STONEHENGE

Suite for Nonlinear Analysis of Energy Harvesting Systems

www.github.com/americocunhajr/STONEHENGE

J.P.C.V. Norenberg, J.V.L.L. Peterson, V.G. Lopes, R. Luo,
M.C. Pereira, J.G. Telles Ribeiro, A. Cunha Jr

Authors

- *João Pedro C. V. Norenberg* [1] (jp.norenberg@unesp.br)
- *João Victor L. L. Peterson* [2] (ligier.peterson@gmail.com)
- *Vinicius G. Lopes* [2] (vinigolop@gmail.com)
- *Roberto Luo* [2] (cai.roberto@graduacao.uerj.br)
- *Marcelo da Cruz Pereira* [2] (marcelopereira086@gmail.com)
- *José Geraldo Telles Ribeiro* [2] (telles@eng.uerj.br)
- *Americo Cunha Jr* [2] (americo.cunha@uerj.br)

[1] São Paulo State University - UNESP
[2] Rio de Janeiro State University - UERJ

What is STONEHENGE?

STONEHENGE: Suite for Nonlinear Analysis of Energy Harvesting Systems

It is a repository with an ensemble of codes developed to conduct nonlinear analysis on a bistable piezoelectric-magneto-elastic energy harvester.

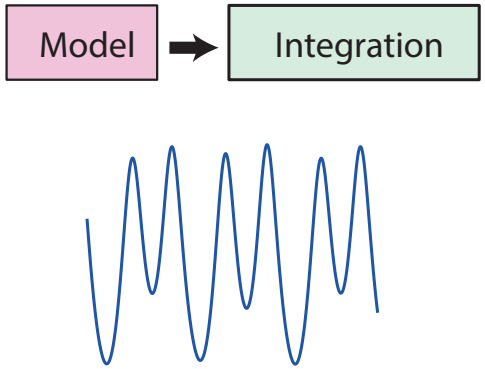
Mission:

- analyze bistable piezoelectric-magneto-elastic energy harvester;
- provides a didactic and intuitive tool for interested audiences.

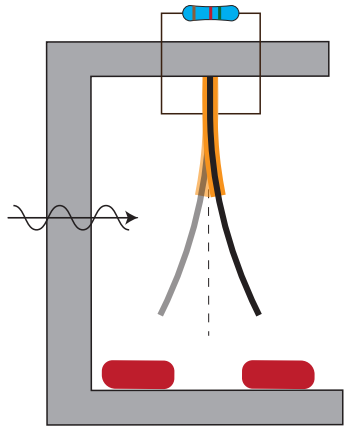
STONEHENGE Modules

- **Initial Value Problem (IVP)**
- **Animation**
- **Nonlinear Analysis Tools**
- **Sensitivity Analysis**
- **Stochastic Simulation**
- **Chaos Control**

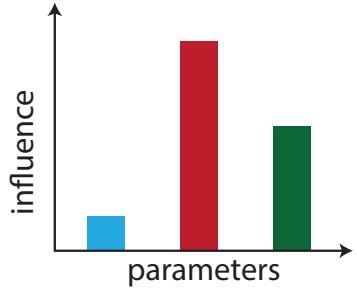
Initial Value Problem



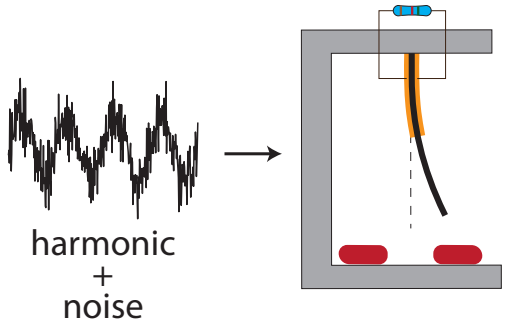
Dynamic Animation



Sensitivity Analysis

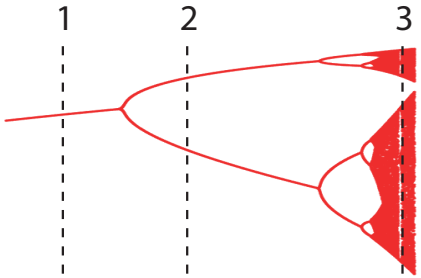


Stochastic Simulation

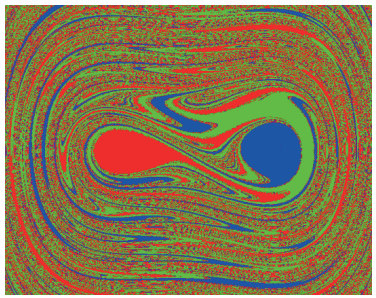


Nonlinear Tools

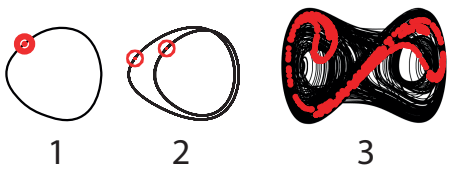
Diagram Bifurcation

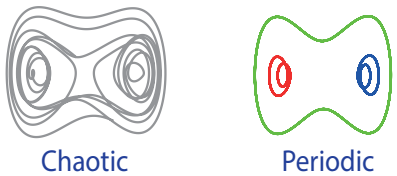


Basins of Attraction

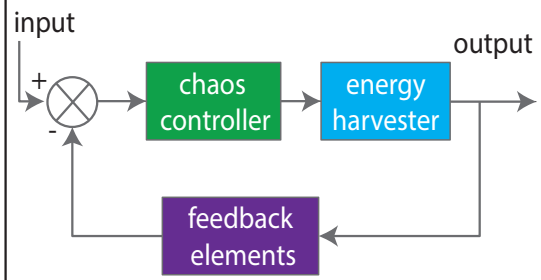


Poincare Section

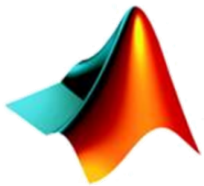




Chaos Control



Software



MATLAB

STONEHENGE is compatible with the proprietary software MATLAB© and C++.

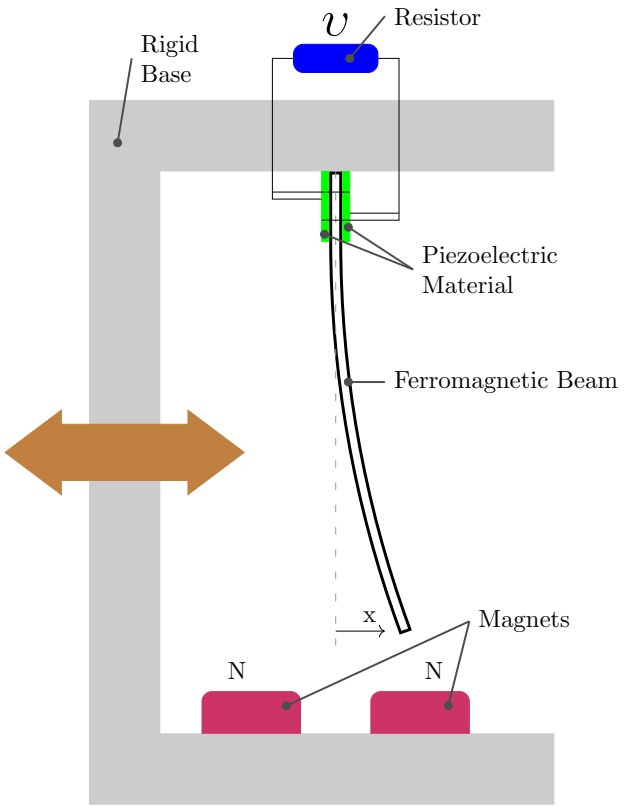
Summary

Bistable Energy Harvester

Bistable energy harvester models

1. Classical bistable energy harvester
2. Classical bistable energy harvester with nonlinear piezoelectric coupling
3. Asymmetric bistable energy harvester

Classical bistable energy harvester



Equation of motion (2-DOF):

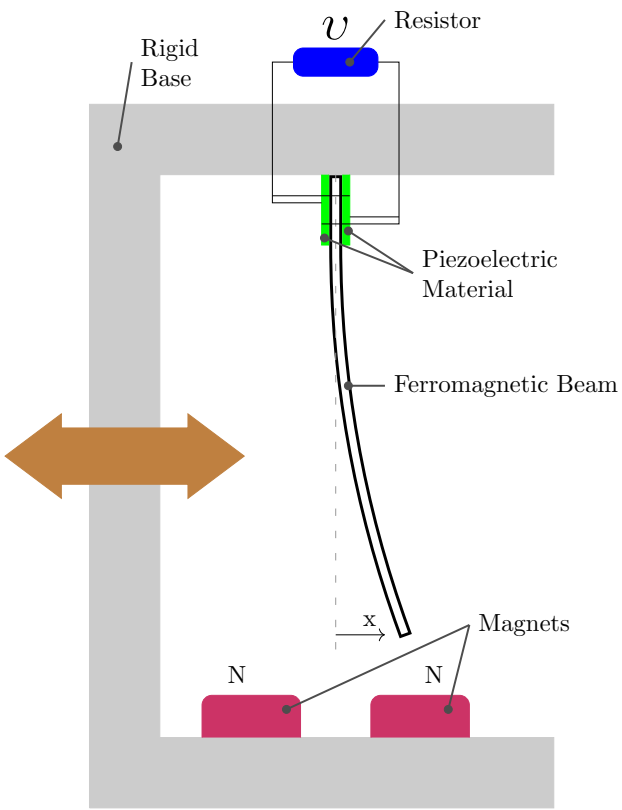
$$\ddot{x} + 2\xi\dot{x} - \frac{1}{2}x(1-x^2) - \chi v = f\cos(\Omega t)$$

$$\dot{v} + \lambda v + \kappa\dot{x} = 0$$

$$+ \text{ initial conditions}$$

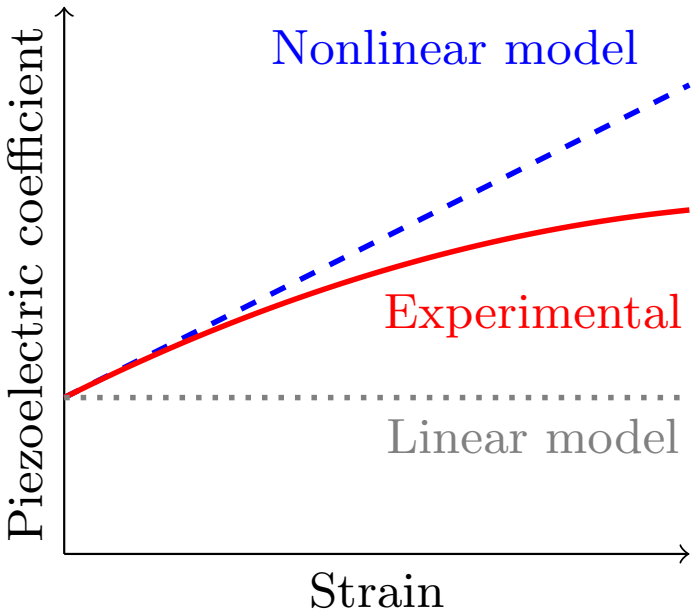
Average power:

$$P_{avg} = \frac{1}{T}\int_{t_i}^{t_i+T} \lambda\, v(t)^2 \; dt$$



- x: dimensionless displacement
- v: dimensionless voltage
- ξ: damping ratio
- χ: piezoelectric mechanical coupling
- f: amplitude excitation
- Ω: frequency excitation
- λ: reciprocal time constant
- κ: piezoelectric electrical coupling

Nonlinear electromechanical coupling



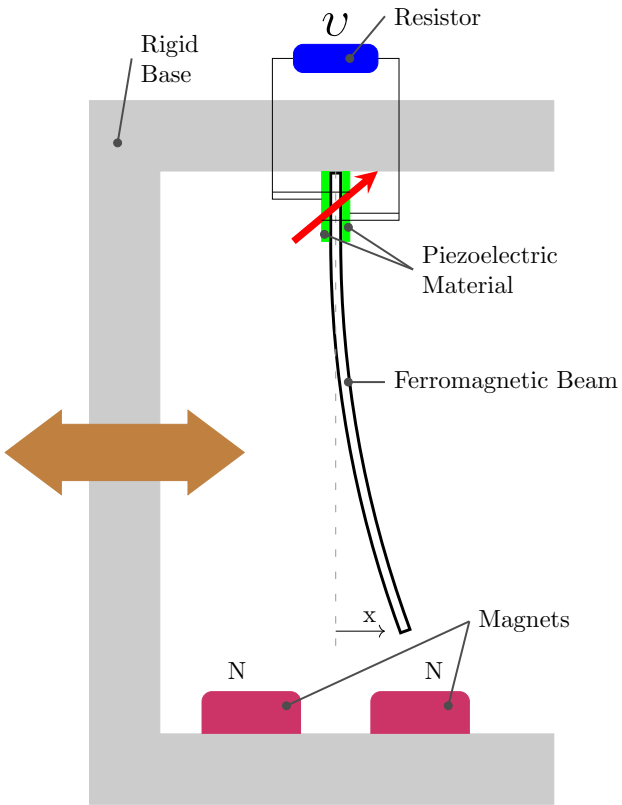
Nonlinear coupling by Triplett and Quinn (2009):

$$\widehat{\Theta}(x) = \theta(1 + \beta\,|x|)$$

- θ: Linear coupling
- β: Nonlinear coefficient coupling

A. Triplett and D. D. Quinn **The Effect of Non-linear Piezoelectric Coupling on Vibration-based Energy Harvesting**. J Intell Mat Syst Struct, 20(16), 1959–1967. (2009).

Bistable energy harvester with nonlinear coupling



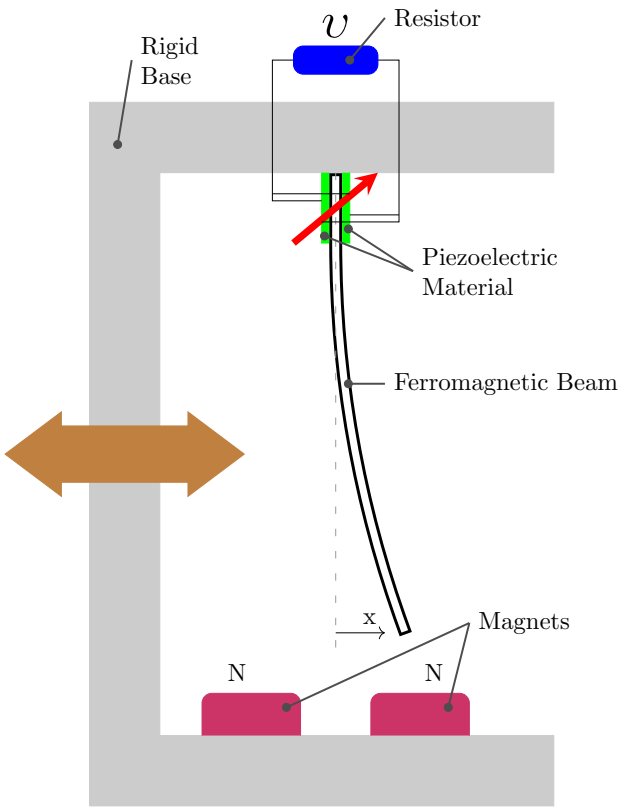
Equation of motion (2-DOF):

$$\ddot{x} + 2\xi\dot{x} - \frac{1}{2}x(1-x^2) - (1+\beta\,|x|)\chi v = f\cos(\Omega t)$$

$$\dot{v} + \lambda v + (1+\beta\,|x|)\kappa\dot{x} = 0$$

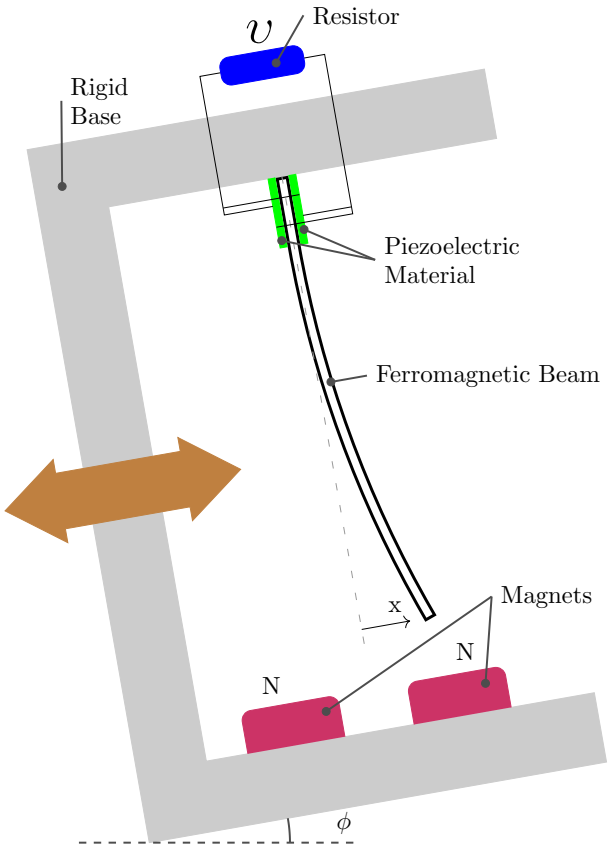
$+$ initial conditions

Note that whether $\beta = 0$, the electromechanical coupling has only the linear part.



