COBEM 2021 26th International Congress of Mechanical Engineering
November 22-26, 2021, Virtual Congress, Brazil

COB-2021-0454
A JULIA CODE FOR COMPUTING BASINS OF ATTRACTION OF ONE-DEGREE-OF-FREEDOM SYSTEMS

Rodrigo Provasi
provasi@usp.br
Vitor Schwenck Franco Maciel
vitor.maciel@usp.br
Guilherme Rosa Franzini
gfranzini@usp.br

Offshore Mechanics Laboratory (LMO), Escola Politécnica, University of São Paulo, Brazil

Abstract. *A plethora of phenomena on dynamics are governed by nonlinear equations. In some cases, the linearization of the mathematical model can lead to results without physical sense. In the last three decades, the sensitivity of the response of nonlinear equations with respect to changes in the initial conditions has received numerous research efforts. In this scenario, the definition of the basins of attraction is a key aspect that demands significant computational efforts. Although the use of high-level languages simplifies the implementation, it may not lead to acceptable performance, thus leaning development to lower-level languages. Recently, Julia language has been developed aiming to combine the benefits of high-level programming environments, especially ease of use, with the performance of lower-level languages. The present paper focuses on presenting a numerical code programmed in Julia for obtaining basins of attraction of a one degree-of-freedom oscillator. The performance of the code is compared with its C++ counterpart for classical Helmholtz-Duffing oscillator for different values of the forcing amplitude.*

Keywords: *Julia Language, C++ Language, Basin of Attraction, Nonlinear Dynamics*

1. INTRODUCTION

It is common for physical dynamical phenomena found in engineering applications to be governed by nonlinear equations. This characteristic usually brings forth a more complex analysis when compared to systems governed by linear dynamics. For instance, the presence of nonlinearities in the mathematical model may lead to the presence of multiple stability points and loss of significance of the standard (linear) modal expansion. Moreover, when dealing with nonlinear mechanical systems, analyses of the sensitivity of the response with respect to changes in the initial conditions may be of great importance. This latter aspect is the focus of the present contribution. Rega and Lenci (2015), Lenci and Rega (2019) and Rega and Settimi (2021) are examples of recent references that carry out a number of discussions regarding the sensitivity of nonlinear oscillators with respect to changes in the initial conditions.

Following Nayfeh and Balachandran (1995), the basin of attraction of a given attractor $\tilde{\boldsymbol{x}}$ can be defined as the region of the phase space $\mathcal{D} \subset \mathbb{R}^n$ such that if an initial condition $\boldsymbol{x_0} \in \mathcal{D}$, then $\tilde{\boldsymbol{x}}$ is achieved when $t \to \infty$. The evaluation of basins of attraction requires significant computational efforts since a large number of numerical integration of the equations of motion are necessary for sufficient accuracy. Usually, the cell mapping method or variations of it are adopted; see Hsu (1987) and van der Spek (1994). As can be seen in Belardinelli and Lenci (2016, 2017) and in Andonovski and Lenci (2020), parallel computation is also interesting aiming at enhancing the efficiency of the computations.

In addition to the calculation of the basins of attraction, the concept of integrity measurement (IM) is also relevant in global dynamics, provided it quantifies the erosion of the basins of attraction. Following Rega and Lenci (2005), three possible IM criteria are the global integrity measurement (GIM), the local integrity measurement (LIM) and the integrity factor (IF). GIM is the normalized magnitude of a certain basins with respect to the hipervolume (area, for 2D cases). Despite its simplicity for calculations (for example, taking the ratio between the number of cells associated pertaining to a certain basins and the total number of cells considered), GIM does not provide useful information regarding the erosion (and consequently, the sensitivity of the response with respect to the initial conditions). LIM can be defined as the normalized radius of the largest hipersphere (circle, for 2D cases) centered at the attractor that is fully inside its basin of attraction. Finally, IF is the normalized radius of the largest hipersphere (circle, for 2D cases) pertaining to a certain basins of attraction.

Indeed, computational efficiency is a keypoint in the development of codes for the numerical assessment of basins of attraction and the choice of the language is crucial. In this direction, one has a number of languages, including both

classical and well-established such as MatLab, Python, C and C++ and recently developed ones. This paper focuses on the use of Julia, which belongs to the latter class of languages.

As it can be seen in Julia Language (2021) documentation, the language tries to bridge two different situations in scientific computing: the day-to-day work and the high-performance final codes. In daily work, one is more interested in having all moving parts of the project working, disregarding the time it takes to achieve the solution. Once the solution is finally obtained, it then is converted into performance-driven codes, such as C++. By having an easy syntax, reminiscent from MatLab and Python while achieving performance comparable with C and C++, Julia language emerges as an interesting alternative. On the other hand, C++ language is a well established language and is an evolution from C with classes; see Hamilton (2021). It implements the orient-object paradigm and allows a great control of the resources available. This manifests especially in the memory control, where the programmer not only can but must control allocation and freeing of the memory. Since there is no such thing as garbage collection, some programs can show memory leaks if no attention is taken when allocating and freeing memory. On contrast, great control can significantly increase performance.

Comparing both languages, there is clearly advantages and disadvantages. While Julia is easy-to-use and manages memory usage, C++ provides more raw power at the cost of skilled programming. On the other hand, Julia can perform poorly in comparison while C++ can have associate memory leak problems.

