Tutorial for DynIn toolbox v1.0.1, a toolbox for the rapid computation of the dynamical integrity measure.

Giuseppe Habib

January 12, 2023

Department of Applied Mechanics, Faculty of Mechanical Engineering, MTA-BME Lendület "Momentum" Global Dynamics Research Group, Budapest University of Technology and Economics, Budapest, Hungary.
E-mail: habib@mm.bme.hu

Contents

Previous versions

- v1.0.0

New features

New feature:

- Possibility of a checking if an identified periodic solution is too small and, in that case, ignore it. If check_min_periodic is set to 1, the this option is activated. If the algorithm identify a periodic solution for which the maximal distance between two points of the solution is smaller than min_periodic_radius, the periodic solution is ignored. This option help to avoid artifact periodic solutions around a slowly converging stable focus.

Two additional input arguments are related to this new feature, namely min_periodic_radius and check_min_periodic, and the new function maximal_distance, as explained below.

Bugs fixed:

- Removed the possibility of generating initial conditions out of the phase space for type_x0=2
- Warning instead of Error for system of dimension larger than 8
- arrays cell_c, cell_d and cell_dout reduced to a single column array, as the second column was useless.
- eliminated check on coherence between dim and length of xe, as it is automatically verified.

1 Introduction

This document serves as a short tutorial for the MATLAB Dynamical Integrity (DynIn) Toolbox. Explanations about the toolbox's main idea, scope, and role in the existing scientific literature are explained in [1]. I hope you will find the toolbox useful, and I will be glad to hear your feedback. Please feel free to write me about any technical issue you might experience or if clarifications are needed. If you use the toolbox for your research, I would be glad if you could please cite my paper [1].

2 Installation

To install DynIn Toolbox, you need to open the main folder of the program (DynIn_v1.0.X) in MATLAB and run the command install in the Command Window. The installation will only add to your MATLAB path the folder functions, required to use the toolbox. The function uninstall will do exactly the opposite. If you already installed a previous version of DynIn, we suggest to uninstall that one before installing the new version to avoid conflicting versions of the same function.

Some illustrative examples are provided to help the user understand the toolbox. To use them, you should set as your current folder the one of the example you are interested in. Then, open the Live Script file Starter.mlx where it is explained how to proceed further. More information is provided below.

3 What the **DynIn** toolbox does and will (hopefully) do in the future

DynIn toolbox is conceived to rapidly compute the local integrity measure (LIM) of a steady state of a dynamical system. The LIM is the radius of the largest hypersphere entirely included in the basin of attraction of a steady state and centered at the solution (the definition in the literature is still unclear regarding steady-state solutions different from a fixed point). For detailed explanations please look at [1, 2].

The current version (v1.0) of DynIn is able to handle only this case:

- LIM of equilibrium points of ordinary differential equations (ODEs)

However, we plan to extend the algorithm to the following cases in the near future (in order of probable implementation):

- equilibrium points of difference equations (maps)
- periodic solutions of ODEs
- periodic solutions of maps
- equilibrium solutions of delayed differential equations (DDEs)
- periodic solutions of DDEs.

Besides, currently DynIn has only a command line version. In the future, we plan to develop a GUI version as well.

Currently, DynIn *cannot* compute basins of attraction; however, we might include this option in a future toolbox version.

4 How to use it

The main function of the toolbox is compute_LIM_FP, where FP stands for fixed point. This function requires a minimal number of input arguments, which are:

- xe: is a single line array containing the coordinates of the equilibrium point.
- par: is an array containing the parameter values of the system.

Apart from that, the algorithm requires a file providing the ODE of the system. This must be included in a file called sistema.m, which should be placed in MATLAB's current folder. Please refer to the examples provided to understand the structure of the file sistema.m.

The function compute_LIM_FP provides numerous outputs and has many optional inputs. For a full list of them, we invite you to type compute_LIM_FP and click F1, right-click on the name of the function and click then on help, or open the compute_LIM_FP file and look at the first lines. The main output of the function is the finally estimated LIM value.

To compute the LIM of your system, after setting the xe and par, you should type:

```
compute_LIM_FP(xe,par)
```

which will provide the estimated LIM value. Additional inputs can be provided in the following way. If, for instance, you want to specify the number of steps of the iteration as 200 and the discretization of the phase space in 1001 cells in each direction, you should type:

```
compute_LIM_FP(xe,par,'number_of_steps',200,'discr',1001)
```

The order of the optional inputs is irrelevant, while the order of xe and par cannot be changed.

