DOI: 10.18605/2175-7275/cereus.v10n2p122-136

<<Recebido: 05/06/2018. Aceito: 09/07/2018 >>

ARTIGO ORIGINAL

REVISTA CEREUS ISSN: 2175-7275

Parallelization of the Complex Nonlinear Regression procedure applied in electrochemical impedance data for a wide potential range

Marco André Abud Kappel, Ricardo Fabbri [2], Roberto Pinheiro Domingos [2], Ivan Napoleão Bastos[2]

ABSTRACT

Electrochemical impedance spectroscopy (EIS) is a widely used technique in electrochemical systems characterization. Modeling this data is usually done using equivalent electrical circuits. These circuits have parameters that need to be fitted correctly, in order to enable the simulation of impedance data. Furthermore, the circuit fitting can be made for a wide potential range, allowing a characterization of the circuit elements evolution according to potential. At first, this work presents a sequential fitting methodology with high computational cost, using the optimization method Differential Evolution in each applied potential. The fitted parameters obtained for each potential step are used in the next, accelerating the fitting process and ensuring the smoothness necessary for the evolution of the circuit. Then, a parallelized algorithm is proposed for the problem, in order to reduce the fitting runtime, keeping the dependency relationship among applied potentials. Finally, results show that the parallelized algorithm is almost 50 times faster than the original and reaches the correct fitted values with the same accuracy.

Keywords: Electrochemical Impedance, Differential Evolution, Optimization, Stochastic Methods, Parallel Computing.

[1] Professor no Centro Federal de Educacão Tecnológica Celso Suckow da Fonseca – CEFET/RJ.

E-mail: marco.kappel@cefet-rj.br

[2] Professor na Universidade do Estado do Rio de Janeiro - UERJ.

1. INTRODUCTION

The use of electrical equivalent circuits on analysis of impedance data has become the main technique for this objective (ORAZEM & TRIBOLLET, 2008). Mathematically simple and physically coherent, electrical circuit models have the purpose to represent an abstraction of the main features of the studied system, providing important conclusions about the involved processes. Each element used in the circuit must have a physical match in the electrochemical system. A resistor, for example, can represent the electrolyte resistance of the electrochemical cell, while a capacitor may represent the capacitive processes occurring in the electrode double layer. Typically, this analogy is made by inspection and comparison with results obtained by other independent physico-chemical techniques.

Different circuits are used according to the application, with their topologies directly linked to the characteristics of the involved processes. After identification of the circuit that best fits the physical characteristics of the system, a complex nonlinear regression needs to be performed to fit the circuit parameters to values closer to the experimental data. Using these values allows the EIS data to be used in practical predictions of corrosion (SILVERMAN, 1993).

The intensity of electrochemical phenomena strongly depends on the applied potential. Thus, in some works, especially those studying some specific electrochemical mechanism, the impedance measurements are made on several potentials (MATTOS & BARCIA, 1990; HU et al., 2004). Furthermore, recent studies (BASTOS et al., 2013; KAPPEL et al., 2014) perform the application of the EIS for a wide potential range, providing a large amount of experimental impedance data to be analyzed and fitted to equivalent electrical circuits. Therefore, it is possible to use not only the transient impedance data in the fitting, as well as the stationary response data, available only when a wide potential range is used (KAPPEL et al., 2016).

The main objective of this work is to improve the existing software of impedance fitting and analysis on wide potential ranges (KAPPEL et al., 2015), in order to reduce its processing time. This will be done by parallelizing the program logic and running it on a high performance cluster. In order to identify the most appropriate parallelizing strategy, a brief bibliographical research will be done on the possible parallelization techniques to be applied. In addition to improving the speed of execution, the algorithm needs to keep the dependence

between the circuit parameters in consecutive potentials, besides using the stationary response in the objective function of the fitting process.

2. METHODOLOGY

The software that needs to be parallelized applies the optimization method Differential Evolution to adjust an electrical circuit model equivalent to experimental impedance data in various potentials. For each potential, the impedance varies with frequency. This process is performed in a wide range of potential, in order to better characterize the corrosion phenomenon. Thus, for each potential, a series of frequencies is applied and measurements are made. The measured experimental values can then be modeled by an equivalent electrical circuit. The impedance Z is a complex quantity, i.e., has a real and imaginary part.

In the studied case, the fitting needs to be done in a potential sequence, which leads to the necessity to resolve a large number of optimization problems. For each potential, impedance data for the whole frequency range are fitted to an equivalent electrical circuit. On the current version of the software, the fitting procedure is performed for each potential, serially. The results obtained for the lowest potential are used in the next potential, increasing the probability of the optimal solution to be found more rapidly, taking into consideration that the parameters vary smoothly between sequential potentials. Figure 1 shows a scheme that represents the adopted procedure.

