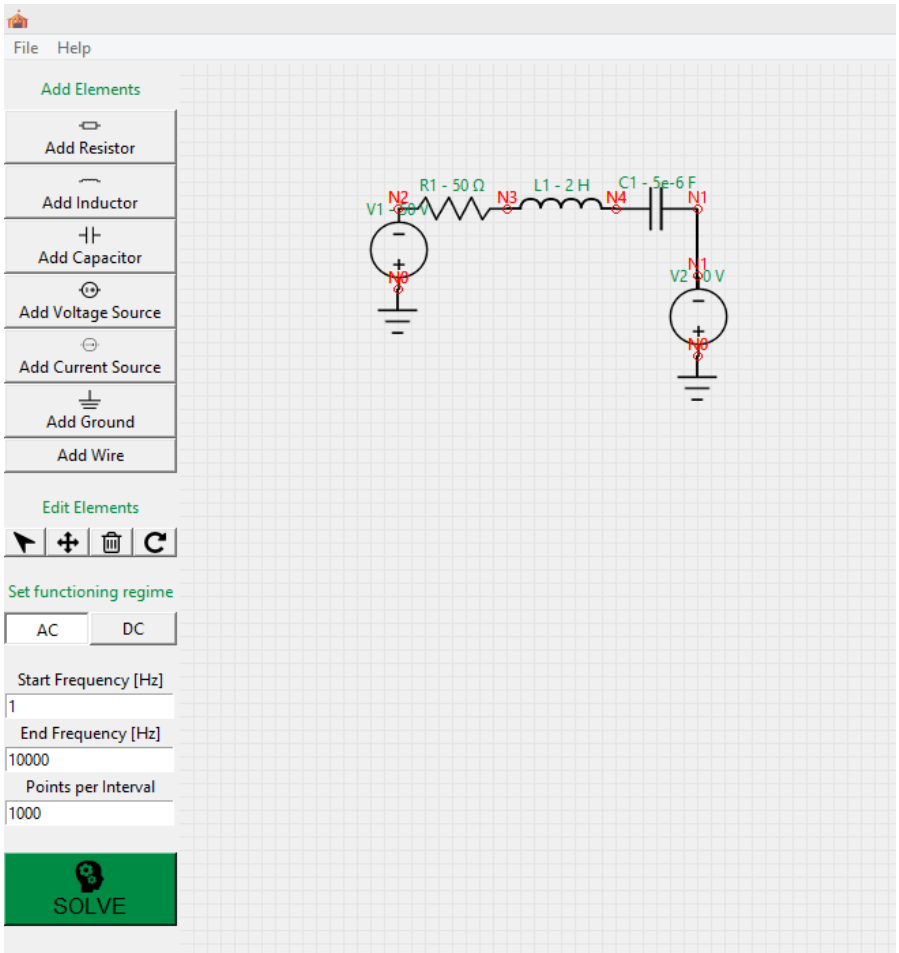



Fig. 6.14 Electric diagram of RLC network built with CIRCUS GUI

After running the program, the result window is displayed as in fig. 6.15.

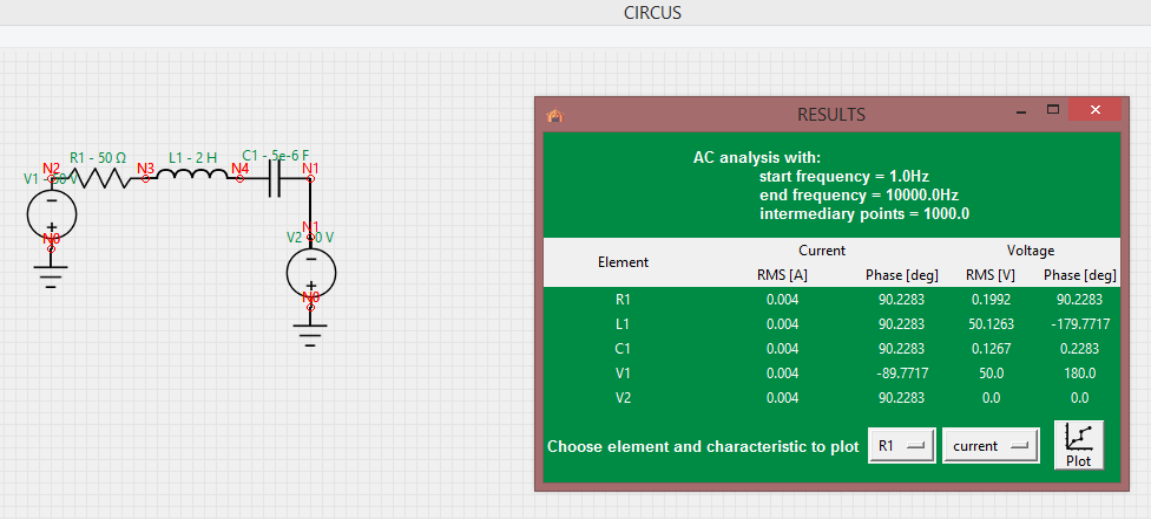



Fig. 6.15 CIRCUS result window for AC analysis

In order to display the frequency characteristics, of any branch current or voltage, the user choses the circuit element and the quantity of interest by clicking on the buttons from the bottom-right corner of the window. The desired option is chosen from dropdown lists. The current which flows through the resistor R1 was chosen and shown in fig. 6.16. Both amplitude-frequency and phase-frequency are displayed. It is obvious the phenomenon of electric resonance for a certain frequency, known from the theory [5].

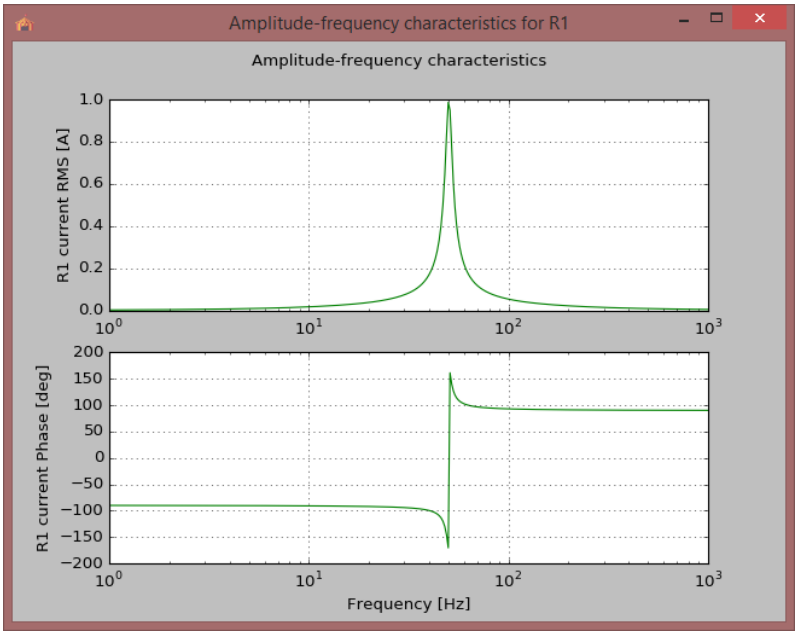



Fig. 6.16 Frequency characteristics generated by CIRCUS

I completed a witness SPICE simulation for the same circuit, with the SPICE input diagram shown in fig. 6.17. The frequency characteristics of the current are shown in fig. 6.18.

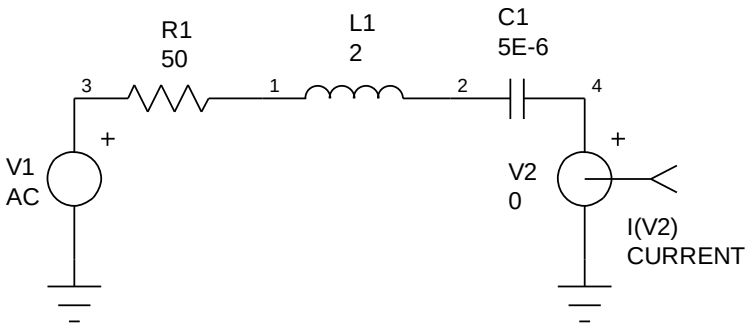



Fig. 6.17 SPICE input diagram of RLC circuit

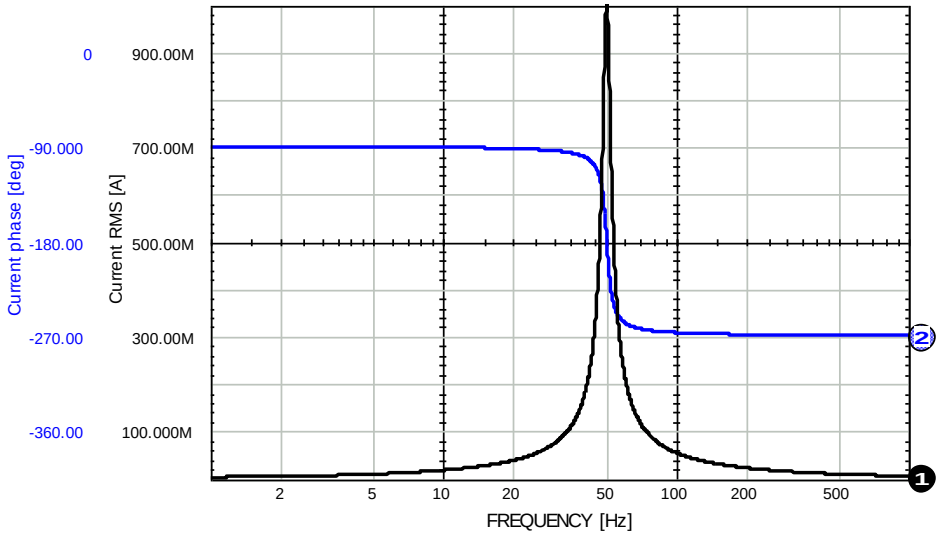



Fig. 6.18 Frequency characteristics of the current computed by a SPICE AC simulation

The similarity of the results returned by CIRCUS and SPICE is obvious. It is noticeable that CIRCUS translates the angular phases in the 1st and 4th quadrant of the complex plan.

# 7 CONCLUSION

## 7.1 Summary

In order to accomplish the purpose of the project, I began with an extensive study of the scientific literature. I intended to find the most appropriate software resources to obtain a reliable, robust and flexible software application. Then I identified the most convenient simulation algorithms with a high level of generality, which require a relatively reduced amount of computation effort while still providing accurate results. I decided to use Python and MATLAB due to their acclaimed popularity in the scientific community and their facilities suitable for my purpose.

The software application CIRCUS developed within the project accomplishes the requirements highlighted before. It is composed of computation modules organized in three layers. Since it is modular, it can be extended with newer computation facilities.