In the present work, a code developed at Escola Politécnica, University of São Paulo in Julia environment named PoliBoA (Poli Basins of Attraction) is presented. Basins of attraction for a harmonically forced Helmholtz-Duffing oscillator are obtained for some values of forcing amplitude. The computational time needed for the Julia code is compared with that associated with C++ version of the code. The results show that the Julia code combines simplicity in the sintaxes with excellent performance, thus being an interesting alternative. At least to the best of the authors' knowledge, this is the first code for the evaluation of basins of attraction programmed in Julia.

The remainder of the paper is structured as follows. In the next section, the implemented algorithm and the investigated oscillator are described. Details concerning the implementation of the algorithm, in both languages, are presented in section 3, while the main results are shown and discussed in section 4. Finally, the main conclusions are gathered in section 5.

2. INVESTIGATED OSCILLATOR AND ALGORITHM DESCRIPTION

In this paper, we present results for the forced Helmholtz-Duffing oscillator given by Equation 1. Notice that Equation 1 is non-autonomous and periodic of period $\bar{T} = 2\pi/\Omega$. Hence, the stroboscopic Poincaré's section defined at $t = n\bar{T}, n = 0, 1, 2, \ldots$ is obvious for the analysis.

$$\ddot{y} + a\dot{y} + by + cy^2 + dy^3 = e\sin\Omega t \tag{1}$$

Equation 1 can be rewritten in the form of a system of first-order differential equations. For this, we define $x_1 = y$ and $x_2 = \dot{y}$, which leads to the following system:

$$\dot{x}_1 = x_2 \tag{2}$$

$$\dot{x}_2 = -ax_2 - bx_1 - cx_1^2 - dx_1^3 + e\sin\Omega t \tag{3}$$

The algorithm description is similar to other similar codes. Firstly, a certain region of the phase space $x_1 \times x_2$ that is of interest for the analysis is discretized in a number of regular cells. Each cell is properly numbered, such that each position of the mapped region is uniquely assigned to a particular cell. The first simulation considers the coordinates of the center of the first cell (first point of the trajectory) as initial conditions and lasts N_t excitation periods $\bar{T}$, leading to the point (x_1^t, x_2^t). This simulation includes the initial transitory, such that the second point of the trajectory is the number of the cell that contains (x_1^t, x_2^t). Subsequently, Equations 2 and 3 are integrated during $\bar{T}$ with initial condition (x_1^t, x_2^t), leading to the point (x_1^{t+1}, x_2^{t+1}). The number of the cell that contains (x_1^{t+1}, x_2^{t+1}) is appended to the trajectory. This step is repeated until a cell appears twice in the computed trajectory. At a first glance, this allows identifying the attractor using the stroboscopic Poincaré's section. For example, if the cell N^* appears sequentially, its center defines a period-1 attractor. On the other hand, if part of the computed trajectory has the following sequence (N^*,$\hat{N}$,N^*), a period-2 attractor defined by the center of the cells $(\hat{N}, N^*)$ is identified. Once the cells corresponding to the first attractor are identified, they are labeled by number 2, and the remaining cells pertaining to the trajectory (part of the basin of attraction) are labeled by 1. Notice that, contrary to the method described in van der Spek (1994), the mathematical model is integrated during $\bar{T}$ using as initial condition the last point computed in the trajectory and not the center of the corresponding cell. Hence, the adopted algorithm does not "break" the trajectory.

The trajectory that starts at the center of the cell number 2 is computed following the steps above described. If a cell already mapped appears in the new trajectory, the number of the basins associated with the repeat cell is assigned to all

