Tutorial for `DynIn` toolbox v1.0.2, a `MATLAB` toolbox for the rapid computation of the dynamical integrity measure.

Giuseppe Habib

January 25, 2024

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

- v1.0.1

New features of v1.02

- It is not anymore required to provide the exact equilibrium solution; it is sufficient to provide an initial condition which will lead to the desired equilibrium solution. By default, the algorithm check if the provided initial condition is an equilibrium point and, if it is not, it looks for an equilibrium starting a trajectory from the given point. To disable this feature, the variable check_equilibrium should be set to 0. By default it is 1.
- Output data is now collected in three variables. The first one provides the final estimated LIM value, the second one, OutL (where 'L' stands for 'light'), collects all other output variables which require small memory and they form a structured variable, the third one, OutH ('H' for 'heavy'), collects all large arrays.
- The time series are not anymore obtained from an ode45 command directly in the function simulation_DI, which was hard to modify from users. Conversely, the new function sys_solver is the only function used to generate trajectories, which facilitate customization. Details of the function are provided below.
- An additional check in the function checkoninput which gives an error if the equilibrium point is outside of the space boundary.
- If the equilibrium or initial condition for finding the equilibrium is a column array instead of a row array, this is automatically corrected.
- Warning and error messages are modified for improved clarity.
- Additional examples, relative to wheel shimmy, are provided. Already existing examples were slightly modified to facilitate their understanding. In some cases, parameters are organized in a structured variable instead of an array.
- The variable new_solution_c was eliminated as it was useless.

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 (v1.0.2) of DynIn is able to handle these cases:

- equilibrium points.
- periodic solutions of exactly known period (reduced to a point through stroboscopic poincaré mapping.)
- non-smooth systems (examples will be included in future versions of the toolbox. However, if you are interested in this feature for your system and you have any difficulty, please contact me.)
- systems defined by ODE, maps, and black-box models.
  - time delayed systems can be studied through the DynInD toolbox, available at https://gdrg.mm.bme.hu/software.php [3]. DinIn and DynInD will be merged in a unique toolbox in one of the next releases.

We plan to extend the algorithm to limit cycle oscillations and quasiperiodic solutions in the near future. 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 will probably include this feature in a future release.

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:

- xe0: is a single line array containing the coordinates of the equilibrium point, or of a point in the basin of attraction of the equilibrium point.
- par: is an array or structured variable containing the parameter values of the system.

Apart from that, the algorithm requires a file providing the ODE of the system (alteratives are possible, see Sect. 4.1). 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 estimated LIM value.

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

```
compute_LIM_FP(xe0,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 (which are two optional inputs, as eexplained below), you should type:

```
compute_LIM_FP(xe0,par,'number_of_steps',200,'discr',1001)}
```

The order of the optional inputs is irrelevant, while the order of xe0 and par cannot be changed.

The output parameters are provided in the following order:

```
[R_final, OutL, OutH]
```

- R_final provide the scalar value of the estimated LIM
- OutL stands for 'output light' and provide all outputs of limited size in the form of a structures variable
- OutH stands for 'output heavy' and provide all outputs of possibly large size in the form of a structures variable

The meaning of each parameter is explained in the help of the function compute_LIM_FP. If only the LIM value is required, you can simply type

```
LIM = compute_LIM_FP(xe0,par)
```

if also some other parameters of the computation can be interesting, but you would like to limit the used memory, you can type

```
[LIM, OutL] = compute_LIM_FP(xe0,par)
```

while if you would like to have all output, then type

```
[LIM, OutL, OutH] = compute_LIM_FP(xe0,par)
```

4.1 Non-smooth systems, systems defined by maps, and other types of system descriptions

In order to use the algorithm, it is required a script which provides the evolution of the system after a given time step. This is given by the function sys_solver.m, which by default is

```
function x = sys_solver(Ts, x0, par, tols)
option = odeset('RelTol', tols(1), 'AbsTol', tols(2));
[~, x] = ode45 (@(t,xt) sistema(t, xt, par), [0 Ts], x0, option);
```

sys_solver.m gets in input the time step Ts, the initial conditions x0, the system parameters par and the tolerances tols, while it provides in output the status of the system in the phase space after the time Ts. sys_solver.m can be modified as the user desires, as far as the inputs and output remain unchanged. For example, event functions can be added in the case of non-smooth systems, another solver can be used instead of ode45, a map can be used, or a black box model can be inserted. In any case, the name of the function, its inputs and output should be unchanged. It is not an issue if some parameters will be unused because of the form of sys_solver.m.

The modified sys_solver.m function can be placed in the working `MATLAB` folder, which would replace the one included in the function folder.

4.2 Most commonly required optional input arguments

The most commonly used optional input arguments 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], or Section 2.3 of [3]. If unsure, do not insert any value, which will lead to a default vector of ones.

- spaceboundary: this line vector defines the boundaries of the phase space of interest. The equilibrium point must be within these boundaries. Its structure is