The application is accompanied by an advanced user interface to facilitate the interaction with less experienced and specialized users. For now, from the perspective of a common user, CIRCUS has the following capabilities:

- Allows specifying the input data of the circuit thanks to an advanced GUI;
- Performs topological analyses for linear analog circuits in DC behavior and AC behavior;
- Validates the analysis result using the power balance;
- Returns the simulation results in a user friendly manner, as text or/and graphics.

What is more, I used witness simulations using a well-known commercial simulation program to prove the accuracy of the results for some relevant examples, two of which are described in the paper.

## 7.2 Problems Encountered

The process of developing the project was fairly straightforward, the majority of software tools I used have detailed documentation and helpful communities, however I encountered some issues with the Pillow library (a fork of PIL) caused by malfunctions in its methods; they are introduced hereinafter.

Python Imaging Library (PIL) doesn't support converting an EPS file as PNG image on Windows; I found in the source for the Ghostscript() function in EpsImagePlugin.py that it says "Unix only". TkInter Canvas allows saving its contents as an EPS file and I unsuccessfully tried converting that to PNG using PIL

Another PIL bug is that when a PhotoImage object is garbage-collected by Python, namely when you return from a function which stored an image in a local variable, the image is cleared even if it's being displayed by a Tkinter widget. To avoid this, I kept an extra reference to the image object.

## 7.3 Future Development Direction

I plan to exploit the growth potential of this application as much as possible by adding new functionality that is also innovative in this field.

First and foremost, I have in view to develop the application entirely in Python and with the help of free software and also make it independent of the operating system, all in all I wish to make it as widely available as possible and to distribute the application freely for high-schools and other educational establishment whose students may benefit from it.

A significant ambition is to give the user the possibility to load an image of a hand-drawn electric diagram as input and the application to be able to solve it. This requires a complex image recognition module along with comprehensive machine learning research.

I plan to add support for electronic circuits which require controlled sources and nonlinear elements.

Another intention is to add more the computational modules for solving circuits in transient behavior; this is difficult because the mathematical models consist of nonlinear differential equations which require complicated algorithms.

# REFERENCES


[1] D.O. Pederson, *A Historical Review of Circuit Simulation*, IEEE Solid-State Circuits Magazine, Year: 2011, Volume: 3, Issue: 2, Pages: 43 – 54.

[2] L.O. Chua, C.A. Desoer, E.S. Kuh, *Linear and nonlinear circuits*, McGraw-Hill, Inc., New York, 1987.

[3] L.O. Chua, P.M. Lin, *Computer-aided analysis of electronic circuits – algorithms and computational techniques*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[4] D. Topan, L. Mandache, Roland Suesse, *Advanced Analysis of Electric Circuits*, Wissenschaftsverlag Thüringen, Langenwiesen, 2011.

[5] D. Topan, *Bazele electrotehnicii*, notițe de curs, lecture notes, Faculty of Automation, Computers and Electronics Craiova, academic year 2012-2013.

[6] M. Iordache, L. Dumitriu, *Simularea asistată de calculator a circuitelor analogice: algoritmi şi tehnici de calcul*, Ed. Politehnica Press, Bucureşti 2002.

[7] R. Militaru, Numerical methods, lecture notes, Faculty of Automation, Computers and Electronics Craiova, academic year 2012-2013.

[8] Matthew N. O. Sadiku, Ph.D., *Numerical Techniques in Electromagnetics – Second Edition*, CRC Press, Boca Raton, 2001.

[9] A. Vladimirescu, SPICE (in Romanian – translation from English), Ed. Tehnica, Bucharest, 1999.

[10] M. Ghinea, V. Firețeanu, *MATLAB – calcul numeric, grafică, aplicații*, Editura Teora, Bucureşti, 1997.

[11] The Mathworks Inc., *MATLAB 8.5 User's Guide*, 2015.

[12] Fredrik Lundh, *An Introduction to Tkinter*, 1999.

[13] Allen B. Downey, *Think Python Version 2.0.17*, Green Tea Press, Needham, Massachusetts, 2012.

[14] David Houcque, *Introduction to MATLAB for engineering students*, Northwestern University, Evanston, Illinois, 2005.

[15] John E. Grayson, *Python and Tkinter Programming*, Manning Publications, Greenwich, 2000.