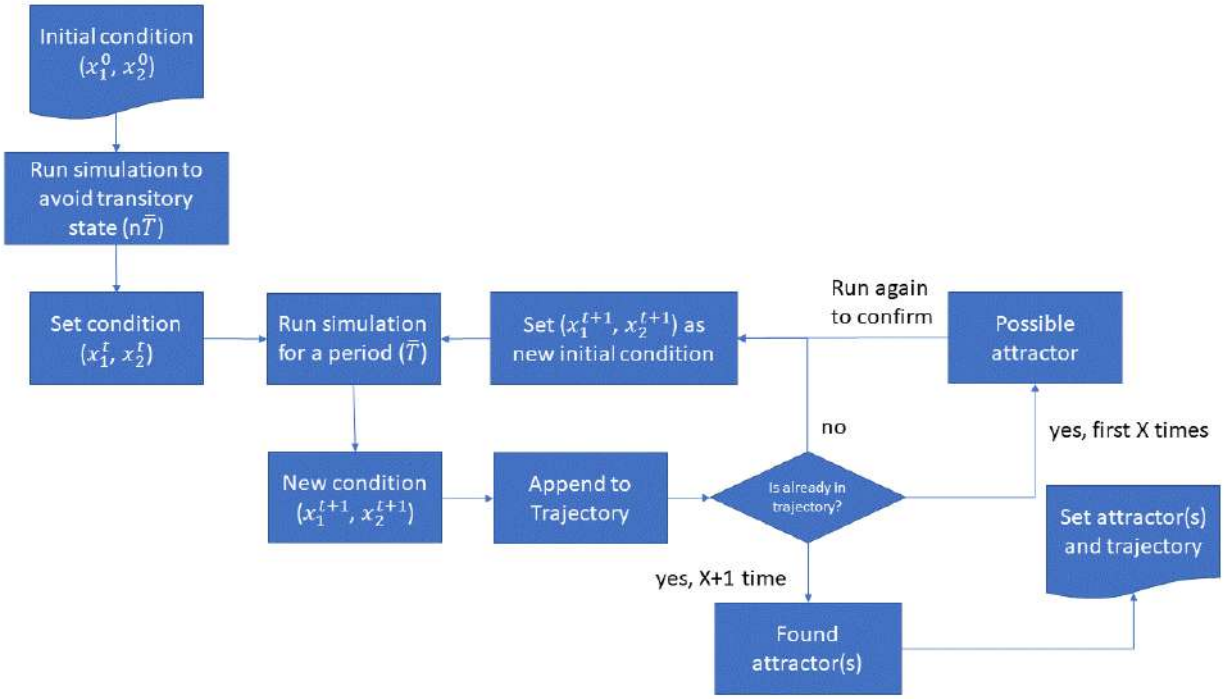


Figure 1: Trajectory definition

points of the new trajectory, boosting the calculations. On the other hand, 3 is assigned to the points of the new trajectory and the cells associated with the new attractor are labeled as 4. The trajectory algorithm can be seen in Figure 1.

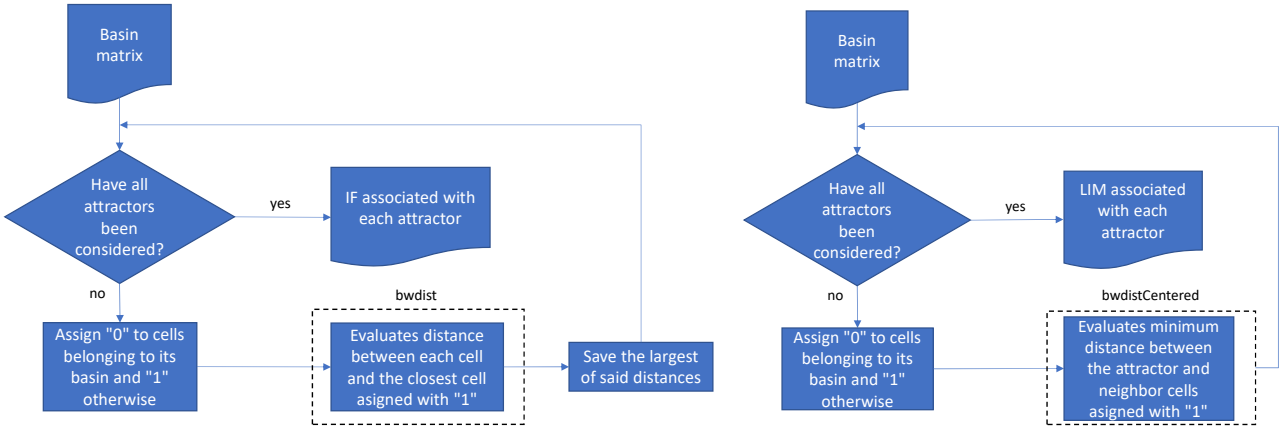
Two more aspects regarding the algorithm must be emphasized. The first one is the use of an extended region, defined by the user, around the investigated region of the phase space. If, in a certain part of the algorithm, the trajectory falls in this extended region, we continue to run up to certain number of periods (also defined by the user). If the trajectory return to the mapped region, the algorithm continues as described. If the prescribed number of periods is achieved and the trajectory does not return to the mapped region, we assign -1 to the trajectory, indicating that the attractor is divergent or that the attractor does not belong to the mapped region. In addition, -1 is also assigned if the trajectory escapes from the extended region. The second aspect is related to an additional verification, following the detection of an attractor. If the algorithm finds an attractor, then the mapping process is continued for a certain number of steps (defined by the user) to assure the identification of the attractor.

At the end of the algorithm, a vector of length equal to the number of cells and composed of even numbers that represent the different attractors, odd numbers corresponding to their basins of attraction is obtained and -1 if the corresponding initial conditions leads to an attractor outside the mapped region. Finally, this vector is transformed into a matrix and plotted in as a colormap, in which the basin of attraction of each attractor is assigned to a different color.

The final detailed aspect of the analysis is the evaluation of the LIM and of the IF from the basins of attraction obtained using the algorithm described above. Considering first the IF, the first step is to create a binary matrix from the basins of attraction while using the following conditions: for a given attractor, assign 0 to all cells belonging to its basin and 1 otherwise. Now considering this binary matrix, another matrix can be obtained by imposing that each of its entries contains the minimum distance, in the binary matrix, between each cell and the closest cell to which 1 is assigned. Finally, the largest of said distances is saved and the next attractor is considered, which is done until all attractors have been analyzed. Subsequently, the IF for each attractor is plotted by also saving the cell from which it is obtained. The algorithm described in this paragraph is presented in Figure 2 (a).

Now considering the evaluation of the LIM for each of the attractors, the first steps of the algorithm are identical as when evaluating the IF. Hence, from the basins of attraction, binary matrices are constructed for each of its attractors. The main difference between the algorithms is that, upon possession of the binary matrix, the LIM for that attractor can be directly obtained by evaluating the normalized distance between its cell and the closest one with 1 assigned to it (and thus the normalized distance between it and the boundary of its basin is obtained). The algorithm described in this paragraph is presented in Figure 2 (b).