  ```
  spaceboundary = [x1min, x2min, x3min, x1max, x2max, x3max];
  ```

- 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]. This method provide a relatively uniform investigation of the phase space, but it is also the slowest.

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. This is the fastest method, which also usually provide the most accurate results. However, it might overlook some regions of the phase space, and focus only on some others.
3. Initial conditions are chosen randomly within the hypersphere of convergence.

- 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. It the algorithm is applied for *periodic solutions*, then Ts must be set to the value of the period (the periodic solution is then reduced to an equilibrium point.)

4.3 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 (*much 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 or speed (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 (for ode45 solvers or similar).
- abstol: absolute tolerance of the simulations (for ode45 solvers or similar).
- 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 'converging'. However, convergence 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 and options are:

- 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.
- check_min_periodic: if set to 1, the algorithm checks the 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.
- num_fig: number of the figure where the trajectories are plotted.

- check_equilibrium: if set to 1, the algorithm checks if the inserted xe0 is an equilibrium. If it is not, perform some simulations starting from xe0 until it does not reach an equilibrium. If set to 0, it assumes that the inserted xe0 value is already an equilibrium point.

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.

4.4 Periodic solutions reduced to fixed points

Although the algorithm can handle only fixed points, the dynamical integrity of periodic solutions is often studied through Poincaré maps, which reduce them to fixed points. 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.

4.5 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.

- **Slow convergence**
  - set type_x0 = 2, i.e., use the bisection method for selecting initial conditions. Although this choice might overlook some regions of the phase space, in general, it can produce more accurate results in shorter time.
  - reduce discr, i.e., reduce the resolution of the phase place discretization. The computational time should reduce almost linearly with the discr value (see Fig. 5 of [1]). Too small values of discr might reduce accuracy and facilitate the identification of fake periodic solutions.
  - set plot_results = 0. This prevents the generation of figures at each iteration.
  - increase reltol and abstol. These are the tolerances used for the integration scheme. Increasing them reduces the accuracy, but should reduce computational time for obtaining the solutions.
  - if type_x0 = 1 and you do not want to change it, then reduce num_gen, num_almost_clones, newcomers and num_crossover. These are the parameters of the genetic algorithm used for identifying the best initial condition, which might be lengthy, especially if number_of_steps is large; (see the red line in Figs. 4c, 7d and 11d in [1].)
- **The algorithm identifies non-existent (fake) periodic solutions**
  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. We note that this problem is typical also for the cell-mapping method [4]. Possible solutions to this problem are:

- increase rep_periodic, which is the number of time a trajectory must pass through the same cell such that the trajectory is classified as periodic.
- increase discr, i.e., increase the resolution of the phase space discretization, which reduces the cells dimension. This might increase computational time, which is approximately linearly proportional with discr. (See Fig. 5 of [1]).
- modify spaceboundary, in order to reduce the dimension of the space boundary. This enables to reduce the cell dimension, without increasing the number of cells.

If the fake periodic solution is small and it is around the equilibrium point of interest, then

- increase num_of_cell_around_equilibrium (default value is 3). This will increase the number of cells around the equilibrium already classified as converging, which might avoid the identification of fake periodic solutions and also reduce computational time.
- set check_min_periodic = 1. This forces the algorithm to ignore periodic solutions whose maximal diameter (distance between two points) is smaller than min_periodic_radius. In this case, min_periodic_radius should also be set conveniently (the default value is 0.005).

If the fake periodic solution is part of a trajectory that would have diverged from the desired solution anyway, then its detection will not affect the approximated LIM value.

- **Uncategorized trajectory before tfinal is reached**
  It can happen that the algorithm is unable to classify a trajectory within the allowed simulation time, given by tfinal. By default, in this case the algorithm pauses the computation and asks to the user to decide how the trajectory should be classified. If this is due to the small damping in the vicinity of the desired equilibrium, leading to long transients before reaching it, then possible solutions to this problem include:

  - increase tfinal, i.e., increase allowed time for each simulation. This is an obvious and effective solution. However, it might excessively increase computational time in some cases.
  - increase num_of_cell_around_equilibrium (default value is 3). This will increase the number of cells around the equilibrium already classified as converging, which might help if the trajectory takes too long time to reach the equilibrium, while it is clearly doing so.

  If the system fails to identify a periodic solution before the allowed computational time is reach, leading to a non-defined trajectory, then:

  - increase tfinal, to allow the simulation to run for a longer time.

- reduce rep_periodic, which is the number of time a trajectory must pass through the same cell such that the trajectory is classified as periodic.
- reduce discr, to reduce cell dimension and facilitate the recognition of periodic solutions.

If you are almost sure that all non-classified trajectories are not converging, then:

- set automatic = 1, such that the algorithm will automatically set all trajectories not converged in the allowed time as diverging.

If the computation is performed on a large parameter space and the algorithm is let run unsupervised, it is strongly suggested to set automatic = 1, to avoid interruptions to the computation.

5 Examples

Various examples are provided, namely:

1. unforced Duffing oscillator
2. van der Pol-Duffing oscillator with an attached nonlinear tuned vibration absorber
3. chain of four masses
4. towed wheel undergoing shimmy vibrations
5. towed wheel, with an attached nonlinear tuned vibration absorber, undergoing shimmy vibrations

The first three examples are better described in [1], while the last two are detailed in [5]. For understanding the algorithm and for using the provided code as a base for your own system, we suggest looking at the unforced Duffing oscillator, which is the simplest system.

The examples are organized in different folders, where the files Starter.mlx, sistema.m and sys˙solver.m are included. To run one example, set the example's folder as `MATLAB` current folder. 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).

6 Conclusions

We hope this tutorial and the toolbox will be useful for you. We 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] B. Szaksz, G. Stepan, and G. Habib, "Dynamical integrity estimation in time delayed systems: a rapid iterative algorithm," *Journal of Sound and Vibration*, vol. 571, p. 118045, 2024.

[4] 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.

[5] G. Habib and A. Epasto, "Towed wheel shimmy suppression through a nonlinear tuned vibration absorber," *Nonlinear Dynamics*, vol. 111, no. 10, pp. 8973–8986, 2023.