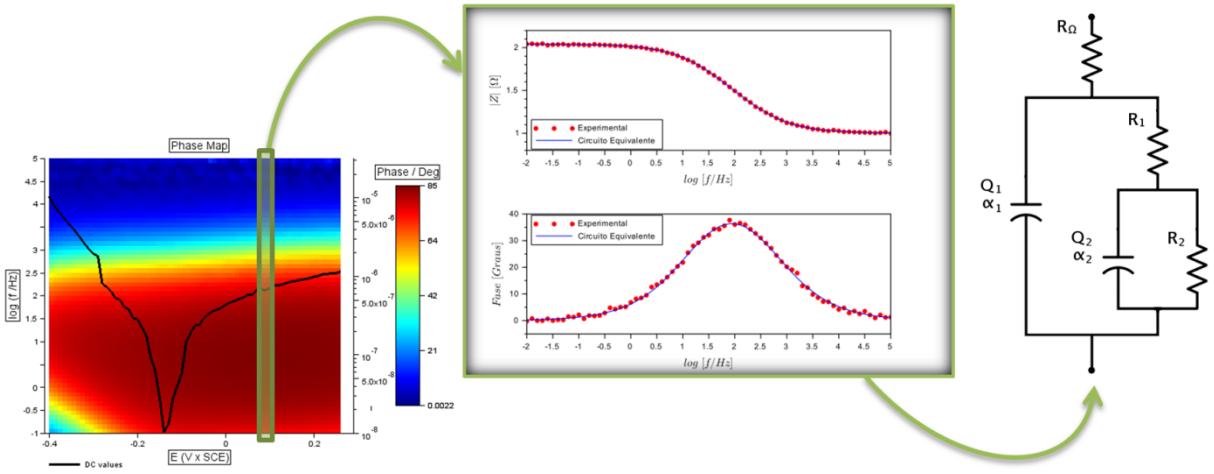

Figure 1. Fitting procedure scheme.

The mathematical model of the equivalent electric circuit used in this work is given by:

$$Z_{eq}(\omega) = R_{\Omega} + \frac{R_1 + \frac{R_2}{1 + (i\omega)^{\alpha_2} Q_2 R_2}}{1 + Q_1 \left(R_1 + \frac{R_2}{1 + (i\omega)^{\alpha_2} Q_2 R_2} \right) (i\omega)^{\alpha_1}} \quad (1)$$

It is desired to obtain the set of parameters R_{Ω}, R_1, R_2, Q_1, Q_2, α_1, e α_2. For this, the following objective function must be minimized:

$$f_{obj}(\omega) = \sum \left(Z_{real}^{calc} - Z_{real}^{exp} \right)^2 + \sum \left(Z_{imag}^{calc} - Z_{imag}^{exp} \right)^2 + \left(Z_{f=0}^{calc} - Z_{f=0}^{exp} \right)^2 \quad (2)$$

The ultimate goal is to find the variation of the equivalent circuit parameters according to the potential, enabling realistic simulation of electrochemical impedance using the equivalent circuit.

2.1. Differential Evolution

Differential Evolution (DE) is a metaheuristic, i.e., a generic form method for solving optimization problems, developed by STORN and PRICE (1997). The main idea is to use evolutionary operators to modify an initial population of real value vectors, which characterizes solution points of the problem space. Then, at each iteration, the algorithm provides a new population of the same size until a prescribed stop condition is met. The applied operators are: mutation, crossover and selection. There are several possible DE variants. The one used in the present work was DE/rand/1/bin. The algorithmic flow of the DE operation implemented is shown in Figure 2.

First, an initial population of possible solutions is randomly created within the search space. Each individual represents a solution candidate composed by one real value of each parameter to be estimated in the model. The size of the population is defined as a DE configuration parameter. For the creation of a new population representing the next generation, the mutation operator is initially employed. For each population individual v_i, a mutant vector v_{mut} is created. The mutation consists of adding the difference, weighted by a