- x: dimensionless displacement
- v: dimensionless voltage
- ξ: damping ratio
- χ: piezoelectric mechanical coupling
- f: amplitude excitation
- Ω: frequency excitation
- λ: reciprocal time constant
- κ: piezoelectric electrical coupling
- β: nonlinear coupling term

Asymmetric bistable energy harvester



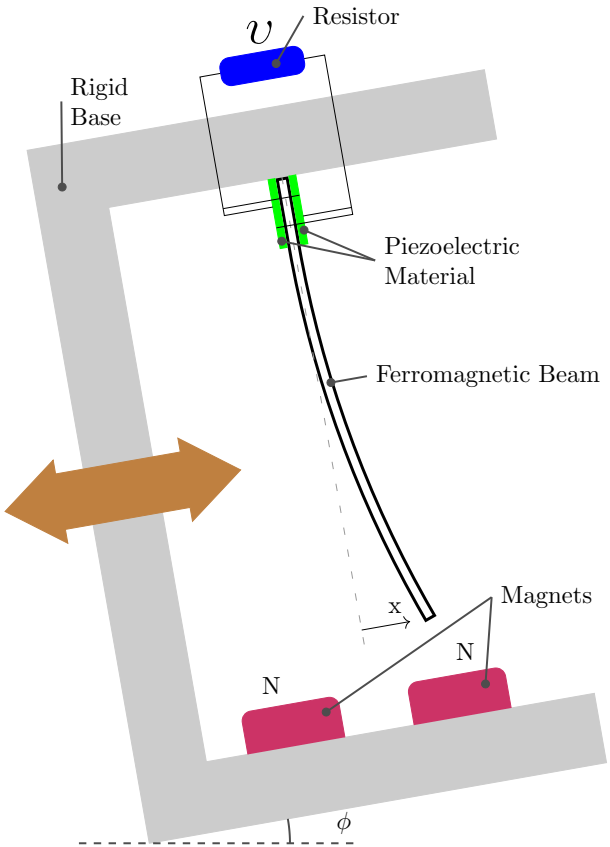
Equation of motion (2-DOF):

$$\ddot{x} + 2\xi\dot{x} - \frac{1}{2}x(1 + 2\delta x - x^2) - \chi v = f\cos(\Omega t) + p\sin\phi$$

$$\dot{v} + \lambda v + \kappa\dot{x} = 0$$

$+$ initial conditions

The electromechanical coupling can be added to this system in the same way has been done previously.



- x: dimensionless displacement
- v: dimensionless voltage
- ξ: damping ratio
- χ: piezoelectric mechanical coupling
- f: amplitude excitation
- Ω: frequency excitation
- λ: reciprocal time constant
- κ: piezoelectric electrical coupling
- δ: asymmetric coeff. of potential energy
- ϕ: bias angle

Initial Value Problem (IVP)

file .m: function for equation of motion

Function to solve equation of motion by **ode45** integrator method.

```
function [time,Y] = pmehna_eom(X,IC,tspan)
    % ...
    % ODE solver optional parameters
    opt = odeset('RelTol',1.0e-6,'AbsTol',1.0e-9);

    % ODE right hand side (Z = [z; zdot])
    dYdt = @(t,y) [y(2);
                   -2.*ksi.*y(2) + 0.5.*y(1).*(1.0+2*delta*y(1)-y(1).^2) + ...
                   (1+beta*abs(y(1)))*chi.*y(3) + f.*cos(Omega.*t)       + ...
                   p*sin(phi*pi/180);
                   -lambda.*y(3) - (1+beta*abs(y(1)))*kappa.*y(2)];

    % ODE solver Runge-Kutta45
    [time,Y] = ode45(dYdt,tspan,IC,opt);
```

Input data:

- X: model paramters
- IC: initial conditions vector
- tspan: time series sampling

Output data:

- time: time series
- Y: space-state response
  $Y = [x\ \dot{x}\ v]$

file .m: function to average power

This function computes the power of a piezo-magneto-elastic beam.

```
function [power,mean_power] = piezomagbeam_power(series,phys_param)

    % physical parameters
    lambda = phys_param.lambda;

    % vector input
    time  = series.time;
    Qvolt = series.Volt;

    % temporal interval of analysis
    T = time(end)-time(1);

    % output power
    power = lambda*Qvolt.^2;
    mean_power = (1/T)*trapz(time,power);

end
```

Input data:

- series: time-series response
- phys_param: model parameters

Output data:

- power: instantaneous power
- mean_power: average power

file .m: main equation of motion solve

The main file solves the equation of motion.

```
% defining parameter of the model
Xpar.ksi    = 0.01;  % mechanical damping ratio
Xpar.chi    = 0.05;  % dimensionless piezoeletric coupling term (mechanical)
Xpar.lambda = 0.05;  % reciprocal time constant
Xpar.kappa  = 0.5;   % dimensionless piezoeletric coupling term (eletrical)
Xpar.f      = 0.083; % amplitude of excitation
Xpar.Omega  = 0.8;   % frequency of excitation
Xpar.beta   = 0.0;   % nonlinear electromechanical coupling term
Xpar.delta  = 0.0;   % asymmetric coefficient of potential energy
Xpar.phi    = 0;     % bias angle

% time interval integration
tspan = 0:0.01:1000;

% initial condition
x0    = 1;
xdot0 = 0;
v0    = 0;
IC    = [x0 xdot0 v0];

% system response function
[time,Y] = pmeh_eom(Xpar,IC,tspan);

% serial time response
Qdisp = Y(:,1);      % displacement-time of system
Qvelo = Y(:,2);      % velocity-time of system
Qvolt = Y(:,3);      % voltage-time of system
```

Models selection

If adding a value for the following parameters the model change:

$\beta \neq 0 \longrightarrow$ add nonlinear piezoelectric coupling

$\delta \neq 0 \longrightarrow$ add asymmetric coefficient of potential energy

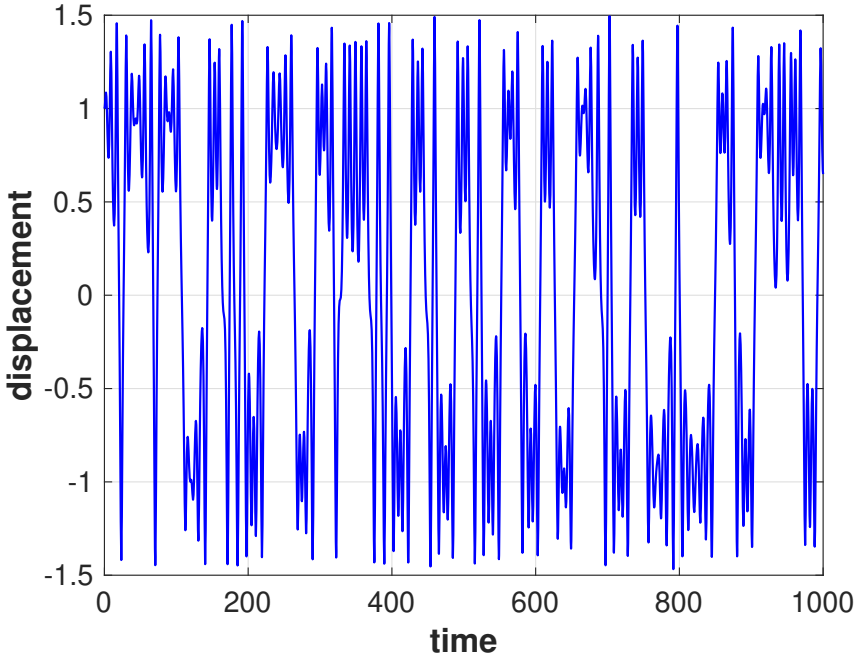
$\phi \neq 0 \longrightarrow$ add bias angle

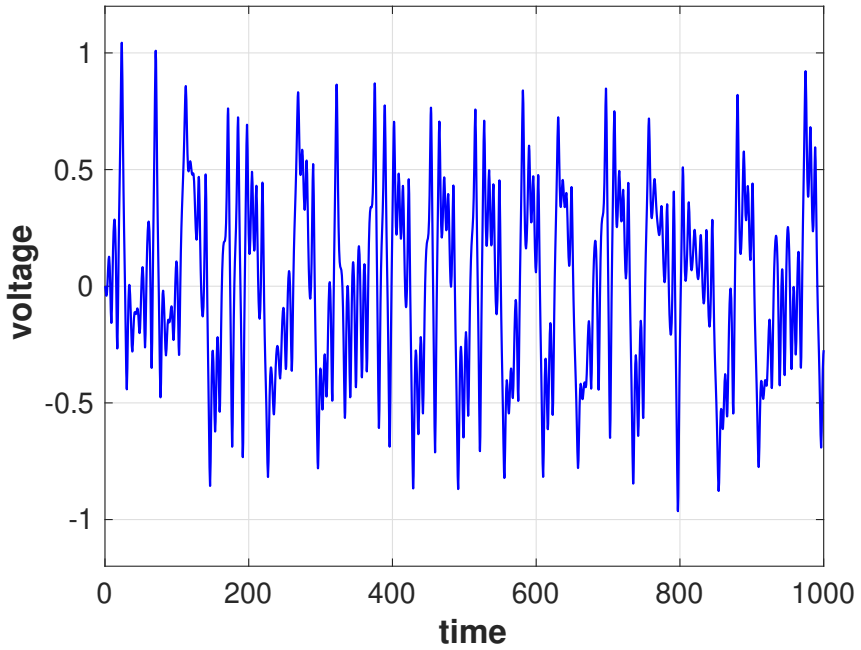
file .m: plot time-serial response

To visualize the time-serial is plot the system response.

```
% post-processing (plot)
% plot displament
figure(1)
plot(time,Qdisp,'b','LineWidth',1)
set(gca, 'FontName' , 'Arial' , 'FontSize' ,13 );
xlabel('     time      ', 'FontSize' , 15 , 'FontWeight' , 'bold');
ylabel(' displacement ', 'FontSize' , 15 , 'FontWeight' , 'bold');
grid

% plot voltage
figure(2)
plot(time,Qvolt,'b','LineWidth',1)
set(gca, 'FontName' , 'Arial' , 'FontSize' ,13 );
xlabel('    time   ', 'FontSize' , 15 , 'FontWeight' , 'bold');
ylabel('  voltage ', 'FontSize' , 15 , 'FontWeight' , 'bold');
grid
```

Graph of the time-serial response





Animation

Function: plot_piezomagbeam_animation_inertial.m

This function plot the animation of bistable energy harvesting.
This code is divided into main three parts:

- assigning the dimension, values, and coordinates of parameters
- cantilever beam mode shape function
- loop for plotting animation of each time-step

file .m: Function plot animation inertial frame

Defining a function to plot the animation of a bistable energy harvester

```
function plot_piezomagbeam_animation_inertial(series,f,Omega,dim,beta,IC)
    ⋮
```

Input data:

- series: time-series response
- f: amplitude of excitation
- Omega: frequency of excitation
- dim: limited dimension
- IC: initial condition

Canteliver beam mode shape function

The mode shapes for a continuous cantilever beam is given as

$$\phi_1 = \sinh(\beta_1 \mathcal{Z}) - \sin(\beta_1 \mathcal{Z}) - \frac{\sinh(\beta_1) + \sin(\beta_1)}{\cosh(\beta_1) + \cos(\beta_1)}(\cosh(\beta_1 \mathcal{Z}) - \cos(\beta_1 \mathcal{Z}))$$

where β_1 is a constant and $\mathcal{Z}$ is the mesh.

```
% mesh for beam mode shape
zmesh = beam_heigth*linspace(0.0,1.0,10);

% beam mode shape
b1 = (0.5*pi + 0.3042)*300e-3;
phi1 = sinh(b1*zmesh) - sin(b1*zmesh) - ...
       ((sinh(b1)+sin(b1))/(cosh(b1)+cos(b1)))*(cosh(b1*zmesh)-cos(b1*zmesh));
```

Loop for plotting animation of each time-step

To animate the graph is created Loop, such that each step is plotted the system states.

```
% number of time steps
Ndt = length(time);
% number of jump time steps
Njump = 50;

% loop to construct the video
    for n=1:Njump:Ndt
        % ...
        % processing plot
        % ...
    end
```

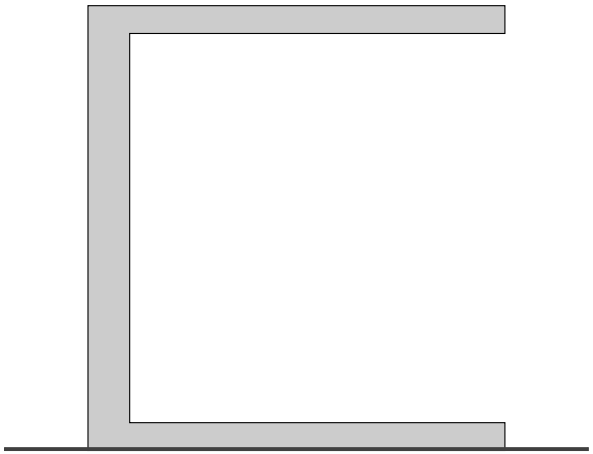
Inside the loop has all time-dependent variables and plot functions.
The figure has three subplot:

- **sp1**
- **sp2**
- **sp3**

processing plot (part 1)

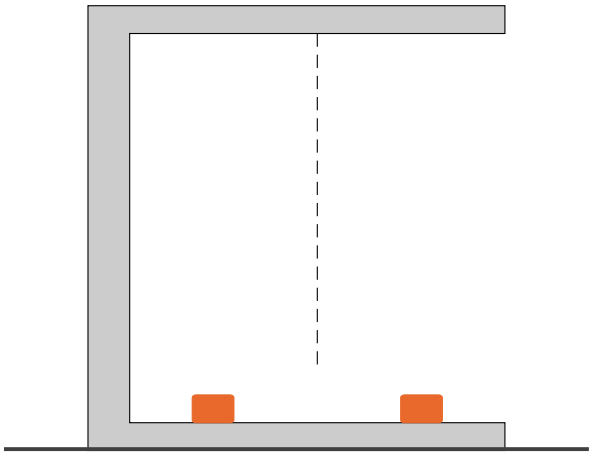
“sp1” is the subplot of system graphs

```
sp1 = subplot(3,1,1);

% draw rigid base bottom part
fh01 = fill(xrb,yrb,[0.8,0.8,0.8]);

hold on

% draw ground
fh02 = rectangle('Position',
                 [xv1_0,ymin,1.4,0.01], ...
                  'EdgeColor',
                 [0.25 0.25 0.25]      , ...
                  'FaceColor',
                 [0.25 0.25 0.25] );
```



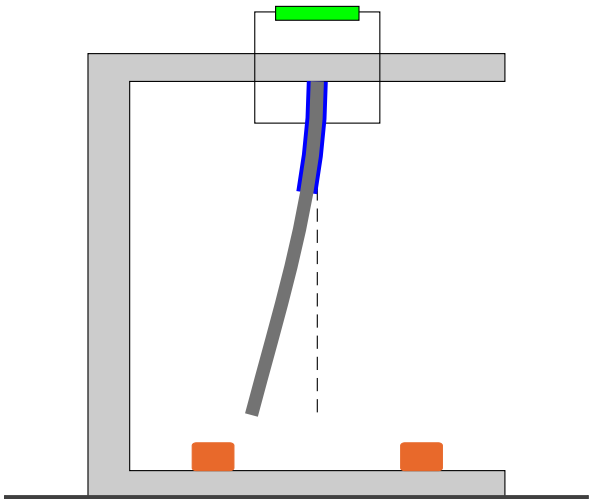
processing plot (part 2)

```
% draw dashed vertical line
fh03 = plot([xvline1 xvline2],
            [yvline1 yvline2], 'k--');

% draw magnets
fh04 = rectangle('Position' ,
                 [xm1                ym1,
                 0.1*rb_length 0.1*rb_length],
                 'EdgeColor'              , ...
                 [0.9100 0.4100 0.1700] , ...
                 'FaceColor'              , ...
                 [0.9100 0.4100 0.1700] , ...
                 'Curvature',    0.2);
```



processing plot (part 3)

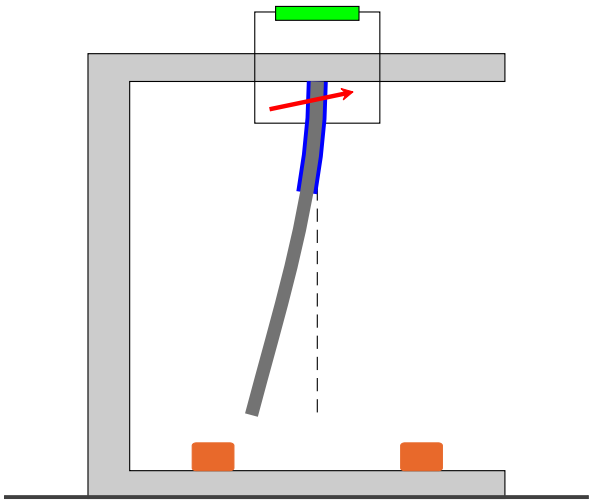
```
% draw resistor
fh06 = rectangle('Position',
                 [xr2          ,      yr2      ,
                 0.3*rb_length,0.4*rb_length]);

fh07 = rectangle('Position',
                 [xr1          ,      yr1,      ,
                 0.2*rb_length,.05*rb_length] ,
                 'FaceColor'   , [0,1,0]       )
                      ;

% draw pzt
fh08 = plot(xvline1+phi1(1:4)*Qdisp(n) ,
            yvline1 - zmesh(1:4)       , ...
            '-','color',  [0,0,1]      , ...
            'LineWidth',  9.5          );

% draw beam
fh09 = plot(xvline1+phi1*Qdisp(n)      ,
            yvline1 - zmesh            , ...
            'color' , [0.45,0.45,0.45] , ...
            'LineWidth', 6             );
```



processing plot (part 4)

If the nonlinear electromechanical coupling is considered, adds:

```
% nonlinear coupling
fh10 = annotation('arrow',
        [0.52 - cos_Omega_t*0.17
         0.61 - cos_Omega_t*0.17]    , ...
        [0.81  0.835]                , ...
        'color'      ,     [1,0,0]   , ...
        'HeadLength' , 5             , ...
        'HeadWidth'  , 7             , ...
        'LineWidth'  , 2             );
```



processing plot (part 5)

“sp2” and **“sp2”** are displacement and voltage time-series, respectively:

```
% plot displacement time series
sp2 = subplot(3,1,2);
fh11 = plot(time    , Qdisp      , 'b-', ...
            time(n), Qdisp(n)   , 'or', ...
            'MarkerFaceColor'   , 'r' , 'MarkerSize',5);
set(gca,'FontSize',16);

% set axis
ylabel('displacement','FontSize',14,'FontName','Helvetica')
set(gca,'xtick',[])
ylim([-2 2])

% plot voltage time series
sp3 = subplot(3,1,3);
fh12 = plot(time    , Qvolt    , 'b-', ...
            time(n), Qvolt(n), 'or', ...
            'MarkerFaceColor',  'r', 'MarkerSize',5);
set(gca,'FontSize',16);

% set axis
xlabel('time','FontSize',14,'FontName','Helvetica')
ylabel('voltage','FontSize',14,'FontName','Helvetica')
volt_max = max(Qvolt);
ylim([-3 3])
```

Main file: main_piezomagbeam_ivp.m

This file is the main file for a program that simulates the nonlinear dynamics of a piezo-magneto-elastic beam.

