2019 International Symposium on Lightning Protection (XV SIPDA), São Paulo, Brazil, 30th September – 4th October 2019.

The Usage of Julia Programming in Grounding Grids Simulations

An alternative to MATLAB and Python

[a]Rodolfo A. R. Moura, [a]Marco A. O. Schroeder, [a]Samuel J. S. Silva, [a]Erivelton G. Nepomuceno, [a]Pedro H. N. Vieira and [b]Antonio C. S. Lima.

[a]Department of Electrical Engineering (DEPEL)
Federal University of São João del-Rei (UFSJ)
São João del-Rei, Brazil
moura@ufsj.edu.br, schroeder@ufsj.edu.br, samuelsj10@ufsj.edu.br, nepomuceno@ufsj.edu.br, pedrohnv@hotmail.com

[b]Department of Electrical Engineering (DEE)
Federal University of Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
acsl@dee.ufrj.br

***Abstract*—Matlab and Python have been widely used in science field, mainly used by scientists who are involved in numerical and technical computing. However, other programming languages have arisen in recent years aim to create a combination of power, and efficiency for free. In light of this, this work presents a comparison between three computational environments (MATLAB, Python and Julia) – on the study of grounding grids for lightning transients – to evaluate the best programming language (among all studied). The main parameter presented on this paper is computational time. The results showed Julia programming language provided simulations six times faster than Matlab and twenty-eight faster than Python.**

Keywords—Computational Efficiency; Grounding and Earthing;Transients; Numerical Solution.

I. INTRODUCTION

When dealing with power systems, one element that must be accurately modeled is grounding grids. Its low-frequency response is relatively well-known [1]. Nevertheless, when request by faster transients (such as lightning) it presents a very complex response. One well-known technique to obtain such response is the Hybrid Electromagnetic Model (HEM) [2]. This particular method deals with an extensive matrix system requiring Electrical Engineers and researchers to use computational simulation.

To simulate such problems there are several computational environments, each one with its particularity. Among all programming languages, according to IEEE Spectrum [3], Python is the most popular. Moreover, according to Mathworks (developer of MATLAB) [4], MATLAB is a just-in-time compiled language fast to work with Matrices.

With the proposal of overcoming all scientific programming languages, Julia is a daring-just-in-time compilation and open source option. It is a newborn language with strong inspiration of C, FORTRAN, Python, MATLAB among others. Its basis is written in C language, however its standard library is written in Julia itself [5].

Thereby, the main idea behind this paper is to present a comparison between MATLAB, Python and Julia for grounding grids studies with a "native" computer implementation, i.e. straightforward use without much effort given to performance optimization. To analyze it, this paper is divided into four sections, including this introduction. The second section handle numerical model and the usage of each language, the third one shows the results obtained for the studied case and finally a conclusion is presented.

II. GROUNDING GRIDS MODELING

A. Hybrid Electromagnetic Model (HEM)

The HEM is a model proposed to evaluate impedance, overvoltage and electromagnetic fields generate by current that flows along conductors. It can be applied in several applications, such as overhead transmission lines [6], groundings [7] and lightning channel modeling [8].

Basically, it consists of two parts: i) discretizing the conductor element into cylindrical segments and, for each element, obtaining an average potential and a voltage drop; ii) applying electrical circuit theory to determine the node voltages. Although it seems an easy task, there is a heavy burden on the computational processing since, for all discretized element, double integral needs to be solved to relate the average potential and the voltage drop. Eq. (1) and (2) illustrate two parameters that compute each entity, Z_T (associated with average potential) and Z_L (associated with

voltage drop), between segments R (receptor segment) and S (source segment). More details can be found at [2,9].

