ORBITADOR 1.0.0 - User Guide

May 14, 2021

Contents

Chapter 1

ORBITADOR overview

ORBITADOR is a tool that aims to study the robustness of dynamical systems commanded by differential equations. It implements a formal method to prove formally the stability of time-periodic systems and some other useful features.

1.1 Features

1.1.1 Platforms

ORBITADOR can be run under Linux, MacOS X and Windows. See the specified instructions for each platform.

1.1.2 Core

The core files for ORBITADOR are written in Python and released under Pycharm License. User supplied functions can be written in plain Python, and do not require advanced programming skills since they mostly translate the mathematical equations of the problem.

1.1.3 Key features

- Gives a simple condition of set inclusion which guarantees that the system is open-loop stable in the presence of perturbation when it is generating by periodic control that should be intrinsically robust or stable against possible perturbations or uncertainties.
- Proves the existence of a limit cycle (LC) of a dynamic system Σ and construct an enclosure of the LC which is an invariant set of Σ. The method essentially consists in adding a small additive perturbation to Σ, and finding an approximation T of the exact period such that the set of solutions of the perturbed system at the instant $(i+1)T$ is included in the set of solutions at iT for some integer i.

1.2 Algorithm

Consider a differential system with bounded uncertainty of the form $\{\dot{x}(t) = f(x(t), w(t))\}$ satisfying the equation:

$$\langle f_u(y_1,w_1) - f_u(y_2,w_2), y_1 - y_2\rangle \leq \lambda_u\|y_1 - y_2\|^2 + \gamma_u\|y_1 - y_2\|\|w_1 - w_2\| \quad (1.1)$$

where, there exist constants $\lambda \in \mathbb{R}$ and $\gamma \in \mathbb{R}_{\geq 0}$ such that, for all $y_1, y_2 \in \mathcal{S}$ and $w_1, w_2 \in \mathcal{W}$. Recall that λ has to be computed in the absence of perturbation ($w = 0$). The additional constant γ is used for taking into account the uncertainty w. Given λ, the constant γ can be computed itself using a nonlinear optimization solver. Instead of computing them globally for S, it is advantageous to compute λ and γ locally depending on the subregion of S occupied by the system state during a considered interval of time.

Consider a point $x_0 \in \mathcal{S}$ and a point $y_0 \in \mathcal{B}(x_0,\varepsilon)$.Let $x(t;y_0)$ be the exact solution of the system $\frac{dx(t)}{dt} = f(x(t),w(t))$ with bounded uncertainty $\mathcal{W}$ and initial condition y_0, and $\tilde{x}(t;x_0)$ the Euler approximate solution of the system $\frac{dx(t)}{dt} = f(x(t),0)$ *without uncertainty* ($|\mathcal{W}| = 0$) with initial condition x_0. We have, for all $w(\cdot) \in \mathcal{W}$ and $t \in [0,\tau]$:

$$\|x(t;y_0) - \tilde{x}(t;x_0)\| \leq \delta_{\varepsilon,\mathcal{W}}(t).$$

with:

- if $\lambda < 0$:

$$\delta_{\varepsilon,\mathcal{W}}(t) = \left(\frac{C^2}{-\lambda^4}\left(-\lambda^2t^2 - 2\lambda t + 2e^{\lambda t} - 2\right) + \frac{1}{\lambda^2}\left(\frac{C\gamma|\mathcal{W}|}{-\lambda}\left(-\lambda t + e^{\lambda t} - 1\right) + \lambda\left(\frac{\gamma^2(|\mathcal{W}|/2)^2}{-\lambda}(e^{\lambda t} - 1) + \lambda\varepsilon^2 e^{\lambda t}\right)\right)\right)^{1/2} \quad (1.2)$$