This code is divided into main three parts:

- input parameters
- run pmeh_eom.m
- run plot_piezomagbeam_animation_inertial.m

Input parameters

Model parameters, initial conditions, and integration time interval

```
% defining parameter of the model
Xpar.ksi    = 0.01;  % mechanical damping ratio
Xpar.chi    = 0.05;  % dimensionless piezoeletric coupling term (mechanical)
Xpar.lambda = 0.05;  % reciprocal time constant
Xpar.kappa  = 0.5;   % dimensionless piezoeletric coupling term (eletrical)
Xpar.f      = 0.083; % amplitude of excitation
Xpar.Omega  = 0.8;   % frequency of excitation

% time interval integration
tspan = 0:0.01:1000;

% initial condition
x0    = 1;
xdot0 = 0;
v0    = 0;
IC    = [x0 xdot0 v0];
```

Run pmeh_eom.m

Solving the initial value problem

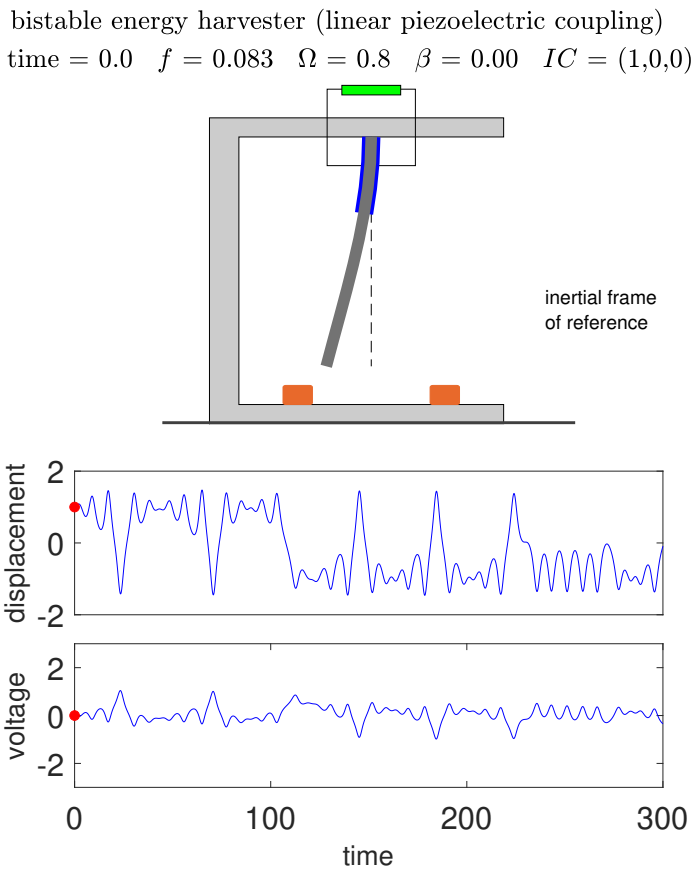
```
% system response function
[time,Y] = pmeh_eom(Xpar,IC,tspan);

% serial time response
Qdisp = Y(:,1);       % displacement-time of system
Qvelo = Y(:,2);       % velocity-time of system
Qvolt = Y(:,3);       % voltage-time of system

% Sctruct input
series.time = time;
series.Disp = Qdisp;
series.Velo = Qvelo;
series.Volt = Qvolt;
```

Run plot_piezomagbeam_animation_inertial.m

Computing the animation plot

```
% Dimension graphs
dim.xmin    = -1;
dim.xmax    =  1;
dim.ymin    = -1;
dim.ymax    =  1;

% run function of animation
plot_piezomagbeam_animation_inertial(series,  Xpar.f   , Xpar.Omega,...
                                              dim  , Xpar.beta, IC          );
```

Animation



Global sensitivity analysis of (a)symmetric energy harvesters (2021)
J. P. Norenberg, A. Cunha Jr, S. da Silva, P. S. Varoto

Press "Enter" to start the animation

Options

There are another two options to plot:

- plot_piezomagbeam_animation_mobile.m
  The animation is on the mobile frame of reference.
- plot_piezomagbeam_animation_mobile_gif.m
  Animation is plotted as gif archive with grater time increment.

Remarks

For asymmetric model is necessary to use:

- main_piezomagbeam_asymmetric_ivp.m
  The main file animates an asymmetric bistable energy harvester.
- plot_piezomagbeam_animation_mobile_asymmetric.m
  The animation is on mobile frame of reference.
- plot_piezomagbeam_animation_inertial_asymmetric.m
  The animation is on inertial frame of reference.

processing plot

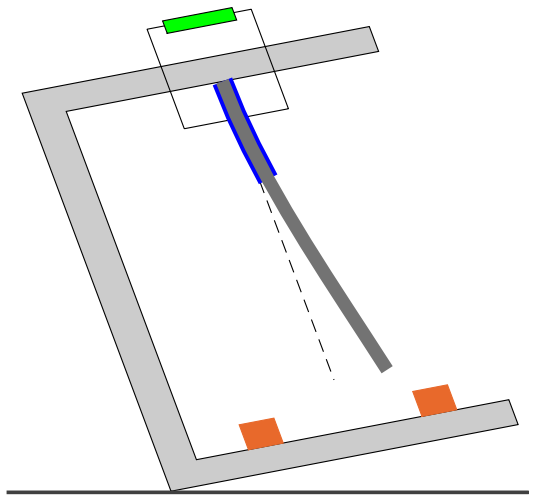
The code works by:

Rotation matrix:

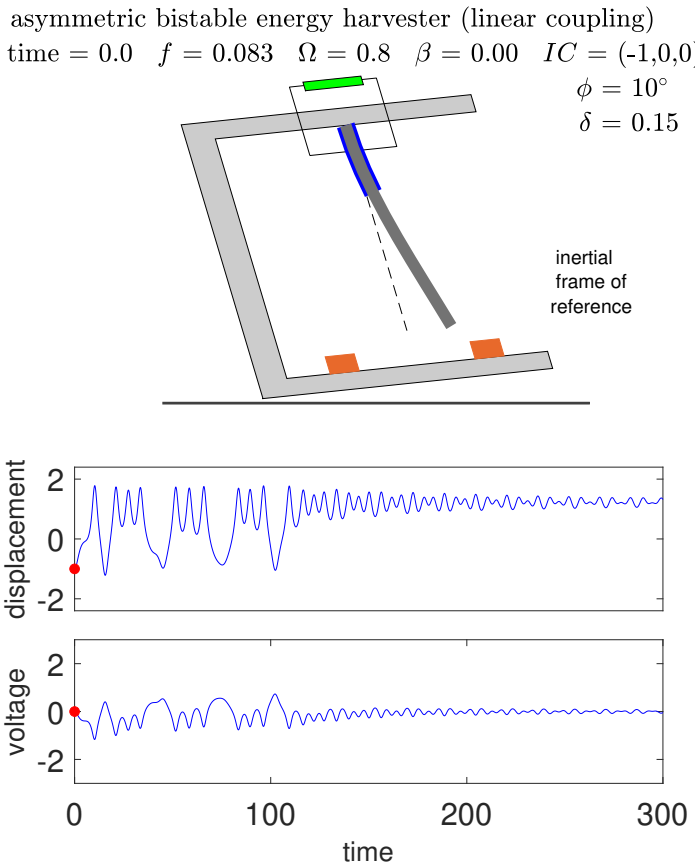
$$T = \begin{bmatrix} cos(\phi\frac{\pi}{180}) & -sin(\phi\frac{\pi}{180}) \\ sin(\phi\frac{\pi}{180}) & cos(\phi\frac{\pi}{180}) \end{bmatrix}$$

T multiply the coordinate vector of:

- rigid base
- dashed vertical line
- magnets
- resistor and circuit
- piezoelectric layers and beam



Animation



Global sensitivity analysis of (a)symmetric energy harvesters (2021)
J. P. Norenberg, A. Cunha Jr, S. da Silva, P. S. Varoto

Press "Enter" to start the animation

Nonlinear Analysis Tools

Which tools are we using?

1. **Poincaré Section**
2. **Bifurcation Diagram**
3. **Basins of attraction**

Poincaré Section

Poincaré Section

The **Poincaré section** is the intersection of a periodic orbit in the state space of a dynamical system.

Two codes run to plot the Poincaré section. Also, it is plotted with phase portrait.

- *poincare.m*: function to compute the Poincaré section.
- *phase_portrait.m*: function to compute the phase portrait.
- *main_PhasePortrait_Poincare*: main code to run the phase portrait and Poincaré section functions.

Function: poincare.m

This function calculates the Poincaré section of the bistable energy harvester.

function name file: $[poincare\_disp, poincare\_velo] = poincare(Xpar, IC)$

Input data:

- Xpar: model parameters
- IC: initial conditions

Output data:

- poincare_disp: Poincaré map of displacement
- poincare_velo: Poincaré map of velocity

Function: poincare.m

First of all, it checks the number of arguments:

```
% check number of arguments
    if nargin > 2
        error('Too many inputs.')
    elseif nargin < 2
        error('Too few inputs.')
    end
```

The integration time is determined by a forcing cycle period:

```
% period of a forcing cycle
T = 2*pi/Omega;
% number of forcing cycles
Nf =  5000;
% initial dimensionless time
t0 = 0.0;
% final dimensionless time
t1 = t0 + Nf*T;
% number of samples per forcing cycle
Nsamp = 10000;
% time series sampling points
tspan = t0:(T/Nsamp):t1;
```

Function: poincare.m

To solve the equation of motion:

```
% ODE solver Runge-Kutta45
    [time,y] = piezomagbeam_asymmetric(Xpar,IC,tspan);
```

The time-series:

```
% time series of dimensionless displacement
Qdisp = y(:,1);

% time series of dimensionless velocity
Qvelo = y(:,2);

% time series of dimensionless voltage
Qvolt = y(:,3);
```

Function: poincare.m

Finally, only the steady-state is required:

```
% number of dimensionless time steps
Ndt = length(time);

% number of steps for steady state
Nss = round(0.99*Ndt);

% number of steps to initiates Poincare map
Npm = round(0.10*Ndt);

% Poincare maps
poincare_disp = Qdisp(Npm:Nsamp:Ndt);
poincare_velo = Qvelo(Npm:Nsamp:Ndt);
```

Function: phase_portrait.m

This function computes the phase portrait.

function name file: $[disp, vel] = phase\_portrait(Xpar, IC)$

Input data:

- Xpar: model parameters
- IC: initial conditions

Output data:

- disp: displacement time-series (only the steady-state)
- vel: velocity time-series (only the steady-state)

Function: phase_portrait.m

It is checked the number of arguments:

```
% check number of arguments
    if nargin > 2
        error('Too many inputs.')
    elseif nargin < 2
        error('Too few inputs.')
    end
```

Then, determine the input parameters:

```
% number of forcing cycles
    Nf =  10000;

% initial dimensionless time
    t0 = 0.0;

    t1 = t0 + Nf;

    tspan = t0:0.001:t1;
```

Function: phase_portrait.m

To solve the equation of motion:

```
% ODE solver Runge-Kutta45
    [time,y] = piezomagbeam_asymmetric(Xpar,IC,tspan);
```

The time-series:

```
% Output of function, only with the steady state
    disp = y(round(0.5*end):end,1);
    vel  = y(round(0.5*end):end,2);
```

main file: main_PhasePortrait_Poincare.m

This is the main file for a program that responds to the phase portrait and Poincaré section for the piezo-magneto-elastic beam.

The main file is divided into two sections:

- Processing: computes the phase portrait and Poincaré section
- Plotting

Processing

Input data of Poincaré function is defined:

```
% physical parameters:
Xpar.ksi    = 0.01;     % mechanical damping ratio
Xpar.chi    = 0.05;     % dimensionless piezoeletric coupling term (mechanical)
Xpar.lambda = 0.05;     % dimensionless time constant reciprocal
Xpar.kappa  = 0.5;      % dimensionless piezoeletric coupling term (eletrical)
Xpar.Omega  = 0.8;      % dimensionless excitation frequency
Xpar.beta   = 0;        % nonlinear electromechanical coupling
Xpar.delta  = 0;        % asymmetric coefficient of potential energy
Xpar.phi    = 0;        % bias angle (degree)

% initial conditions
IC      = [1,0,0];

% range of amplitude excitation
frang = [0.041  0.06  0.083  0.091  0.105  0.115  0.147  0.200  0.250];
```

Processing

Then, computes the phase portrait and Poincaré section for each condition of excitation.

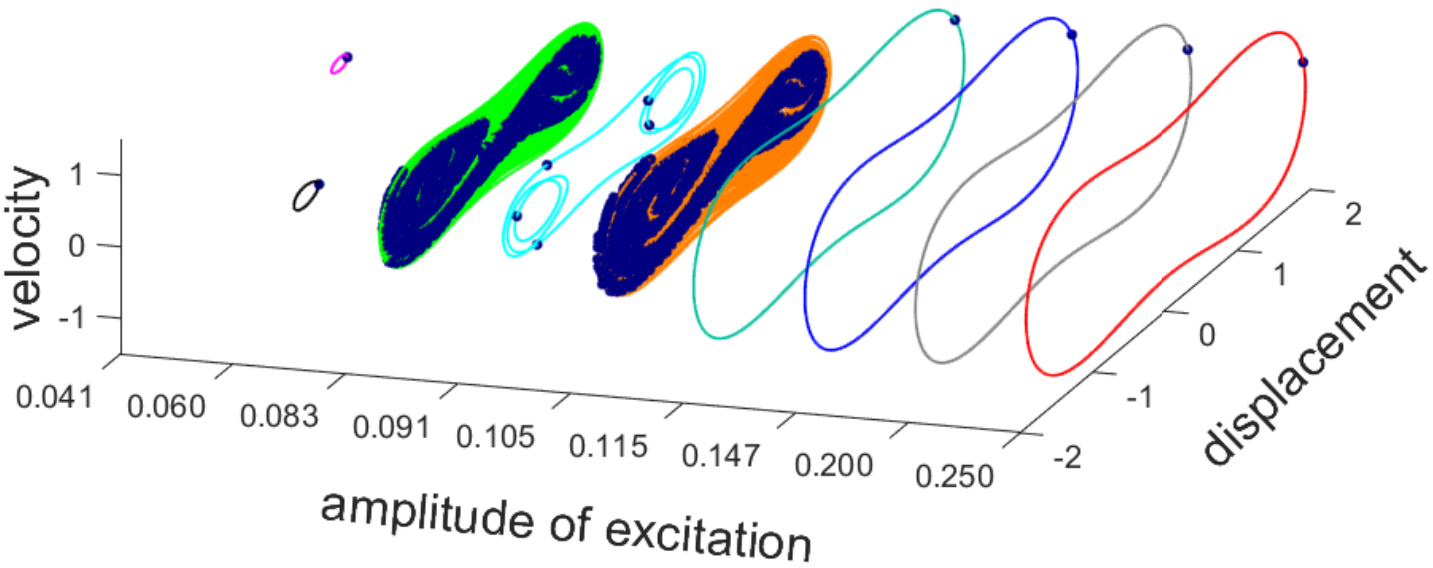
```
% Loop to compute Phase Portait and Poincare Section
for k = 1:length(frang)
    Xpar.f = frang(k);
    [displ(:,k),velo(:,k)] = phase_portait(Xpar,IC);
    [poincare_disp(:,k),poincare_velo(:,k)] = poincare(Xpar,IC);
end

% Save struct variables
Phase_Poincare.poincare_disp = poincare_disp;
Phase_Poincare.poincare_velo = poincare_velo;
Phase_Poincare.displ         = displ;
Phase_Poincare.velo          = velo;
Phase_Poincare.frang         = frang;
```

Plotting

Now it plots the phase portrait and Poincaré section:

```
% Ploting
fh1 = figure(1);
set(fh1, 'Position', [488 342 600 360])

% ... setup plotting ...

% Loop to plotting 3D phase portait and Poincare section
for k = 1:length(frang)
    plot3(k*ones(length(displ(2/3*end:end,k)),1),displ(2/3*end:end,k), ...
          velo(2/3*end:end,k),'.','color',colorMap(k,:),'MarkerSize',2 )
    hold on
    plot3(k*ones(length(poincare_displ(:,k)),1) ,poincare_displ(:,k) , ...
          poincare_velo(:,k) ,'.','color',   [0 0 .5]   ,'MarkerSize',10)
    hold on
end

view(18,50)
xticklabels({'0.041'  , '0.060' , '0.083' , '0.091' , ' 0.105 ' , ...
             '0.115 ' , '0.147' , '0.200' , '0.250' })

xlabel('amplitude of excitation', 'FontSize',15,'FontName','Helvetica');
ylabel(    'displacement'       , 'FontSize',15,'FontName','Helvetica');
zlabel(      'velocity'         , 'FontSize',15,'FontName','Helvetica');
```