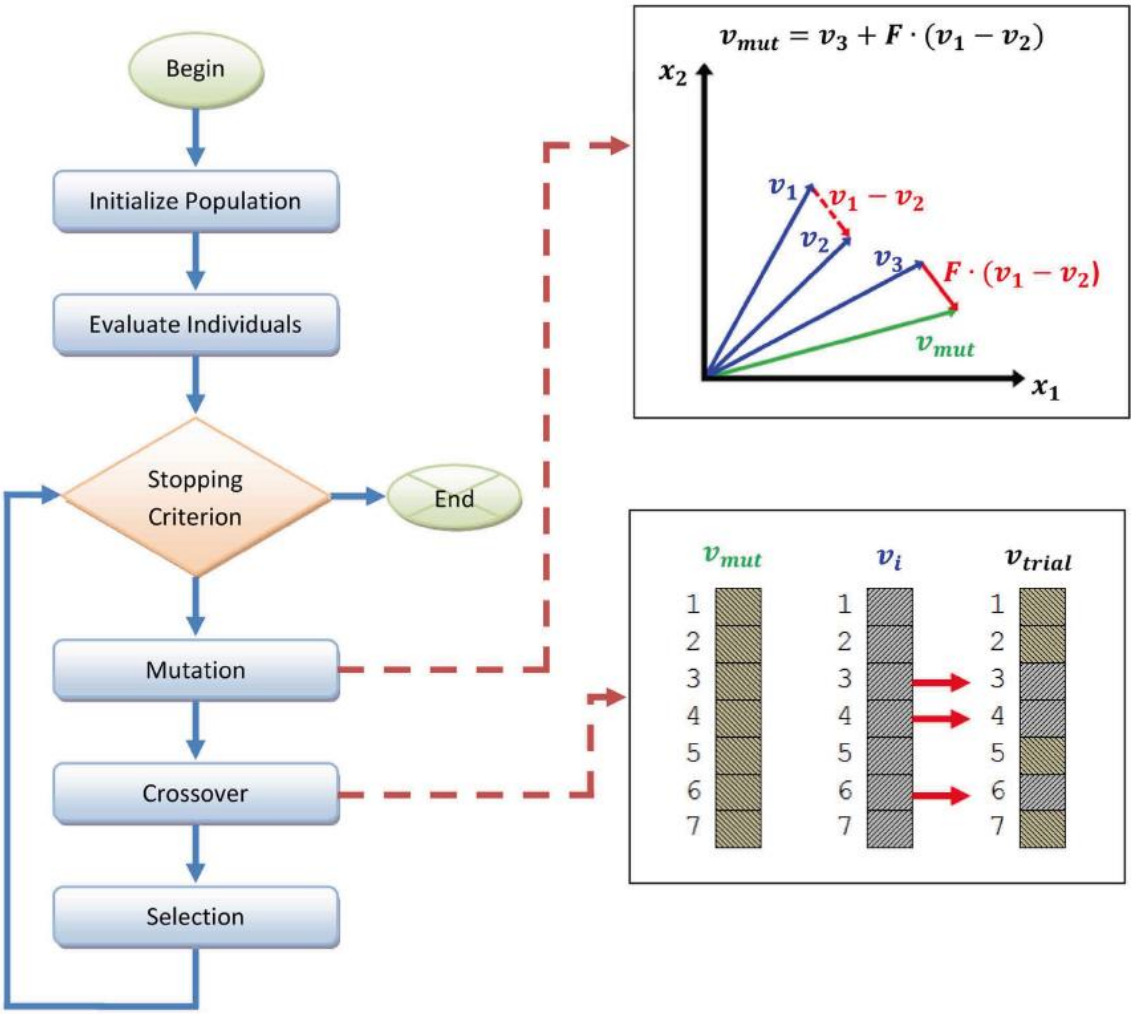

Figure 2. Differential Evolution algorithm flowchart.

factor F, of two randomly selected vectors, v_1 and v_2, to a third, also random, v_3, resulting a mutation vector v_{mut}. F is a configuration parameter of DE that controls the amplification of the variation of the difference of the vectors. Then, the crossover occurs, where each target vector v_i on the population, is mixed with the mutation vector, thus generating a trial vector vtrial. Finally, selection occurs, the step in which an objective function is applied to the trial vector and v_i in order to find out the best result. This output becomes the new individual that goes to the next generation. This process is done until the stipulated objective is achieved, generally characterized by minimization or maximization of a given function. The stop criterion adopted in the present work was a fixed number of generations, simulating that the optimum parameters are not known.

Because it is a direct stochastic search technique, Differential Evolution has the ability to deal with non-differentiable, non-linear and multimodal objective functions. This fact is extremely beneficial in cases of actual physical problems like the one in this work, which

usually have these features. Another good advantage of Differential Evolution is the ease of use by requiring the entry of only a few configuration variables. Additionally, a study from STORN and PRICE (1997) shows that the method has good convergence properties, especially for the global minimum, in independent tests.

Usually, stochastic optimization methods, such as Differential Evolution, are not present among the options provided by the owners of impedance fitting tools. Thus, despite its advantages, few studies make use of these techniques in modeling impedance using equivalent circuits, such as the reference (SHARIFI-ASL et al., 2013).

2.2. Sequential Implementation

As the original system was implemented in an academic environment, with few functionalities and with the need to easily manipulate matrices of complex experimental data, the programming language used to develop the software was Scilab. Initially, the software reads the experimental data from data sheets. Then, some configuration parameters necessary to solve the problem, as the circuit model being used and some physical characteristics of the experimental data, are set.

The optimization procedure is then initiated. The algorithm process the data and calls the optimization function. The logic for a wide potential range fitting process is defined in this function. First, the program creates an array that represents the initial population of solution candidates. Assuming that, in total, 7 parameters are going to be estimated, and 700 individuals are used in the population, the matrix will have the dimensions 7 x 700, where each column vector corresponds to a candidate set of parameters to the solution of the problem. In this population, the differential evolution algorithm is applied to the first potential data. At each iteration, the genetic operators are applied to the population of the matrix, evolving candidates to the solution until an optimum individual is found, which is the solution of the problem in the first potential. This solution is then written in an output file.

In the second potential, the same procedure is adopted, with one difference: the solution estimated for the previous potential is inserted into the optimization process. To do this, several "clones" of the previous solution are created in the initial population of this potential. For example, if the random population consists of 700 individuals, 10 individuals, identical to the previous solution, are added in the population, increasing the number of columns to 710. This procedure assists the fitting, since individuals near to the optimum are already present in the initial population of each potential, after the first. Furthermore, in

general, depending on the complexity and number of circuit parameters, different parameters can lead to similar responses of the objective function, featuring an ill-posed inverse problem (SGURA & BOZZINI, 2005). Using the previous solutions in the fitting of the following potential helps ensure that the progress of the found parameters is actually a result of real physical development, i.e., it increases the probability of the results obtained to be close to each other. Figure 3 exemplifies this process.