- if $\lambda > 0$:

$$\delta_{\varepsilon,\mathcal{W}}(t) = \frac{1}{(3\lambda)^{3/2}}\left(\frac{C^2}{\lambda}\left(-9\lambda^2t^2 - 6\lambda t + 2e^{3\lambda t} - 2\right) + 3\lambda\left(\frac{C\gamma|\mathcal{W}|}{\lambda}\left(-3\lambda t + e^{3\lambda t} - 1\right) + 3\lambda\left(\frac{\gamma^2(|\mathcal{W}|/2)^2}{\lambda}(e^{3\lambda t} - 1) + 3\lambda\varepsilon^2 e^{3\lambda t}\right)\right)\right)^{1/2} \quad (1.3)$$

- if $\lambda = 0$:

$$\delta_{\varepsilon,\mathcal{W}}(t) = \left(C^2\left(-t^2 - 2t + 2e^t - 2\right) + \left(C\gamma|\mathcal{W}|\left(-t + e^t - 1\right) + \left(\gamma^2(|\mathcal{W}|/2)^2(e^t - 1) + \varepsilon^2 e^t\right)\right)\right)^{1/2} \quad (1.4)$$

where $|\mathcal{W}|$ is the diameter of $\mathcal{W}$ (maximum distance between 2 points of $\mathcal{W}$).

Chapter 2

Getting started

ORBITADOR files include:

- Core files (analyser.py, funcs.py and plot_funcs.py): sources files for ORBITADOR
- Problem file: each problem has its own file in which definition of the system and inputs are stored also the output files including figures and a **.res* file will be saved after the analysis the problem.

ORBITADOR features are organized in 3 main modules

1. Problem Definition
   In this module, the user defines the names of the states that constitute the system also specify the names and the values of the parameters. After that, the user gives the differential equations of the system. If the problem contains a guard condition and a reset, it is possible to add them in the problem definition part.

2. Problem Configuration
   In the problem configuration module, the user can set the options to run ORBITADOR. Among these options, we find the time steps, the initial conditions of the system or the starting point at initial time t_0 the value of the perturbation and the number of the periods.

3. Visualization
   In the last module, the user specify the states to display. It can be a 1D plot where the state is shown as a function of t also or a phase portal where a state is shown as a function of an other state, in this case it can be 2D or 3D plot.

2.1 Test problem: Biped

The files corresponding to the Biped problem are given in appendix C.

We consider a simple model of biped walker, seen as a hybrid oscillator. The fig. 2.1 shows a schematic diagram of the model, where l is the length of the legs; M and m are the masses of the hip and the foot, respectively; ϕ_1 and ϕ_2 specify the angles of the swing and support legs and γ is the angle of the slope. This model exhibits a stable limit-cycle oscillation for appropriate parameter values that corresponds to periodic movements

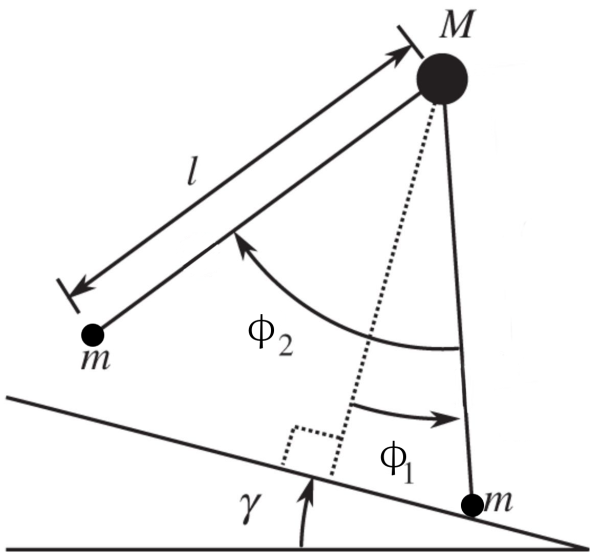


Figure 2.1: Biped system

of the legs. The model has a unique mode ($U = \{1\}$) and a continuous state variable $\mathbf{x}(t) = (\phi_1(t), \dot{\phi_1}(t), \phi_2(t), \dot{\phi_2}(t))^\top$. The dynamics is described by<