Phase portrait and Poincaré section

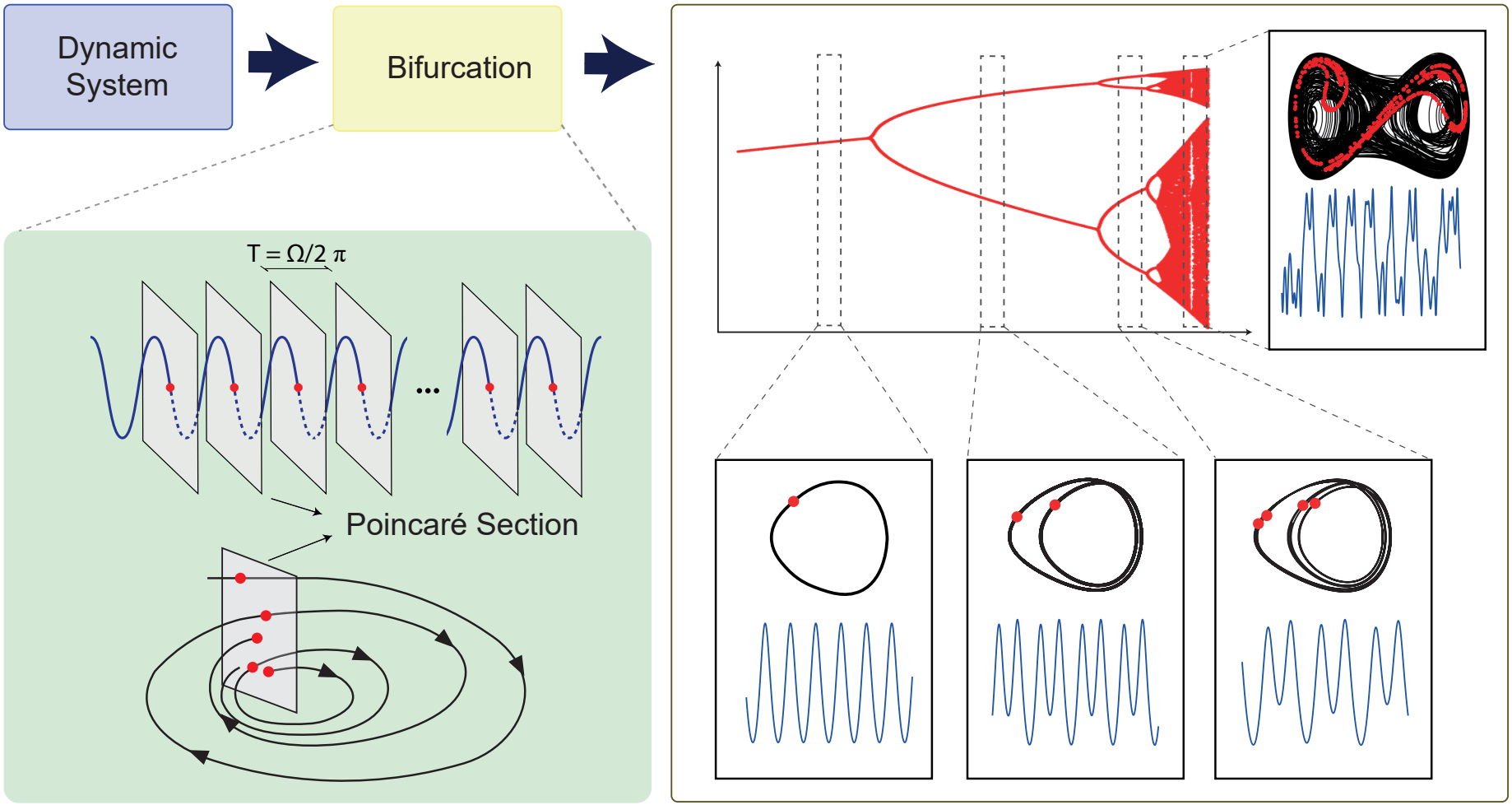


Bifurcation Diagram

Bifurcation Diagram

Bifurcation diagram is a sweep up and down of Poincaré section by throughout control parameter.

It is a visual summary of the succession of period-doubling produced as control parameters vary.


Dynamic System
Bifurcation
T = Ω/2 π
Poincaré Section

Bifurcation Diagram

The bifurcation diagram is divided into three codes:

- *bifurcation_3d*: function to compute the Poincaré section for each control parameter.
- *plot_3D_bifurcation*: function to plot the bifurcation diagram.
- *main_3D_bifurcation*: main code to run the bifurcation and plotting.

Function: bifurcation_3d.m

This function defines the bifurcation diagram for amplitude and frequency of excitation as a control parameter.

function name file: $[bifurc\_inf] = bifurcation\_3d(var\_input, name\_file)$

Input data:

- var_input: input parameter in the system
- name_file: name of the file save

Output data:

- bifurc_inf: struc variable of the bifurcation diagram

Function: bifurcation_3d.m

Initially, it checks the number of arguments and fields in a struct:

```
% check number of arguments
    if nargin > 2
        error('Too many inputs.')
    elseif nargin < 2
        error('Too few inputs.')
    end

% check number of field in struct
    if length(struct2cell(var_input)) < 5
        error('Too few inputs in struct')
    elseif length(struct2cell(var_input)) > 5
        error('Too many inputs in struct')
    end

% check number of parameters in field struct X_params
    if length(struct2cell(var_input.X_params)) < 5
        error('Too few inputs in parameters model')
    elseif length(struct2cell(var_input.X_params)) > 5
        error('Too many inputs in parameters model')
    end
```

Function: bifurcation_3d.m

Next, it defines the input parameter:

```
% physical parameters
    ksi    = var_input.X_params.ksi;    % mechanical damping ratio
    chi    = var_input.X_params.chi;    % piezoeletric coupling term (mechanical)
    lambda = var_input.X_params.lambda; % time constant reciprocal
    kappa  = var_input.X_params.kappa;  % piezoeletric coupling term (eletrical)
    beta   = var_input.X_params.beta;   % nonlinear electromechanical coupling
    delta  = var_input.X_params.delta;  % asymmetric coefficient
    phi    = var_input.X_params.phi;    % bias angle

% bifurcation control parameters
    Omega_int = var_input.Par1_rang.Omega_rang; % Omega interval
    num_omega = var_input.N1_rang.N_omega;      % Size of Omega interval
    f_int     = var_input.Par2_rang.f_int;      % f interval
    int_param = var_input.N2_rang.int_param;    % Increment of f interval

% range of frequency of excitation
    Omega_rang = linspace(min(Omega_int),max(Omega_int),num_omega);
```

Function: bifurcation_3d.m

Then starts computing the Poincaré for each control parameter step. The first loop is for the frequency of excitation.

```
% loop for Omega bifurcation
for k = 1:length(Omega_rang)
    % Omega value in step
    Omega = Omega_rang(k);

    % vector of sweeping up f
    f_rang_up = min(f_int):int_param:max(f_int);

    % initial condition
    x0 = [1,0,0];
```

Then there are two loops for the amplitude of excitation (sweeping up and down).

Function: bifurcation_3d.m

The amplitude excitation by sweeping up is computed:

```
% loop for bifurcation processing 1 (sweeping up)
for i = 1:length(f_rang_up)
    % f value in step
    f = f_rang_up(i);

    % ... same processing as in poincare section ...

    % vector of bifurcation point
    Bifurc_up(:,i,k) = Y1(Nss:Nsamp:Npp,3);

    % rewrite initial condition
    x0 = [Y1(end,1) Y1(end,2) Y1(end,3)];
end
```

Note that for each amplitude step the initial condition is up to date by the final condition of the last step to decrease the transient regime.

Function: bifurcation_3d.m

The amplitude excitation by sweeping down is computed:

```
        % vector of sweeping down f
        f_rang_down = max(f_int):-int_param:min(f_int);

        % loop for bifurcation processing 2 (sweeping down)
        for i = 1:length(f_rang_down)
            % f value in step
            f = f_rang_down(i);

            % ... same processing as in poincare section ...

            % vector of bifurcation point
            Bifurc_down(:,i,k) = Y1(Nss:Nsamp:Npp,3);

            % rewrite initial condition
            x0 = [Y1(end,1) Y1(end,2) Y1(end,3)];
        end
    end
```

Note that the first "for" is ended.

Function: bifurcation_3d.m

Finally, the bifurcation processing is concluded saving the output data.

```
% save data
bifurc_inf.Bifurc_up   = Bifurc_up;
bifurc_inf.Bifurc_down = Bifurc_down;
bifurc_inf.Omega_rang  = Omega_rang;
bifurc_inf.f_rang_up   = f_rang_up;
bifurc_inf.f_rang_down = f_rang_down;

save(name_file,'bifurc_inf')
```

Function: plot_3D_bifurcation.m

This function plot the bifurcation diagram by graph 3D.

function name file: $[fig] = plot\_3D\_bifurcation(bifurc\_inf, name\_file)$

Input data:

- inf_input: struc variable of the bifurcation diagram
- name_file: name of the file save

Output data:

- bifurc_inf: figure

Function: plot_3D_bifurcation.m

First of all, is check the number of arguments:

```
% check number of arguments
if nargin > 2
    error('Too many inputs.')
elseif nargin < 2
    error('Too few inputs.')
end
```

Next is extracted the variables within the struct one:

```
% bifurcation input parameters
Bifurc_up   = bifurc_inf.Bifurc_up;
Bifurc_down = bifurc_inf.Bifurc_down;
Omega_rang  = bifurc_inf.Omega_rang;
f_rang_up   = bifurc_inf.f_rang_up;
f_rang_down = bifurc_inf.f_rang_down;
```

and is generate a 2D mesh grid and colormap vector:

```
% generate 2D mesh grid
[X1,Y1] = meshgrid(f_rang_up,Omega_rang);
[X2,Y2] = meshgrid(f_rang_down,Omega_rang);
% color map
colorMap = winter(5);
colorMap2= autumn(5);
```

Function: plot_3D_bifurcation.m

Finally is ploted the 3D bifurcation by a loop for plotting each bifurcation:

```
% loop for plotting each bifurcation
    for k = 1:5
        plot3(X1(k,:),Y1(k,:),Bifurc_up(:,:,k),'.',
              'color', colorMap(k,:) , 'MarkerSize', 4.2)
        hold on
        plot3(X2(k,:),Y2(k,:),Bifurc_down(:,:,k),'.',
              'color', colorMap2(k,:), 'MarkerSize', 4.2)
    end
```

The next lines of the code are just to set up the plot and save the figure:

```
% set-up plot
% labels, grids, view, axis limits, etc.
% ...
saveas(fig,name_file,'eps');
```

main file: main_3D_bifurcation.m

This is the main file for a program that traces a bifurcation diagram for the piezo-magneto-elastic beam.

The main file is divided in two section:

- Processing: computes the bifurcation
- Plotting

Processing

The input data is attributed:

```
ksi    = 0.01;    % mechanical damping ratio
chi    = 0.05;    % dimensionless piezoeletric coupling term (mechanical)
lambda = 0.05;    % dimensionless time constant reciprocal
kappa  = 0.5;     % dimensionless piezoeletric coupling term (eletrical)
beta   = 0;       % nonlinear electromechanical coupling term
delta  = 0;       % asymmetric coefficient of potential energy
phi    = 0;       % bias angle

% struct variable physical parameters
var_input.X_params.ksi    = ksi;
var_input.X_params.chi    = chi;
var_input.X_params.lambda = lambda;
var_input.X_params.kappa  = kappa;
var_input.X_params.beta   = beta;
var_input.X_params.delta  = delta;
var_input.X_params.phi    = phi;
```

Note that only the excitation conditions are not defined because they are control parameters defined in bifurcation conditions.

Processing

Bifurcation conditions is defined:

```
% min and max of frequency
var_input.Par1_rang.Omega_rang = [0.64  0.96];

% size frequency vector
var_input.N1_rang.N_omega = 5;

% min and max of amplitude
var_input.Par2_rang.f_int = [0.02  0.30];

% increment of amplitude
var_input.N2_rang.int_param = 0.001;
```

Next, the file name to save the bifurcation data:

```
% file name
name_file = ['bifurc_3D_beta', num2str(beta*10)];
```

Finally, it is computed the bifurcation by function:

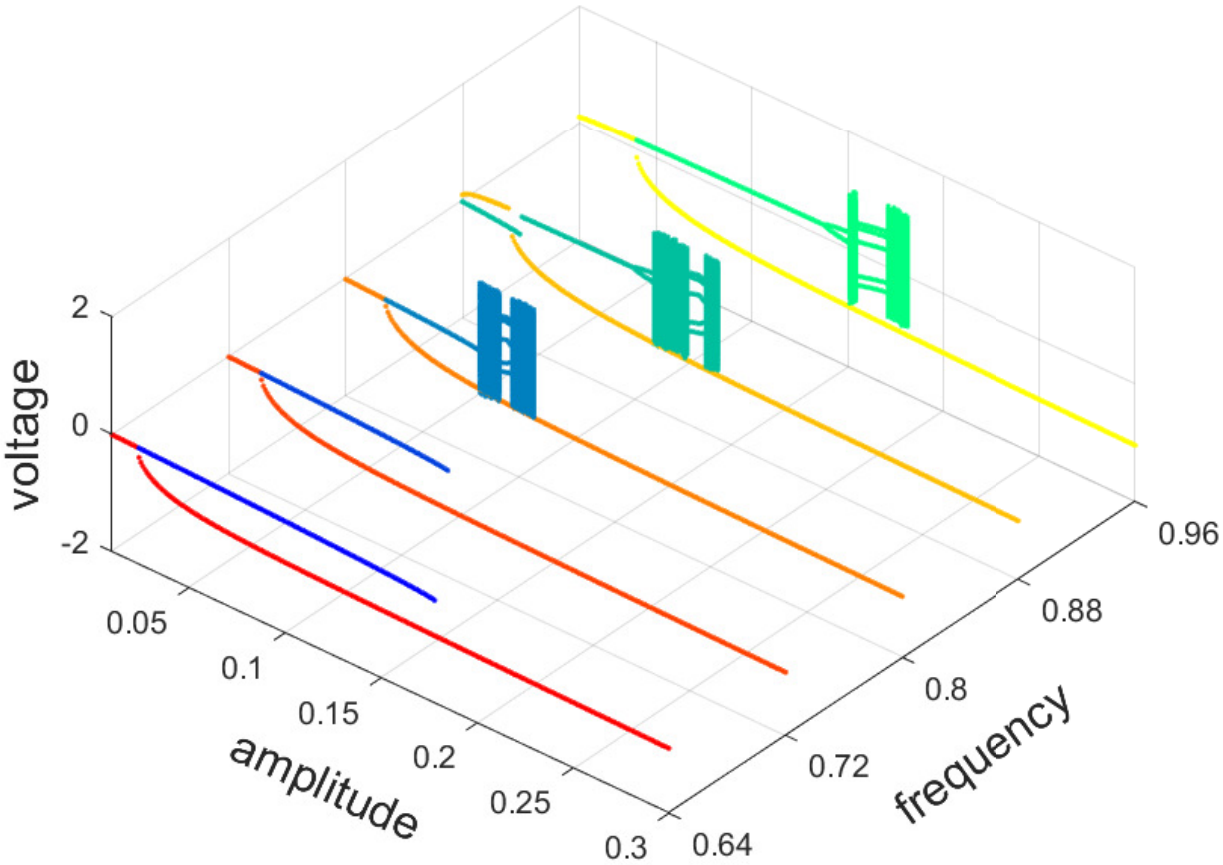
```
% compute bifurcation
[bifurc_inf] = bifurcation_3d(var_input,name_file);
```

Plotting

The bifurcation data is ploted using the bifurcation function:

```
% plotting bifurcation (graph 3-D)
name_file = 'teste';
fig = plot_3D_bifurcation(bifurc_inf,name_file);
```

Bifurcation diagram



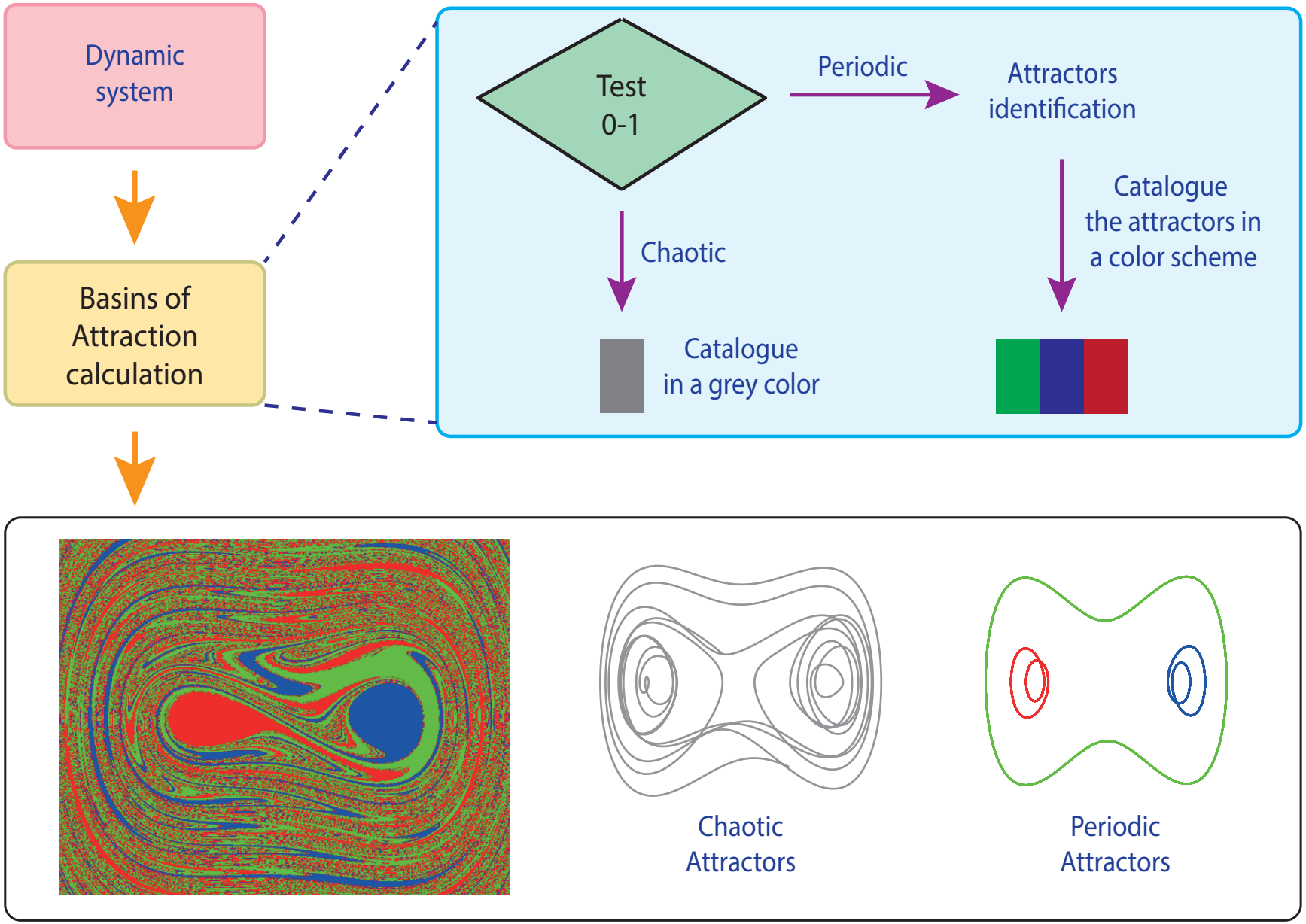
Basins of attraction

Basins of attraction

Basin of attraction is the set of initial conditions leading to long-time behavior that approaches an attractor.