$$Z_{T(RS)} = \frac{1}{4\pi[\sigma(\omega)+j\,\omega\varepsilon(\omega)]\,\ell_S\,\ell_R}\int_{\ell_S}\int_{\ell_R}\frac{e^{-\gamma r}}{r}\,dl_R\,dl_S \qquad (1)$$

$$Z_{L(RS)} = -\frac{j\,\omega\mu}{4\pi}\int_{\ell_S}\int_{\ell_R}\frac{e^{-\gamma r}}{r}\,d\vec{l}_R \bullet d\vec{l}_S \qquad (2)$$

where ℓ_S and ℓ_R are the segment lengths; r is the distance between the infinitesimal segments dl_S and dl_R; σ, ε and μ are the conductivity, permittivity and permeability of the soil, respectively; $\omega = 2\pi f$ is the angular frequency; $\gamma(\omega)=\sqrt{j\,\omega\mu[\sigma+j\,\omega\varepsilon]}$ is the propagation constant.

Furthermore, the main interest of this paper is analyzing the usage of HEM for studies of grounding electrodes. Thus, since the grounding electrode is found close to a boundary of two half-spaces (earth and air), the method of images is used [10].

B. Computational enviroment

Several software were developed to solve computational issues, such as numerical integration. One may highlight two easy general-purpose programming languages well-known among electrical engineers and researchers, MATLAB and Python. The first one is well-known for its applicability and manipulation with matrices while the other has a vast community to help, has shown usage in almost every area of engineering and is open source.

Released at 2012 by Massachusetts Institute of Technology a new language has been presented, Julia. It presents syntax similar to MATLAB, however, free and faster (according to its creators, velocity similar to C programming) [5]. Moreover, it is expected to deal similarly and faster with matrices than MATLAB. For now, the biggest disadvantage of Julia programming lies on: lack of users when compared to other popular languages, few books and courses as it got out of development recently, on August 2018, when version 1.0 was released. Currently it is on version 1.1.1, released on May 2019.

In this paper the double integrals were numerically solved by the Gauss-Legendre method [11]. This method was chosen because among all integral tested, this presented the fastest convergence. Since this kind of system respect the reciprocity theorem, only upper diagonal elements of each matrix is calculated, to improve computational efficiency.

To ensure comparability, all routines were written by the same programmer and the same algorithms compared. Besides, the same system were simulated 100 times (for each programming language) to guarantee a convergence on the average values of computational time. It was used a computer with I5-8250U (1.6 GHz) processor and 8 GB RAM (2133 MHz). Same simulations took place on both Linux (Mint 19.1) and Windows 10 (home version 1803) operating systems.

III. RESULTS

A. Premisses and test case

The main objective of study in this paper is grounding when requested by lightning currents. Thus, one parameter of utmost importance is harmonic impedance (Z(ω)). Therefore, to evaluate the computational effort, the case test is a simple grounding grid composed of four squares of 25 m² (total of 100 m²), as illustrated in Fig. 1. The soil resistivity is 100 Ω.m and permittivity of 10, each electrode presents radius of 7 mm.

Fig. 2 depicts the module of Z(ω) considering the injection point of current in any of the four corners. Its values were found for every computational environment leading to the conclusion that the correct result is obtained, regardless of any programming language adopted here. Fig. 3 illustrate the percentage difference, considering Julia as the reference, with simulations performed on Windows and Linux operating system, respectively. These figures were plotted considering Eq. (1). For both cases, the maximum difference was less than 0.5%.

$$diff[\%] = \frac{Z(\omega) - Z_{Reference}(\omega)}{Z_{Reference}(\omega)} * 100 \qquad (1)$$

According to Fig. 3, Julia programming has a better match with MATLAB for the lower frequency range. On the other hand, for the higher frequency spectrum, Python presents a better match. However, it is important mention that comparing Julia and MATLAB the differences seem to be more systematic, i.e., with a monotonic growth.

Further, it is important mentioning that, no parallel computation, i.e., only sequential loops were used. It was simulated in both operating systems and for every programming language considered on this paper (Julia, MATLAB and Python). Each simulation was made considering a frequency range from 100 Hz to 10 MHz logarithmically separated in 32 points.