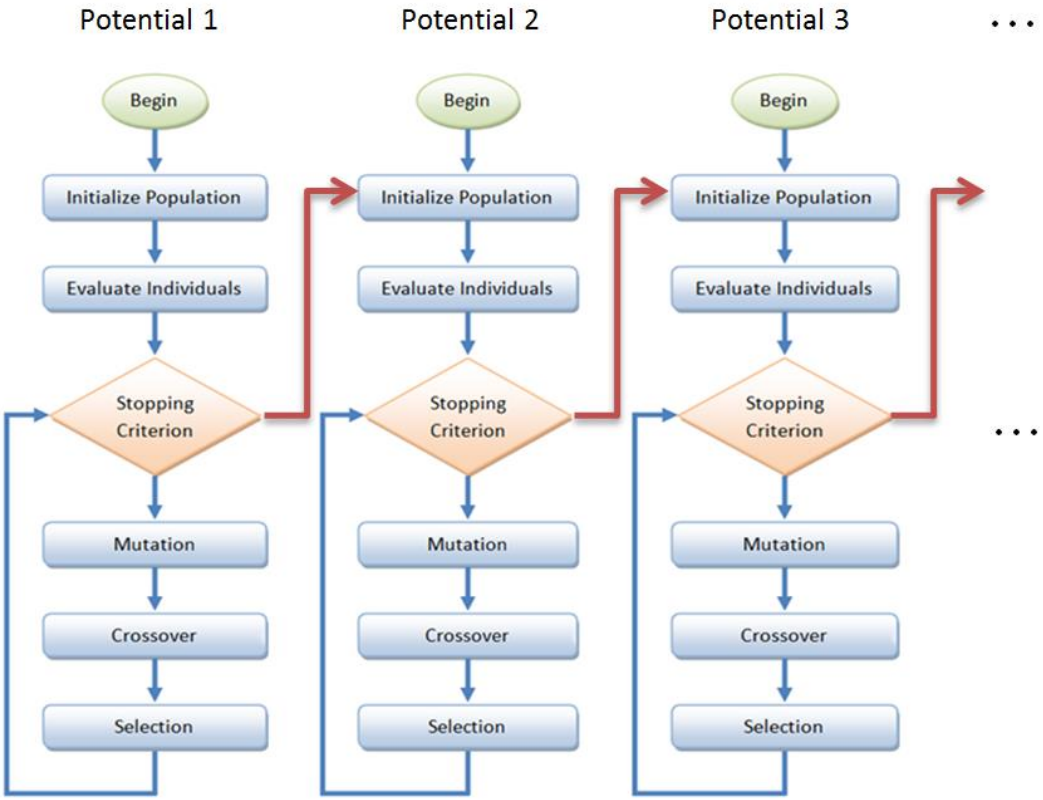

Figure 3. Flowchart of the sequential implementation functioning.

2.2. Parallelized Implementation

Scilab is not the most efficient choice in terms of run time. Thus, an approach that can greatly improve program performance and increase the range of parallelization technology options to be used is to re-implement the system on a lower-level language. On the other hand, using the parallelization tools available in Scilab itself to reduce the runtime required to fit all the impedance data, with minimal impact on the original software, would avoid a reimplementation of the software on another language. This is the approach adopted in this work.

The main purpose of parallelizing the equivalent electrical circuits impedance fitting software is to reduce the processing time required for obtaining solutions to all experimental data corresponding to a wide potential range. In sequential implementation, the duration of the system processing exceeds 20 hours. Increasing the speed to obtain the results is required, especially, because of the use of a stochastic optimization method. Due to the random component in this type of algorithm, its application must be accompanied by a statistical analysis of their results (KAPPEL et al., 2017). In order to do this, it is necessary to replicate software executions, making possible to obtain statistical information about the fitting, as mean and standard deviation of the parameters, demonstrating that the solution provided by the method was not achieved by chance, being statistically valid.

After careful analysis of the problem, the parallelization strategy chosen for the system was the use of threads. Since the system is implemented in Scilab, and in order to avoid the rework of converting the entire implementation to a lower level language like C/C++, it was understood that the parallelization library Scilab suited well to the needs of this problem.

2.2.1. Parallel Computing with Scilab

Parallel computing using Scilab wasn't possible until recent years, prior to version 5 of the language. Since this version, Scilab provides a number of tools, albeit limited, that enable script parallelization, and the use of distributed systems like clusters (BAUDIN & LEDRU, 2013). These tools are basically separated into three categories: multithread programming, using multicore processors, currently available on practically any computer; distributed computing, for use in physically separate computers or clusters, but in network; and programming using a graphics card.

Essentially, there are two multithread programming techniques in Scilab. The first, denominated implicit, automatically parallels matrix operations such as multiplication and division. Therefore, the first step to be taken in the system parallelization process is to refactor the code, trying to replace operations applied in simple variables or vectors for matrix operations. Like this, Scilab will automatically use all available processors for these procedures, optimizing its operation. The second way is to use the tool "parallel_run" which explicitly creates different threads according to the user's desire. Thus, iterative loops as "for" can run simultaneously in different threads. In the current version of Scilab, this feature still does not work on Windows operating systems, but has good efficiency in Linux systems.

Distributed computing can be made using MPI multiprocessing. It is simply needed to run the Scilab from the command line using the directive “mpirun”. This kind of solution would be useful to simultaneously perform replica of adjustments. In this case, the more available nodes in the cluster, the greater the number of replicas that could be executed simultaneously. Unfortunately, the documentation available from the MPI for Scilab is still very incomplete, not showing how to do simple things, like getting the rank of each process. For this reason, this approach has not been adopted in this work, but may be implemented further, in future works.