Each different attractor is cataloged with a color.

To classify the dynamic behavior is used Test 0-1.


Dynamic system
Basins of Attraction calculation
Test 0-1
Periodic
Attractors identification
Chaotic
Catalogue the attractors in a color scheme
Catalogue in a grey color
Chaotic Attractors
Periodic Attractors

Test 0-1

The **Test 0-1** is used to characterize the dynamic numerically, determining if the system evolves to the chaotic or regular response. This classification is obtained through time-series response.

First is divided the time-serial $\phi(t)$ into two coordinates:

$$p_n(c) = \sum_{j=1}^{n} \phi(t_j) \cos jc \quad ; \quad q_n(c) = \sum_{j=1}^{n} \phi(t_j) \sin jc$$

which c is a random value with support $(0, 2\pi)$ and $n = 1, 2, \ldots, N$

Test 0-1

Then the mean square deviation is calculated by:

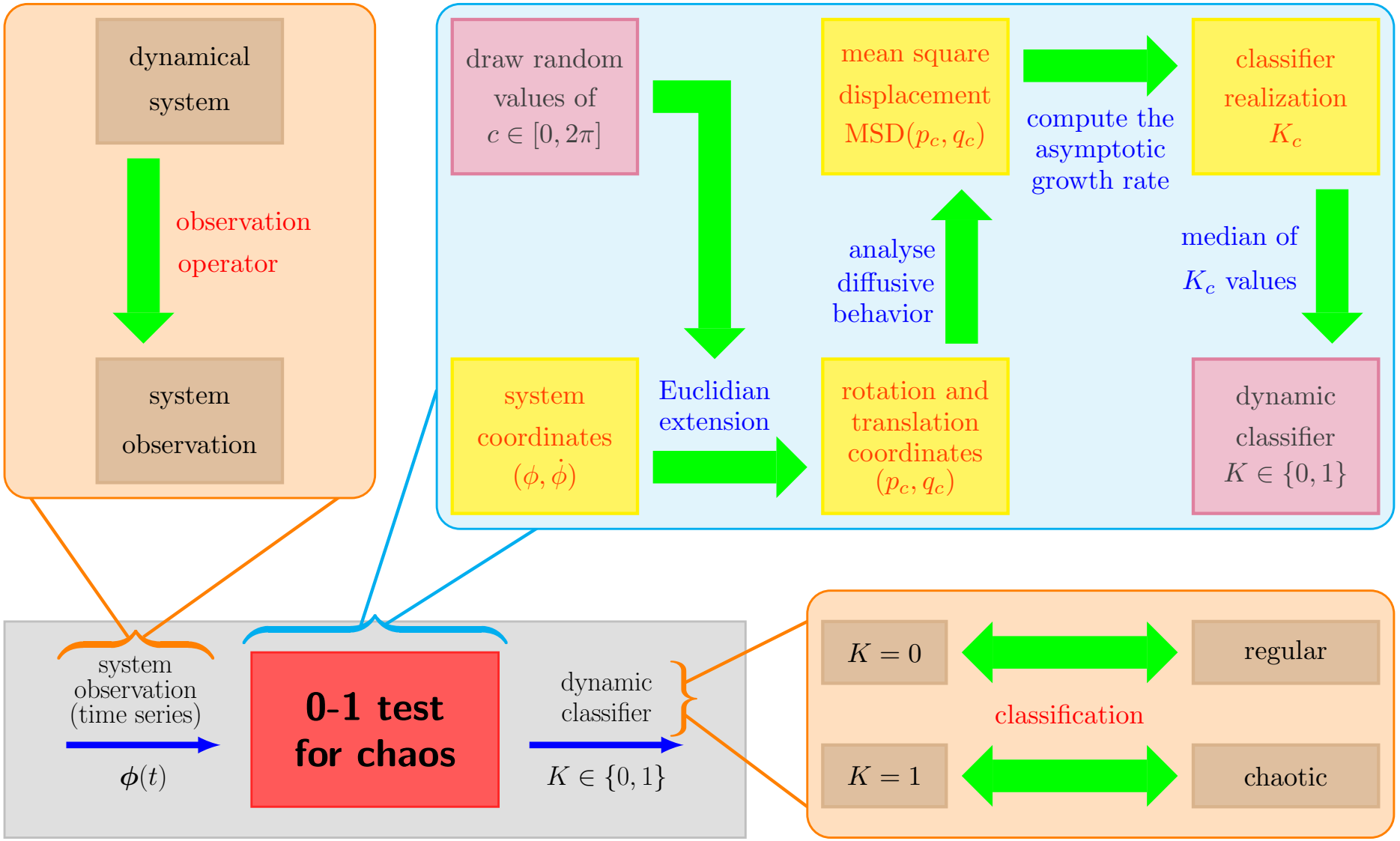
$$M_n(c) = \lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{n} ([p_{j+n}(c) - p_j(c)]^2 + [q_{j+n}(c) - q_j(c)]^2)$$

Finally, the classifier is defined by

$$K_c = \lim_{N\to\infty} \frac{cov(t_n, M_n)}{\sqrt{var(t_n)var(M_n)}}$$

where $M_n = (M_1, M_2, \ldots, M_n)$, $t_n = (t_1, t_2, \ldots, t_n)$, *cov* and *var* are covariance and variance operator.

It is worthwhile to mention that, the K_c is calculated for several c values and a median is computed, in order to a definitive value.


dynamical system
observation operator
system observation
draw random values of $c \in [0, 2\pi]$
mean square displacement $\mathrm{MSD}(p_c, q_c)$
compute the asymptotic growth rate
classifier realization K_c
median of K_c values
dynamic classifier $K \in \{0, 1\}$
system coordinates $(\phi, \dot{\phi})$
Euclidian extension
rotation and translation coordinates (p_c, q_c)
analyse diffusive behavior
system observation (time series)
$\phi(t)$
0-1 test for chaos
dynamic classifier
$K \in \{0, 1\}$
$K = 0$
regular
classification
$K = 1$
chaotic

Code

Basins of attraction code is compatible with the C++.

Project C++ code

Three files are develop:

1. call_function.h : library to named aid function
2. function.cpp : library to define aid function
3. main.cpp : main file

Function-aid

- linspace
- test
- RKF4
- pvi
- fill_AT
- z1test
- cumsum
- sum
- mean
- corr
- median
- my_sort

These functions are defined as a void structure to aid the main file.

Main file

Calling libraries:

```
#include <iostream>
#include <thread>
#include <mutex>
#include <fstream>  // ofstream
#include <cmath>    // round
#include "header.h"
#include <stdlib.h>
#include <cstdlib>
#define pi 3.14159265
```

Main file

Body code:

```
using namespace std;
void task_t2(double *d_params, int *i_params, double *param,
    double** AT,double** IC);
void task_t3(double *d_params, int *i_params, double *param,
    double** AT, double** IC);
void task_t4(double *d_params, int *i_params, double *param,
    double** AT, double** IC);

int main()
{   ... }
```

task_t2, task_t3 and task_t4 are same lines code.

They are implemented to parallelize the computing.

void task function

Input data:

```
double t0          = d_params[0];
double t1          = d_params[1];
double h_q         = d_params[2];
double tol         = d_params[3];
double x0_min      = d_params[4];
double x0_max      = d_params[5];
double xdot0_min = d_params[6];
double xdot0_max = d_params[7];
double v0          = d_params[8];
int Ndt     = i_params[0];
int Nss     = i_params[1];
int Nat     = i_params[2];
int N_x     = i_params[3];
int N_xdot = i_params[4];
```

void task function

Input vector and matrix:

```
// vectors
double *x0 = new double[N_x];          // vector of x0 values
linspace(x0_min, x0_max, N_x, x0);

double *xdot0 = new double[N_xdot]; // vector of xdot0 values
linspace(xdot0_min, xdot0_max, N_xdot, xdot0);

double ic[3];
ic[2] = v0;

// response matrix (dimensions: [Ndt][3])
double **y = new double*[Ndt];
for(int i=0; i<Ndt; i++){y[i] = new double[3];}

double *Qvolt = new double[(Ndt-Nss)/50];

// response matrix (dimensions: [Ndt][3])
double **y = new double*[Ndt];
for(int i=0; i<Ndt; i++){y[i] = new double[3];}

double *Qvolt = new double[(Ndt-Nss)/50];
```

void task function

Main loop:

```
// calculating
for(int nx=1; nx<N_x; nx=nx+4)
{ ...
}
```

This loop computes the dynamic serial for each initial condition.
It is the only difference code to another task function.

- task_t2: nx = 1
- task_t3: nx = 2
- task_t4: nx = 3

void task function

Inside of main loop:

```
ic[0] = x0[nx];                                    // updating displacement

for(int nxdot=0; nxdot<N_xdot; nxdot++){
    ic[1] = xdot0[nxdot];                          // updating velocity
    // Integrate the dynamical system with RKF4:
    RKF4(ic, 3, param, t0, t1, h, pvi, y);
    // steady-state response:
    double **yss = new double *[Ndt-Nss+1];
    for(int i=0; i<Ndt-Nss+1; i++){
        yss[i] = new double[3];
        yss[i][0] = y[Nss+i-1][0];
        yss[i][1] = y[Nss+i-1][1];
        yss[i][2] = y[Nss+i-1][2];
    }
    for(int i=0; i<(Ndt-Nss)/50; i++){
        Qvolt[i] = yss[(i*50)][1];
    }

// ... continuing
```

void task function

Inside of main loop:

```
// ... continuing
    for(int nat=0; nat<Nat; nat++){        // cataloging attractors
        if(AT[0][nat*3] == 0){             // if attractor is empty
            // fill attractor:
            fill_AT(yss, 0, Ndt-Nss, AT, nat*3, nat*3+2);
            // fill IC matrix
            IC[nx][nxdot] = nat+1;
            break;
        }
        // if same attractor:
        else if(test(yss, Ndt-Nss+1, AT, nat*3, tol)){
            // fill IC matrix:
            IC[nx][nxdot] = nat+1;
            break;
        }
    }
    // ... continuing
```

void task function

Ending the main loop:

```
// ... continuing
    if(abs(z1test(Qvolt, (Ndt-Nss)/50)) > 0.8){    // chaotic
        // fill IC matrix
        IC[nx][nxdot] = 0;
    }

    for(int i=0; i<Ndt-Nss+1; i++){delete [] yss[i];}
    delete [] yss;
    yss = 0;
}
```

int main processing

The int main() is the starting place of program running.

It is divided on:

1. Simulation parameters
2. Physical parameters
3. Run tasks function
4. Save attractors(AT) output matrix and IC output matrix

Plotting

To plot the basins of attraction and the phase portrait is used the Matlab©.

Matlab files:

1. plot_IC: plot the basins of attractors
2. plot_AT: plot the attractors
3. graphrb_contourf_pnt: function of plot set up

graph_contourf_pnt file

This function plots the contour map of a scalar function.

```
%  input:
%  x       - x mesh vector
%  y       - y mesh vector
%  F       - scalar field
%  gtitle  - graph title
%  xlab    - x axis label
%  ylab    - y axis label
%  xmin    - x axis minimum value
%  xmax    - x axis maximum value
%  ymin    - y axis minimum value
%  ymax    - y axis maximum value
%  gname   - graph name
%  flag    - output file format (optional)
%
%  output:
%  gname.eps - output file in eps format (optional)
```

graph_contourf_pnt file

First, is check the number of arguments:

```
function fig = graph1_contourf_pnt(x,y,F,gtitle,xlab,ylab,...
                                   xmin,xmax,ymin,ymax,gname,flag)

    % check number of arguments
    if nargin < 11
        error('Too few inputs.')
    elseif nargin > 12
        error('Too many inputs.')
    elseif nargin == 11
        flag = 'none';
    end
```

Then, generate a 2D mesh grid, name the figure and also create a contour matrix

```
% generate 2D mesh grid
    [Xq,Yq] = meshgrid(x,y);

    fig = figure('Name',gname,'NumberTitle','off');

    [C,h] = contourf(Xq,Yq,F);
    set(h, 'LineColor', 'none');
```

graph_contourf_pnt file

Set the colormap (these colors are going to be the same when defining to the attractors)

```
colormap([
            0.0  0.0  1.0;  % blue
            1.0  0.0  0.0;  % red
            0.0  1.0  0.0;  % green
            1.0  0.0  1.0;  % magenta
            0.0  1.0  1.0;  % cyan
            1.0  1.0  0.0;  % yellow
           ]);

 caxis([0  10])
```

plot_IC file

This code plot the basins of attractors using an IC output matrix file.
First, import the IC file

```
% Import IC matrix file
buffer = importdata('IC_[-3,3]_[-3,3]_1200X1200_a-25_d15.dat');

% Loop to organize data file
for i=1:1200
    data(i,:) = buffer(end-i+1,:);
end
```

Then, set the inputs parameters to create a grid

```
% Plot parameters to create a grid (x-axis)
x0_min = -3;
x0_max = 3;
N_x0 = 1200;
% Plot parameters to create a grid (y-axis)
xdot0_min = -3;
xdot0_max = 3;
N_xdot0 = 1200
```

plot_IC file

The next step is a discretization of this grid equally spaced

```
% Grid discretization
x0 = linspace(x0_min, x0_max, N_x0);
xdot0 = linspace(xdot0_min, xdot0_max, N_xdot0);
```

Finally, set the inputs parameters to plot using the graphrb_contourf_pnt function

```
% Input parameters to plot using the function graphrb_contourf_pnt
x      = x0;
y      = xdot0;
F      = data;
gtitle = '';
xlab   = 'initial displacement';
ylab   = 'initial velocity';
xmin   = x0_min;
xmax   = x0_max;
ymin   = xdot0_min;
ymax   = xdot0_max;
gname  = '';
flag   = '';

% plotting
fig = graphrb_contourf_pnt(x,y,F,gtitle,xlab,ylab,...
                           xmin,xmax,ymin,ymax,gname,flag);
```

plot_AT file

This file plot the attractors in the phase space. First, import the AT data file

```
% Import the AT data file
data = importdata('AT_[-3,3]_[-3,3]_1200X1200_a-35_d15.dat');
```

Then, each attractor has one of a kind color

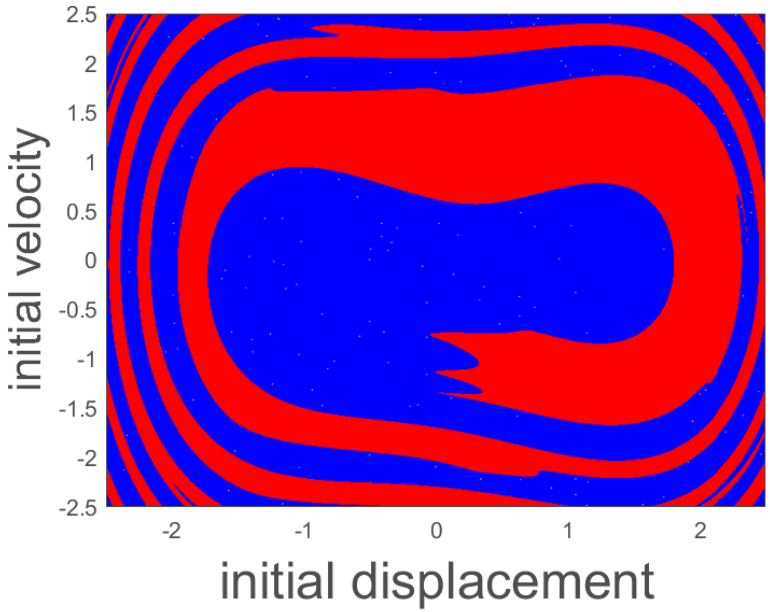
```
% Vector colors
color = [
        0.0 0.0 1.0; % blue
        1.0 0.0 0.0; % red
        0.0 1.0 0.0; % green
        1.0 0.0 1.0; % magenta
        0.0 1.0 1.0; % cyan
        1.0 1.0 0.0; % yellow
        ];
```