The function referred to as “bwdist”, in Figure 2 (a), evaluates the Euclidean distance transform of the binary matrix given as input. The function referred to as “bwdistCentered”, in Figure 2 (b), in turn, evaluates the distance between the attractor’s cell and the nearest cell assigned with 1. Since there is no native function in Julia which can carry out such computations, this function is also conceived by the authors. Essentially, searches for cells with 1 assigned are made by



(a) IF evaluation. (b) LIM evaluation.

Figure 2: Flowchart for obtaining the integrity measures of the basins of attraction.

considering squared shaped "rings" around a given cell in the binary matrix.

3. IMPLEMENTATION

The code implementation is similar in both languages, differing due to the capabilities and resources of each language. Figure 3 shows the basic workflow for the implementation. Since the method is the same, there is not much difference in how the code works, other than the implementation. One important point that is worth mentioning is that no multi-threading optimization is done, thus only sequential, single core is implemented.

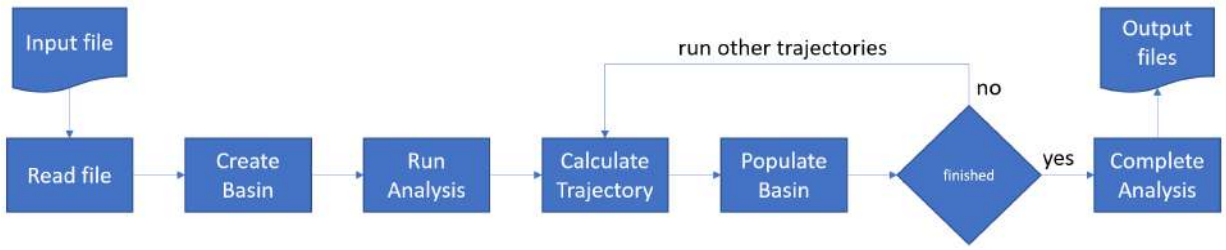


Figure 3: Flowchart for Basin of attraction implementation

C++ implementation follows the usual object-oriented paradigm. Different from many languages, the use of external libraries is not straightforward and demands an installation and posterior linking to the code. For the Basin of Attraction code, the odeint library was used (Ahnert and Mulansky (2021)). The structure consists in defining classes and methods for the basin itself, the regions (center and extend ones) and the analysis. For internal storage, the Standard Template Libraries (*STL*) vector class was used, since it provides methods for searching and item count.

Julia implementation uses the main characteristics of the language: optional typing and multiple dispatch. The last one is a core feature, consisting in defining functions that work differently depending on the parameters, which should be typed. There is no object-oriented structure in the language, only *structs*. These *structs* can be marked as mutable to enable changes in their fields.