Programming using graphics cards are available in Scilab through the toolbox sciGPGPU. Essentially, this toolbox uses CUDA functions. In this case, this approach was not the most suitable because of the very nature of the technique. Normally, graphics cards processing is recommended for very thin-grained problems, in which many calculations need to be done simultaneously, as in highly complex simulations. In the problem at hand, the use of the large number of available threads on GPUs does not bring enough benefit to justify its use. The operation of copying the data to the board memory, to process them and return them to the main memory brings a further difficulty in programming, which makes it necessary further code refactoring. Thus, if a large refactoring needs to be done, it would be more advantageous to port the system to a lower level language like C/C++.

2.2.2. Problem Parallelization

In the case of this problem, the parallelization technique adopted is to separate the application of the optimization process in each potential in different threads. Thus, all the fitting procedures occur simultaneously, dramatically reducing the time required to get solutions in all potentials.

However, as there is dependence on each solution with the previous potential, it is necessary to adopt a synchronization strategy in the applications of the optimization method Differential Evolution. The idea is that, at each iteration, the best individual of the previous potential population is “cloned” to the next potential population. Thus, the fitting made at each potential, after the second, uses the best result from the previous potential, improving the process as a whole. Figure 4 shows the functioning of this idea.

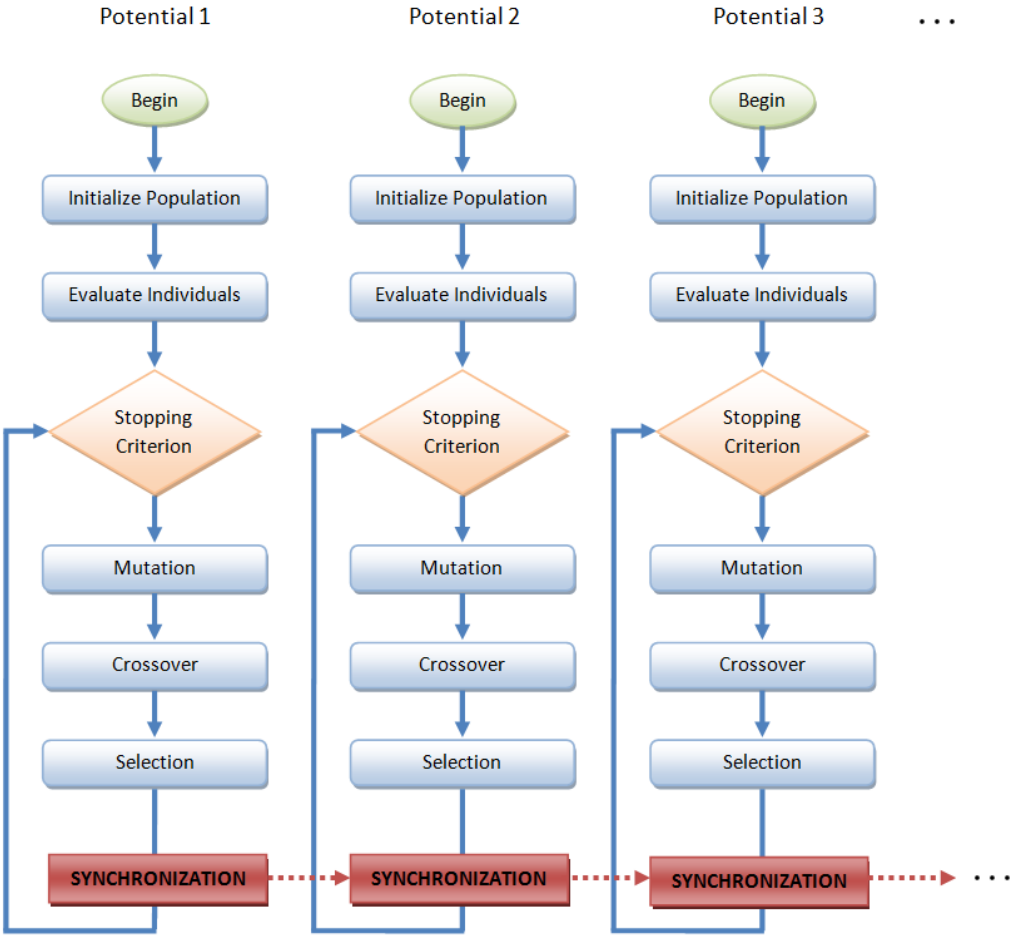

Figure 4. Flowchart of the sequential implementation functioning.

To accomplish this synchronization, it was necessary to use a logic similar to that described in (MCKENNEY, 2015) for statistical counters with implementation based on arrays. This logic is based on creating an array where each element is the unique counter for each thread. Thus, each thread has access to the corresponding counter element in the array. As only the very thread accesses that element, the atomicity of the operation is guaranteed.

Similarly, in the problem at hand, a matrix was created in shared memory scope, among the threads. This matrix has the number of columns equal to the number of applied potentials, that is, the number of optimization problems to be solved simultaneously. Each thread will only write in the column that corresponds to the potential being solved by it. The columns of the shared array will keep the best solution candidate of their population, at the time. At the end of each iteration of Differential Evolution, each thread will write in the matrix the temporary best individual of the population. At the beginning of each iteration running on threads, each thread will check if there is any value written in the column that corresponds

to the previous potential of the shared array. If so, this individual is “cloned” to the thread population, which shall consider this solution in its processing. If there isn’t an individual in the previous potential, it means that the corresponding thread to such potential has not finished running your iteration. In this case, the optimization process continues normally and the verification occurs again in the next iteration.