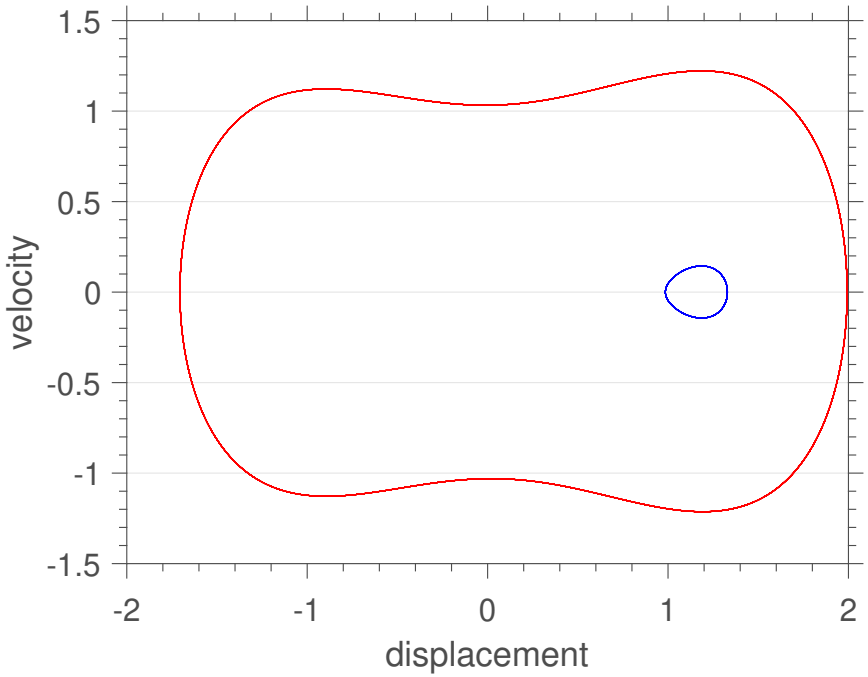
plot_AT file

Finally, in this loop, the graph will be plotted

```
% This loop plot the attractors in the phase space
for b=1:n % where "n" is the number of attractos
    fig1 = plot(data(round(0.8*end):end,3*b-2), data(round(0.8*end):end,3*b-1), '
        Color', color(b,:));

    % graph's customization
    xlabel('displacement', 'FontSize', 20, 'FontName', 'Helvetica');
    ylabel('velocity', 'FontSize', 20, 'FontName', 'Helvetica');
    xlim([-2.5,2.5]);
    ylim([-2,2]);
    set(gcf,'color','white');
    set(gca,'Box','on');
    set(gca,'TickDir','out','TickLength',[.02 .02]);
    set(gca,'XMinorTick','on','YMinorTick','on');
    set(gca,'XGrid','off','YGrid','on');
    set(gca,'XColor',[.3 .3 .3],'YColor',[.3 .3 .3]);
    set(gca,'FontName','Helvetica');
    set(gca,'FontSize',20);

    hold on
end
```

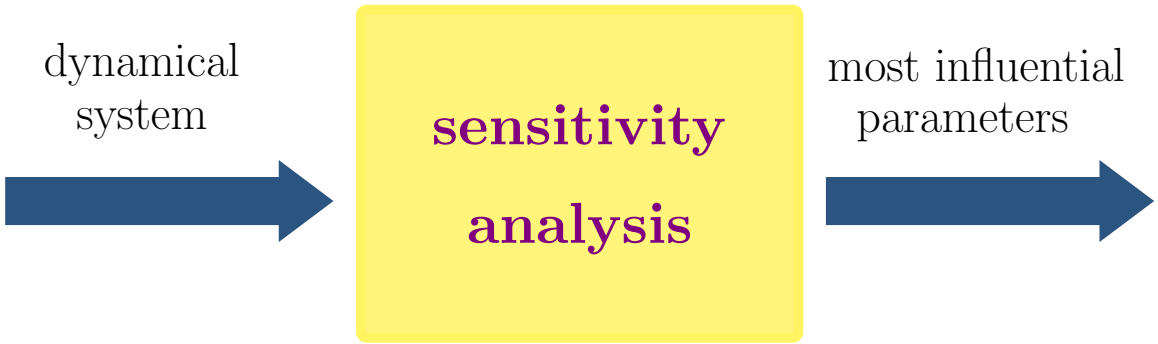
Plotting


initial velocity
initial displacement
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-2
-1
0
1
2


velocity
displacement
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-1
0
1
2

Sensitivity Analysis

Sensitivity analysis



Main contributions:

- Simpler probabilistic model constructions
- Decision making
- Nontrivial insight into the behavior
- Important step for robustness and optimization problems

Variance-decomposition sensitivity analysis

Mathematical Model:

$$Y = \mathcal{M}(\mathbf{X}) \quad , \quad X_i \sim \mathcal{U}(0,1)$$

Hoeffding-Sobol decomposition:

$$Y = \mathcal{M}_0 + \sum_{i=1}^{k} \mathcal{M}_i\left(X_i\right) + \sum_{i<j}^{k} \mathcal{M}_{ij}\left(X_i, X_j\right) + \cdots + \mathcal{M}_{1\ldots k}\left(X_1 \ldots X_k\right)$$

An orthogonal decomposition in terms of conditional expectations:

- $\mathcal{M}_0 = \mathbb{E}\left\{Y\right\}$
- $\mathcal{M}_i(X_i) = \mathbb{E}\left\{Y \,|X_i\right\} - \mathcal{M}_0$
- $\mathcal{M}_{ij}(X_i, X_j) = \mathbb{E}\left\{Y \,|X_i, X_j\right\} - \mathcal{M}_i - \mathcal{M}_j - \mathcal{M}_0$
- $\ldots$

Sobol' indices

First-order Sobol' indices:

$$S_i = Var[\mathcal{M}_i(X_i)] \ / \ Var[\mathcal{M}(\mathbf{X})]$$

(quantify the additive effect of each input separately)

Second-order Sobol' indices:

$$S_{ij} = Var[\mathcal{M}_{ij}(X_i, X_j)] \ / \ Var[\mathcal{M}(\mathbf{X})]$$

(quantify interaction effect of inputs X_i and X_j)

I.M. Sobol **Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates**. Mathematics and Computers in Simulation, 55(1-3): 271-280, 2001.

Polynomial Chaos Expansion (PCE)

Polynomial Chaos Expansion is defined as:

$$Y = \mathcal{M}(X) \approx \sum_{\alpha \in \mathcal{A}} y_\alpha \psi_\alpha(X)$$

where

- y_α: deterministic coefficients (to be determined)
- ψ_α: multivariate orthogonal polynomial bases

B. Sudret **Global sensitivity analysis using polynomial chaos expansions**. Reliability Engineering and System Safety, 93: 964-979, 2008.

Post-processing techniques PCE

Mean:

$$\hat{\mu} = \mathbb{E}\left[\mathcal{M}^{PC}(X)\right] = y_0$$

Variance:

$$\hat{\sigma}^2 = \mathbb{E}\left[\left(\mathcal{M}^{PC}(X) - \hat{\mu}\right)^2\right] = \sum_{\substack{\alpha \in \mathcal{A} \\ \alpha \neq 0}} y_\alpha^2$$

Then, Sobol' indices:

$$S_i = \sum_{\substack{\alpha \in \mathcal{A}_i \\ \alpha \neq 0}} y_\alpha^2 \Bigg/ \sum_{\substack{\alpha \in \mathcal{A} \\ \alpha \neq 0}} y_\alpha^2$$


B. Sudret **Global sensitivity analysis using polynomial chaos expansions**. Reliability Engineering and System Safety, 93: 964-979, 2008.

Code

The code is developed at Matlab© with the UQLab toolbox.

UQLab is developed at the Chair of Risk, Safety and Uncertainty Quantification of ETH Zurich under the supervision of Prof. B. Sudret and Dr. S. Marelli.

UQLab: A Framework for Uncertainty Quantification in MATLAB, Stefano Marelli and Bruno Sudret, In The 2nd International Conference on Vulnerability and Risk Analysis and Management (ICVRAM 2014), University of Liverpool, United Kingdom, July 13-16, 2014, pp. 2554–2563.

https://www.uqlab.com/

Code 1: one scenario

First, the Sobol Indices are calculated to one specific nominal condition of excitation.

Three files are develop:

1. piezo_magneto_harvesting.m
2. plot_sobol.m
3. main_pmeh_sobol.m

piezo_magneto_harvesting.m

This function computes the power of a piezo-magneto-elastic beam.

function name file: $[mean\_power] = piezo\_magneto\_harvesting(X)$

Input data:

- X: uncertainty model parameters

Output data:

- mean_power: mean power output

piezo_magneto_harvesting.m

The code is separed as:

1. model parameters
2. equation of motion integration
3. calculate the mean power output

plot_sobol.m

This function plots the Sobol Indices to the piezo-magneto-elastic beam.

function name file: $[fig] = plot\_sobol(mySobolAnalysis, order, method)$

Input data:

- mySobolAnalysis: Sobol data
- order: Sobol order to plot
- method: based method

Output data:

- fig: figure

plot_sobol.m

A conditional logical is defined:

- if order == 1
- elseif order == 2

Second-order Sobol indices are plotted only the top five joint parameters.

main_pmeh_sobol.m

This script is the main file for a program that performs a global sensitivity analysis in the nonlinear dynamics of a Piezo-Magneto-Elastic-Beam Harvesting via Sobol indices.

It is divided on three cells:

1. create input data
2. perform Sobol indices based on MC
3. perform Sobol indices based on PCE

Create input data

Nominal parameter values are defined.

```
% physical parameters
%-------------------------------------------------------------------------

ksi1    = 0.01;         % damping ratio
chi1    = 0.05;         % piezoelectric coupling (mechanical)
lambda1 = 0.05;         % reciprocal time constant
kappa1  = 0.5;          % piezoelectric coupling (electrical)
f1      = 0.147;        % amplitude of external force
Omega1  = 0.8;          % frequency of external force
```

Mathematical model is established

```
ModelOpts.mFile        = 'piezo_magneto_harvesting';
ModelOpts.isVectorized = false;
myModel = uq_createModel(ModelOpts);
```

Create input data

Input parameters and associated distributions:

```
% dispersion with respect to the nominal value
delta = 0.2;

InputOpts.Marginals(1).Name = 'ksi';
InputOpts.Marginals(1).Type = 'Uniform';
InputOpts.Marginals(1).Parameters = [1-delta 1+delta]*ksi1;

InputOpts.Marginals(2).Name = 'chi';
InputOpts.Marginals(2).Type = 'Uniform';
InputOpts.Marginals(2).Parameters = [1-delta 1+delta]*chi1;

InputOpts.Marginals(3).Name = 'lambda';
InputOpts.Marginals(3).Type = 'Uniform';
InputOpts.Marginals(3).Parameters = [1-delta 1+delta]*lambda1;

InputOpts.Marginals(4).Name = 'kappa';
InputOpts.Marginals(4).Type = 'Uniform';
InputOpts.Marginals(4).Parameters = [1-delta 1+delta]*kappa1;

InputOpts.Marginals(5).Name = 'f';
InputOpts.Marginals(5).Type = 'Uniform';
InputOpts.Marginals(5).Parameters = [1-delta 1+delta]*f1;

InputOpts.Marginals(6).Name = 'Omega';
InputOpts.Marginals(6).Type = 'Uniform';
InputOpts.Marginals(6).Parameters = [1-delta 1+delta]*Omega1;

myInput = uq_createInput(InputOpts);
```

Sobol indices based on MC

To perform Sobol indices based on the Monte Carlo method:

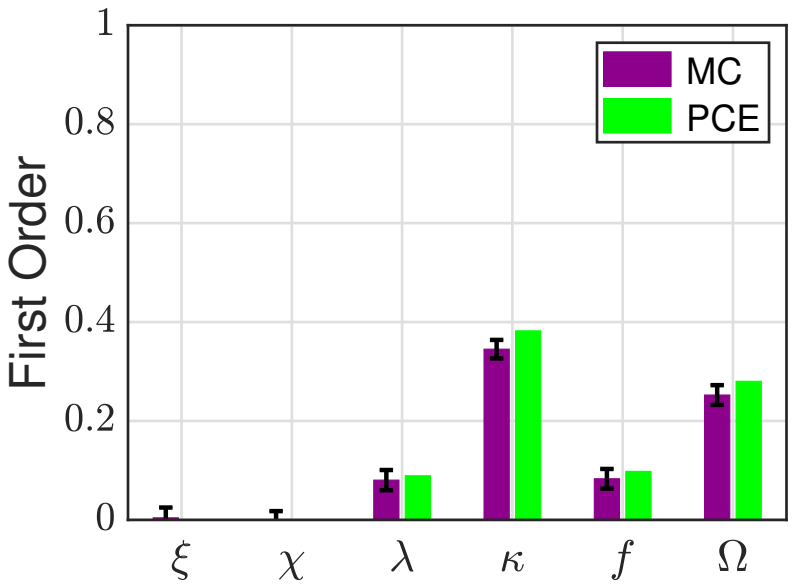
```
% MC-based Sobol indices
% ------------------------------------------------------------------------

Order         = 1;
SampleSize    = 10;
repBootstrap = 5;
alpha         = 0.05;

SobolAnalysisMC = sobol_mc(Order,SampleSize,repBootstrap,alpha);

file_name = ['Sobol_MC_Nsamp', num2str(SampleSize), ...
             '_ord'          , num2str(Order)     , ...
             '_f'            , num2str(f1*1e3)    , ...
             '_O'            , num2str(Omega1*1e1), '.mat'];

save(file_name,'SobolAnalysisMC')

plot_sobol(SobolAnalysisMC,1,'MC')
% ------------------------------------------------------------------------
```

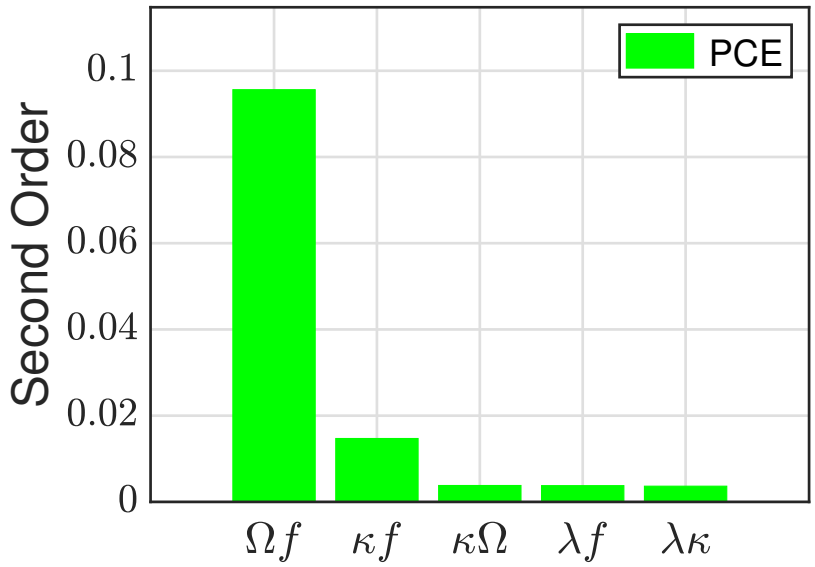
Sobol indices based on PCE

To perform Sobol indices based on Polynomial Chaos Expansion method:

```
% PCE-based Sobol indices
% ------------------------------------------------------------------------
degreePCE = 3;
Nsamples  = 100;
order     = 2;
SobolAnalysisPCE = sobol_pce(myInput,myModel,degreePCE,Nsamples,order);

file_name = ['Sobol_PCE_Nsamp', num2str(Nsamples)   , ...
             '_ord'           , num2str(order)      , ...
             '_f'             , num2str(f1*1e3)     , ...
             '_O'             , num2str(Omega1*1e1), '.mat'];
save(file_name,'SobolAnalysisPCE')

plot_sobol(SobolAnalysisPCE,1,'PCE')
% ------------------------------------------------------------------------
```

Sobol indices based on PCE





Code 2: several scenarios

Finally, the Sobol Indices are calculated to the wide condition of excitation.

Three files are develop:

1. piezo_magneto_harvesting.m *
2. main_pmeh_sobol_frang_pce.m
3. main_plot_sobol_frang.m

* The same function showed previously

main_pmeh_sobol_frang_pce.m

This script is the main file for a program that performs a global sensitivity analysis in the nonlinear dynamics of a bistable energy harvester via Sobol indices based on PCE, sweeping the amplitude of excitation.

It is organized as:

1. define input data
2. create a loop for each amplitude of excitation condition
3. build a PCE model
4. perform Sobol indices

Create input data

Nominal parameter values are defined.

```
% physical parameters
%------------------------------------------------------------------
ksi1    = 0.01;         % damping ratio
chi1    = 0.05;         % piezoelectric coupling (mechanical)
lambda1 = 0.05;         % reciprocal time constant
kappa1  = 0.5;          % piezoelectric coupling (electrical)
Omega1  = 0.8;          % frequency of external force

% range of amplitude of excitation
frang = [0.041 0.060 0.083 0.091 0.105 0.115 0.147 0.2 0.25];
```

Loop

Loop for each amplitude of excitation condition reads

```
for i = 1:length(frang)

    f1       = frang(i);   % amplitude of external force

    % mathematical model
    % ------------------------------------------------------------------
    ModelOpts.mFile          = 'piezo_magneto_harvesting';
    ModelOpts.isVectorized = false;
    myModel = uq_createModel(ModelOpts);

    % ... probabilistic input model

    % ... PCE-building

    % ... Sobol calculating

    % ... Saving

end
```

The process is the same as present in code 1.

main_plot_sobol_frang.m

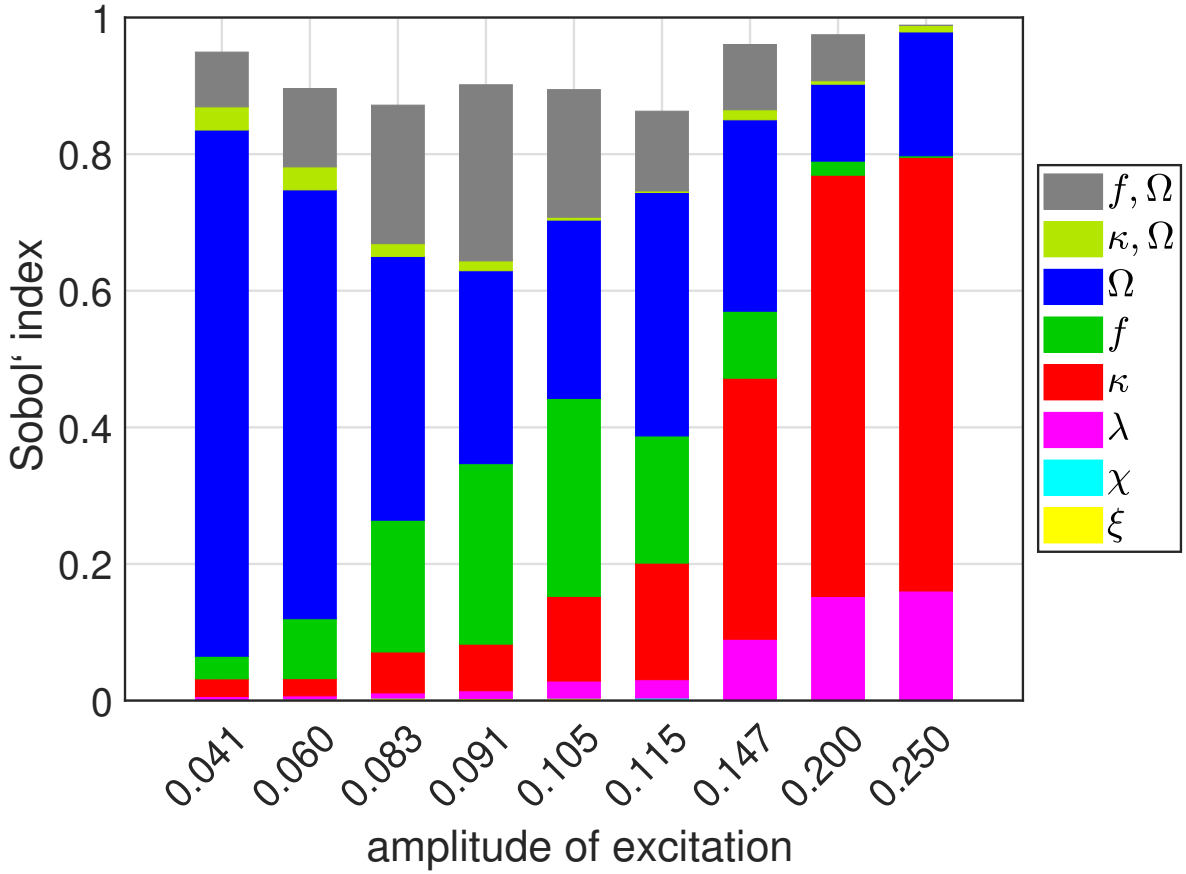
This script is the main file for a program that plots the global sensitivity analysis of bistable energy harvester via Sobol indices based on PCE sweeping the amplitude of excitation.