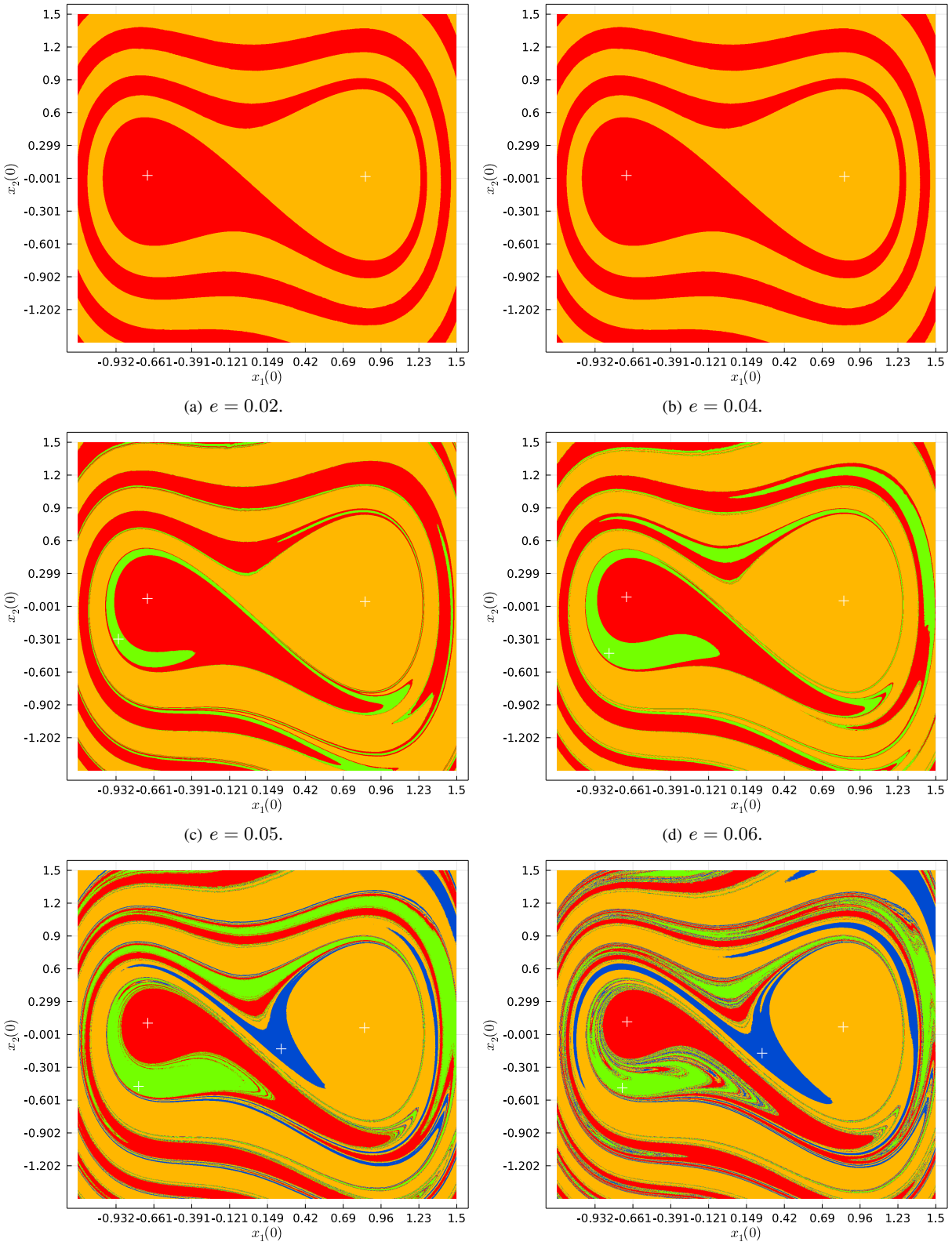
4. RESULTS AND DISCUSSION

For the sake of illustration of the results, only the forcing amplitude e is varied while the remainder parameters of Equation 3 are $a = 0.10$, $b = 1.20$, $c = -0.30$, $d = 2$ and $\Omega = 1.17$. This section is organized in two subsections. Subsection 4.1 presents examples of basins of attraction and integrity measurements obtained for different values of forcing amplitude e using the Julia version of PoliBoA. The performance of the Julia code is compared with its C++ counterpart in subsection 4.2

4.1 Basins of attraction and integrity measurements

Figure 4 shows the results for different values of e using a $1{,}200 \times 1{,}200$ grid of initial conditions for discretizing the region of the phase space $-1.2 \leq x_1 \leq 1.5 \times -1.5 \leq x_2 \leq 1.5$. Throughout the paper, white crosses indicate the

obtained attractors. Considering $e = 0.02$ and $e = 0.04$ (see Figures 4(a) and 4(b)), two attractors are identified and their basins of attraction are compact (i.e., no important erosion is observed).



(a) $e = 0.02$. (b) $e = 0.04$.

(c) $e = 0.05$. (d) $e = 0.06$.

(e) $e = 0.07$. (f) $e = 0.077$.

Figure 4: Basins of attraction for different values of forcing amplitude. A 1,200 $\times$ 1,200 grid of initial conditions is employed.

When $e = 0.05$, a third attractor appears (see the region shadded in green in Figure 4(c)). The basins of attraction of

this third attractor is enlarged when $e = 0.06$ is considered (see Figure 4(d)). If the amplitude of the external forcing is $e = 0.07$, a fourth attractor whose basins of attraction is assigned in blue appears; Figure 4(e). As can be seen in Figure 4(f), an increase in the amplitude of the external forcing to $e = 0.077$ increases the erosions of the basins of attraction. Such erosion is pronounced in the region $-0.12 \leq x_1(0) \leq 0.149 \times -0.601 \leq x_2(0) \leq -0.301$, where small and unavoidable distubarances in the initial conditions may lead to a different attractor.

Regarding the computational performance, Julia took around 3 minutes to fully simulate the Helmoltz-Duffing oscillator for the given parameters while using a 500×500 basin discretization. The same code using C++ took around 15 minutes. This difference seems to remain a constant as the same problem was simulated using a finer discretization and the simulation times were 50 minutes for Julia and 150 minutes for C++.

The differences can be summarized and explained in a simple way: Julia provides most of the necessary tools native to the language and implements them in a much more optimized way than a programmer can achieve without investing a great deal of energy. Examples in the code is the native vector support, including search and multiple values appearances count. Even though C++ can provide this using the standard template library (STL) vector, there is an overheading in code. Slicing is another feature that hugely accelerates the process and there is no equivalent in C++.

Of course, both implementations can be enhanced and optimized. For the current implementations are within the authors knowledge and abilities when considering single-thread oriented programming. It must be noted also that the selection of libraries used in C++ can also enhance the performance and, most importantly, switching to a multi-thread paradigm.

In the multi-threading scenario the differences may tip in favor of C++ since the language has not only great established library (such as *MPI* and *OpenMP*) but also support to different technologies (such as the GPU-bound tools). It remains to see how Julia fair against such plethora of libraries and to determine its performance.

4.2 Performance: Julia *vs* C++ implementations

The code performance when comparing C++ to Julia, in our implementation favors Julia. Even though the memory usage is smaller in C++, being practically negligible, the computing times display a great reduction. For the tests a i7-8700 processor machine with 32 *GB* of RAM memory was employed. It is worth noting that, even though this paper is focused in a single thread implementation, a multi-thread scenario could heavily favor C++.