The biggest advantage of this implementation is in the speed that the full problem, in all potentials, is solved, keeping the dependence between the partial results of each individual optimization problem. The greater the number of threads used, the faster a complete solution is obtained, but the lower is the dependency between individual solutions. Thus, it is necessary to make a balance between the need to increase the speed and the need for the smooth evolution of the solutions, by the dependency between each potential.

3. Results and Discussion

The experimental data used in this work were taken from (KAPPEL et al., 2016). In this reference, the sequential implementation of the software was used to fit all the experimental data to the equivalent electrical circuit shown in section 2. Figure 5 shows the experimental data in the form of impedance maps.

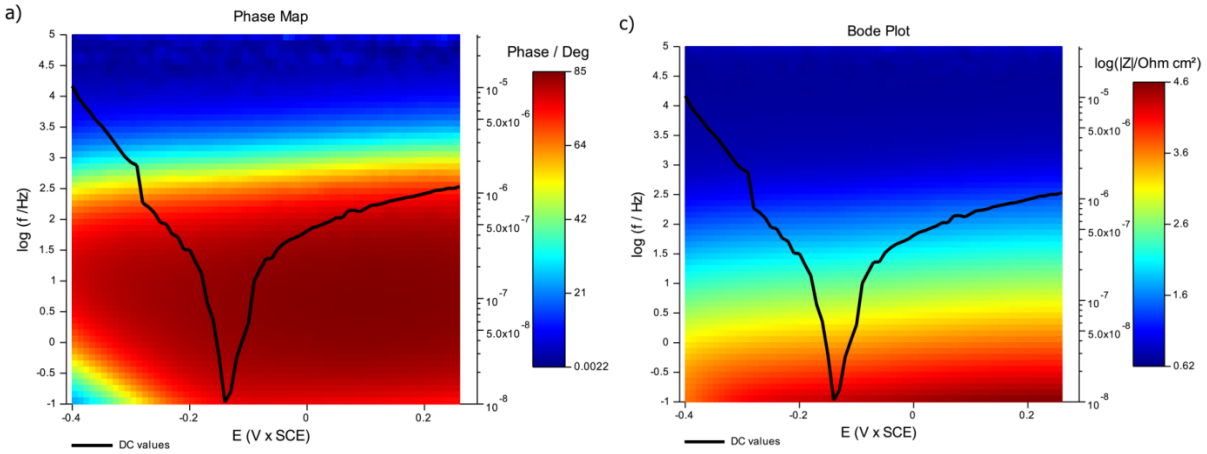

Figure 5. a) Phase angle and b) module of the experimental impedance, overlapped by the DC values.

This data set consists of 67 impedance module and phase diagrams, each corresponding to measurements made at a different potential. Thus, using the parallelized by threads implementation, it was decided that, in this case, the optimal number of threads

that corresponds to a good balance between runtime and dependence among the solutions was 15. Thus, in the cluster execution, we used only 1 node and 15 cores.

Figure 6 shows a schematic of the sequential implementation execution in terms of run time. In it, each color represents the optimization process in a potential. The program was executed on the author's personal computer, a Dell XPS L502X notebook with 8GB of RAM and an Intel i7-2670QM 8-core processor, running Windows 7 Home Premium. In this run, the entire process was completed in approximately 23 hours.

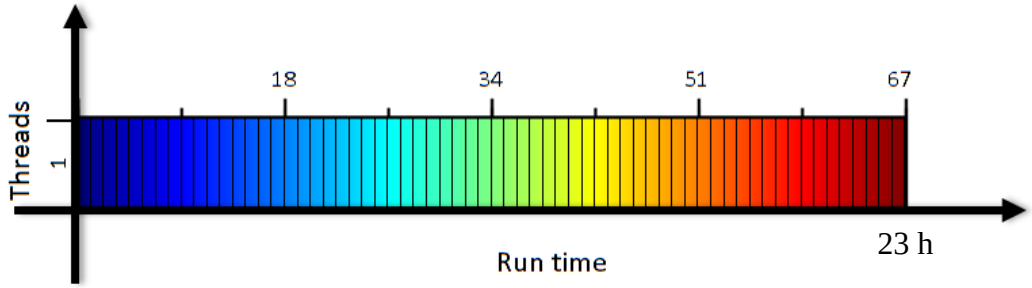

Figure 6. Execution of the sequential implementation regarding processing time. Each color represents the optimization process in a potential.

The parallelized implementation was executed on the IPRJ-UERJ cluster, a high-performance system installed and configured on a Dell 44U Rack, consisting of 7 PowerEdge servants units with 2 Intel Xeon E5-2620 processors each, for a total of 140 threads available for processing computer simulations. In this case, we used only one node, and the number of threads was limited to 15, in order to not lose the dependency between the results for each potential. To check the distribution of the optimization processes by threads, a statement was placed on the paralleled implementation, so that at every beginning and end of a thread execution, the processed potential number is wrote into a file. Thus, it was possible to construct the graph in Figure 7, which shows how the parallelization logic worked, regarding the total run time. This run has completed the full procedure of equivalent circuit fitting in 29 minutes, making the algorithm almost 50 times faster than the original. This is due not only to the parallelization of explicit threads, but also to the Scilab implicit parallelization due to refactoring of the operations, making the most of matrix operations instead of operations on simple variables.