# APPENDIX

## gui.py

```
from Tkinter import *
from PIL import Image, ImageTk
from tkFileDialog import *

import matplotlib
matplotlib.use("TkAgg")
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

from circuit import *


class GUI(Tk):
    def __init__(self,parent):
        Tk.__init__(self,parent)
        self.parent = parent
        self.resizable(width=True, height=True)

        self.myColor = 'SpringGreen4'
        self.defaultColor = self.cget("bg")

        self.canvas_width = 800
        self.canvas_height = 640
        self.canvas_width_total = 1600
        self.canvas_height_total = 1200

        self.gridSpacing = 10
        self.grid_vertical = []
        self.grid_horizontal = []

        self.iconFile = 'images/buttons/circus.ico'

        self.imagesPil = {ElementsEnum.Resistor:
Image.open(Resistor.imgFile),
                          ElementsEnum.Inductor: Image.open(Inductor.imgFile),
                          ElementsEnum.Capacitor:
Image.open(Capacitor.imgFile),
                          ElementsEnum.VoltageSource:
Image.open(VoltageSource.imgFile),
                          ElementsEnum.CurrentSource:
Image.open(CurrentSource.imgFile),
                          ElementsEnum.Ground: Image.open(Ground.imgFile)}

        self.imagesTk = {ElementsEnum.Resistor:
ImageTk.PhotoImage(self.imagesPil[ElementsEnum.Resistor]),
                         ElementsEnum.Inductor:
ImageTk.PhotoImage(self.imagesPil[ElementsEnum.Inductor]),
                         ElementsEnum.Capacitor:
ImageTk.PhotoImage(self.imagesPil[ElementsEnum.Capacitor]),
                         ElementsEnum.VoltageSource:
ImageTk.PhotoImage(self.imagesPil[ElementsEnum.VoltageSource]),
```

```
                        ElementsEnum.CurrentSource:
ImageTk.PhotoImage(self.imagesPil[ElementsEnum.CurrentSource]),
                        ElementsEnum.Ground:
ImageTk.PhotoImage(self.imagesPil[ElementsEnum.Ground])}

        self.buttonImages = {"move":
ImageTk.PhotoImage(Image.open("images/buttons/move.png")),
                             "rotate":
ImageTk.PhotoImage(Image.open("images/buttons/rotate.png")),
                             "delete":
ImageTk.PhotoImage(Image.open("images/buttons/delete.png")),
                             "run":
ImageTk.PhotoImage(Image.open("images/buttons/head.png")),
                             "cursor":
ImageTk.PhotoImage(Image.open("images/buttons/cursor.png")),
                             "resistor":
ImageTk.PhotoImage(Image.open("images/buttons/resistor.png")),
                             "inductor":
ImageTk.PhotoImage(Image.open("images/buttons/inductor.png")),
                             "capacitor":
ImageTk.PhotoImage(Image.open("images/buttons/capacitor.png")),
                             "voltage source":
ImageTk.PhotoImage(Image.open("images/buttons/voltage_src.png")),
                             "current source":
ImageTk.PhotoImage(Image.open("images/buttons/current_src.png")),
                             "ground":
ImageTk.PhotoImage(Image.open("images/buttons/ground.png")),
                             }

        '''
            Menu
        '''
        self.menubar=Menu(self)
        filemenu=Menu(self.menubar,tearoff=0)
        filemenu.add_command(label="New", command=self.new_CB)
        filemenu.add_command(label="Load", command=self.file_load_CB)
        filemenu.add_command(label="Save", command=self.file_save_CB)
        filemenu.add_command(label="Save as Image",
command=self.image_save_CB)
        filemenu.add_separator()
        filemenu.add_command(label="Exit", command=self.quit)
        self.menubar.add_cascade(label="File", menu=filemenu)

        helpmenu=Menu(self.menubar,tearoff=0)
        helpmenu.add_command(label="Help",command=self.cursor_CB)
        self.menubar.add_cascade(label="Help",menu=helpmenu)

        self.config(menu=self.menubar)

        self.mainFrame = Frame(self)

        self.frame1 = Frame(self.mainFrame)
        Label(self.frame1, text="Add Elements",
fg=self.myColor).pack(pady=5)
        self.b1 = Button(self.frame1, text='Add Resistor',
image=self.buttonImages["resistor"], compound="top", command=lambda:
self.selectElement_CB(ElementsEnum.Resistor))
        self.b1.pack(fill=X, expand=1)
        self.b2 = Button(self.frame1, text='Add Inductor',
image=self.buttonImages["inductor"], compound="top", command=lambda:
self.selectElement_CB(ElementsEnum.Inductor))
```

```
        self.b2.pack(fill=X, expand=1)
        self.b3 = Button(self.frame1, text='Add Capacitor',
image=self.buttonImages["capacitor"], compound="top",command=lambda:
self.selectElement_CB(ElementsEnum.Capacitor))
        self.b3.pack(fill=X, expand=1)
        self.b4 = Button(self.frame1, text='Add Voltage Source',
image=self.buttonImages["voltage source"], compound="top", command=lambda:
self.selectElement_CB(ElementsEnum.VoltageSource))
        self.b4.pack(fill=X, expand=1)
        self.b5 = Button(self.frame1, text='Add Current Source',
image=self.buttonImages["current source"], compound="top", command=lambda:
self.selectElement_CB(ElementsEnum.CurrentSource))
        self.b5.pack(fill=X, expand=1)
        self.b6 = Button(self.frame1, text='Add Ground',
image=self.buttonImages["ground"], compound="top", command=lambda:
self.selectElement_CB(ElementsEnum.Ground))
        self.b6.pack(fill=X, expand=1)
        self.b7 = Button(self.frame1, text='Add Wire', command=lambda:
self.selectElement_CB(ElementsEnum.Wire))
        self.b7.pack(fill=X, expand=1)
        self.elementsButtons = [self.b1, self.b2, self.b3, self.b4,
self.b5, self.b6, self.b7]
        self.frame1.pack(side=TOP, anchor=NW, fill=X, pady=5)


        self.frame2 = Frame(self.mainFrame)
        Label(self.frame2, text="Edit Elements",
fg=self.myColor).pack(pady=5)
        self.b8 = Button(self.frame2, text='Cursor',
image=self.buttonImages["cursor"], command=self.cursor_CB).pack(fill=X,
side=LEFT, expand=1)
        self.b9 = Button(self.frame2, text='Move',
image=self.buttonImages["move"], command=self.move_CB).pack(fill=X,
side=LEFT, expand=1)
        self.b10 = Button(self.frame2, text='Delete',
image=self.buttonImages["delete"], command=self.delete_CB).pack(fill=X,
side=LEFT, expand=1)
        self.b11 = Button(self.frame2, text='Rotate',
image=self.buttonImages["rotate"], command=self.rotate_CB).pack(fill=X,
side=LEFT, expand=1)
        self.frame2.pack(fill=X, pady=5)


        self.frame3 = Frame(self.mainFrame)
        Label(self.frame3, text="Set functioning regime",
fg=self.myColor).pack(pady=5)
        self.regime = StringVar()
        self.regime.set("dc")
        self.acRadioButton = Radiobutton(self.frame3, text="AC",
variable=self.regime, value="ac", indicatoron=0 ,
command=self.toggleRegime_CB).pack(side=LEFT, fill=X, expand=1)
        self.dcRadioButton = Radiobutton(self.frame3, text="DC",
variable=self.regime, value="dc", indicatoron=0,
command=self.toggleRegime_CB).pack(side=LEFT, fill=X, expand=1)
        self.frame3.pack(fill=X, pady=5)

        self.frame4 = Frame(self.mainFrame)
        self.startFreq = StringVar()
        self.startFreqLabel = Label(self.frame4, text="Start Frequency
[Hz]")
```

```
        self.startFreqEntry = Entry(self.frame4,
textvariable=self.startFreq)
        self.endFreq = StringVar()
        self.endFreqLabel = Label(self.frame4, text="End Frequency [Hz]")
        self.endFreqEntry = Entry(self.frame4, textvariable=self.endFreq)
        self.points = StringVar()
        self.pointsLabel = Label(self.frame4, text="Points per Interval")
        self.pointsEntry = Entry(self.frame4, textvariable=self.points)

        self.b12 = Button(self.mainFrame, text='SOLVE', font=('bold'),
bg=self.myColor, image=self.buttonImages["run"], compound="top",
command=self.run_CB).pack(fill=BOTH, pady=10, anchor=S, side=BOTTOM)

        self.mainFrame.pack(side=LEFT, anchor=N)

        self.canvas = Canvas(self, width=self.canvas_width,
height=self.canvas_height)

        self.xsb = Scrollbar(self, orient="horizontal",
command=self.canvas.xview)
        self.ysb = Scrollbar(self, orient="vertical",
command=self.canvas.yview)
        self.canvas.configure(yscrollcommand=self.ysb.set,
xscrollcommand=self.xsb.set)


self.canvas.configure(scrollregion=(0,0,self.canvas_width_total,self.canvas
_height_total))
        self.xsb.pack(fill=X, side=BOTTOM)
        self.ysb.pack(fill=Y, side=RIGHT)
        self.canvas.pack(side=RIGHT, expand=TRUE, fill=BOTH)

        self.drawGrid()
        self.setFlag()

        self.canvas.bind("<Motion>", self.canvasMotion)
        self.canvas.bind("<ButtonPress-1>", self.canvasClickDown)
        self.canvas.bind("<ButtonRelease-1>", self.canvasClickUp)
        self.canvas.bind("<ButtonPress-3>", self.canvasRightClickDown)
        self.bind('<Escape>', self.endWire_CB)

        '''
            Logic of Application related variables
        '''
        self.selectedElement = None
        self.moveElement = False
        self.deleteElement = False
        self.rotateElement = False

        self.circuit = Circuit()

        self.nodeIdList = []
        self.nodeLabelIdList = []

        self.click = "up"
        self.xold, self.yold = None, None
        self.start = None
        self.newItem = None
        self.oldItem = None

    def drawGrid(self):
```

```
        # vertical lines at an interval of "line_distance" pixel
        if self.grid_horizontal:
            self.deleteGrid()

        for x in range(self.gridSpacing, self.canvas_width_total ,
self.gridSpacing):
            line = self.canvas.create_line(x, 0, x,
self.canvas_height_total, fill="gray90", state=DISABLED)
            self.grid_vertical.append(line)
            self.canvas.tag_lower(line)
        # horizontal lines at an interval of "line_distance" pixel
        for y in range(self.gridSpacing,self.canvas_height_total
,self.gridSpacing):
            line = self.canvas.create_line(0, y, self.canvas_width_total,
y, fill="gray90", state=DISABLED)
            self.grid_horizontal.append(line)
            self.canvas.tag_lower(line)

    def deleteGrid(self):
        for x in self.grid_vertical:
            self.canvas.delete(x)
        self.grid_vertical = []
        for y in self.grid_horizontal:
            self.canvas.delete(y)
        self.grid_horizontal = []

    def new_CB(self):
        self.circuit.clear_circuit()
        self.canvas.delete("all")
        self.drawGrid()

    def file_save_CB(self):
        f = asksaveasfile(mode='w', defaultextension=".txt")

        if f is None: # asksaveasfile return `None` if dialog closed with
"cancel".
            return

        for e in self.circuit.elements:
            line = e.printToFile() + "\n"
            f.write(line)
        for w in self.circuit.wires:
            line = w.printToFile() + "\n"
            f.write(line)
        f.close()

    def file_load_CB(self):
        mask = \
        [("Text files","*.txt"),
        ("All files","*.*")]
        filename = askopenfilename(filetypes=mask)

        if filename is '':  # asksaveasfilename return `` if dialog closed
with "cancel".
            return

        self.circuit.clear_circuit()
        self.canvas.delete("all")
        self.drawGrid()

        f = open(filename)
```

```
        lines = f.read().splitlines()
        self.circuit.loadFromFile(lines)
        f.close()
        self.drawFromFile()

    def image_save_CB(self):
        self.canvas.postscript(file="tests/file_name.eps",
colormode='color')
        img = Image.open("file_name.eps")
        img.save("file_name.png", "png")        #works only on Unix

    def selectElement_CB(self, e):
        self.selectedElement = e
        if e == ElementsEnum.Resistor:
            for btn in self.elementsButtons:
                if btn is self.b1:
                    btn.config(relief="sunken", bg=self.myColor)
                else:
                    btn.config(relief="raised", bg=self.defaultColor)
        elif e == ElementsEnum.Inductor:
            for btn in self.elementsButtons:
                if btn is self.b2:
                    btn.config(relief="sunken", bg=self.myColor)
                else:
                    btn.config(relief="raised", bg=self.defaultColor)
        elif e == ElementsEnum.Inductor:
            for btn in self.elementsButtons:
                if btn is self.b2:
                    btn.config(relief="sunken", bg=self.myColor)
                else:
                    btn.config(relief="raised", bg=self.defaultColor)
        elif e == ElementsEnum.Capacitor:
            for btn in self.elementsButtons:
                if btn is self.b3:
                    btn.config(relief="sunken", bg=self.myColor)
                else:
                    btn.config(relief="raised", bg=self.defaultColor)
        elif e == ElementsEnum.VoltageSource:
            for btn in self.elementsButtons:
                if btn is self.b4:
                    btn.config(relief="sunken", bg=self.myColor)
                else:
                    btn.config(relief="raised", bg=self.defaultColor)
        elif e == ElementsEnum.CurrentSource:
            for btn in self.elementsButtons:
                if btn is self.b5:
                    btn.config(relief="sunken", bg=self.myColor)
                else:
                    btn.config(relief="raised", bg=self.defaultColor)
        elif e == ElementsEnum.Ground:
            for btn in self.elementsButtons:
                if btn is self.b6:
                    btn.config(relief="sunken", bg=self.myColor)
                else:
                    btn.config(relief="raised", bg=self.defaultColor)
        elif e == ElementsEnum.Wire:
            for btn in self.elementsButtons:
                if btn is self.b7:
                    btn.config(relief="sunken", bg=self.myColor)
                else:
                    btn.config(relief="raised", bg=self.defaultColor)
```

```
        self.setFlag("draw")
        self.config(cursor='arrow')

    def run_CB(self):
        self.circuit.regime = self.regime.get()
        if self.circuit.regime == 'ac':
            self.circuit.startFreq = self.startFreq.get()
            self.circuit.endFreq = self.endFreqEntry.get()
            self.circuit.intPoints = self.points.get()

        if self.nodeIdList or self.nodeLabelIdList:
            for i in self.nodeIdList:
                self.canvas.delete(i)
            for i in self.nodeLabelIdList:
                self.canvas.delete(i)

        self.circuit.run_interpreter()
        self.circuit.print_elements()

        for n in self.circuit.terminals:
            if n.node is not None:
                r = 3
                x = n.x
                y = n.y
                self.nodeIdList.append(self.canvas.create_oval(x-r, y-r,
x+r, y+r, outline='red'))
                self.nodeLabelIdList.append(self.canvas.create_text(x, y-
3*r, text=n.node, fill='red'))

        try:
            self.circuit.run_matlab()
        except Exception as e:
                top = Toplevel()
                top.title("ERROR")
                top.iconbitmap(self.iconFile)
                photo =
ImageTk.PhotoImage(Image.open("images/buttons/error.png"))
                Label(top, image=photo).pack(pady=10)

                msg = e.message
                lines = msg.split("\n")
                if len(lines) >= 2:
                    Label(top, text=lines[2], fg="indian red", font=(None,
12, 'bold')).pack(pady=5)
                else:
                    Label(top, text=lines, fg="indian red", font=(None, 12,
'bold')).pack(pady=5)
                raise

        top = Toplevel(bg=self.myColor)
        top.title("RESULTS")
        top.iconbitmap(self.iconFile)
        '''
            Show results of DC analysis
        '''
        if self.circuit.regime == "dc":
            rows = self.circuit.number_of_branches + 2  # cols = 5
            Label(top, bg=self.myColor, fg="white", justify=LEFT,
font=(None, 10, "bold"), text="DC analysis").grid(row=0, columnspan=5,
pady=10)
```

```
            Label(top, text="Element").grid(row=1, column=0, pady=10)
            Label(top, text="Current [A]").grid(row=1, column=1, pady=10)
            Label(top, text="Voltage [V]").grid(row=1, column=2, pady=10)


            for i in range(0,rows-2):
                r = i+2 #row number
                Label(top, bg=self.myColor, fg="white",
text=self.circuit.branch_elements[i].name).grid(row=r, column=0)
                Label(top, bg=self.myColor, fg="white",
text=self.circuit.currents[i][0]).grid(row=r, column=1)
                Label(top, bg=self.myColor, fg="white",
text=self.circuit.voltages[i][0]).grid(row=r, column=2)
        '''
            Show results of AC analysis
        '''
        if self.circuit.regime == "ac":
            rows = self.circuit.number_of_branches + 3  # cols = 5
            if self.circuit.endFreq == 0:
                Label(top, bg=self.myColor, fg="white", justify=LEFT,
font=(None, 10, "bold"), text="AC analysis at frequency = " +
str(self.circuit.startFreq) + " Hz").grid(row=0, columnspan=5, pady=10)
            else:
                Label(top, bg=self.myColor, fg="white", justify=LEFT,
font=(None, 10, "bold"),
                      text="AC analysis with: \n\tstart frequency = " +
str(self.circuit.startFreq) + "Hz\n\tend frequency = " +
str(self.circuit.endFreq)
                           + "Hz\n\tintermediary points = "+
str(self.circuit.intPoints)).grid(row=0, columnspan=5, pady=10)

            Label(top, text="Element").grid(row=1, rowspan=2, column=0,
sticky="wens")
            Label(top, text="Current").grid(row=1, column=1, columnspan=2,
sticky="wens")
            Label(top, text="RMS [A]").grid(row=2, column=1, sticky="wens")
            Label(top, text="Phase [deg]").grid(row=2, column=2,
sticky="wens")
            Label(top, text="Voltage").grid(row=1, column=3, columnspan=2,
sticky="wens")
            Label(top, text="RMS [V]").grid(row=2, column=3, sticky="wens")
            Label(top, text="Phase [deg]").grid(row=2, column=4,
sticky="wens")

            for i in range(0,rows-3):
                r = i+3 #row number
                Label(top, bg=self.myColor, fg="white",
text=self.circuit.branch_elements[i].name).grid(row=r, column=0)
                Label(top, bg=self.myColor, fg="white",
text=self.circuit.currents[i]).grid(row=r, column=1)
                Label(top, bg=self.myColor, fg="white",
text=self.circuit.currents_phases[i]).grid(row=r, column=2)
                Label(top, bg=self.myColor, fg="white",
text=self.circuit.voltages[i]).grid(row=r, column=3)
                Label(top, bg=self.myColor, fg="white",
text=self.circuit.voltages_phases[i]).grid(row=r, column=4)

            if self.circuit.endFreq != 0:
```

```
                Label(top, bg=self.myColor, fg="white", justify=LEFT,
font=(None, 10, "bold"), text="Choose element and characteristic to
plot").grid(row=rows+1, column=0, columnspan=2, pady=20)

                element_names = [e.name for e in self.circuit.elements if
e.type is not ElementsEnum.Ground]
                self.plot_element_name = StringVar()
                self.plot_element_name.set(element_names[0]) # default
value
                w1 = apply(OptionMenu, (top, self.plot_element_name) +
tuple(element_names))
                w1.grid(row=rows+1, column=2)

                self.plot_characteristic_name = StringVar()
                self.plot_characteristic_name.set("current") # default
value
                w2 = OptionMenu(top, self.plot_characteristic_name,
"current", "voltage")
                w2.grid(row=rows+1, column=3)

                photo =
ImageTk.PhotoImage(Image.open("images/buttons/diagram.png"))
                b = Button(top, text='Plot', image=photo, compound="top",
command=self.plot_CB)
                b.config(height=35, width=35)
                b.image = photo
                b.grid(row=rows+1, column=4)

    def plot_CB(self):
        element_to_plot =
self.circuit.get_element_by_name(self.plot_element_name.get())
        measurement_to_plot = self.plot_characteristic_name.get()

        plotWindow = Toplevel(bg=self.myColor)
        plotWindow.title("Amplitude-frequency characteristics for " +
element_to_plot.name)
        plotWindow.iconbitmap(self.iconFile)

        x = self.circuit.freq_plot
        if measurement_to_plot == "current":
            y1 = self.circuit.I_ef_plot[element_to_plot.branch-1]
            y2 = self.circuit.I_faza_plot[element_to_plot.branch-1]
            unit = " [A]"
        elif measurement_to_plot == "voltage":
            y1 = self.circuit.U_ef_plot[element_to_plot.branch-1]
            y2 = self.circuit.U_faza_plot[element_to_plot.branch-1]
            unit = " [V]"

        f = Figure()
        f.suptitle('Amplitude-frequency characteristics for '+
element_to_plot.name)
        a = f.add_subplot(211)
        b = f.add_subplot(212)

        a.semilogx(x,y1,'g-')
        a.grid(True)
        a.set_ylabel(element_to_plot.name + " " + measurement_to_plot + "
RMS" + unit)

        b.semilogx(x, y2, 'g-')
        b.grid(True)
```

```
        b.set_xlabel('Frequency [Hz]')
        b.set_ylabel(element_to_plot.name + " " + measurement_to_plot + "
Phase [deg]")

        canvas = FigureCanvasTkAgg(f, master=plotWindow)
        canvas.get_tk_widget().pack()

    def cursor_CB(self):
        self.setFlag("cursor")
        self.config(cursor='arrow')

    def move_CB(self):
        self.setFlag("move")
        self.config(cursor='fleur')

    def delete_CB(self):
        self.setFlag("delete")
        self.config(cursor='X_cursor')

    def rotate_CB(self):
        self.setFlag("rotate")
        self.config(cursor='exchange')

    def setValue_CB(self, window, e, value1, value2=None):
        e.setValue(value1)
        l = e.getLabelId()
        self.canvas.itemconfig(l, text=e.labelText())
        if value2 is not None:
            e.phase = value2
        window.destroy()

    def toggleRegime_CB(self):
        regime = self.regime.get()
        self.circuit.regime = regime
        if regime == "ac":          #enable entries
            self.startFreqLabel.pack()
            self.startFreqEntry.pack()
            self.startFreq.set("0")
            self.endFreqLabel.pack()
            self.endFreqEntry.pack()
            self.endFreq.set("0")
            self.pointsLabel.pack()
            self.pointsEntry.pack()
            self.points.set("0")
            self.frame4.pack(fill=X, pady=10)
        elif regime == "dc":        #disable entries
            self.startFreqLabel.pack_forget()
            self.startFreqEntry.pack_forget()
            self.endFreqLabel.pack_forget()
            self.endFreqEntry.pack_forget()
            self.pointsLabel.pack_forget()
            self.pointsEntry.pack_forget()
            self.frame4.pack_forget()

    def endWire_CB(self, event):   #esc key binding
        if self.selectedElement is ElementsEnum.Wire:
            if self.start is not None:
                self.start = None
        self.canvasClickUp(event)

    def setFlag(self,f=None):
```

```
        if f is "move":
            self.selectedElement = None
            self.moveElement = True
            self.deleteElement = False
            self.rotateElement = False
            self.modifyElement = False
        elif f is "delete":
            self.selectedElement = None
            self.moveElement = False
            self.deleteElement = True
            self.rotateElement = False
            self.modifyElement = False
        elif f is "rotate":
            self.selectedElement = None
            self.moveElement = False
            self.deleteElement = False
            self.rotateElement = True
            self.modifyElement = False
        elif f is "draw":
            self.moveElement = False
            self.deleteElement = False
            self.rotateElement = False
            self.modifyElement = False
        elif f is "cursor":
            self.selectedElement = None
            self.moveElement = False
            self.deleteElement = False
            self.rotateElement = False
            self.modifyElement = True
        elif f is None:
            self.selectedElement = None
            self.moveElement = False
            self.deleteElement = False
            self.rotateElement = False
            self.modifyElement = False

        if self.selectedElement == None:
            for btn in self.elementsButtons:
                btn.config(relief="raised", bg=self.defaultColor)

    def canvasClickDown(self,event):
        self.click = "down"
        x = self.canvas.canvasx(event.x, self.gridSpacing)
        y = self.canvas.canvasy(event.y, self.gridSpacing)

        '''
                Add circuit elements
        '''
        if self.selectedElement is ElementsEnum.Resistor:               #
ADD RESISTOR
            img = self.imagesTk[ElementsEnum.Resistor]
            if not self.overlaps(x,y,img):
                    r = Resistor(x, y)
                    r.setId(self.canvas.create_image(x, y, image=img,
tags='image'))
                    r.setLabelId(self.canvas.create_text(x, y-r.h,
text=r.labelText(), fill=self.myColor))
                    self.circuit.add_element(r)
        elif self.selectedElement is ElementsEnum.Inductor:               #
ADD INDUCTOR
            img = self.imagesTk[ElementsEnum.Inductor]
```

```
            if not self.overlaps(x,y,img):
                i = Inductor(x, y)
                i.setId(self.canvas.create_image(x, y, image=img,
tags='image'))
                i.setLabelId(self.canvas.create_text(x, y-i.h,
text=i.labelText(), fill=self.myColor))
                self.circuit.add_element(i)
        elif self.selectedElement is ElementsEnum.Capacitor:
# ADD CAPACITOR
            img = self.imagesTk[ElementsEnum.Capacitor]
            if not self.overlaps(x,y,img):
                c = Capacitor(x, y)
                c.setId(self.canvas.create_image(x, y, image=img,
tags='image'))