The results shows this difference consistently: using a 500×500 grid yield a 3-minute simulation in Julia *vs* a 20-minute simulation in C++ while for a $1{,}000 \times 1{,}000$ basin Julia took 14 minutes while C++ took 106 minutes. For both languages it was decided to study how the basin size (the total number of cells) influences in computational times, for the two extreme cases shown in section 3. Figure 5 shows different basins sizes and the computational time in second it took to run for two different amplitudes (0.02 and 0.077) using Julia. Figure 5 plots the information summarized in Table 1, displaying a quasi-linear relation between the basin size and time spent.

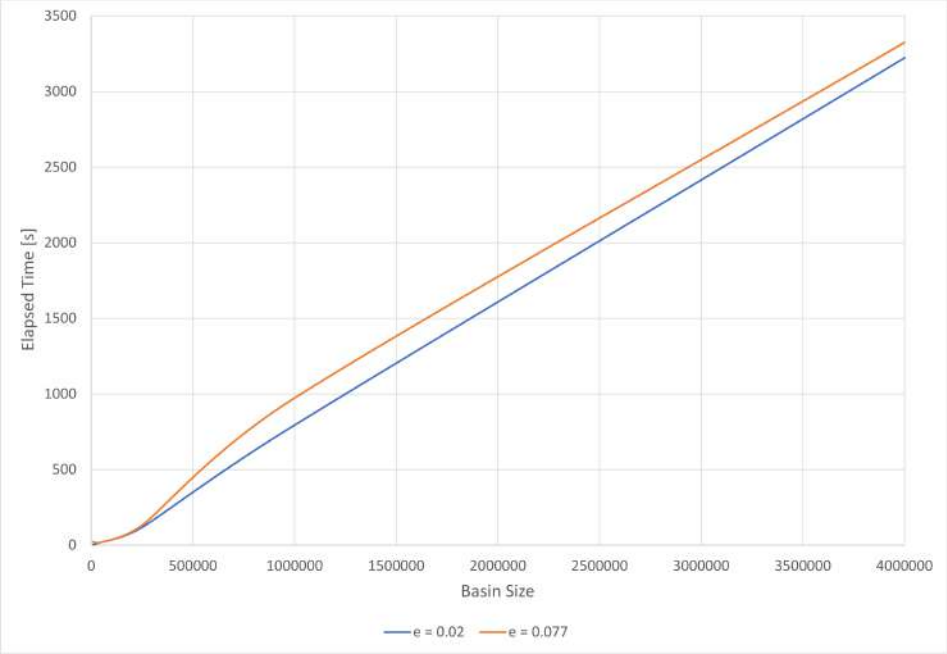


Figure 5: Basin total elements *vs* time taken in seconds. Performed using Julia.

For C++, the same tests were performed and the results are summarized in Table 2 and shown in Figure 6.

Table 1: Basins divisions, total elements and time taken in seconds. Performed using Julia.

Number of divisions	Basin Size	$e = 0.02$ Elapsed Time[s]	$e = 0.077$ Elapsed Time[s]
100	10,000	4	23
200	40,000	16	17
500	250,000	11	131
1,000	1,000,000	794	974
2,000	4,000,000	3,222	3,325

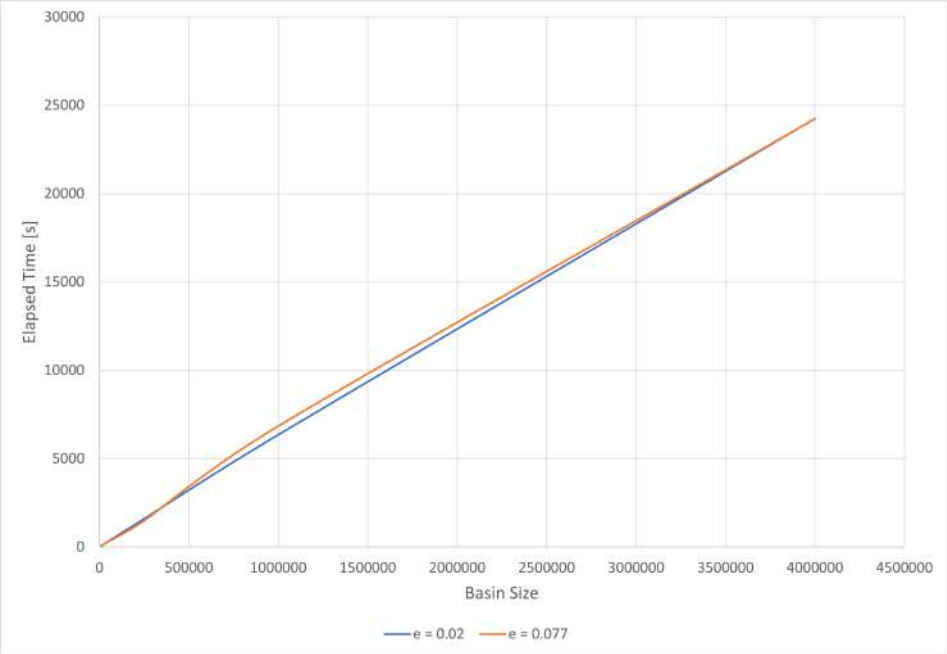


Figure 6: Basin total elements *vs* time taken in seconds. Performed using C++.

Figures 7 and 8 show the comparison between Julia and C++ for the amplitudes of 0.02 and 0.077 respectively.