$$f(\mathbf{x}) = \begin{pmatrix} \dot{\phi_1} \\ sin(\phi_1 - \gamma) \\ \dot{\phi_2} \\ sin(\phi_1 - \gamma) + \dot{\phi}_1^2 sin\phi_2 - cos(\phi_1 - \gamma) sin\phi_2 \end{pmatrix} \tag{2.1}$$

$$Reset(\mathbf{x}) = \begin{pmatrix} -\phi_1 \\ \dot{\phi_1} sin(2\phi_1) \\ -2\phi_1 \\ \dot{\phi_1} cos2\phi_1(1 - cos2\phi_1) \end{pmatrix} \tag{2.2}$$

$$Guard(\mathbf{x}) = 0 \equiv (2\phi_1 - \phi_2 = 0 \ \wedge \phi_2 < -\delta), \tag{2.3}$$

We consider: $\gamma = 0.009$, $\delta = 0.1$ and initial condition $x(0) = (0.200236, -0.199756, 0.40055, -0.01585)$.

- Numerical simulations: problem biped

Discretization: Euler with time steps $dt = 2 * 10^{-5}$

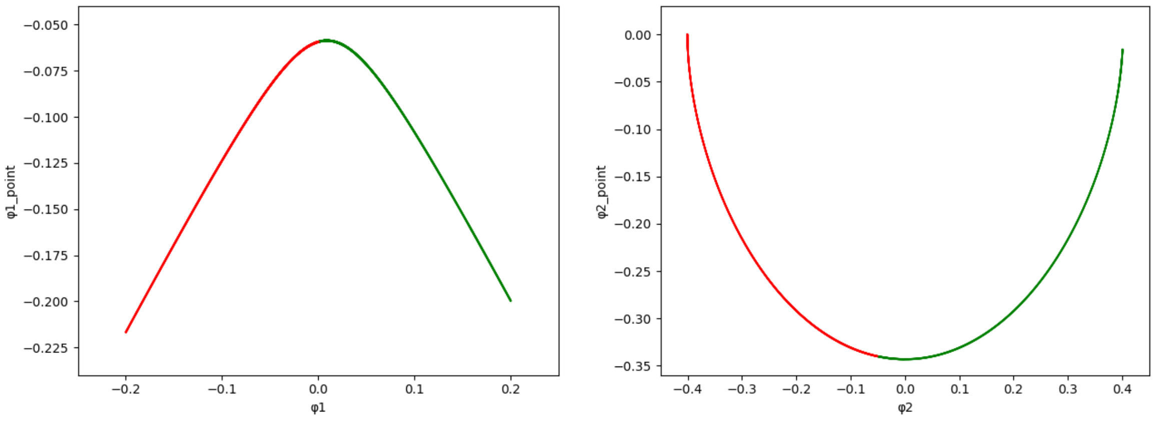


Figure 2.2: Biped: A cyclic trajectory for plan ϕ_1 (left) and ϕ_2 (right)

Appendix A

Readme and Install files

A.1 Readme

ORBITADOR is a software tool for robustness analysis of dynamical systems commanded by differential equations. It implements a formal method to prove formally the stability of time-periodic systems and some other useful features. The options available for IMITATOR are explained in the following

A.2 Install

A.2.1 Prerequisites and dependencies

ORBITADOR has some dependencies (like *numpy*, *sympy*, *scipy* and *matplotlib*) that need to be installed. We give here some recommendations to ensure that ORBITADOR works correctly:

- General:
  - Use Python 3.6 vesion or higher.
  - The dependencies can be installed separately using the commands below:

    ```
    pip3 install numpy
    pip3 install sympy
    pip3 install scipy
    sudo apt-get install python3-matplotlib -y
    sudo apt-get install python3-tk
    ```

    Or, they can be installed using the *requirements.txt* file in *orbitador* directory:

    ```
    python -m pip install -r requirements.txt
    ```

- Windows: For Windows users, we strongly recommend to use MinGW when running ORBITADOR command.

A.2.2 How to lunch ORBITADOR (COMMAND LINE)

To launch ORBITADOR without using the GUI, the user needs to run , in the terminal, the executable file *orbitador* followed by the name of the input file.