                c.setLabelId(self.canvas.create_text(x, y-c.h/1.8,
text=c.labelText(), fill=self.myColor))
                self.circuit.add_element(c)
        elif self.selectedElement is ElementsEnum.VoltageSource:
# ADD VOLTAGE SOURCE
            img = self.imagesTk[ElementsEnum.VoltageSource]
            if not self.overlaps(x,y,img):
                vs = VoltageSource(x, y)
                vs.setId(self.canvas.create_image(x, y, image=img,
tags='image'))
                vs.setLabelId(self.canvas.create_text(x, y-vs.h/2,
text=vs.labelText(), fill=self.myColor))
                self.circuit.add_element(vs)
        elif self.selectedElement is ElementsEnum.CurrentSource:
# ADD CURRENT SOURCE
            img = self.imagesTk[ElementsEnum.CurrentSource]
            if not self.overlaps(x,y,img):
                cs = CurrentSource(x, y)
                cs.setId(self.canvas.create_image(x, y, image=img,
tags='image'))
                cs.setLabelId(self.canvas.create_text(x, y-cs.h/2,
text=cs.labelText(), fill=self.myColor))
                self.circuit.add_element(cs)
        elif self.selectedElement is ElementsEnum.Ground:               #
ADD GROUND
            img = self.imagesTk[ElementsEnum.Ground]
            if not self.overlaps(x,y,img):
                g = Ground(x, y)
                g.setId(self.canvas.create_image(x, y, image=img,
tags='image'))
                self.circuit.add_element(g)
        elif self.selectedElement is ElementsEnum.Wire:
# ADD WIRE
            if self.start is None:
                self.start = (x, y)
            else:
                if self.start == (x, y):    #end wire when double clicked
                    self.start = None
                else:
                    x0 = self.start[0]
                    y0 = self.start[1]
                    dx = abs(x - x0)            # delta x
                    dy = abs(y - y0)            # delta y
                    if dx >= dy:
                        id = self.canvas.create_line(x0, y0, x, y0,
fill="black", width=2.0, tag='wire')
                        w = Wire(x0, y0, x, y0, id)
```

```
                        self.circuit.add_wire(w)
                        self.start = (x, y0)
                    else:
                        id = self.canvas.create_line(x0, y0, x0, y,
fill="black", width=2.0, tag='wire')
                        w = Wire(x0, y0, x0, y, id)
                        self.circuit.add_wire(w)
                        self.start = (x0, y)


        self.canvas.update()

        '''
                Delete element
        '''
        if self.deleteElement is True:
            if self.canvas.find_withtag(CURRENT) and 'image' in
self.canvas.gettags(CURRENT):   # Delete Element
                e = self.getCurrentElement()
                self.canvas.delete(e.getId())         #delete image
                if e.type is not ElementsEnum.Ground:
                    self.canvas.delete(e.getLabelId())    #delete label
                self.circuit.remove_element(e)
            else:                                                       #
Delete Wire
                id = self.canvas.find_closest(x, y, halo=5)[0]
                if 'wire' in self.canvas.gettags(id):
                    w = self.circuit.get_wire_by_id(id)
                    self.circuit.remove_wire(w)
                    self.canvas.delete(id)

        '''
                Rotate element
        '''
        if self.rotateElement is True:
            if self.canvas.find_withtag(CURRENT) and 'image' in
self.canvas.gettags(CURRENT):
                itemId = self.getCurrentElementID()
                e = self.getCurrentElement()
                self.canvas.delete(itemId)
                angle = e.rotate()
                e.im = ImageTk.PhotoImage(self.imagesPil[e.type].rotate(-
90*angle, expand=True))
                inst = self.canvas.create_image(e.center.x, e.center.y,
image=e.im, tags='image')
                e.setId(inst)
                self.canvas.update()

        '''
                Modify Element Attributes
        '''
        if self.modifyElement is True:
            if self.canvas.find_withtag(CURRENT) and 'image' in
self.canvas.gettags(CURRENT):
                e = self.getCurrentElement()
                if e.type is not ElementsEnum.Ground:
                    top = Toplevel()
                    top.title(e.name)
                    top.iconbitmap(self.iconFile)
                    Label(top, text="Set parameters for " +
e.name).grid(row=0, columnspan=3)
```

```
                    var1 = StringVar(top, value=e.value)
                    Label(top, text=e.param_name).grid(row=1, column=0)
                    Entry(top, width=10, font=('Arial', 12,),
textvariable=var1).grid(row=1, column=1)
                    Label(top, text=e.unit).grid(row=1, column=2)
                    if e.type == ElementsEnum.CurrentSource or e.type ==
ElementsEnum.VoltageSource:
                        var2 = StringVar(top, value=e.phase)
                        Label(top, text="Phase (for AC)").grid(row=2,
column=0)
                        Entry(top, width=10, font=('Arial', 12,),
textvariable=var2).grid(row=2, column=1)
                        Label(top, text="deg").grid(row=2, column=2)
                        Button(top, bg=self.myColor, fg="white", text="OK",
                               command=lambda: self.setValue_CB(top, e,
var1.get(), var2.get())).grid(row=3, columnspan=3, pady=10)
                    else:
                        Button(top, bg=self.myColor, fg="white", text="OK",
                               command=lambda: self.setValue_CB(top, e,
var1.get())).grid(row=2, columnspan=3, pady=10)
                    self.canvas.update()

    def canvasClickUp(self,event):
        self.click = "up"
        self.xold = None           # reset the line when you let go of the
button
        self.yold = None
        if self.oldItem is not None:
            self.canvas.delete(self.oldItem)
        if self.newItem:
            self.canvas.delete(self.newItem)
        self.newItem = None
        self.oldItem = None
        self.canvas.update()

    def canvasMotion(self,event):
        '''

            DRAG

        '''
        if self.click == "down":
            if self.xold is None and self.yold is None:
                x2 = self.canvas.canvasx(event.x, self.gridSpacing)
                y2 = self.canvas.canvasx(event.y, self.gridSpacing)
                x1 = x2
                y1 = y2
            else:
                x1 = self.canvas.canvasx(self.xold, self.gridSpacing)
#old positions of cursor
                y1 = self.canvas.canvasx(self.yold, self.gridSpacing)
                x2 = self.canvas.canvasx(event.x, self.gridSpacing)
#new positions of cursor
                y2 = self.canvas.canvasx(event.y, self.gridSpacing)

            '''
                    Move element
            '''
            if self.moveElement is True:
                if self.canvas.find_withtag(CURRENT) and 'image' in
self.canvas.gettags(CURRENT):
```

```
                    newX = x2 - x1      #deplasament
                    newY = y2 - y1
                    e = self.getCurrentElement()
                    self.canvas.move(e.getId(), newX, newY)
                    if e.type is not ElementsEnum.Ground:
                        self.canvas.move(e.getLabelId(), newX, newY)
                    xy = self.canvas.coords(CURRENT)
                    e.move(xy[0], xy[1])
                    self.canvas.update()
        '''

            MOTION

        '''
        if self.click == "up":
            x = self.canvas.canvasx(event.x, self.gridSpacing)       #new
positions of cursor
            y = self.canvas.canvasy(event.y, self.gridSpacing)
            if self.selectedElement is ElementsEnum.Resistor:
# RESISTOR
                img = self.imagesTk[ElementsEnum.Resistor]
                if not self.overlaps(x,y,img):
                    self.oldItem = self.newItem
                    self.newItem = self.canvas.create_image(x, y,
image=img, tags='image')
                    self.canvas.delete(self.oldItem)
            elif self.selectedElement is ElementsEnum.Inductor:
# INDUCTOR
                img = self.imagesTk[ElementsEnum.Inductor]
                if not self.overlaps(x,y,img):
                    self.oldItem = self.newItem
                    self.newItem = self.canvas.create_image(x, y,
image=img, tags='image')
                    self.canvas.delete(self.oldItem)
            elif self.selectedElement is ElementsEnum.Capacitor:
#  CAPACITOR
                img = self.imagesTk[ElementsEnum.Capacitor]
                if not self.overlaps(x,y,img):
                    self.oldItem = self.newItem
                    self.newItem = self.canvas.create_image(x, y,
image=img, tags='image')
                    self.canvas.delete(self.oldItem)
            elif self.selectedElement is ElementsEnum.VoltageSource:
#  VOLTAGE SOURCE
                img = self.imagesTk[ElementsEnum.VoltageSource]
                if not self.overlaps(x,y,img):
                    self.oldItem = self.newItem
                    self.newItem = self.canvas.create_image(x, y,
image=img, tags='image')
                    self.canvas.delete(self.oldItem)
            elif self.selectedElement is ElementsEnum.CurrentSource:
#  CURRENT SOURCE
                img = self.imagesTk[ElementsEnum.CurrentSource]
                if not self.overlaps(x,y,img):
                    self.oldItem = self.newItem
                    self.newItem = self.canvas.create_image(x, y,
image=img, tags='image')
                    self.canvas.delete(self.oldItem)
            elif self.selectedElement is ElementsEnum.Ground:
#  GROUND
                img = self.imagesTk[ElementsEnum.Ground]
```

```
                if not self.overlaps(x,y,img):
                    self.oldItem = self.newItem
                    self.newItem = self.canvas.create_image(x, y,
image=img, tags='image')
                    self.canvas.delete(self.oldItem)
            elif self.selectedElement is ElementsEnum.Wire:
#  WIRE
                self.oldItem = self.newItem
                if self.start is not None:
                    x0 = self.start[0]
                    y0 = self.start[1]
                    dx = abs(x - x0)            # delta x
                    dy = abs(y - y0)            # delta y
                    if dx >= dy:
                        self.newItem = self.canvas.create_line(x0, y0, x,
y0, fill="black", width=2.0)
                    else:
                        self.newItem = self.canvas.create_line(x0, y0, x0,
y, fill="black", width=2.0)
                    self.canvas.delete(self.oldItem)

        self.canvas.update()
        self.xold = event.x
        self.yold = event.y

    def canvasRightClickDown(self, event):
        pass

    def getCurrentElementID(self):
        itemId = self.canvas.find_withtag(CURRENT)[0]
        return itemId

    def getCurrentElement(self):
        itemId = self.getCurrentElementID()
        return self.circuit.get_element_by_id(itemId)

    def getCurrentWire(self):
        itemId = self.getCurrentElementID()
        return self.circuit.get_wire_by_id(itemId)

    def overlaps(self, x, y, img):
        h = img.height()
        w = img.width()

        x1 = x-w/2
        y1 = y-h/2
        x2 = x+w/2
        y2 = y+h/2

        canvas_w = self.canvas_width_total
        canvas_h = self.canvas_height_total
        if x1 <= 1 or y1 <= 1 or x2 >= canvas_w or y2 >= canvas_h:
#image is inside canvas
            self.canvas.delete(self.newItem)
            return True

        for e in self.circuit.elements:
            if e.isOverlapping(x1, y1, x2, y2):
                self.canvas.delete(self.oldItem)
                return True
```

```
        return False

    def drawFromFile(self):
        for e in self.circuit.elements:
            img = self.imagesTk[e.type]
            if e.rotated == 0:
                e.setId(self.canvas.create_image(e.center.x, e.center.y,
image=img, tags='image'))
            else:
                angle = e.rotated
                e.im = ImageTk.PhotoImage(self.imagesPil[e.type].rotate(-
90*angle, expand=True))
                e.setId(self.canvas.create_image(e.center.x, e.center.y,
image=e.im, tags='image'))
                self.canvas.update()
            if e.type is not ElementsEnum.Ground:
                if e.type is ElementsEnum.Resistor or e.type is
ElementsEnum.Inductor:
                    e.setLabelId(self.canvas.create_text(e.center.x,
e.center.y-e.h, text=e.labelText(), fill=self.myColor))
                else:
                    e.setLabelId(self.canvas.create_text(e.center.x,
e.center.y-e.h/2, text=e.labelText(), fill=self.myColor))
        for w in self.circuit.wires:
            w.id = self.canvas.create_line(w.start.x, w.start.y, w.end.x,
w.end.y, fill="black", width=2.0, tag='wire')


if __name__ == "__main__":
    app = GUI(None)
    app.title('CIRCUS')
    app.iconbitmap(app.iconFile)
    app.mainloop()
```