The output parameters are provided in the following order:

```
[R_final, R,time_sim, time_x0, time_steps, time_steps_sim, total_time, distance, vc, vd, vdout, tipo, other_solutions_p, num_points_inside, x0]
```

The meaning of each parameter is explained in the help of the function compute_LIM_FP.

4.1 Most commonly required optional input arguments

Of all optional input arguments, the ones that should be used in most cases are:

- weight: this is a line vector that scales distances in the various directions of the phase space; this is required to define a hypersphere since the phase space usually is not isotropic. It can be either defined through practical considerations, such as the expected value of a perturbation in each given direction, using energetic considerations, or by transforming the system in its Jordan normal form. For further explanations, we address the reader to Sections 3.2.1 and 4 of [1].
- spaceboundary: this line vector defines the boundaries of the phase space of interest. The equilibrium point must be within these boundaries.
- type_x0: this parameter can assume the values 1, 2 or 3. It defines the algorithm used for selecting the initial conditions of each simulation. In particular:
  1. The system looks for the farthest point from all other tracked points within the hypersphere of convergence using a genetic algorithm. This method is explained in [1].
  2. A bisection method is used for finding initial conditions as close as possible to the basin boundary. The procedure is combined with a random scheme, to avoid that some regions of the phase space are completely overlooked.
  3. Initial conditions are chosen randomly within the hypersphere of convergence

  The default value of type_x0 is 1.

4.2 Good to know

The optional input arguments related to the visualization of the results are:

- plot_results; 0: no figure is shown during the computation (fastest computation), 1: figures are generated and updated at each iteration.
- plot_results_final; 0: no figure is produced at the end of the computation, 1: figures illustrating the results are shown when the computation ends.
- var1: variable plotted in the x-axis of each generated figure.
- var2: variable plotted in the y-axis of each generated figure.

Among the input arguments which might increase accuracy (note that higher accuracy usually implies longer computational time), there are:

- number_of_steps: number of iterations performed by the algorithm.
- reltol: relative tolerance of the simulations (ode45).
- abstol: absolute tolerance of the simulations (ode45).
- discr: number of cells in each direction of the phase space for its discretization (see Section 3.1.1 of [1]).
- rep_fix_point: number of required repetition in a cell before it is assumed that a trajectory reached a so far unknown equilibrium point (see Section 3.1.1 of [1]).
- rep_periodic: number of required repetition in a cell before it is assumed that a trajectory reached so far unknown periodic solution (see Section 3.1.1 of [1]).
- num_of_cell_around_equilibrium: the algorithm initially marks only the cell where the desired equilibrium position lies as attractive. However, attraction in its proximity might be slow. To speed up the computation, a hypercube of cells surrounding the equilibrium cell might also be directly marked as attractive. This hypercube's size (in number of cells per size length) is given by num_of_cell_around_equilibrium, which should be an odd positive integer number. Its default value is 3.

In the case of initial conditions chosen as the most remote point in the hypersphere of convergence (type_x0=1), some additional input arguments, mainly related to the genetic algorithm, can be adjusted:

- divR: a parameter to eliminate points very close to each other and speed up computation. The larger this number, the more points are kept inside the hypersphere of convergence as individual points.
- num_gen: number of generations of the genetic algorithm.
- num_almost_clones: number of near clones of the best individual of the previous generation.

- selected_opt: number of selected best individuals of each generation.
- newcomers: new randomly chosen individual of each generation.
- num_crossover: number of individuals generated as a crossover between the fittest individuals of the previous generation.

In the case of initial conditions chosen through the bisection method (type_x0=2), an additional input argument can be adjusted:

- bis_iter_max: marks the maximal number of consecutive steps of the bisection method for selecting initial conditions before moving to a new randomly chosen point.