A.2.3 Some options of ORBITADOR (COMMAND LINE)

We list here some options available for ORBITADOR:

- –show-figures or -s **(default: disabled)**: Show figures directly after analysis
- –no-save-figures or -n **(default: disabled)**: Block save figures
- –save-results or -r **(default: disabled)**: Save results in a file named ”example_name”.res

Appendix B

Description of input file

The input file contains all specifications of the computed system. As we mentioned in chapter 2, it can be divided in 3 main modules:

1. The definition of the system:
   This module has essentially 5 components:

   - The list states: it contains all the states that constitute the system. The states are separated by commas and each state is enclosed in two quotation marks.
   - The dictionary parameters: it contains the set of parameters that have been integrated in the system. The parameters are separated by commas and for each parameter the user first gives the name of the parameter placed between two quotes followed by a colon and its value.
   - The function differential_system: it contains the differential equations that define the system. For each state (placed between two quotes), the user gives the differential equation which corresponds to it.
   - The function guard: if the system is hybrid, the user gives in this function the condition of the reset.
   - The function reset: it holds the assignment of the reset for the states.

2. The configuration of the system:
   This module is taken up by a dictionary called system_config that have 8 items:

   - dt: The time steps.
   - initial_gap_epsilon: It defines the initial value of the error bounded δ between the exact solution of the ODE and its Euler approximation.
   - w: The value of the perturbation.
   - num_periods: The number of the periods of the computation.
   - num_system_simulations: The number of simulations that are generated randomly.
   - auto_period_value: The user chooses whether the period value of the system should be determined automatically or not.
   - period_value: If the period *auto_period_value* is False, the user can gives in this item the value of a period.

- init_system: This item contains the initial condition of the system. The user must give the initial value of the states in the same order as given in the states list.

3. The display of the system:
   This module is grouped into a dictionary called system_display that have 11 items:

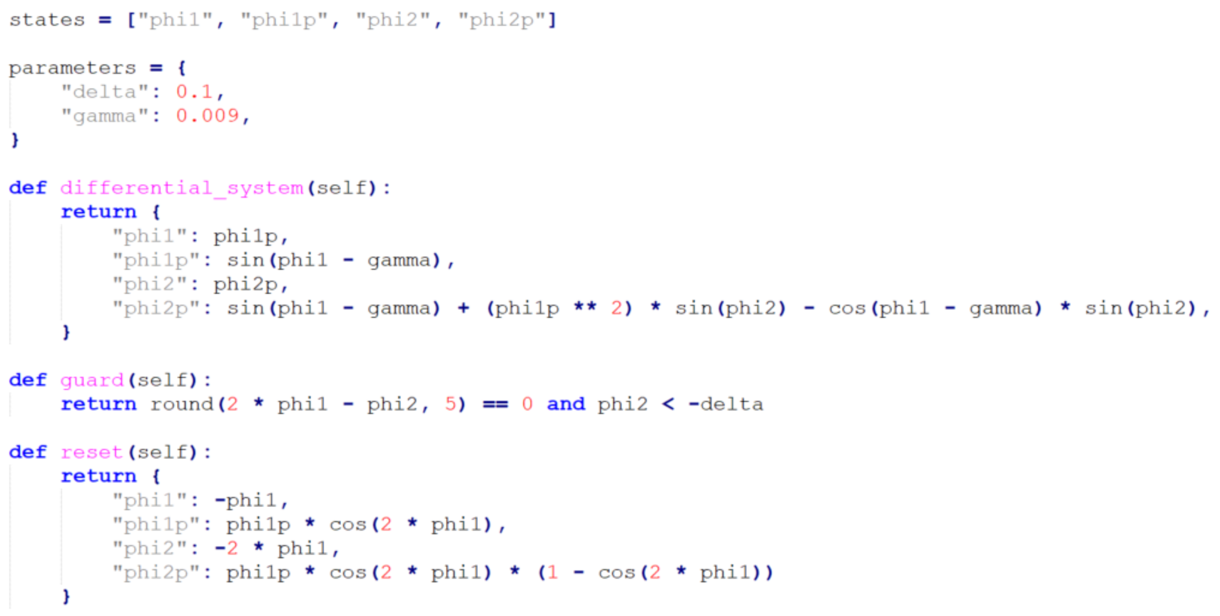
   - 1D_plot: In this item, the user selects the states to display as a function of time t.
   - 2D_plot: In this item, the user selects two states in each list to show in 2D plot.
   - 3D_plot: In this item, the user selects three states in each list to show in 3D plot.
   - delta_curve: If it is True, it shows the curve of δ in green in the plots 1D of the states that are based on time t.
   - color_depend_on_lambda_sign: If it is True, it shows the curve of the state according to the sign of lambda (red if lambda>0 and green otherwise).
   - simulations_curves: If it is True, it show the curves of the random simulations in blue.
   - min_max_tube: If it is True, it shows the lowest point of the top part of the tube in gray and the highest point of the bottom part of the tube in cyan. This will be shown in the plots that are a function of time t.
   - plot_lambda: If this item is True, it shows the evolution of λ as a function of t.
   - plot_delta: If this item is True, it shows the evolution of δ as a function of t.
   - plot_gamma: If this item is True, it shows the evolution of γ as a function of t.
   - step_plot: The value of the step for all the plots.