B. Computational time

Table 1 and Table 2 depict it for the three computational environments considering, respectively, Windows and Linux operating system. According to the results, regardless of operating system, for the grounding system analyzed, Julia is the fastest one while Python is the slowest.

Even more, it is possible to realize that, for every case, windows consume more computational time. Apparently, there are many tasks running in the background slowing the main algorithm down.

C. Performance

It is important mentioning that the authors know that one may achieve higher performance using MATLAB and Python when properly vectorizing the code instead of using "for loops". For further studies, the authors intend to implement parallel computing and vectorization to reevaluate the studied case. Such vectorization seems to be unneeded in Julia due to its type inference system. As Bezanson notes:

It is not that "for loops" are inherently slow in themselves. The slowness comes from the fact that in the case of most dynamic languages, the system does not have access to the types of the variables within a loop. Since programs often spend much of their time doing repeated computations, the slowness of a particular operation due to lack of type information is magnified inside a loop [12, p. 78].

Table 1 - Computational time for each computational environment. Windows operating system.

Programming Language	Average Time [s]	Standard deviation
Julia	23.87	0.745
MATLAB	147.91	2.138
Python	615.28	1.738

Table 2 - Computational time for each computational environment. Linux operating system.

Programming Language	Average Time [s]	Standard deviation
Julia	22.04	0.640
MATLAB	133.15	1.924
Python	565.43	1.597

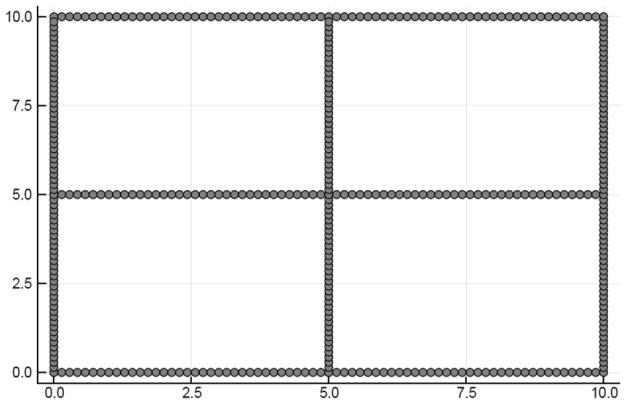

Figure 1 - Grounding grid configuration.

I. CONCLUSION

This paper deals with the usage of few computational environments for programming. The results shown that, for the particular study proposed in this paper, Julia is the fastest solution to deals with extensive matrices. Of course, to present a more incisive conclusion, other programming languages should be implemented (such as FORTRAN and C). However, the scope of this paper is to work with high-level syntax programming languages.

Moreover, in this specific case presented Linux provided more efficient than Windows. The authors strong believe that it occurs due to background tasks. Therefore, it is concluded that the best option for studying grounding grids is using Julia on Linux.

Finally, it is important to comment that other grounding configurations (such as tower groundings, bigger grids, horizontal electrode) were studied. The same pattern was found leading to the same conclusions presented on this paper.

Future works will deal with parallel programming (to enhance results) and low-level syntax languages (such as C and FORTRAN).

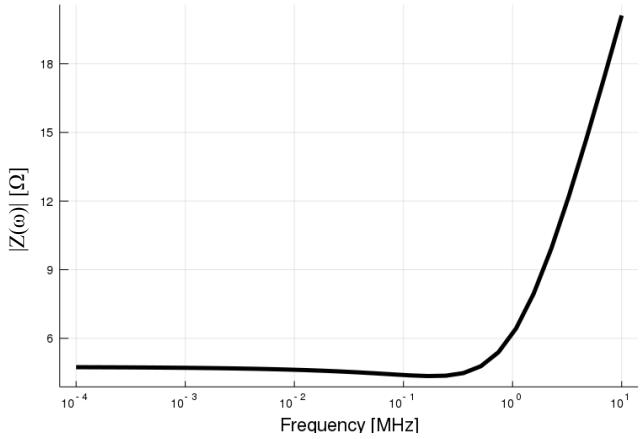

Figure 2 – Module of Z(ω) of the grounding grid understudy. Frequency range from 100 Hz to 10 MHz.