Other adjustable parameters are:

- tfinal: maximal final time of each simulation (note that simulations should be classified *before* this time is reached. So selecting a very large tfinal might be completely irrelevant for the computation; conversely, having too short tfinal might cut simulations before they are classified).
- Ts: time step between two subsequent points in a trajectory.
- automatic: can be set as 0 or 1. 0: in case of an undefined time series (once the maximal allowed time per simulation tfinal is reached), the computation is paused, and it is asked to the user to decide about its convergence based on a plot. 1: if a time series is not classified in the available time for the simulation, the algorithm automatically considers it diverging.
- num_fig: number of the figure where the trajectories are plotted.
- check_min_periodic: if set to 1, the algorithm perform a check on identified periodic solutions. If the maximal distance between two points of the periodic solution is larger than min_periodic_radius the periodic solution is ignored. This option help to avoid artifact periodic solutions around a slowly converging stable focus.
- min_periodic_radius: minimal distance between points of a periodic solution such that the solution is taken into account. Works if check_min_periodic is set to 1.

We remark that most of the toolbox's code lines include a comment. We hope this will enable users to smoothly utilize the program, avoiding the "black-box" feeling, which is not particularly appreciated, especially in academia.

4.3 Fixed points as periodic solutions

Although the algorithm can handle only fixed points, the dynamical integrity of periodic solutions is often studied through Poincaré maps. This is particularly easy to do in the case of forced vibrations if the period of oscillation equals the excitation period. In this case, by imposing Ts equal to the period of oscillation, the periodic solution is reduced to a fixed point, and the algorithm will study its dynamical integrity, restricted to the Poincaré map in which it lies.

4.4 Other special cases

In some cases it is not convenient to simulate the motion of the system with a simple ode45 function, for instance in the case of stiff problems. Similarly, some systems are better simulated through a series of ode45 commands separated by so-called events (for example, in the case of impacts). In those case, the program can be modified by editing the command line

```
[t,x]=ode45 (@(t,xt) sistema(t,xt,par), [0 Ts], x0, option); % single simulation of length Ts (one time steps)
```

of the function simulation_DI; the option, where an event function can be inserted, is defined in the command line

```
option=odeset('RelTol', reltol, 'AbsTol', abstol); % tollerances for the simulations
```

of the function compute_LIM_FP. The important thing is that that line generates an array x whose last raw contains the state variables of the system. This information is later processed to characterize the time series and, in a while loop, it is used to mark the initial condition of the following time step simulation. It is not particulalry relevant how this information is obtained. We also remark that this is the only command line calling the equations of motion contained in sistema. Therefore, if this line is changed, the file sistema must be also changed accordingly.

4.5 Typical problems

The most typical problems that can be encountered with DynIn are slow convergence towards the expected LIM value, identification of nonexistent periodic solutions, and the missed characterization of trajectories before the final allowed time (tfinal) is reached.

In the case of slow convergence, we suggest using the selection of initial conditions through the bisection method (type_x0=2). Although this choice might overlook some regions of the phase space, in general, it can produce more accurate results in shorter time. Reducing the resolution of the phase place discretization (discr), the computational time reduces almost linearly (see Fig. 5 of [1]). Computational time can also be reduced by preventing the generation of figures (set plot_results to 0) or increasing the relative and absolute tolerance of the Runge-Kutta scheme used for the simulations (reltol and abstol, respectively). If initial solutions are chosen as the farthest point in the hypersphere of convergence (type_x0=1), the algorithm exploits a genetic algorithm for choosing them; therefore, reducing the number of generations (num_gen) or the number of individuals per generation (num_almost_clones, newcomers and num_crossover) will reduce the time required for choosing the initial conditions of each simulation; this might speed up the computation, especially for high iteration numbers of the algorithm (see the red line in Figs. 4c, 7d and 11d in [1]).

Systems with very low damping might undergo spiraling motions with very slowly varying amplitude. This can induce the algorithm to confuse a non-stationary spiraling motion with a stationary periodic motion, in particular in the vicinity of a stable equilibrium. If this happens, it is convenient to thicken the cell discretization of the phase space, i.e., increase the discr value. Similarly, the phase space of interest can be reduced (spaceboundary), which will reduce the dimension of the cells if discr is unchanged. Alternatively, the rep_periodic value can be increased, which requires that a trajectory passes more times through the same cells before it is assumed that the trajectory is periodic. Any of these solutions increase the time required for characterizing the trajectories, which might be larger than the available time for each simulation tfinal. Accordingly, it might be convenient to increase its value. We notice that this problem is typical also for the cell-mapping method [3].

In contrast, if the algorithm is unable to characterize trajectories before tfinal is reached, either its values should be increased, or the number of cells dividing the phase space can be reduced (discr). Similarly, rep_periodic or rep_fix_point can be reduced, or num_of_cell_around_equilibrium can be increased, depending on what prevents the algorithm from characterize the trajectories. If the problem persists, an unpractical but effective solution is to manually indicate to the algorithm whether an uncharacterized trajectory is converging towards the desired fixed point. To exploit this option set automatic to 0.