Appendix C

Files for the Biped problem

C.1 Input file

In figs. C.1 to C.3, we give the composition of the input file *biped.py* that leads to the computation of the system.

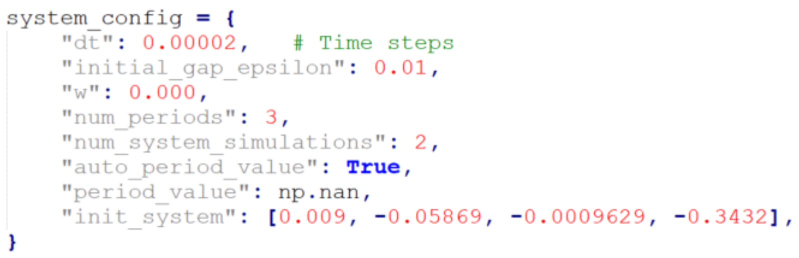


```
states = ["phi1", "phi1p", "phi2", "phi2p"]

parameters = {
    "delta": 0.1,
    "gamma": 0.009,
}

def differential_system(self):
    return {
        "phi1": phi1p,
        "phi1p": sin(phi1 - gamma),
        "phi2": phi2p,
        "phi2p": sin(phi1 - gamma) + (phi1p ** 2) * sin(phi2) - cos(phi1 - gamma) * sin(phi2),
    }

def guard(self):
    return round(2 * phi1 - phi2, 5) == 0 and phi2 < -delta

def reset(self):
    return {
        "phi1": -phi1,
        "phi1p": phi1p * cos(2 * phi1),
        "phi2": -2 * phi1,
        "phi2p": phi1p * cos(2 * phi1) * (1 - cos(2 * phi1))
    }
```

Figure C.1: Biped: definition of the system in the input file

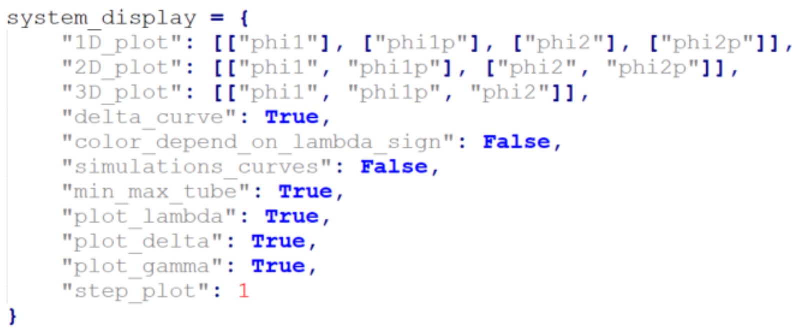
C.2 Output files

In figs. C.4 to C.6, we show the plots that are generated by ORBITADOR. The fig. C.7 presents the result file that shows whether the system is robust or not.



```
system_config = {
    "dt": 0.00002,    # Time steps
    "initial_gap_epsilon": 0.01,
    "w": 0.000,
    "num_periods": 3,
    "num_system_simulations": 2,
    "auto_period_value": True,
    "period_value": np.nan,
    "init_system": [0.009, -0.05869, -0.0009629, -0.3432],
}
```