## elements.py

```
import global_var


class ElementsEnum:
    Resistor, Inductor, Capacitor, VoltageSource, CurrentSource, Ground,
Wire = range(7)


class UnitEnum:
    Ohm = u"\u03A9hm" #Ohm
    Henry = "Henry"
    Farad = "Farad"
    Volt = "Volt"
    Ampere = "Ampere"


class ParamEnum:
    Resistance = "Resistance"
    Inductance = "Inductance"
    Capacity = "Capacity"
    RMSvoltage = "RMS Voltage"
    RMScurrent = "RMS Current"
```

```
class Element(object):

    def __init__(self, x, y):
        self.center = Point(x,y)

        self.rotated = 0    # 0 = 0deg, 1 = 90deg, 2 = 180deg, 3 =  270deg
clockwise
        self.im = None      # attribute used as workaround to save rotated
image, avoid garbage collector bug in PhotoImage
        self.nodeLeft = None
        self.nodeRight = None
        self.value = '0'
        self.branch = 0

    def setValue(self, v):
        self.value = v

    def setId(self, i):
        self.id = i

    def getId(self):
        return self.id

    def setLabelId(self, i):
        self.labelId = i

    def getLabelId(self):
        return self.labelId

    def labelText(self):
        return self.name + " - " + self.value + " " + self.unit[0]

    def isOverlapping(self, x1, y1, x2, y2):
        if self.x1 < x2 and self.x2 > x1 and self.y1 < y2 and self.y2 > y1:
            return True
        else:
            return False

    def update(self):
        if self.rotated is 0 or self.rotated is 2:
            self.x1 = self.center.x - self.w / 2
            self.y1 = self.center.y - self.h / 2
            self.x2 = self.center.x + self.w / 2
            self.y2 = self.center.y + self.h / 2
        elif self.rotated is 1 or self.rotated is 3:
            self.x1 = self.center.x - self.h / 2
            self.y1 = self.center.y - self.w / 2
            self.x2 = self.center.x + self.h / 2
            self.y2 = self.center.y + self.w / 2
        if self.type is not ElementsEnum.Ground:
            if self.rotated is 0:
                self.terminalLeft = TerminalPoint(self.center.x - self.w /
2, self.center.y, self, 'left')
                self.terminalRight = TerminalPoint(self.center.x + self.w /
2, self.center.y, self, 'right')
            elif self.rotated is 1:
                self.terminalLeft = TerminalPoint(self.center.x,
self.center.y - self.w / 2, self, 'left')
                self.terminalRight = TerminalPoint(self.center.x,
self.center.y + self.w / 2, self, 'right')
```

```
            elif self.rotated is 2:
                self.terminalLeft = TerminalPoint(self.center.x + self.w /
2, self.center.y, self, 'left')
                self.terminalRight = TerminalPoint(self.center.x - self.w /
2, self.center.y, self, 'right')
            elif self.rotated is 3:
                self.terminalLeft = TerminalPoint(self.center.x,
self.center.y + self.w / 2, self, 'left')
                self.terminalRight = TerminalPoint(self.center.x,
self.center.y - self.w / 2, self, 'right')
        else:
            if self.rotated is 0:
                self.terminalLeft = TerminalPoint(self.center.x,
self.center.y - self.h / 2, self, 'left')
            elif self.rotated is 1:
                self.terminalLeft = TerminalPoint(self.center.x + self.h /
2, self.center.y, self, 'left')
            elif self.rotated is 2:
                self.terminalLeft = TerminalPoint(self.center.x,
self.center.y + self.h / 2, self, 'left')
            elif self.rotated is 3:
                self.terminalLeft = TerminalPoint(self.center.x - self.h /
2, self.center.y, self, 'left')

    def printElement(self):
        print self.name, ": between ", self.terminalLeft, " and ",
self.terminalRight

    def printToFile(self):
        if self.type is ElementsEnum.Resistor: id = "R"
        elif self.type is ElementsEnum.Inductor: id = "L"
        elif self.type is ElementsEnum.Capacitor: id = "C"
        elif self.type is ElementsEnum.VoltageSource: id = "V"
        elif self.type is ElementsEnum.CurrentSource: id = "I"
        elif self.type is ElementsEnum.Ground: id = "G"
        return id + " " + str(self.center.x) + " " + str(self.center.y) + "
" + str(self.rotated) + " " + str(self.value) + " " + self.name


class Resistor(Element):

    imgFile = "images/resistor.png"
    w = 80
    h = 18

    type = ElementsEnum.Resistor

    code = 3

    unit = UnitEnum.Ohm

    param_name = ParamEnum.Resistance

    index = 0

    def __init__(self, x, y):
        super(Resistor, self).__init__(x,y)

        self.terminalLeft = TerminalPoint(x - self.w / 2, y, self, 'left')
        self.terminalRight = TerminalPoint(x + self.w / 2, y, self,
'right')
```

```
        Resistor.index += 1
        self.name = "R" + str(Resistor.index)
        # edges
        self.x1 = self.center.x - self.w / 2
        self.y1 = self.center.y - self.h / 2
        self.x2 = self.center.x + self.w / 2
        self.y2 = self.center.y + self.h / 2
        self.value = "1000"

    def rotate(self):
        self.rotated = (self.rotated + 1) % 4
        self.update()
        return self.rotated

    def move(self, x, y):
        self.center = Point(x,y)
        self.update()


class Inductor(Element):

    imgFile = "images/inductor.png"
    w = 80
    h = 18

    type = ElementsEnum.Inductor

    code = 4

    unit = UnitEnum.Henry

    param_name = ParamEnum.Inductance

    index = 0

    def __init__(self, x, y):
        super(Inductor, self).__init__(x,y)

        self.terminalLeft = TerminalPoint(x - self.w / 2, y, self, 'left')
        self.terminalRight = TerminalPoint(x + self.w / 2, y, self,
'right')

        Inductor.index += 1
        self.name = "L" + str(Inductor.index)
        # edges
        self.x1 = self.center.x - self.w / 2
        self.y1 = self.center.y - self.h / 2
        self.x2 = self.center.x + self.w / 2
        self.y2 = self.center.y + self.h / 2
        self.value = "0.1"

    def rotate(self):
        self.rotated = (self.rotated + 1) % 4
        self.update()
        return self.rotated

    def move(self, x, y):
        self.center = Point(x, y)
        self.update()
```

```
class Capacitor(Element):

    imgFile = "images/capacitor.png"
    w = 60
    h = 40

    type = ElementsEnum.Capacitor

    code = 2

    unit = UnitEnum.Farad

    param_name = ParamEnum.Capacity

    index = 0

    def __init__(self, x, y):
        super(Capacitor, self).__init__(x,y)

        self.terminalLeft = TerminalPoint(x - self.w / 2, y, self, 'left')
        self.terminalRight = TerminalPoint(x + self.w / 2, y, self,
'right')

        Capacitor.index += 1
        self.name = "C" + str(Capacitor.index)

        # edges
        self.x1 = self.center.x - self.w / 2
        self.y1 = self.center.y - self.h / 2
        self.x2 = self.center.x + self.w / 2
        self.y2 = self.center.y + self.h / 2
        self.value = "1e-6"

    def rotate(self):
        self.rotated = (self.rotated + 1) % 4
        self.update()
        return self.rotated

    def move(self, x, y):
        self.center = Point(x, y)
        self.update()


class VoltageSource(Element):

    imgFile = "images/voltage_source.png"
    w = 60
    h = 60

    type = ElementsEnum.VoltageSource

    code = 1

    unit = UnitEnum.Volt

    param_name = ParamEnum.RMSvoltage

    index = 0

    def __init__(self, x, y):
```

```
        super(VoltageSource, self).__init__(x,y)

        self.terminalLeft = TerminalPoint(x - self.w / 2, y, self, 'left')
        self.terminalRight = TerminalPoint(x + self.w / 2, y, self,
'right')

        VoltageSource.index += 1
        self.name = "V" + str(VoltageSource.index)
        # edges
        self.x1 = self.center.x - self.w / 2
        self.y1 = self.center.y - self.h / 2
        self.x2 = self.center.x + self.w / 2
        self.y2 = self.center.y + self.h / 2
        self.value = "1"
        self.phase = '0'

    def rotate(self):
        self.rotated = (self.rotated + 1) % 4
        self.update()
        return self.rotated

    def move(self, x, y):
        self.center = Point(x,y)
        self.update()


class CurrentSource(Element):

    imgFile = "images/current_source.png"
    w = 60
    h = 60

    type = ElementsEnum.CurrentSource

    code = 5

    unit = UnitEnum.Ampere

    param_name = ParamEnum.RMScurrent

    index = 0

    def __init__(self, x, y):
        super(CurrentSource, self).__init__(x,y)

        self.terminalLeft = TerminalPoint(x - self.w / 2, y, self, 'left')
        self.terminalRight = TerminalPoint(x + self.w / 2, y, self,
'right')

        CurrentSource.index += 1
        self.name = "I" + str(CurrentSource.index)
        # edges
        self.x1 = self.center.x - self.w / 2
        self.y1 = self.center.y - self.h / 2
        self.x2 = self.center.x + self.w / 2
        self.y2 = self.center.y + self.h / 2
        self.value = "0"
        self.phase = '0'

    def rotate(self):
        self.rotated = (self.rotated + 1) % 4
```

```
        self.update()
        return self.rotated

    def move(self, x, y):
        self.center = Point(x,y)
        self.update()


class Ground(Element):

    imgFile = "images/ground.png"
    w = 40
    h = 40

    type = ElementsEnum.Ground

    unit = " "

    nodeLeft = "N0"

    def __init__(self, x, y):
        super(Ground, self).__init__(x,y)
        self.terminalLeft = TerminalPoint(x, y - self.h / 2, self, 'left')
        self.terminalLeft.node = "N0"
        self.terminalRight = None
        self.name = "GND"
        # edges
        self.x1 = self.center.x - self.w / 2
        self.y1 = self.center.y - self.h / 2
        self.x2 = self.center.x + self.w / 2
        self.y2 = self.center.y + self.h / 2

    def rotate(self):
        self.rotated = (self.rotated + 1) % 4
        self.update()
        return self.rotated

    def move(self, x, y):
        self.center = Point(x,y)
        self.update()


class Wire(object):

    type = ElementsEnum.Wire

    index = 0

    def __init__(self, x_start, y_start, x_end, y_end, id=0):
        self.start = Point(x_start, y_start)
        self.end = Point(x_end, y_end)
        self.id = id
        Wire.index += 1
        self.name = "W" + str(Wire.index)
        self.connectedTerminals = []
        self.connectedWires = []
        self.visited = False

    def intersection(self, w):
        left = max(min(self.start.x, self.end.x), min(w.start.x, w.end.x))
        right = min(max(self.start.x, self.end.x), max(w.start.x, w.end.x))
```

```
        top = max(min(self.start.y, self.end.y), min(w.start.y, w.end.y))
        bottom = min(max(self.start.y, self.end.y), max(w.start.y,
w.end.y))

        if top > bottom or left > right:
           return False                    #('NO INTERSECTION',list())
        if (top,left) == (bottom,right):
           return Point(left,top)          #('POINT
INTERSECTION',list((left,top)))
        return Point(left, bottom)         #('SEGMENT INTERSECTION',
list((left,bottom,right,top)))

    def dfs(self):
        for w in self.connectedWires:
            if w.visited is False:
                w.visited = True
                w.getNameFrom(self)
                w.dfs()

    def setConnectedWires(self, wireList):
        for w in wireList:
            if self.intersection(w):
                self.connectedWires.append(w)

    def addTerminal(self, t, isGround=False):
        if t not in self.connectedTerminals:
            if isGround:
                self.connectedTerminals.insert(0, t)
            else:
                self.connectedTerminals.append(t)

    def importTerminals(self, w):
        for t in w.connectedTerminals:
            if t not in self.connectedTerminals:
                if t.element.type == ElementsEnum.Ground:
                    self.connectedTerminals.insert(0,t)
                else:
                    self.connectedTerminals.append(t)

    def setNodes(self):
        for i in range(len(self.connectedTerminals)):
            for j in range(i + 1, len(self.connectedTerminals)):
                t1 = self.connectedTerminals[i]
                t2 = self.connectedTerminals[j]
                t1.setNode(t2)


    def containsPoint(self, p):
        crossproduct = (p.y - self.start.y) * (self.end.x - self.start.x) -
(p.x - self.start.x) * (self.end.y - self.start.y)
        if abs(crossproduct) != 0 :
            return False
        dotproduct = (p.x - self.start.x) * (self.end.x - self.start.x) +
(p.y - self.start.y)*(self.end.y - self.start.y)
        if dotproduct < 0 :
            return False
        squaredlengthba = (self.end.x - self.start.x)*(self.end.x -
self.start.x) + (self.end.y - self.start.y)*(self.end.y - self.start.y)
        if dotproduct > squaredlengthba:
            return False
        return True
```

```
    def setId(self, i):
        self.id = i

    def getId(self):
        return self.id

    def getNameFrom(self, w):
        self.name = w.name

    def printWire(self):
        print self.name, ": "
        for t in self.connectedTerminals:
            print "\t", t.element.name, t.position

    def printToFile(self):
        return "W " + str(self.start.x) + " " + str(self.start.y) + " " +
str(self.end.x) + " " + str(self.end.y)


class Point(object):

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def equal(self, p):
        if p.x == self.x and p.y == self.y:
            return True
        else:
            return False

    def setCoords(self, x, y):
        self.x = x
        self.y = y

    def getCoords(self):
        return self.x, self.y

    def getX(self):
        return self.x

    def getY(self):
        return self.y

    def __str__(self):
        return "(" + str(self.x) + ", " + str(self.y) + ")"


class TerminalPoint(Point):

    def __init__(self, x, y, e, pos):
        super(TerminalPoint, self).__init__(x, y)
        self.element = e
        self.position = pos         # 'left' or 'right'
        self.node = None

    def setNode(self, t):
        if self.element.type is ElementsEnum.Ground:
            t.node = 'N0'
        elif t.element.type is ElementsEnum.Ground:
```

```
            self.node = 'N0'
        elif self.node is None:
            if t.node is None:
                global_var.nodeIndex += 1
                self.node = "N" + str(global_var.nodeIndex)
                t.node = "N" + str(global_var.nodeIndex)
            else:
                self.node = t.node
        else:
            t.node = self.node

    def clearNode(self):
        node = None

    def __str__(self):
        return str(self.node)
```