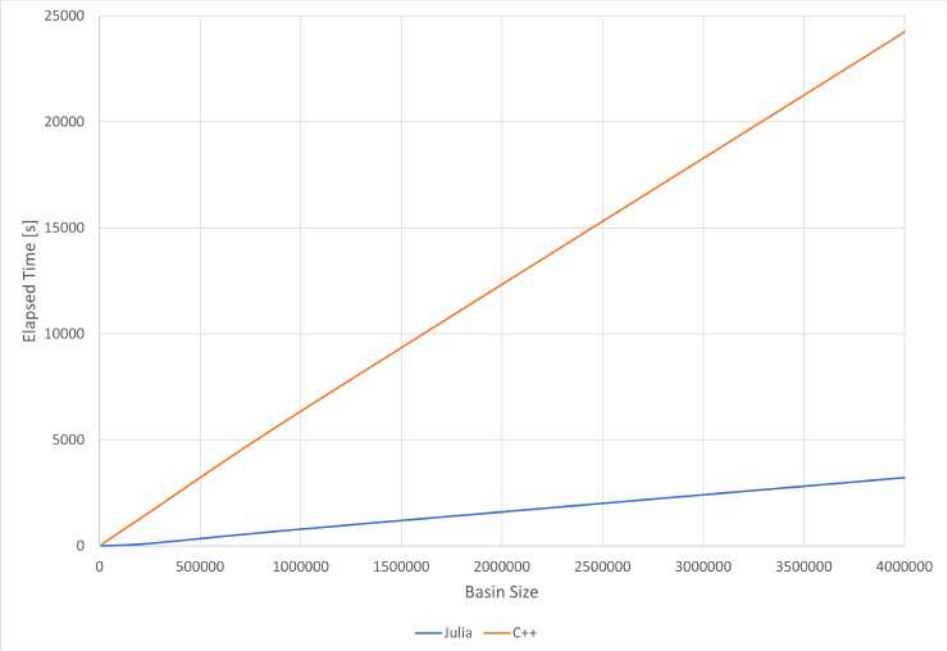


Figure 7: Julia *vs* C++ for the Helmholtz-Duffing oscillator using $e = 0.02$

It was selected these two different basins parameters since they lead to different number of attractors, (see Figures 4(a) and 4(f)) and some time difference is expected. As it can be observed, the difference is small and it maintains with the

Table 2: Basins divisions, total elements and time taken in seconds. Performed using C++.

Number of divisions	Basin Size	$e = 0.02$ Elapsed Time[s]	$e = 0.077$ Elapsed Time[s]
100	$10,000$	62	71
200	$40,000$	265	247
500	$250,000$	1,602	1,489
1,000	1,000,000	6,358	6,845
2,000	4,000,000	24,249	24,236

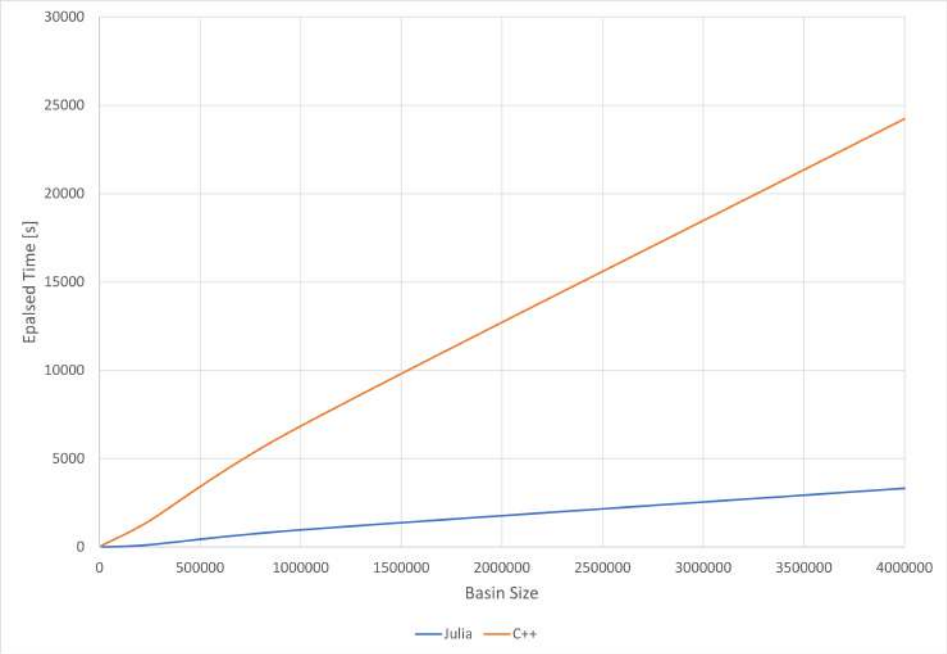


Figure 8: Julia *vs* C++ for the Helmholtz-Duffing oscillator using $e = 0.077$

increase of the basin size. This is due to the nature of the trajectory determination algorithm. It also explains the almost linear behavior instead of the expected quadratic behavior. Since when a already known cell is encountered, *i.e.*, the cell already contains a previously established trajectory, the algorithm exists and sets all prior points in the trajectory to the same *ID*. With a small number of attractors, the chances to step a period in the algorithm and obtain a cell with values is high enough that reduces considerably the computational efforts.