Figure C.2: Biped: configuration of the system in the input file



```
system_display = {
    "1D_plot": [["phi1"], ["phi1p"], ["phi2"], ["phi2p"]],
    "2D_plot": [["phi1", "phi1p"], ["phi2", "phi2p"]],
    "3D_plot": [["phi1", "phi1p", "phi2"]],
    "delta_curve": True,
    "color_depend_on_lambda_sign": False,
    "simulations_curves": False,
    "min_max_tube": True,
    "plot_lambda": True,
    "plot_delta": True,
    "plot_gamma": True,
    "step_plot": 1
}
```

Figure C.3: Biped: configuration of the visualization of the system in the input file

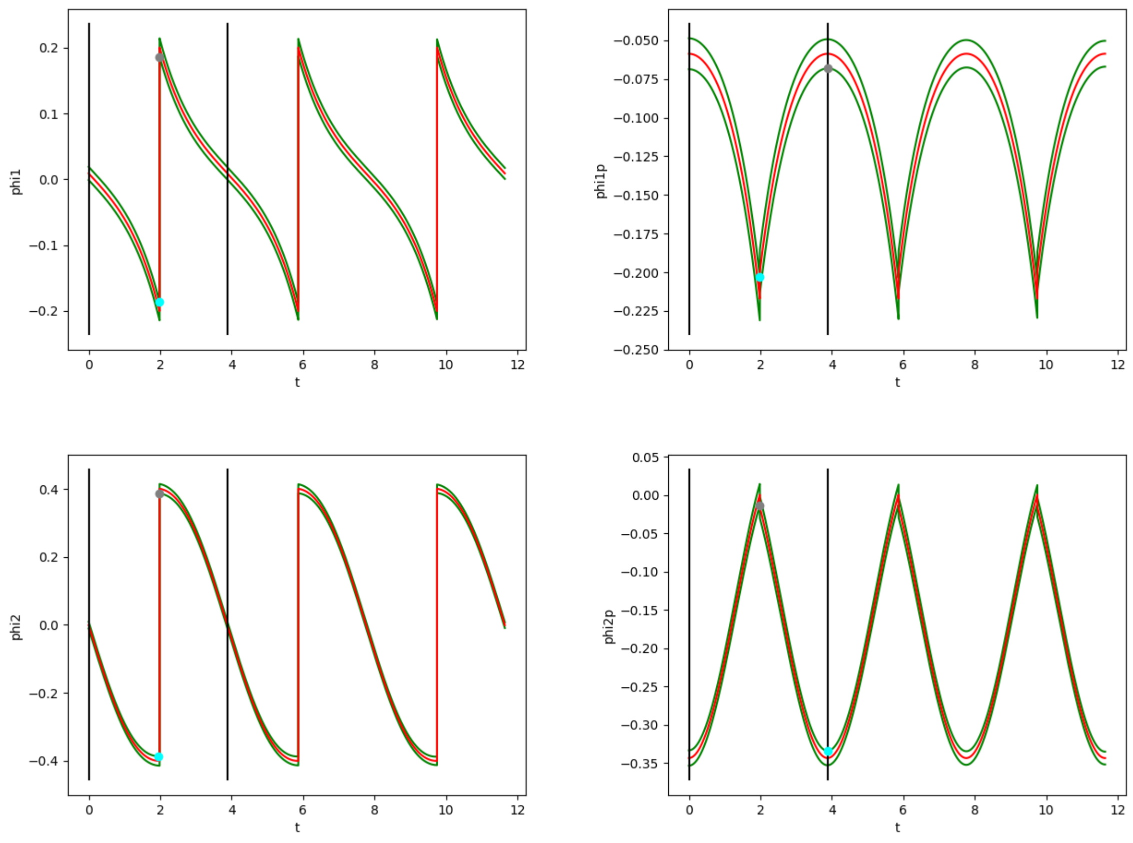


Figure C.4: Biped: The evolution of $\phi_1(t)$, $\dot{\phi_1}(t)$, $\phi_2(t)$ and $\dot{\phi_2}(t)$ presented by the red curves (from the top-left to the bottom-right), the green curves correspond to the borders of tube B_w

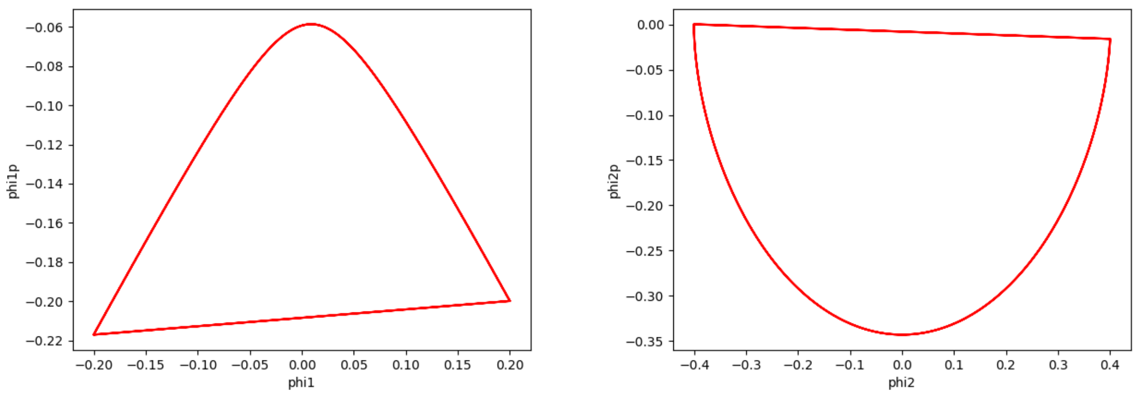


Figure C.5: Biped:The left plot shows the evolution of $\dot{\phi_1}$ as a function ofϕ_1 and the right plot illustrates the evolution of $\dot{\phi_2}$ as a function ofϕ_2 (2D plots)