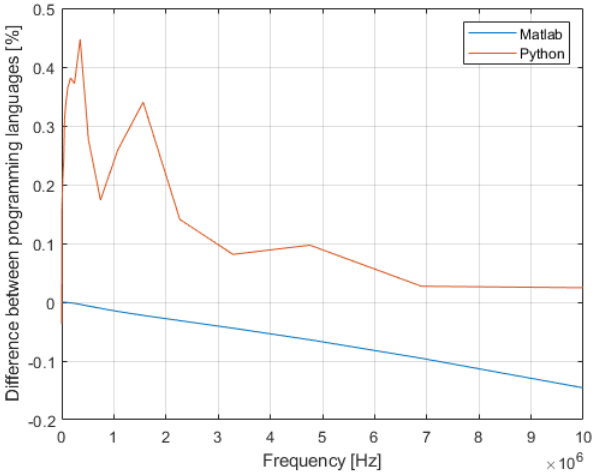

Figure 3 – Perceptual differences between Julia and both Matlab and Python programming languages for calculating harmonic impedance.

II. ACKNOWLEDGMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), Finance Code 001. It also was partially supported by INERGE (Instituto Nacional de Energia Elétrica), CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), FAPEMIG (Fundação de Amparo à Pesquisa do Estado de Minas), and FAPERJ (Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro).

REFERENCES

[1] IEEE Guide for Safety in AC Substation Grounding, IEEE Std. 80-2013, May 2015.

[2] S. Visacro, A. Soares and M.A.O. Schroeder, "An interactive computational code for simulation of transient behavior of electrical system components for lightning currents", in 26th International Conference on Lightning Protection, Cracow, Poland, 2002, pp. 732-737.

[3] "Interactive: The Top Programming Languages 2018", Internet: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018, Jul. 31, 2018 [Apr. 10, 2019].

[4] "Why MATLAB", Internet: https://www.mathworks.com/products/matlab/why-matlab.html, [May. 10, 2019].

[5] "Julia 1.1 Documentation", Internet: https://docs.julialang.org/en/v1/, [Apr. 10, 2019].

[6] R. A. R. Moura, "Representação de linha de transmissão com geometrica não uniforme para estudos de sobretensões atmosféricas [Modeling nonuniform overhead transmission lines for lightning overvoltages studies]", D. Sc. thesis, COPPE, UFRJ, Rio de Janeiro - Brazil, 2018.

[7] M. A. O. Schroeder, M. T. C. Barros, A. C. S. Lima, M. M. Afonso and R. A. R. Moura, Evaluation of the Impact of different frequency dependent Soil models on lightning overvoltages, Electric Power Systems Research, v.159, p. 40-49, 2018.

[8] S. Visacro and F. H. Silveira, Evaluation of current distribution along the lightning discharge channel by a hybrid electromagnetic model, Journal of Electrostatics, Vol. 60, no. 2-4, pp. 111 - 120, 2017.

[9] S. Visacro and A. Soares, "HEM: A model for simulation of lightning-related engineering problems", IEEE Trans. Power Delivery, Vol. 20, no. 2, pp. 1206-1207, 2005.

[10] V. Arnautovski-Toseva and L. Grcev, On the Image Model of a Buried Horizontal Wire, IEEE Trans. Elec. Comp., vol. 58, n.1, pp. 278 - 286, 2015.

[11] M. Abramowicz, I.A. Stegun (Eds.), "Handbook of Mathematical Functions", Dover Publications, 1964.

[12] Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. SIAM review, 59(1), 65-98.