Analyzing Figure 7, it is also possible to realize that the execution of the potentials in each thread occurs consecutively, meaning that the dependency with the previous potential is used in all following optimization processes, except those that are initially processed by the threads. After them, all settings will have a good initial information, further accelerating the optimization process.

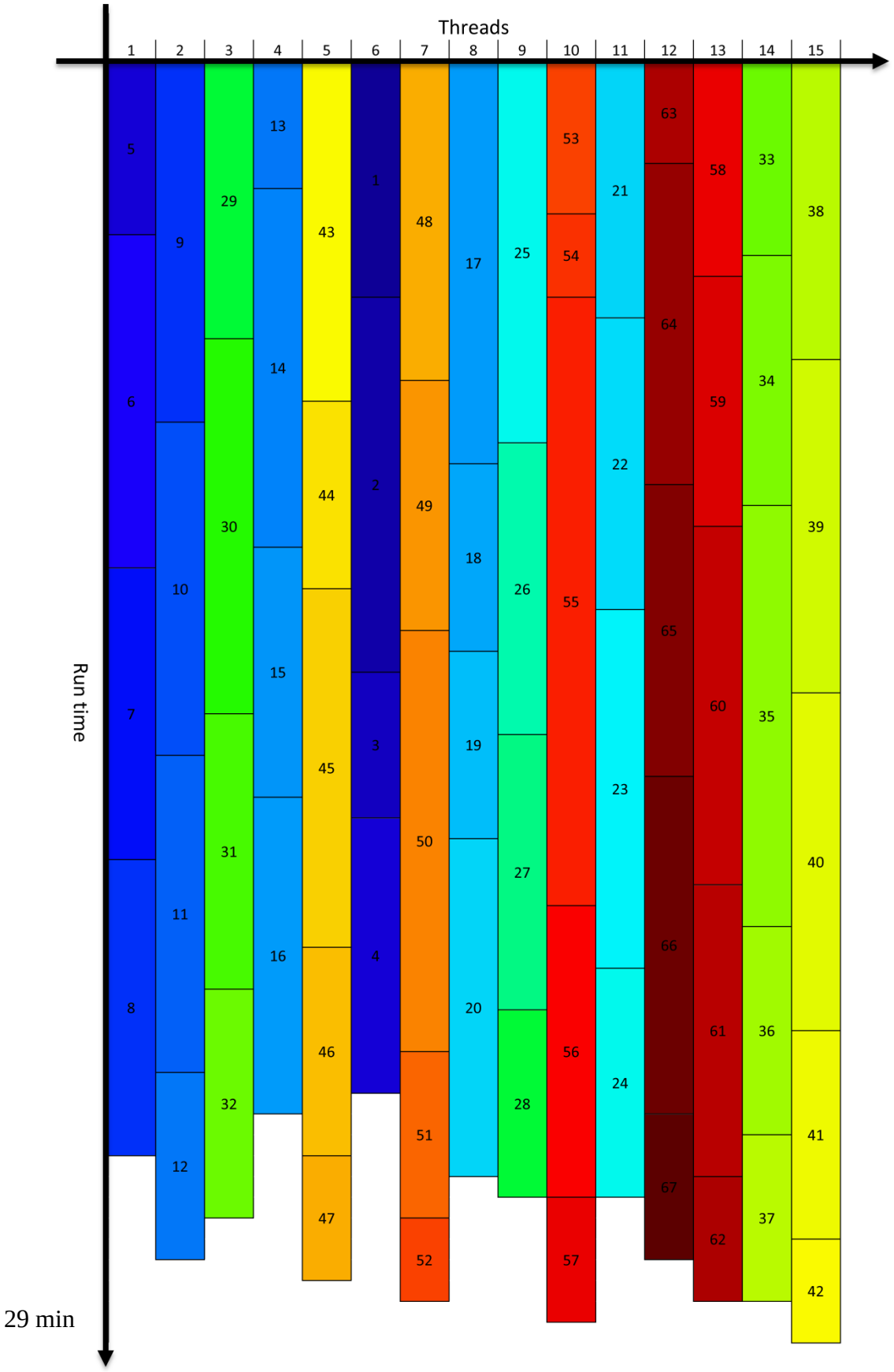

Figure 7. Execution of the parallel implementation regarding processing time. Each color represents the optimization process in a potential.

Finally, Figure 8 shows the results of simulations using the parameters found by the parallelized implementation. Compared with Figure 5, we clearly see that the fitting process was successful, since the simulation corresponds very well to the experimental data.