The characteristics of the algorithm are also something to be taken into consideration when multi-threading is used. Since simultaneous data access cannot happen among threads, some variables should be dealt within each thread and later on synchronized. If one decides to take that route and split a basin within some threads and do the calculation independently, the final times can be comparable with the original single thread one, since the previous determined trajectories would not be available and the full trajectory must be calculated.

5. CONCLUSIONS

This paper describes a code programmed in Julia environment for obtaining the basins of attraction of a one-degree-of-freedom non-linear and harmonically forced system. The code also calculates some measurements of integrity of the basins of attraction. The main aspects of the algorithm are presented and examples of results with a Helmholtz-Duffing oscillation are discussed for different values of forcing amplitude.

The implementation is done in two different languages (Julia and C++) aiming at comparing their performances. As it can be seen from the results, Julia performs remarkably well and is even faster when compared to the C++ implementation done by the authors. Thus, it is a great option for scientific computing problems such as the one presented.

It is worth to notice that no multi-threading implementations were done in the present article. C++ offers a great range of libraries to create such programs. This can be a turning point for performance if Julia performs as well as in the single-thread scenario. Future works regarding multi-thread comparison are especially valuable as the basin of attraction problem scales quickly with the number of divisions for each interest variable and more so with the number of these variables. Other suitable programming languages, such as Rust are going to be explored in future developments, especially in the multi-threading scenario.

6. ACKNOWLEDGEMENTS

The authors thank Profs. Stefano Lenci and Pierpaolo Belardinelli (Università Politecnica delle Marche - UnivPM) and Giuseppe Rega (La Sapienza Università di Roma) for the discussions regarding global dynamics. The third author is grateful to the São Paulo Research Foundation (FAPESP) for supporting his short-term period as a visiting researcher at UnivPM during December/2019 and February/2020 (grant 2019/15046-2) and the Brazilian Scientific Research Council (CNPq) for the grant 305945/2020-3.

7. REFERENCES

Ahnert, K. and Mulansky, M., 2021. "Chapter 1. boost.numeric.odeint". Available at https://www.boost.org/doc/libs/1_66_0/$libs/numeric/odeint/doc/html/index.html$.

Andonovski, N. and Lenci, S., 2020. "Six-dimensional basins of attraction computation on small clusters with semi-parallelized SCM method". *International Journal of Dynamics and Control*, Vol. 8, No. 2, pp. 436–447. ISSN 21952698. doi:10.1007/s40435-019-00557-2. URL https://doi.org/10.1007/s40435-019-00557-2.

Belardinelli, P. and Lenci, S., 2016. "A first parallel programming approach in basins of attraction computation". *International Journal of Non-Linear Mechanics*, Vol. 80, pp. 76–81. ISSN 00207462. doi:10.1016/j.ijnonlinmec.2015.10.016. URL http://dx.doi.org/10.1016/j.ijnonlinmec.2015.10.016.

Belardinelli, P. and Lenci, S., 2017. "Improving the Global Analysis of Mechanical Systems via Parallel Computation of Basins of Attraction". *Procedia IUTAM*, Vol. 22, No. 0, pp. 192–199. ISSN 22109838. doi:10.1016/j.piutam.2017.08.028. URL http://dx.doi.org/10.1016/j.piutam.2017.08.028.

Hamilton, N., 2021. "The a-z of programming languages: C++". Available at https://www.pcworld.idg.com.au/article/250514/a-z$_{p}rogramming_{l}anguages_{c/}$.

Hsu, C.S., 1987. *Cell-to-Cell Mapping - a method of global analysis for nonlinear systems*. Springer Verlag.

Julia Language, 2021. "Julia 1.6 documentation". Available at https://docs.julialang.org/en/v1/.

Lenci, S. and Rega, G., 2019. *Global Nonlinear Dynamics for Engineering Design and System Safety*, Vol. 588. ISBN 978-3-319-99709-4. doi:10.1007/978-3-319-99710-0. URL http://link.springer.com/10.1007/978-3-319-99710-0.

Nayfeh, A.H. and Balachandran, B., 1995. *Applied Nonlinear Dynamics - Analytical, Computational and Experimental Methods*. John Wiley Sons.

Rega, G. and Lenci, S., 2005. "Identifying, evaluating, and controlling dynamical integrity measures in non-linear mechanical oscillators". *Nonlinear Analysis, Theory, Methods and Applications*, Vol. 63, No. 5-7, pp. 902–914. ISSN 0362546X. doi:10.1016/j.na.2005.01.084.

Rega, G. and Lenci, S., 2015. "A Global Dynamics Perspective for System Safety From Macro - to Nanomechanics: Analysis, Control, and Design Engineering". *Applied Mechanics Review*, Vol. 67, pp. 50802–50819. doi:http://dx.doi.org/10.1115/1.4031705.

Rega, G. and Settimi, V., 2021. *Global dynamics perspective on macro- to nano-mechanics*, Vol. 0123456789. Springer Netherlands. ISBN 1107102006. doi:10.1007/s11071-020-06198-x. URL https://doi.org/10.1007/s11071-020-06198-x.

van der Spek, J., 1994. *Cell mapping methods : modifications and extensions*. Phd thesis, Eindhoven University of Technology. doi:10.6100/IR411481. URL https://research.tue.nl/en/publications/cell-mapping-methods-modifications-and-extensions.

8. RESPONSIBILITY NOTICE

The authors are solely responsible for the printed material included in this paper.