## circuit.py

```
from elements import *
import global_var
import matlab.engine

matlab_engine = matlab.engine.start_matlab()

class Circuit(object):

    def __init__(self):
        self.elements = []
        self.wires = []
        self.newWires = []
        self.terminals = []

        self.regime = "dc"
        self.startFreq = 0
        self.endFreq = 0
        self.intPoints = 0

        self.number_of_branches = 0
        self.branch_elements = []

        self.currents = []
        self.currents_phases = []
        self.voltages = []
        self.voltages_phases = []

    def clear_circuit(self):
        self.elements[:] = []
        self.wires[:] = []
        self.newWires[:] = []
        self.terminals[:] = []

        self.number_of_branches = 0
        self.branch_elements[:] = []

        self.currents = []
        self.currents_phases = []
```

```
        self.voltages = []
        self.voltages_phases = []

        global_var.nodeIndex = 0

    def process_wires(self):
        for w in self.wires:                          # for every wire
            w.setConnectedWires(self.wires)           # compute list of
directly connected wires
            for e in self.elements:                   # and list of directly
connected terminals
                if w.start.equal(e.terminalLeft) or
w.end.equal(e.terminalLeft) or w.containsPoint(e.terminalLeft):
                    if e.type is ElementsEnum.Ground:
                        w.addTerminal(e.terminalLeft, True)
                    else:
                        w.addTerminal(e.terminalLeft)
                if e.type is not ElementsEnum.Ground:
                    if w.start.equal(e.terminalRight) or
w.end.equal(e.terminalRight) or w.containsPoint(e.terminalRight):
                        w.addTerminal(e.terminalRight)

        for w in self.wires:        # clear visited
            w.visited = False

        for w in self.wires:        # dfs to rename all connected wires
            if w.visited is False:
                w.visited = True
                w.dfs()

        for i in range(len(self.wires)):                # compute
indirectly connected terminals
            for j in range(i + 1, len(self.wires)):
                w1 = self.wires[i]
                w2 = self.wires[j]
                if w1.name == w2.name:
                    w1.importTerminals(w2)
                    w2.importTerminals(w1)

        self.newWires = list(self.wires)                    # unify wires,
remove duplicates
        for i in range(len(self.wires)):
            for j in range(i + 1, len(self.wires)):
                w1 = self.wires[i]
                w2 = self.wires[j]
                if w1.name == w2.name and w2 in self.newWires:
                    self.newWires.remove(w2)

    def process_terminals(self):
        for w in self.newWires:
            w.setNodes()

        self.terminals = []

        for e in self.elements:
            if e.terminalLeft not in self.terminals:
                if e.type is ElementsEnum.Ground:
                    self.terminals.insert(0, e.terminalLeft)
                else:
                    self.terminals.append(e.terminalLeft)
```

```
            if e.terminalRight not in self.terminals and e.type is not
ElementsEnum.Ground:
                self.terminals.append(e.terminalRight)

        for i in range(len(self.terminals)):
            for j in range(i + 1, len(self.terminals)):
                t1 = self.terminals[i]
                t2 = self.terminals[j]
                if t1.equal(t2):
                    t1.setNode(t2)

    def add_element(self, e):
        self.elements.append(e)

    def remove_element(self, e):
        self.elements.remove(e)
        #self.terminals[:] = [t for t in self.terminals if t.element is not
e]

    def add_wire(self, e):
        self.wires.append(e)

    def remove_wire(self, e):
        self.wires.remove(e)

    def print_elements(self):
        print "elements:"
        for e in self.elements:
            e.printElement()
        print "wires:"
        for w in self.newWires:
            w.printWire()

    def get_element_by_name(self, name):
         for e in self.elements:
            if e.name == name:
                return e

    def get_element_by_id(self, id):
        result = None
        for e in self.elements:
            if e.getId() == id:
                result = e
        return result

    def get_wire_by_id(self, id):
        result = None
        for w in self.wires:
            if w.getId() == id:
                result = w
        return result

    def loadFromFile(self, lines):
        for line in lines:
            tokens = line.split()
            id = tokens[0]
            if id is "W":
                x1 = float(tokens[1])
                y1 = float(tokens[2])
                x2 = float(tokens[3])
                y2 = float(tokens[4])
```

```
                w = Wire(x1, y1, x2, y2)
                self.add_wire(w)
            else:
                x = float(tokens[1])
                y = float(tokens[2])
                rotated = int(tokens[3])
                value = tokens[4]
                name = tokens[5]
                if id is "R":
                    e = Resistor(x,y)
                    e.value = value
                    e.name = name
                elif id is "L":
                    e = Inductor(x,y)
                    e.value = value
                    e.name = name
                elif id is "C":
                    e = Capacitor(x,y)
                    e.value = value
                    e.name = name
                elif id is "V":
                    e = VoltageSource(x,y)
                    e.value = value
                    e.name = name
                elif id is "I":
                    e = CurrentSource(x,y)
                    e.value = value
                    e.name = name
                elif id is "G":
                    e = Ground(x,y)
                    e.value = value
                    e.name = name
                for rot in range(rotated):
                    e.rotate()
                self.add_element(e)

    def run_interpreter(self):
        self.process_wires()
        self.process_terminals()

    def run_matlab(self):
        branch = 0
        input_matrix = []
        for e in self.elements:
            if e.type is not ElementsEnum.Ground:
                if e.terminalLeft.node is None or e.terminalRight.node is
None:
                    raise Exception('\n\nERROR: Be aware of floating
terminals!')
                branch += 1
                e.branch = branch
                N_initial = float(e.terminalLeft.node[1])
                N_final = float(e.terminalRight.node[1])
                param1 = float(e.value)
                if self.regime == "ac" and (e.type ==
ElementsEnum.VoltageSource
                                            or e.type ==
ElementsEnum.CurrentSource):
                    param2 = float(e.phase)
                else:
                    param2 = 0
```

```
                line = [e.code, branch, N_initial, N_final, param1, param2]
                input_matrix.append(line)
        self.number_of_branches = branch

        if self.regime == "ac":
            self.startFreq = float(self.startFreq)
            self.endFreq = float(self.endFreq)
            self.intPoints = float(self.intPoints)
            input_matrix.append([11, self.startFreq, self.endFreq,
self.intPoints, 0, 0]) # frequencies in AC
        else:
            input_matrix.append([0, 0, 0, 0, 0, 0])      # no frequencies
in DC

        input_matrix_matlab = matlab.double(input_matrix)

        with open("input_matrix.txt", 'w') as f:
            f.writelines('\t'.join(str(j) for j in i) + '\n' for i in
input_matrix)

        self.branch_elements = list(self.elements)
        grounds = []
        for e in self.branch_elements:
            if e.type == ElementsEnum.Ground:
                grounds.append(e)
        for g in grounds:
            self.branch_elements.remove(g)

        if self.regime == "dc":
# DC
            I_lat, U_lat, P_ced, P_cons =
matlab_engine.Analysis_DC(input_matrix_matlab, nargout=4)
            self.currents = I_lat
            self.voltages = U_lat
        elif self.regime == "ac":
            if self.endFreq == 0 and self.intPoints == 0:
# AC with one frequency
                I_ef, I_faza, U_ef, U_faza =
matlab_engine.Analysis_AC_1f(input_matrix_matlab, nargout=4)
                self.currents = I_ef[0]
                self.currents_phases = I_faza[0]
                self.voltages = U_ef[0]
                self.voltages_phases = U_faza[0]
            else:
# AC with range of frequencies
                I_ef, I_faza, U_ef, U_faza, freq_plot, I_ef_plot,
I_faza_plot, U_ef_plot, U_faza_plot \
                    = matlab_engine.Analysis_AC_3f(input_matrix_matlab,
nargout=9)
                self.currents = I_ef[0]
                self.currents_phases = I_faza[0]
                self.voltages = U_ef[0]
                self.voltages_phases = U_faza[0]
                #info for plotting
                self.freq_plot = freq_plot
                self.I_ef_plot = I_ef_plot
                self.I_faza_plot = I_faza_plot
                self.U_ef_plot = U_ef_plot
                self.U_faza_plot = U_faza_plot
```