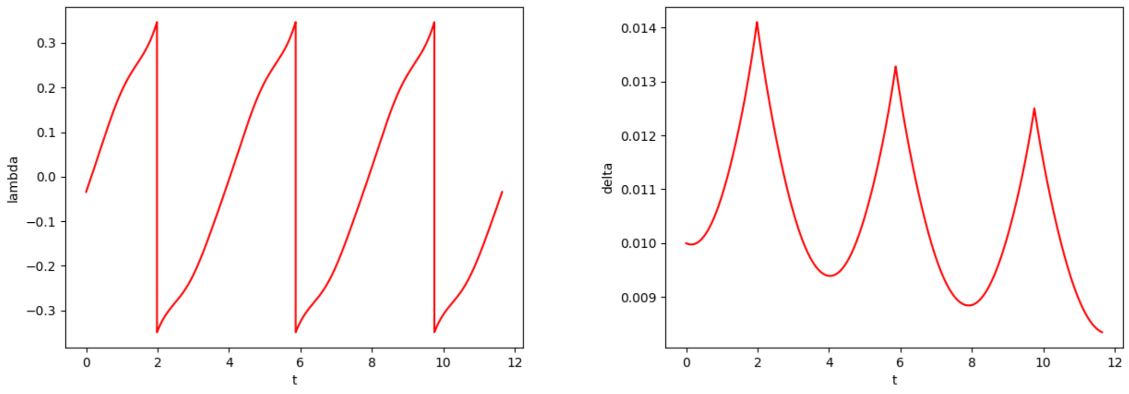


Figure C.6: Biped:The left plot shows the evolution of the one-sided Lipschitz $\lambda(t)$ and the right plot illustrates the evolution of the error bounded $\delta(t)$ between the exact solution of the ODE and its Euler approximation.

```
Launch execution of Biped
dt= 2e-05
w= 0.0
initial_gap_epsilon= 0.01
states= [phi1, phi1p, phi2, phi2p]
parameters= {'delta': 0.1, 'gamma': 0.009}
system_equations= [phi1p, sin(phi1 - 0.009),
phi2p, phi1p**2*sin(phi2) - sin(phi2)*cos(phi1 - 0.009) + sin(phi1 - 0.009)]
Reset= [-phi1, phi1p*cos(2*phi1), -2*phi1,
phi1p*(1 - cos(2*phi1))*cos(2*phi1)]
one period=  3.88254
NodeT=  582382

Check inclusion:
Y_0(0T)=[ 0.009      -0.05869    -0.0009629 -0.3432    ],
delta_w(0T)=0.01, epsilon*e^(lambda*t)=0.01
Y_0(1T)=[0.008989934326328459, -0.05869466628240013, -0.0009845103426641695,
-0.3432136940358442], delta_w(1T)=0.009416162703033158, epsilon*e^(lambda*t)
    =0.008759895947364912
For state phi1: tube at t=1T is included in tube at t=0T
For state phi1p: tube at t=1T is included in tube at t=0T
For state phi2: tube at t=1T is included in tube at t=0T
For state phi2p: tube at t=1T is included in tube at t=0T
For all the states: Tubes at t=1T are included in tubes at t=0T.
Y_0(2T)=[0.008997254971281773, -0.05868306104450253, -0.0010327343230725813,
-0.3432354865820279], delta_w(2T)=0.008866581753879088, epsilon*e^(lambda*t)
    =0.007672243173924995
For state phi1: tube at t=2T is included in tube at t=1T
For state phi1p: tube at t=2T is included in tube at t=1T
For state phi2: tube at t=2T is included in tube at t=1T
For state phi2p: tube at t=2T is included in tube at t=1T
For all the states: Tubes at t=2T are included in tubes at t=1T.
Y_0(3T)=[0.00900325543069183, -0.05868650521084726, -0.0009539181207135081,
-0.3432135425648742], delta_w(3T)=0.008348086599605715, epsilon*e^(lambda*t)
    =0.006718021370124153
For state phi1: tube at t=3T is included in tube at t=2T
For state phi1p: tube at t=3T is included in tube at t=2T
For state phi2: tube at t=3T is included in tube at t=2T
For state phi2p: tube at t=3T is included in tube at t=2T
For all the states: Tubes at t=3T are included in tubes at t=2T.
Conclusion: B(3T) included in B(2T) included in B(1T) included in B(0T)


Check for at least one stat that the lower limit of the top of the tube is less than the
    higher limit of the bottom of the tube:
For the state phi1: the minimum value of the top part of the tube =-0.18620331910499358 ,
the maximum value of the bottom part of the tube =0.18620341748066638
For the state phi1p: the minimum value of the top part of the tube =-0.2029082399107859 ,
the maximum value of the bottom part of the tube =-0.06811082898543329
For the state phi2: the minimum value of the top part of the tube =-0.3865906440516828 ,
the maximum value of the bottom part of the tube =0.3865062663868676
For the state phi2p: the minimum value of the top part of the tube =-0.3337982646481965 ,
the maximum value of the bottom part of the tube =-0.013746299871000452
The proposition is verified for the states: ['phi1', 'phi1p', 'phi2', 'phi2p']


Check that the sum of lambda is negative for each period:
Period number 1: sum(lambda) = -3007.872228097125
Period number 2: sum(lambda) = -3006.9149169381863
Period number 3: sum(lambda) = -3012.8493512652917

Execution time: 67.02307462692261 seconds ---
```

Figure C.7: Biped: output file *Biped.res*