It is organized as:

1. load data of Sobol results
2. plot command
3. plot configurations

The data are localized in the “data” folder.

Sobol indices sweeping the amplitude of excitation



Remarks

To perform Sobol indices for models with nonlinear piezoelectric coupling or asymmetries is necessary to change the mathematical model and add the novel input parameters and associated distributions.

Stochastic Simulation

Stochastic Simulation

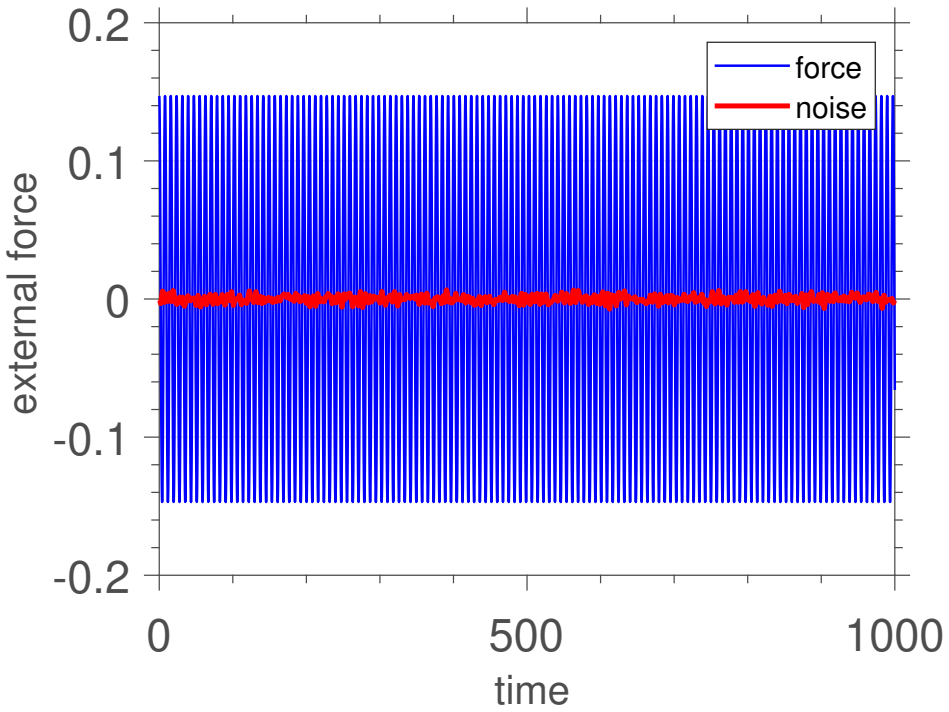
Stochastic simulation module deals with the underlying system subjected to a harmonic excitation disturbed by Gaussian colored noise.

Several numerics experiments suggest a strong influence of noise on the system response, highlighting the demand and the investigation.

External excitation

Harmonic force + Gaussian colored noise:

$$F_{ext} = f\cos(\Omega t) + N_t$$



Gaussian colored noise

It is assumed as a zero-mean Gaussian colored noise with covariance function:

$$\mathrm{COV}_{N_t}(t_1, t_2) = \sigma \ \ exp\left(-\frac{|t_2 - t_1|}{\tau_{corr}}\right)$$

$$\tau_{corr} = \eta \ \ \tau_\Omega \quad \text{end} \quad \tau_\Omega = \frac{2\pi}{\Omega}$$

where τ_{corr} means the correlation time and σ the colored noise standard deviation.

Code

This module has one main file to run and the other six aid files as functions.

Aid-functions:

1. randvar_ksd.m
2. randvar_pdf.m
3. HarvesterPower.m
4. HarvesterColoredNoise.m
5. fredholm_expcorr_eig.m
6. HarvesterColoredNoise_ivp_post

Main-file

1. HarvesterColoredNoise_ivp_main.m

randvar_ksd.m

This functions computes the kernel smooth density estimation of a random process distribution given its numerical series.

function name file: $[data\_ksd, data\_supp] = randvar\_ksd(data, numpts)$

Input data:

- data: (Ns x Ndt) data matrix
- numpts: number of points

Output data:

- data_ksd: (numpts x Ndt) frequency counts matrix
- data_supp: (numpts x Ndt) bins locations matrix

randvar_pdf.m

This functions computes the probability density funtion of a random process given its numerical series.

function name file: $[bins, freq, area] = randvar\_pdf(data, numbins)$

Input data:

- data: (Ns x Ndt) data matrix
- numbins: number of bins

Output data:

- bins: (numbins x Ndt) bins locations matrix (row vector)
- freq: (numbins x Ndt) frequency counts matrix (row vector)
- area: (1 x Ndt) area under the histogram (optional)

HarvesterPower.m

This function computes the mean output power recovered from a bistable piezo-magneto-elastic energy harvester.

function name file:
$[power, power\_mean] = HarvesterPower(time, volt, lambda)$

Input data:

- time: time vector
- volt: voltage vector
- lambda: reciprocal time constant

Output data:

- power: instantaneous power
- power_mean: mean power

HarvesterColoredNoise.m

This function defines the right hand side of the following system of nonlinear ordinary differential equations.

function name file: $[dydt] = HarvesterColoredNoise(t, y, param)$

Input data:

- t: time (s)
- y: state space vector
- param: physical parameters structure

Output data:

- dydt: right hand side function

fredholm_expcorr_eig.m

This is function computes the eigenpairs of Fredholm integral operator, with kernel $K(x1, x2) = exp(-|x1 - x2|/a)$, where $a > 0$ is a correlation length. This kernel is the covariance function of an underlying real-valued random field $S(x), x \in D$, with domain $D = [-b, b]$.

function name file: $[lambda, phi, xmesh, dphidx, omega, iroot, ising, iter] = fredholm\_expcorr\_eig(b, a, Neig, Nx, tol, max\_iter)$

fredholm_expcorr_eig.m

Input data:

- b: domain upper limit
- a: autocovariance correlation length
- Neig: number of eigenpairs to be computed
- Nx: number of mesh points for domain discretization
- tol: numerical tolerance for root find (optional)
- max_iter: maximum of iteration for root find (optional)

fredholm_expcorr_eig.m

Output data:

- lambda: (1 x Neig) vector with eigenvalues
- phi: (Nx x Neig) matrix with eigenfunction (in columns)
- xmesh: (1 x Nx) domain discretization mesh
- dphid x: (Nx x Neig) matrix with eigenfunction derivatives (in columns)
- omega: (1 x Neig) vector with charac. eq. solutions
- iroot: charac. eq. solutions counter
- ising: charac. eq. singularities counter
- iter: iterations counter

HarvesterColoredNoise_ivp_post.m

Script for post processing of the simulation data from HarvesterColoredNoise_ivp_main.m

To plot:

- external force
- displacement, velocity and voltage
- steady state
- power instantaneous
- histograms
- attractors 3D

OBS.: This file has more aid function to plot.

HarvesterColoredNoise_ivp_main.m

This script has a program to simulate the nonlinear dynamics of a bistable piezo-magneto-elastic harvester which evolves according to the following system of differential equations.

First defines the simulation information about noise intensity:

```
% simulation information
% ------------------------------------------------------------------
  case_name = 'HarvesterColoredNoise_corrlen';
%case_name = 'HarvesterColoredNoise_corrlen05';
%case_name = 'HarvesterColoredNoise_corrlen1';
%case_name = 'HarvesterColoredNoise_corrlen10';
%case_name = 'HarvesterColoredNoise_corrlen50';
%case_name = 'HarvesterColoredNoise_corrlen100';
```

HarvesterColoredNoise_ivp_main.m

In sequel are establish the physical parameter and initial conditions :

```
ksi    = 0.01;  % mechanical damping ratio
chi    = 0.05;  % dimensionless piezoeletric coupling term (mechanical)
lambda = 0.05;  % dimensionless time constant reciprocal
kappa  = 0.5;   % dimensionless piezoeletric coupling term (eletrical)
f      = 0.147; % dimensionless excitation amplitude
Omega  = 0.8;   % dimensionless excitation frequency

x0     = 1.0;   % dimensionless initial displacement
xdot0  = 0.0;   % dimensionless initial velocity
v0     = 0.0;   % dimensionless initial voltage
```

HarvesterColoredNoise_ivp_main.m

Noise parameters are defined:

```
% random process mean function
noise_mean = 0.0;

% standard deviation
noise_std = 0.05*f;

% correlation time (s)
tau_corr = 1.0;

% noise intensity
D = noise_std^2*tau_corr;
```

Then are defined the time integration:

```
% initial time
t0 = 0.0;

% final time
t1 = t0 + 1000.0;

% number of samples for time step
Nsamp = 10;

% time step
dt = tau_corr/Nsamp;

% number of time steps
Ndt = round((t1-t0)/dt);
```

HarvesterColoredNoise_ivp_main.m

The autocorr function eigenpairs and noise are computing:

```
% domain upper limit
domain_upp = 0.5*(t1-t0);

% number of eigenpairs to be computed
Neig = 100;

% computing the autocorr function eigenpairs
[lambda_noise, phi_noise, time_noise] = ...
              fredholm_expcorr_eig(domain_upp, tau_corr, Neig, Ndt);

% temporal mesh vector
time_noise = time_noise + domain_upp;

% generate uncorrelated random variables
Y_noise = randn(1, Neig);

% multiply eigenvalues by noise intensity
lambda_noise = lambda_noise*D;

% representation energy level
energy_level = 0.95;

% number of eigenpairs used in KL expansion
[~, Nkl] = max(cumsum(lambda_noise)/sum(lambda_noise) >= energy_level);

% KL representation of random process (Ns x Ndt)
noise = noise_mean + ...
    phi_noise(:,1:Nkl)*sqrt(diag(lambda_noise(1:Nkl)))*Y_noise(:,1:Nkl)';
```

HarvesterColoredNoise_ivp_main.m

The equation of motions with harmonic force + colored noise are integrated, then the instantaneous and mean power are obtained:

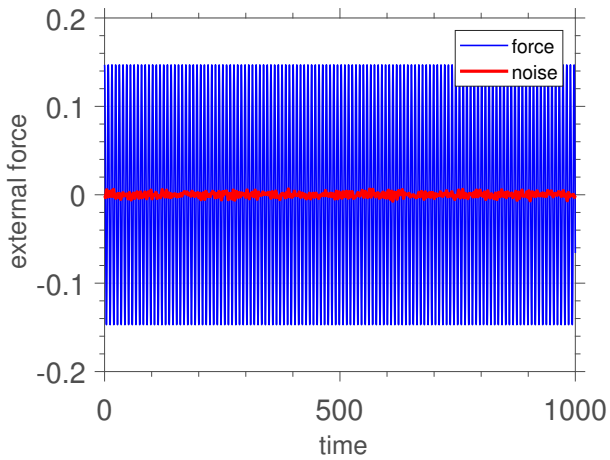
```
% Runge-Kutta45 ODE solver
[time,Y] = ode45(@(t,y)HarvesterColoredNoise(t,y,param),tspan,IC);

% output power
[power,power_mean] = HarvesterPower(time,Qvolt,lambda);
```

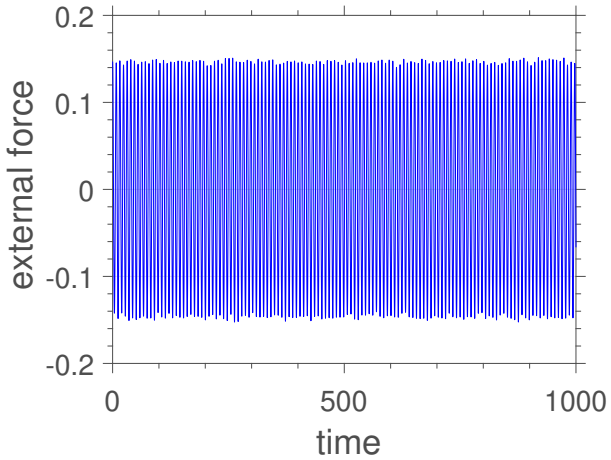
HarvesterColoredNoise_ivp_main.m

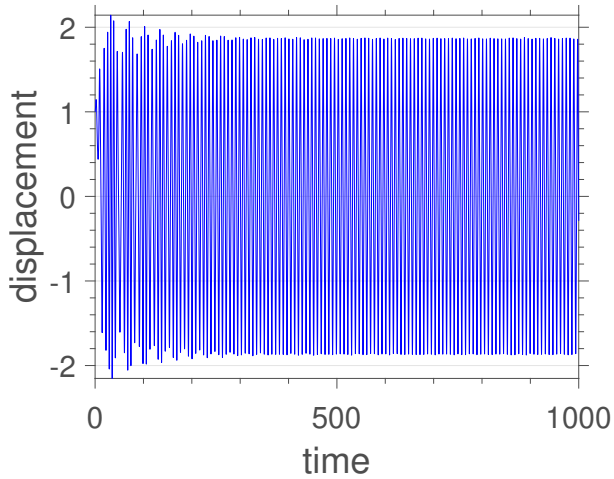
So, the histograms and kernel density are calculated:

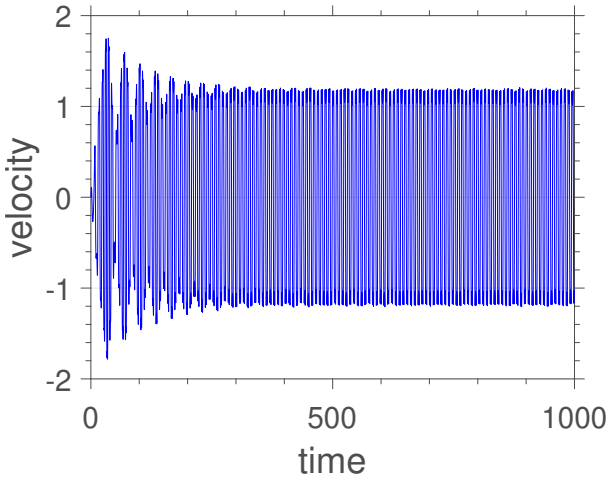
```
% histograms
[Qdisp_bins,Qdisp_freq] = randvar_pdf(Qdisp,Nbins);
[Qvelo_bins,Qvelo_freq] = randvar_pdf(Qvelo,Nbins);
[Qvolt_bins,Qvolt_freq] = randvar_pdf(Qvolt,Nbins);
[power_bins,power_freq] = randvar_pdf(power,Nbins);

% kernel density estimator
[Qdisp_ksd ,Qdisp_supp] = randvar_ksd(Qdisp,Nksd);
[Qvelo_ksd ,Qvelo_supp] = randvar_ksd(Qvelo,Nksd);
[Qvolt_ksd ,Qvolt_supp] = randvar_ksd(Qvolt,Nksd);
[power_ksd ,power_supp] = randvar_ksd(power,Nksd);
```

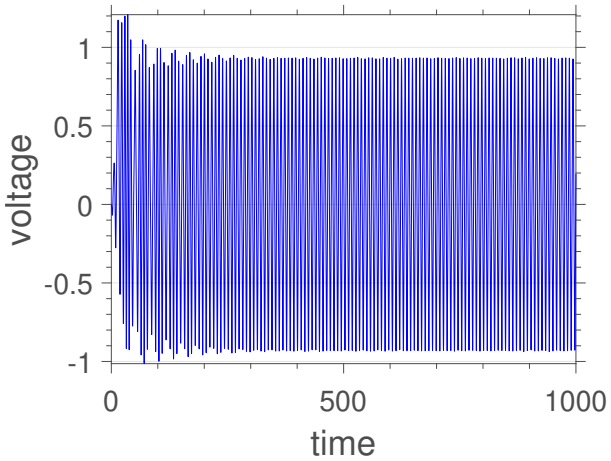
Finally, the graphics are plotted calling the HarvesterColoredNoise_ivp_post.

