Tutorial for `DynIn` toolbox v1.1.0, a `MATLAB` toolbox for the rapid computation of the dynamical integrity measure.

Giuseppe Habib, Dora Patko, Bence Szaksz

February 25, 2025

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, v1.0.2, v1.0.3

New features of v1.1.0

- The toolbox for time delayed systems, previously called `DynInD`, is now incorporated in `DynIn` as a comprehensive toolbox.
- The toolbox can now handle periodic solutions of autonomous and non-autonomous systems; however, this version of the tutorial does not discuss those cases yet.

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, 2]. We hope you will find the toolbox useful, and we 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, we would be glad if you could please cite one of our papers [1, 2].

2 Installation

To install the `DynIn` Toolbox, you need to open the main folder of the program (`DynIn_v1.X.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 uninstalling that one before installing the new version to avoid conflicting versions of the same functions.

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, i.e., the minimal distance between a steady-state solution and any point of the boundary of its basin of attraction. For detailed explanations, please look at [1, 3].

The current version (v1.1.0) 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 us.)
- Systems defined by ODE, maps, and black-box models.
- Limit cycles of autonomous system (although the toolbox can handle this case, this version of the tutorial does not cover those cases yet, a paper describing it in detail is under preparation).
- Fixed points of time delay systems. Note that for this application, there are two versions. In one of them, the LIM is defined based on the headpoint of the initial condition (details are available in [2]). In the other option, the LIM is calculated based on the discretized state of the system, which leads to faster convergence (a paper describing this case in detail is under preparation).
- Limit cycles of time delay systems. This is based on the discretized state approach (a paper describing this case in detail is under preparation).

We plan to extend the algorithm to quasiperiodic solutions in the near future. Besides, currently `DynIn` has only a command line version. In the future, we plan to develop a graphical user interface as well.

Currently, `DynIn` *cannot* compute basins of attraction; however, we plan to include this feature in a future release.

4 How to use it

4.1 Fixed points (equilibrium points)

The function to compute the LIM of fixed points 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 either 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. 4.2). 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 press 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 explained 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` provides 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)
```

4.2 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.

4.3 Periodic solutions reduced to fixed points

The algorithm can handle periodic solutions if reduced to fixed points through stroboscopic Poincaré maps. This can be done 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.

The algorithm can also handle limit cycle oscillations and periodic solutions in their entirety (so not reduced to fixed points). Although this feature is already available, this tutorial will include it only once the paper about it will be publicly available. If you are interested in this feature, please contact us. We will be happy to collaborate with you.

4.4 Time delay systems

In time delay systems with constant delay τ, the state of a solution is a function of time in the time interval $[t-\tau, t]$. Therefore, the initial condition is not defined by a point in the phase space, but by a continuous function (called initial function). The definition of the basin of attraction, and so, the LIM of a fixed point is not straightforward. Here, we propose two approaches:

1. **Headpoint of initial condition (HIC) based approach**: Restrict a specific type of initial functions (constant/linear/free vibration/jump), which are determined by their headpoint of initial condition (HIC) [2]. A point in the phase space belongs to the basin of attraction if it is the headpoint of an initial condition that yields a converging solution. This means that the basin of attraction is defined in a space of dimension equal to the system dimension in absence of time delay. This implies that only the HICs reduce the estimated LIM and not any point of the trajectory, yielding to a slower convergence than with the second approach, discussed below. Details of this approach and its implications are discussed in [2].

2. **Discretized state based approach**: The second approach relies on a discretization of the time delay, leading to a finite phase space, differently from the infinite phase space of time delayed systems. A (discretized) state in the phase space belongs to the basin of attraction of a solution if a trajectory, initiated from that state, converges to the desired solution. For the practical implementation of the approach, initial conditions are restricted to specific functions; otherwise, considering arbitrary initial condition would lead to physically unrealistic states of the system (for example, a non-smooth function that has jumps almost everywhere), making the computation unnecessarily longer. The basin of attraction and the LIM is defined in the discretized phase space. Once the phase space is discretized, the algorithm is carried out as for the case of non-delayed systems, discussed above. Therefore, the whole trajectory is used for the LIM computation and not only its HIC. Details of this approach are provided in a paper currently under preparation.

In both cases, the trajectories are obtained through the semi-discretization method [4]. Our results illustrated that the two methods lead to qualitatively analogous results; however, the second approach is usually faster. Details on how to implement the two approaches are explained below.

4.4.1 Headpoint of initial condition (HIC) based approach

To start utilizing the toolbox for time delay systems we suggest to start from the template:
`Delayed_Simplified_Duffing_Oscillator_HIC`.
The other templates mostly refer to the examples discussed in [2].

If you would like to obtain the LIM of a time delay system with HIC approach, first, construct the structure of parameters `par`. This must contain the time delay τ (`par.tau`), the sampling delay number r (`par.r`), and the time step Δt (`par.dt`). The relation between these is: $\Delta t = \tau/(r + 0.5)$. Additionally, the structure `par` should contain the system parameters similar to the fixed point case (Sect. 4.1).

The equation of motion should be defined in the `dyn[...].m` function:

```
parout = dyn(par)
```

as a symbolic expression. Additionally, this file calls the `semidiscmatr` function that generates the matrices required for the semidiscretization scheme. The output of this function `parout` is the same as the input `par` with the additional variable `parout.mapcell` that contains the matrices for the semi-discretization-based mapping.

Then, the LIM computation for fixed points of time delay systems, based on the HIC, is initiated by the function

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

which works very similarly to `compute_LIM_FP` for fixed points. The output is organized in the same way as in that case, so if you would like to have all the outputs, then type

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

Major differences lie in the way the system is defined.

If one would like to modify the model, then the equation of motion section should be changed in the `dyn[...].m` function. Note that the degree of freedom (DoF) of the system should be adjusted accordingly, which is defined by the variable `n`.

For the details we refer to the template `Delayed_Simplified_Duffing_Oscillator_HIC`. Additional features are available, as discussed below. Note that they are not included in the simplified template, but only in the others.

The `dyn[...].m` function can be called with more output variables:

```
[parout,weight] = dyn[...](par)
```

where the variable `weight` refers to the vector of weights obtained by the modal decomposition of the undamped system [2].

Also, one can transform the system to its modal coordinates in a way that the corresponding transformation matrix `U` is provided to the `semidiscmatr` function as an additional input:

```
[mapcell,x1_q,x2_q] = semidiscmatr(eq,par.dt,par.r,xcell,'U',U)
```

The variables `x1_q` and `x2_q` are the first two original coordinates as functions of the modal coordinates. These can also can be returned from the `dyn[...].m` function as:

```
[parout,weight,x1_q,x2_q] = dyn[...](par)
```

A further feature of this approach is that you can select different type of initial conditions with the 'init_type' option of the compute_LIM_D_FP_HIC function (see details in Sec. 4.6).

Note that the HIC based approach works only with the bisection method.

4.4.2 Discretized state based approach

In the discretized state based approach, we apply two discretization levels to the time delay system. A relatively coarse mesh (discretized state) is used for trajectory comparison and basin of attraction definition, while the numerical integration (based on the semi-discretization method [4]) of the trajectories is done on a fine mesh to obtain accurate results. This way, the problem is transformed back to the estimation of the LIM of a finite phase space, similarly to the case of ODEs.

The LIM estimation of time delay systems with the discretized state based approach is very similar to that with the HIC based approach discussed above. Note that the corresponding templates will be provided later once the related paper is submitted.

You should define your equation of motion in the function `dyn[...].m` and fix the parameter structure **par**. Then type:

```
par = dyn[...](par)
[LIM, OutL, OutH] = compute_LIM_D_FP(xe0,par)
```

The only difference is that the parameter structure should contain information about the discretization. To make it clear let us add some information about the discretization procedure:

The coarse mesh for trajectory comparison consist of N equally distributed sampling points within a time interval of length τ; resulting a time step of $\tau/(N-1)$. Note that it contains the state of the system at both ends of the τ long time interval.

Then, consider that this coarse mesh has k-times larger time steps than the temporal discretization for time integration, that is, the state for time integration consist of $k(N-1)+1$ sampling points.

In the parameter structure `par`, the variable `par.N` refers to the state discretization number, while `par.k` is the multiplier for the extension of the discretized state space to the integration space.

4.4.3 Discretized state based approach for the LIM of limit cycles in time delay systems

The algorithm was extended to limit cycles of time delay systems. However, as the related publication is currently under preparation, no additional detail is provided in the tutorial at this time. If you are interested, please do not hesitate to contact us. Note that the algorithm can also handle non-smooth time delay systems.

4.5 Most commonly required optional input arguments of the `compute_LIM[...].m` function

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 [2]. 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.

Note that LIM estimation for *time delay systems* is currently working only with the bisection method; this optional variable is not included in the corresponding code.

- `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. If 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.) Moreover, in *time delay systems* with the HIC approach, the variable `Ts` is equal to the time delay τ; therefore, there is no need for this additional input.

4.6 Good to know

All the following optional input arguments must be input in the `compute_LIM[...].m` function.

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

- `plot_results`; 0: no figure is shown during the computation (*much faster 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). It is not present in case of *time delay systems*.

- `abstol`: absolute tolerance of the simulations (for `ode45` solvers or similar). In case of *time delay systems*, it is used only to obtain the exact value of the equilibrium.

- `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 the variable named `num_of_cell_around_equilibrium`, which should be an odd positive integer number. Its default value is 3. Note that the number of cells in this hypercube scales exponentially with the dimension of the system; therefore, for large dimensional systems (typically dimension larger than 10) it might be convenient to set it to 1 to avoid memory issues.

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 (note that *time delay systems* are only treated with bisection method, so these parameters are not applicable to them.):

- `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.

In **time delay systems**, when the HIC approach is used, one can set the following arguments:

- `init_type`: type of initial condition, can be set to 0, 1, 2, or 3. In case of time delay systems an initial function is required in the $[-\tau, 0]$ interval. 0: constant initial function. 1: linear initial function. 2: jump type, when the initial function is located in the equilibrium except for the HIC. 3: initial condition is the free vibration of the uncontrolled linearized system.

- `ind_check_conv`: Indices of the state vector in the semi-discretization method, which are considered, when the algorithm checks whether the current trajectory is converging to the fixed point. Used in case of the HIC approach.

- `ind_check`: Indices of the state vector in the semi-discretization method, which are considered when the algorithm checks whether the current trajectory is converging to another trajectory (or to itself). Used in case of the HIC approach.

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 smaller 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.

- `max_check_step`: it is relevant only if `check_equilibrium` is set to 1. `max_check_step` indicates the maximal number of iteration that the algorithm run to look for an equilibrium. If the equilibrium is not found within this number of steps (with an approximation of maximum $10\times$`reltol` or $10\times$`abstol`) an error is given.

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.

4.7 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].)
  - if the LIM of limit cycles is computed, the `num_of_cell_around_equilibrium` argument should be set to 1. Otherwise, it takes long time to obtain these cells.
  - in time delay systems based on the HIC approach, the sampling delay number `r` can be reduced. Note that this effects the accuracy of the time integration.
  - in time delay systems based on the state discretization approach, the parameter `par.k` can be reduced; this somewhat reduces the accuracy of the time integration as well.
- **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 [5]. 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.

– in time delay systems based on the state discretization approach, the state discretization number `par.N` can be increased; in parallel with this, you can also reduce the parameter `par.k` to keep the same accuracy for the time integration. The increase of `par.N` yields that the states of the trajectories are saved in a finer mesh, so it reduces the opportunity to identify non-existent (fake) periodic solutions. Note that this modifies the distance measure and so the LIM as well.

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, with an attached nonlinear tuned vibration absorber, undergoing shimmy vibrations
5. delayed Duffing oscillator (simplified and complete versions)
6. machining (turning operation) subject to delay effects
7. machining (turning operation) with an attached nonlinear tuned vibration absorber (yielding a time delay system)
8. inverted pendulum with an attached nonlinear tuned vibration absorber and with delayed proportional-derivative control.

The first three examples are better described in [1], the fourth and fifth ones are detailed in [6], while the remaining ones are discussed in [2]. 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 (in case of time delay systems these are the `Starter.mlx` and `dyn[...].m` files). 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] 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.

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

[4] T. Insperger and G. Stépán, "Updated semi-discretization method for periodic delay-differential equations with discrete delay," *International journal for numerical methods in engineering*, vol. 61, no. 1, pp. 117–141, 2004.

[5] 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.

[6] 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.