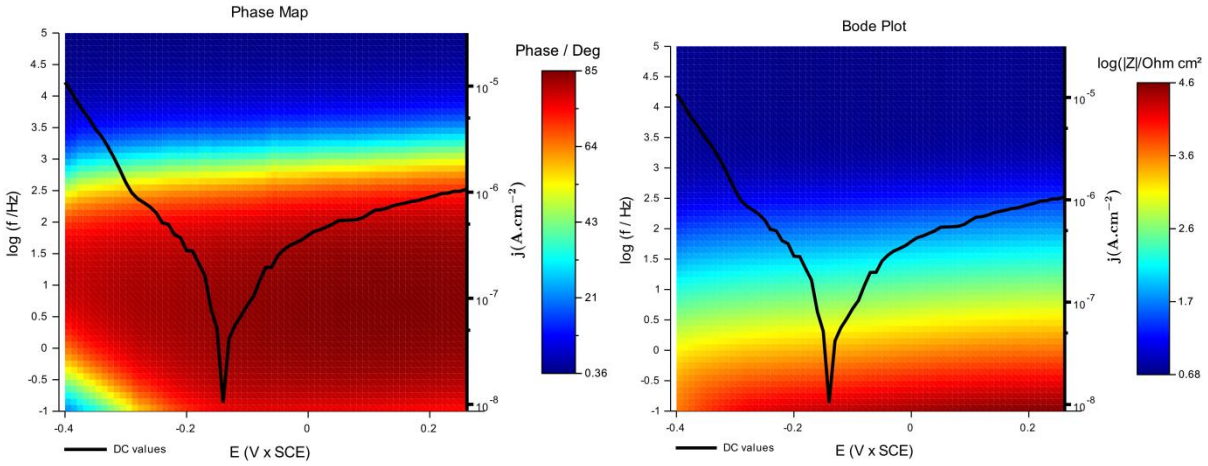

Figure 8. a) Phase angle and b) module of the simulated impedance, overlapped by the DC values.

5. CONCLUSIONS

On the present work, a parallelized version of the equivalent circuit fitting procedure for multi-potential impedance measures was designed and implemented. For this, the parallelization tools provided by Scilab were analyzed and the computing strategy using threads was chosen as the most appropriate for the problem at hand.

Results show that the gain achieved by using the new implementation exceeded expectations in terms of run time. In addition, all the strategies of the original fitting procedure, such as using the stationary response and the inclusion of the best individual in the population of the next potential, remained present in the new parallel version. As the optimization problem involved is highly complex, often ill-posed, the preservation of these aspects of the original process is essential to achieve good results.

REFERENCES

BASTOS, I. N.; CARVALHO, M. P. M.; FABBRI, R.; NOGUEIRA, R. P., Visualization of EIS at large potential range - new insights. **arXiv**: 1310.1629v2 [cond-mat.mtrl-sci], 2013.

BAUDIN, M.; LEDRU, S. Parallel Computing In Scilab, Scilab Online Documentation, 2013. Available at: <http://wiki.scilab.org/Documentation/ParallelComputingInScilab>, accessed

in: June, 2018.

HU, J-M.; ZHANG, J-Q.; CAO, C-N.; HSING, I-M. Kinetics investigation of H2/CO2 electrooxidation in PEFCs by the combined use of equivalent circuit fitting and mathematical modeling of the Faradaic impedance. **Electrochimica Acta**, v. 49, p. 5227-5234, 2004.

KAPPEL, M. A. A.; BASTOS, I. N.; LUZ FILHO, J. M. M.; FABBRI, R.; DOMINGOS, R. P. Mapas de impedância eletroquímica aplicados à corrosão, **Corrosão & Proteção**, ABRACO, Maio/Julho 2014, p. 15-19.

KAPPEL, M. A. A.; BASTOS, I. N.; DOMINGOS, R. P.; FABBRI, R. EIS-Mapper – Electrochemical Impedance Spectroscopy Mapper, Programa de Computador. Número do Registro: BR502013001515-9. Data de Registro: 17/03/2015, INPI - Instituto Nacional da Propriedade Industrial.

KAPPEL, M. A. A.; FABBRI, R.; DOMINGOS, R. P.; BASTOS, I. N. Novel electrochemical impedance simulation design via stochastic algorithms for fitting equivalent circuits., **Measurement**, v. 94, p. 344–354, 2016.

KAPPEL, M. A. A.; PEIXOTO, F.C.; PLATT, G. M.; DOMINGOS, R. P.; BASTOS, I. N. A study of equivalent electrical circuit fitting to electrochemical impedance using a stochastic method, **Applied Soft Computing**, v. 50, p. 183–193, 2017.

MATTOS, O. R.; BARCIA, O.E. The role of chloride and sulphate anions in the iron dissolution mechanism studied by impedance measurement, **Electrochimica Acta**, v. 35, p. 1003-1009, 1990.

MCKENNEY, P. E. **Is Parallel Programming Hard, And, If So, What Can You Do About It?** Linux Technology Center, IBM Beaverton, 2015.

ORAZEM, M. E.; TRIBOLLET, B. **Electrochemical Impedance Spectroscopy**, USA, John Wiley & Sons Inc., 2008.

SGURA, I.; BOZZINI, B. Numerical issues related to the modelling of electrochemical impedance data by non-linear least-squares, **International Journal of Non-Linear Mechanics**, v. 40, p. 557–570, 2005.

SHARIFI-ASL, S.; TAYLOR, M. L.; LU, Z.; ENGELHARDT, G. R.; KURSTEN, B.; MACDONALD, D. D. Modeling of the electrochemical impedance spectroscopic behavior of passive iron using a genetic algorithm approach, **Electrochimica Acta**, v. 102, p. 161– 173, 2013.

SILVERMAN, D. C.; Corrosion Prediction In Complex Environments Using Electrochemical Impedance Spectroscopy, **Electrochimica Acta**. Vol. 38. No. 14, pp. 2075-2078, 1993.

STORN, R.; PRICE, K. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, **Journal of Global Optimization**, v. 11, p. 341–359, 1997.