## Analysis_AC_3f.m

```
function
[I_efectiv,I_faza,U_efectiv,U_faza,freq,Branch_currents_RMS,Branch_currents
_PHASE,Branch_voltages_RMS,Branch_voltages_PHASE]=Analysis_AC_3f(mdg);

cod_E=1;
cod_C=2;
cod_R=3;
cod_L=4;
cod_J=5;
cod_M=10;
cod_f=11;

Types=[];
for k=1:length(mdg(:,1))
    if mdg(k,1)==cod_E
    Types=[Types;'E'];
    elseif mdg(k,1)==cod_C
    Types=[Types;'C'];
    elseif mdg(k,1)==cod_R
    Types=[Types;'R'];
    elseif mdg(k,1)==cod_L
    Types=[Types;'L'];
    elseif mdg(k,1)==cod_J
    Types=[Types;'J'];
    elseif mdg(k,1)==cod_M
    Types=[Types;'M'];
    elseif mdg(k,1)==cod_f
    Types=[Types;'f'];
    end
end

Values=mdg; Values(:,1)=[];

% Generam, structuri de date care contin toata informatia:
% "mutual"; "branch"
branch=[];
for k=1:length(Types)
    if Types(k)=='f';
        if Values(k,2)==0
            disp('AC Analysis performed at one frequency only')
            f=Values(k,1); % Frecventa de analiza
        else
        f(1)=Values(k,1); % Frecventa initiala
        f(2)=Values(k,2); % Frecventa finala
        f(3)=Values(k,3); % Numar de puncte intermediare
        end
    elseif Types(k)=='E';
        p=Values(k,1);
        branch(p).element=Types(k);
        branch(p).cod=cod_E;
        branch(p).nodin=Values(k,2);
        branch(p).nodfin=Values(k,3);
        branch(p).valoare(1)=Values(k,4);
        branch(p).valoare(2)=Values(k,5);
    elseif Types(k)=='C';
        p=Values(k,1);
        branch(p).element=Types(k);
```

```
        branch(p).cod=cod_C;
        branch(p).nodin=Values(k,2);
        branch(p).nodfin=Values(k,3);
        branch(p).valoare(1)=Values(k,4);
    elseif Types(k)=='R';
        p=Values(k,1);
        branch(p).element=Types(k);
        branch(p).cod=cod_R;
        branch(p).nodin=Values(k,2);
        branch(p).nodfin=Values(k,3);
        branch(p).valoare(1)=Values(k,4);
    elseif Types(k)=='L';
        p=Values(k,1);
        branch(p).element=Types(k);
        branch(p).cod=cod_L;
        branch(p).nodin=Values(k,2);
        branch(p).nodfin=Values(k,3);
        branch(p).valoare(1)=Values(k,4);
    elseif Types(k)=='J';
        p=Values(k,1);
        branch(p).element=Types(k);
        branch(p).cod=cod_J;
        branch(p).nodin=Values(k,2);
        branch(p).nodfin=Values(k,3);
        branch(p).valoare(1)=Values(k,4);
        branch(p).valoare(2)=Values(k,5);
    end
end
mutual=[];
m=0;
for k=1:length(Types)
    if Types(k)=='M';
        m=m+1;
        mutual(m).bobina_1=Values(k,1);
        mutual(m).bobina_2=Values(k,2);
        mutual(m).valoare=Values(k,3)*...
            sqrt(branch(mutual(m).bobina_1).valoare*...
            branch(mutual(m).bobina_2).valoare); % Inductivitatea mutula
    end
end
% Numarul de laturi
l=length(branch);
% Numarul de noduri
n=1+max([branch.nodin, branch.nodfin]);

% Construirea matricei laturi-noduri
a=zeros(n,l);
for k=1:l
    a(branch(k).nodin+1,k)=1;
    a(branch(k).nodfin+1,k)=-1;
end
% Matricea redusa laturi-noduri (se elimina linia
% corespunzatoiare nodului "0")
a=a(2:n,:);

% Ordonarea matricei laturi-noduri prin permutari intre coloane
% in sensul crescator al codurilor de element
[codes,order]=sort([branch.cod]);
ao=a(:,order);
% Arborele normal
[g,indep]=rref(ao);
```

```
arb=order(indep);
coarb=order;coarb(indep)=[];
if length(arb)==(n-1)
else
    % Circuitului nu i se poate asocia un arbore normal din cauza
    % unor erori de topologie
    error('TOPOLOGY ERROR: Check electrical diagram !')
end
% Verificare criterii de consistenta:
for k=arb
    if branch(k).cod==cod_J
    error('TOPOLOGY ERROR: Check the electric diagram for Current-source-
cutsets !')
    end
end
for k=coarb
    if branch(k).cod==cod_E
    error('TOPOLOGY ERROR: Check the electric diagram for Voltage-source-
loops !')
    end
end
% Verificare daca toate bobinele cuplate sunt in coarbore:
for k=1:length(mutual)
    if find(mutual(k).bobina_1==coarb)&find(mutual(k).bobina_2==coarb)
    else
       error('Not all coupled inductors are found in the cotree. The
solving method is not applicable')
    end
end

% Partitia matricei A corespunzatoare arborelui normal
ar=a(:,arb);
% Partitia corespunzatoare coarborelui
ac=a(:,coarb);
% Matricea incidentelor esentiale si partitiile ei
d=ar\ac;
% Selectia elementelor E, Za din arbore
    sE=[]; sZa=[];
    lat_E=[];lat_Za=[];
    for k=1:length(arb)
        if
branch(arb(k)).cod==cod_C|branch(arb(k)).cod==cod_R|branch(arb(k)).cod==cod
_L
            sZa=[sZa,k];
            lat_Za=[lat_Za, arb(k)];
        elseif branch(arb(k)).cod==cod_E
            sE=[sE,k];
            lat_E=[lat_E, arb(k)];
        end
    end
    % Selectia elementelor J, Zc din coarbore
    sJ=[];sZc=[];
    lat_J=[];lat_Zc=[];
    for k=1:length(coarb)
        if
branch(coarb(k)).cod==cod_C|branch(coarb(k)).cod==cod_R|branch(coarb(k)).co
d==cod_L
            sZc=[sZc,k];
            lat_Zc=[lat_Zc, coarb(k)];
        elseif branch(coarb(k)).cod==cod_J
            sJ=[sJ,k];
```

```
            lat_J=[lat_J, coarb(k)];
        end
    end

d11=d(sE,sZc);
d12=d(sE,sJ);

d21=d(sZa,sZc);
d22=d(sZa,sJ);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Trasarea caracteristicilor de frecventa
pas_f=(f(2)-f(1))/f(3);
freq=[f(1):pas_f:f(2)]';
Branch_currents=[];
Branch_voltages=[];
for k=1:f(3)+1
omega=2*pi*f(1)*k;
j=sqrt(-1);
% Matricea admitantelor ramurilor (fara mutual)
Ya=[];
for k=lat_Za
    if branch(k).cod==cod_C
        Ya=[Ya, j*omega*branch(k).valoare];
    elseif branch(k).cod==cod_R
        Ya=[Ya, 1/branch(k).valoare];
    elseif branch(k).cod==cod_L
        Ya=[Ya, 1/(j*omega*branch(k).valoare)];
    end
end
Ya=diag(Ya);

% Matricea impedantelor coardelor (fara mutual)
Zc=[];
for k=lat_Zc
    if branch(k).cod==cod_C
        Zc=[Zc, 1/(j*omega*branch(k).valoare)];
    elseif branch(k).cod==cod_R
        Zc=[Zc, branch(k).valoare];
    elseif branch(k).cod==cod_L
        Zc=[Zc, j*omega*branch(k).valoare];
    end
end
Zc=diag(Zc);
for k=1:length(mutual)
    p=find(mutual(k).bobina_1==lat_Zc);
    q=find(mutual(k).bobina_2==lat_Zc);
    Zc(p,q)=j*omega*mutual(k).valoare;
    Zc(q,p)=Zc(p,q);
end

% Vectorii surselor independente
J=[];
for k=1:length(lat_J)
    J=[J;
branch(lat_J(k)).valoare(1)*exp(j*branch(lat_J(k)).valoare(2)*pi/180)];
end
E=[];
for k=1:length(lat_E)
    E=[E; -
branch(lat_E(k)).valoare(1)*exp(j*branch(lat_E(k)).valoare(2)*pi/180)];
```

```
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Construirea modelului matematic - sistem de ec. A*X=N*S
% Matricea sistemului
M=[d21, Ya;...
   Zc, -d21'];

N=[-d22, zeros(length(lat_Za),length(lat_E));...
   zeros(length(lat_Zc),length(lat_J)), d11'];

% Solutia
x=M\(N*[J;E]);
Ic=x(1:length(lat_Zc));
Ua=x(length(lat_Zc)+1:length(lat_Zc)+length(lat_Za));

% Curentii coardelor
I_coarde=[Ic;J];
% Curentii ramurilor
I_ramuri=-d*I_coarde;
% Tensiunile ramurilor
U_ramuri=[E;Ua];
% Tensiunile coardelor
U_coarde=d'*U_ramuri;

% Curentii laturilor
I_lat=zeros(l,1);
for k=1:length(arb)
    I_lat(arb(k))=I_ramuri(k);
end
for k=1:length(coarb)
    I_lat(coarb(k))=I_coarde(k);
end
% Tensiunile laturilor
U_lat=zeros(l,1);
for k=1:length(arb)
    U_lat(arb(k))=U_ramuri(k);
end
for k=1:length(coarb)
    U_lat(coarb(k))=U_coarde(k);
end

% Memorarea solutiei
Branch_currents=[Branch_currents, I_lat];
Branch_voltages=[Branch_voltages, U_lat];
end
% Trasare caracteristici de frecventa
Branch_currents_RMS=abs(Branch_currents);
Branch_currents_PHASE=atan2(imag(Branch_currents),real(Branch_currents))*18
0/pi;
Branch_voltages_RMS=abs(Branch_voltages);
Branch_voltages_PHASE=atan2(imag(Branch_voltages),real(Branch_voltages))*18
0/pi;

%EXEMPLU DE AFISARE
% Afisarea caracteristicilor de frecventa ale curentului I3 si tensiunii
U3:
%figure;plot(freq,Branch_currents_RMS(3,:));grid
%figure;plot(freq,Branch_currents_PHASE(3,:));grid
%figure;plot(freq,Branch_voltages_RMS(3,:));grid
%figure;plot(freq,Branch_voltages_PHASE(3,:));grid
```

```
disp('Curentii laturilor:')
for k=1:l
    Vef=abs(I_lat(k));
    faza=atan2(imag(I_lat(k)),real(I_lat(k)))*180/pi;
    disp(['   I',int2str(k),' = ',num2str(I_lat(k)),...
            '  ==>  ',num2str(Vef),' [A] / ',num2str(faza),' [deg]'])
    I_efectiv(k)=round(Vef,4);
    I_faza(k)=round(faza,4);
end

disp(' ')
disp('Tensiunile laturilor:')
for k=1:l
    Vef=abs(U_lat(k));
    faza=atan2(imag(U_lat(k)),real(U_lat(k)))*180/pi;
    disp(['   U',int2str(k),' = ',num2str(U_lat(k)),...
     '  ==>  ',num2str(Vef),' [V] / ',num2str(faza),' [deg]'])
    U_efectiv(k)=round(Vef,4);
    U_faza(k)=round(faza,4);
end
```