5 Examples

In the first version of the toolbox, three examples are included, although we plan to include other examples in future versions. These examples correspond to the systems studied in Sections 4.1, 4.2 and 4.4 of [1]. The system in Section 4.3 is not included here since it is particularly troublesome because of its small damping.

The three examples are organized in different folders, where the files Starter.mlx and sistema.m are included. The file sistema.m includes the system's equations of motion, which are required for the computation. To use the example, you need to interact with the Starter.mlx file. For faster computation, it is convenient to rewrite all the relevant commands in a classical MATLAB script file (*.m).

Let us consider the simplest example, the unforced Duffing oscillator. After clearing the Workspace

```
clear
```

the parameter values are defined:

```
zeta=0.05;
b=1;
a=1;
par=[zeta b a];
% the equation of motion is x''+2*zeta*x'-a*x+b*x^3=0
```

Then, the weights for computing distances are defined (look at Section 4.1 of [1] for more details)

```
weight=[4 1];
```

and it is specified the equilibrium point

```
xe=[-1,0];
```

The variables to be used for the plot are then defined (this is completely unnecessary for a system of dimension 2 as the present one)

```
var1=1;
var2=2;
```

After that, the number of steps of the iteration is specified

```
number_of_steps=50;
```

the space boundaries are given

```
spaceboundary=[xe-2,xe+4];
```

and also the discretization of the phase space

```
discr=701;
```

In the following three sections, the algorithm is run with the three different schemes for initial condition selection, type˙x0=1, 2 or 3. For a clear understanding of the command line including compute_LIM_FP, we address the reader to the explanation provided above about the possible inputs and outputs of the function.

In each of these sections, after the computation, results are plotted. In particular, the equilibrium point (xe), any other equilibrium point or other solutions identified (other_solutions_p), points of converging and diverging trajectories (vc, vd and vdout), points corresponding to the initial conditions utilized (x0) and a circle representing the hypersphere of convergence. All results are projected in a two-dimensional space, regardless of the dimension of the system. The computation is significantly speeded up if the optional input variable 'plot_results' is set to 0 instead of 1.

The last section of the unforced Duffing oscillator starter estimates the LIM value for a range of cubic stiffness coefficients of the system. The computation is run in a for loop where the parameters, the equilibrium point, and the space boundary are updated at each iteration. The for command can be substituted with parfor for exploiting parallel computation. In the case of parametric analysis and multiple runs of the compute_LIM_FP function, it is always advisable to set plot_results and plot_results_final to 0.

The other two examples, the van der Pol-Duffing oscillator with an attached nonlinear tuned vibration absorber and the chain of four masses, have the same structure, with two differences. First, no parametric analysis is provided; second, the algorithm is run with the system in modal coordinates; therefore, an additional function is included to transform the parameter values (although in the examples it was done analytically, it can be easily done numerically). We remark that the system has several degrees of freedom for these two other examples, so by changing the var1 and var2 values, different projections of the phase space will be displaced.

For more details about the examples, we invite the reader to try them. The most efficient way to learn how to use the toolbox is probably to 'play' with the provided examples and adapt them to your system of interest.

6 Conclusions

I hope this tutorial and the toolbox will be useful for you. I welcome any feedback, especially if it can help to improve the toolbox.

Acknowledgement

This project is supported by the Hungarian Academy of Science under the Lendület framework (MTA-BME Lendület "Momentum" Global Dynamics Research Group) and by the Hungarian National Science Foundation under Grant Numbers OTKA 134496.

References

[1] G. Habib, "Dynamical integrity assessment of stable equilibria: a new rapid iterative procedure," *Nonlinear Dynamics*, vol. 106, no. 3, pp. 2073–2096, 2021.

[2] S. Lenci and G. Rega, *Global Nonlinear Dynamics for Engineering Design and System Safety*, vol. 588. Springer, 2019.

[3] G. A. Vio, G. Dimitriadis, and J. E. Cooper, "Bifurcation analysis and limit cycle oscillation amplitude prediction methods applied to the aeroelastic galloping problem," *Journal of Fluids and Structures*, vol. 23, no. 7, pp. 983–1011, 2007.