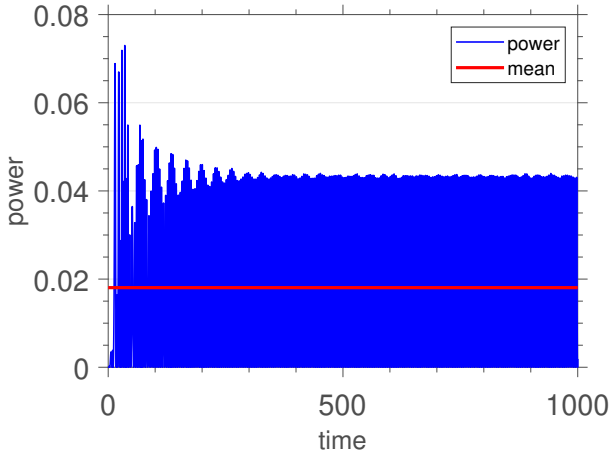

force
noise
external force
time

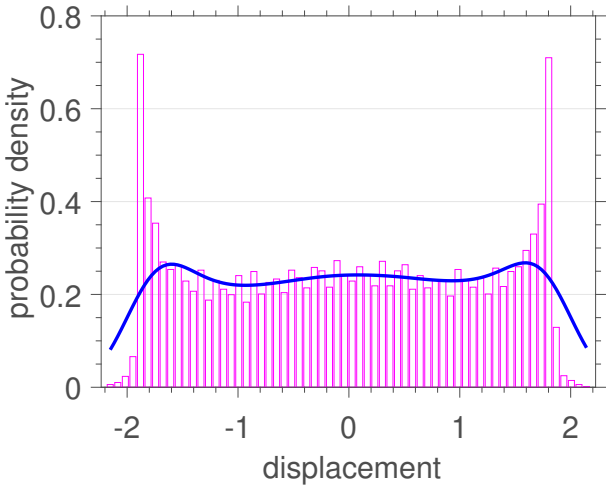

external force
time

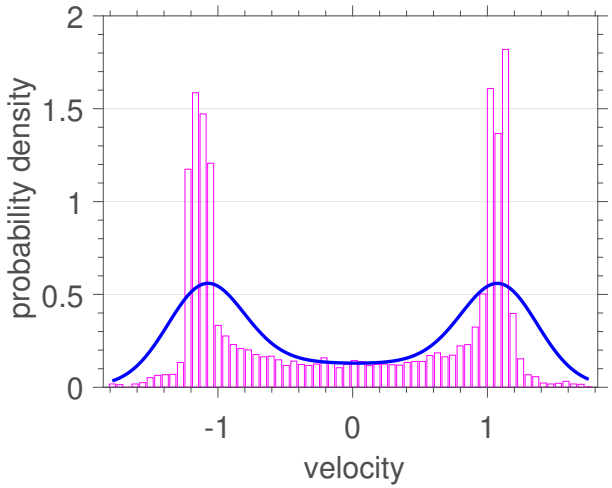

displacement
time

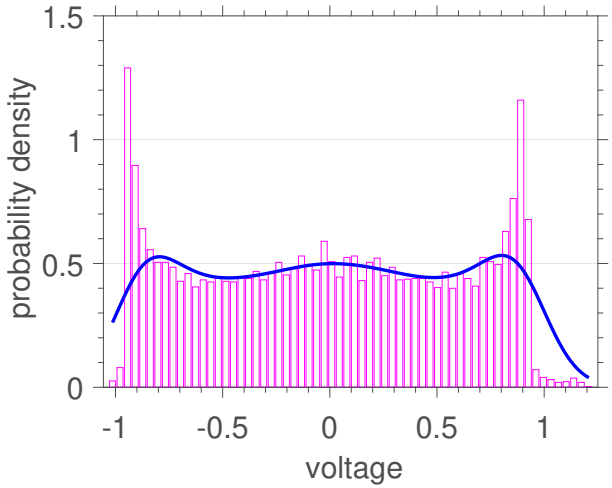

velocity
time

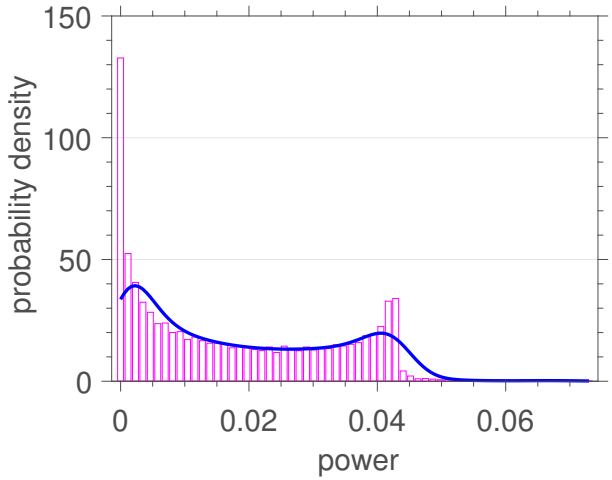

voltage
time


power
mean
power
time


probability density
0.8
0.6
0.4
0.2
0
-2
-1
0
1
2
displacement


probability density
2
1.5
1
0.5
0
-1
0
1
velocity


probability density
1.5
1
0.5
0
-1
-0.5
0
0.5
1
voltage


probability density
150
100
50
0
0
0.02
0.04
0.06
power

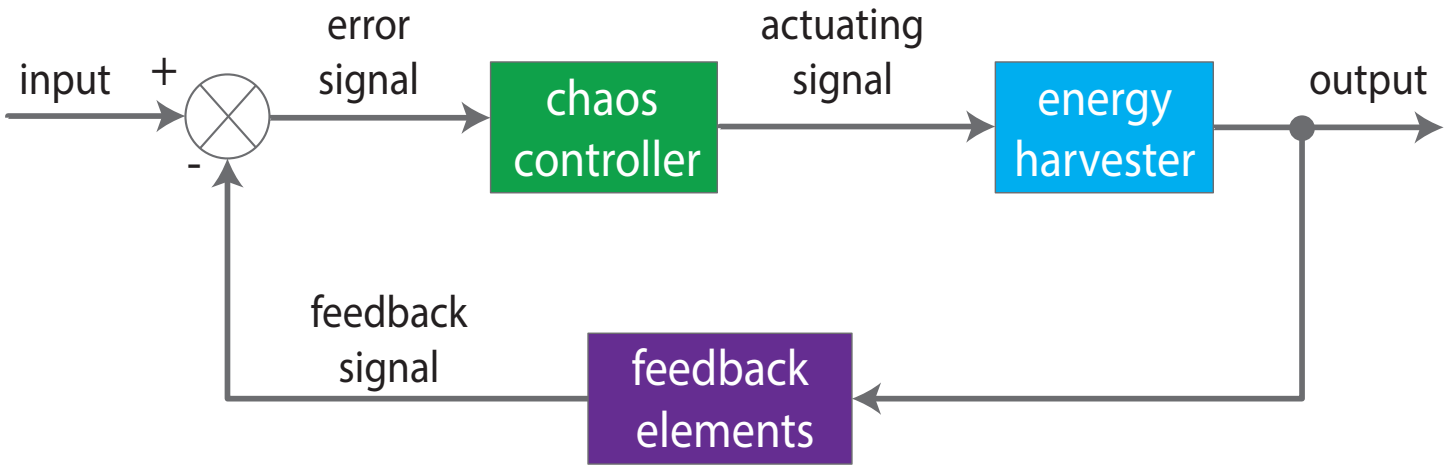
Chaos Control

Chaos Control

This module implements the discrete OGY method and extended time-delayed feedback on the dynamic system.

The goal is to stabilize the chaotic behavior, obtaining its power generation more fruitful, both in signal improvement and power harvesting.

Chaos control methods are based on stabilizing a given chaotic system through small, wisely decided, disturbances to maintain itself in a certain unstable periodic orbit (UPO).



OGY method

OGY method is a discrete method actuates once (if needed) at each period $2\pi\Omega$ of time (a "cycle"), sending a control signal that, through a small controlled force, keeps the system in the chosen UPO.

Such UPO is determined by finding recurrent points in a Poincaré section.

While the UPO is being determined, the method cannot actuate, as the desired orbit is not yet defined.

This stage of determining the UPO is the "learning stage" of the control system. The system will remain uncontrolled until such condition is met.

Code

This module has one main file to run and one function of equation of motion.

Function file:

1. piezomagbeam.m

Main-file

1. main_OGY_R01.m

piezomagbeam.m

This functions define the equation of motion of the bistable energy harvester system.

function name file: $[ydot] = piezomagbeam(t, y, phys_param)$

Input data:

- t: time vector
- y: state variables
- phys_param: physical parameters

Output data:

- ydot: function of the equation of motion

main_OGY_R01.m

This script is the main file to simulate the bistable energy harvester under chaos control by OGY method.

First, it defines the physical parameter:

```
% Piezomagbeam parameters
ksi    = 0.01;  % mechanical damping rate
chi    = 0.05;  % piezoeletric coupling term (mechanical)
lambda = 0.05;  % reciprocal time constant
kappa  = 0.5;   % piezoeletric coupling term (electrical)
Omega  = 0.8;   % frequency of excitation
f      = 0.083; % amplitude of excitation
```

Then, the time vector and initial conditions are established:

```
N_cycles   = 20000;
period     = 2*pi/Omega;
N_sampling = 100;

% time vector
t1      = 0 + N_cycles*period;
time_x0 = 0:period:t1;

% initial conditions
x0 = zeros(3,N_cycles);
x0(1,1) = 1; x0(2,1) = 0; x0(3,1) = 0; x(:,1) = x0(:,1);
```

main_OGY_R01.m

Force excitation for each cycle:

```
% the values of f will be changed in the control
f(:) = f0;

% flags
orbit_flag = 0;
control_flag = 0;
```

The loop starts by integrating the equations of motion:

```
for n=1:N_cycles
    tspan = (n-1)*period : period/N_sampling : n*period;%-period/N_sampling;
    [time_aux, y] = ode45(@(t,y)piezomagbeam(t,y,phys_param),tspan,x0(:,n),opt);
    y = y';

    % update x0 vector
    x0(:,n+1) = y(:,end);

    % update x and time vectors
    x(:,(n-1)*N_sampling+2:n*N_sampling+1) = y(:,2:end);
    time((n-1)*N_sampling+1:n*N_sampling+1) = time_aux(:);

    % orbit starting at n-1 = 2-period orbit
    % orbit starting at n   = 1-period orbit
    orbit_start = n;
```

main_OGY_R01.m

Logical condition is defined inside of the loop to verify the dynamic stabilization and the distance between two points in the Poincaré map is evaluated.

```
% define orbit to stabilize the dynamic
    if n > 1 && orbit_flag == 0
        % distance between two points in the Poincare map
        d = norm(x0(:,orbit_start) - x0(:,n+1));

        if d < tol
           % orbit found
           orbit_flag = 1;

           % save last period evolution
           if orbit_start == n-1
               % for 2-period orbit it is necessary to save 2 cycles
               orbit(:,:) = [buffer(:,:) y(:,2:end)];
           else
               % single-period orbit
               orbit(:,:) = y(:,2:end);
           end

        end
```

main_OGY_R01.m

With orbit already selected, check if next point is near any point of the orbit

```
elseif orbit_flag == 1
    d_vec = sqrt((x0(1,n+1) - orbit(1,:)).^2 + ...
                 (x0(2,n+1) - orbit(2,:)).^2 + ...
                 (x0(3,n+1) - orbit(3,:)).^2);
    d = min(d_vec);

    if d < tol
        % flag indicating in which point the control has been done
        if control_flag == 0
            control_flag = N_sampling*(n+1);

            % find position in d_vec
            pos = find(d_vec == d);
```

If the nearest point is the last of the orbit sequence, seek the first, otherwise, pick the next

```
            if pos == length(d_vec)
                pos = 1;
            else
                pos = pos+1;
            end
```

main_OGY_R01.m

Then, the impulse is calculate:

```
J = [          0                    1          0      ;
     0.5*(1.0 - x(1,n+1)^2)      -2*ksi       chi      ;
                0                -kappa     -lambda  ;];

C = [0  ;  cos(Omega*time_x0(n+1));   0  ];

K(1) = 0.3*sqrt(3)*(1-norm(J))/norm(C);
K(2) = 0.3*sqrt(3)*(1-norm(J))/norm(C);
K(3) = 0.3*sqrt(3)*(1-norm(J))/norm(C);

d1 = orbit(1,pos) - x(1,n+1);
d2 = orbit(2,pos) - x(2,n+1);
d3 = orbit(3,pos) - x(3,n+1);

f(n+1) = -K(1)*d1 -K(2)*d2 -K(3)*d3 + f(n);
```

Otherwise

```
else
    f(n+1) = 0.083;
end
phys_param = [ksi  chi  f(n+1) Omega lambda  kappa];
```

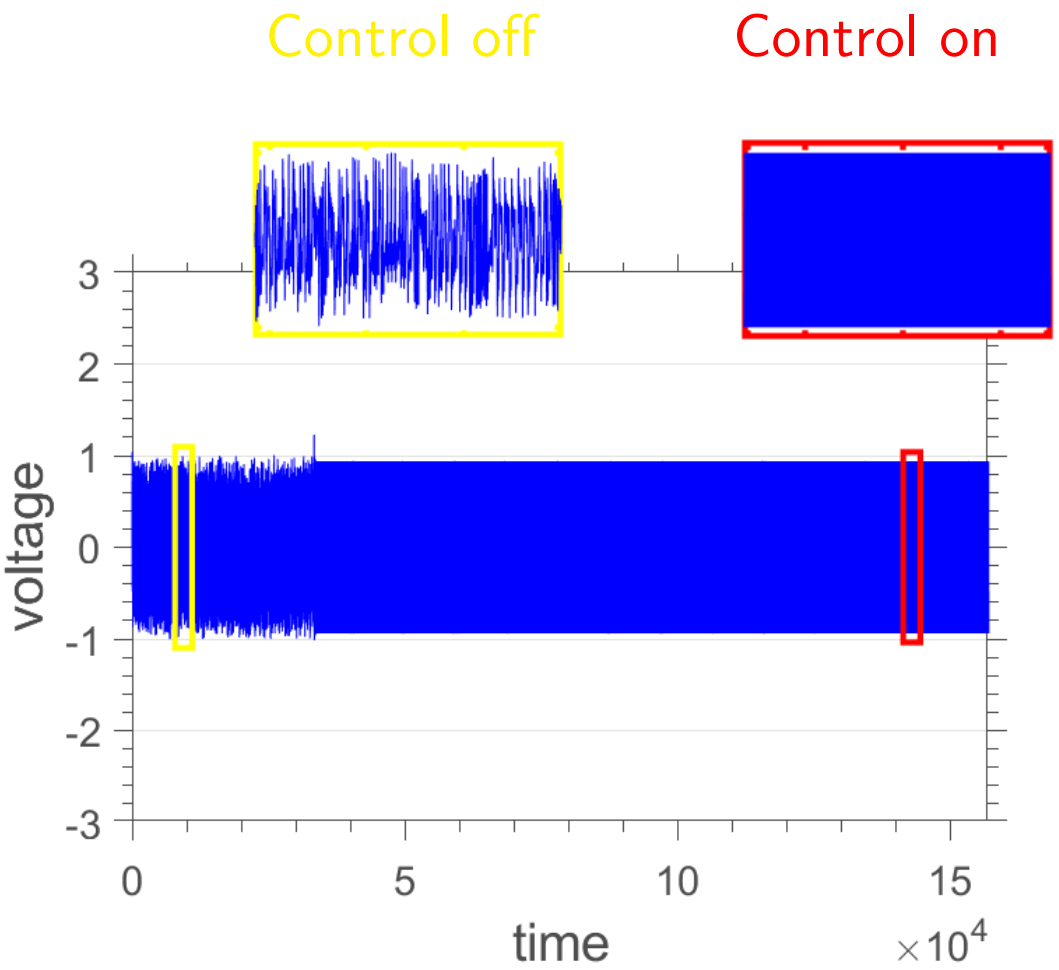
main_OGY_R01.m

Then, the loop is closed:

```
            else
                f(n+1) = 0.083;
            end

        end

        if orbit_start == n-1
            buffer(:,:) = y(:,2:end);
        end

    end
```

Finally, the voltage response is plotted. Initially, the system has a control off, and after about $3x10^4$ of time, the control is on.

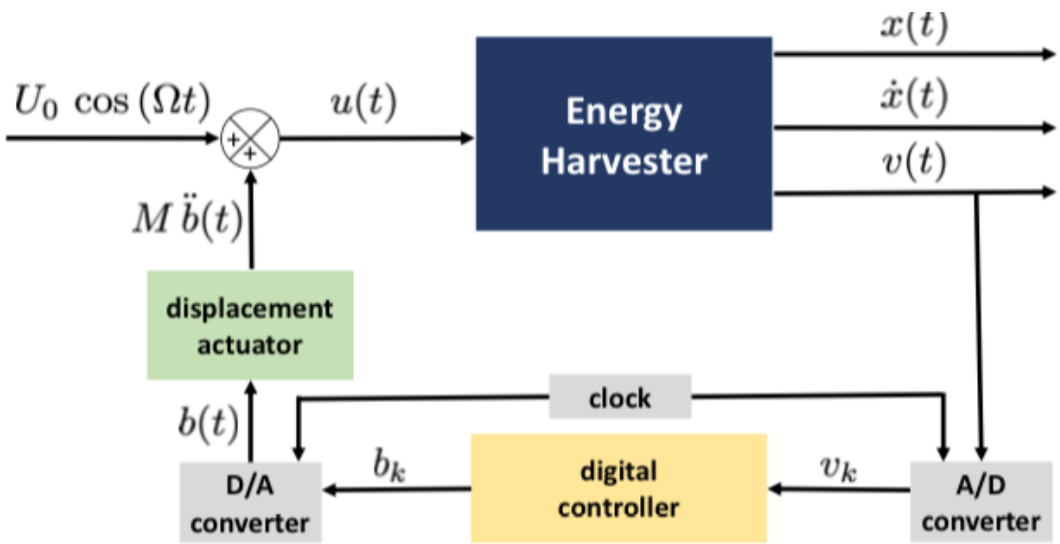
main_OGY_R01.m



Extended time-delayed feedback (ETDF) method

The objective of ETDF is to avoid chaotic response by imposing a controlled displacement.

Controlled displacement is calculated by the digital controller using the voltage as a feedback signal.



Code

This module has one main file to run and simulink file.

Simulink file:

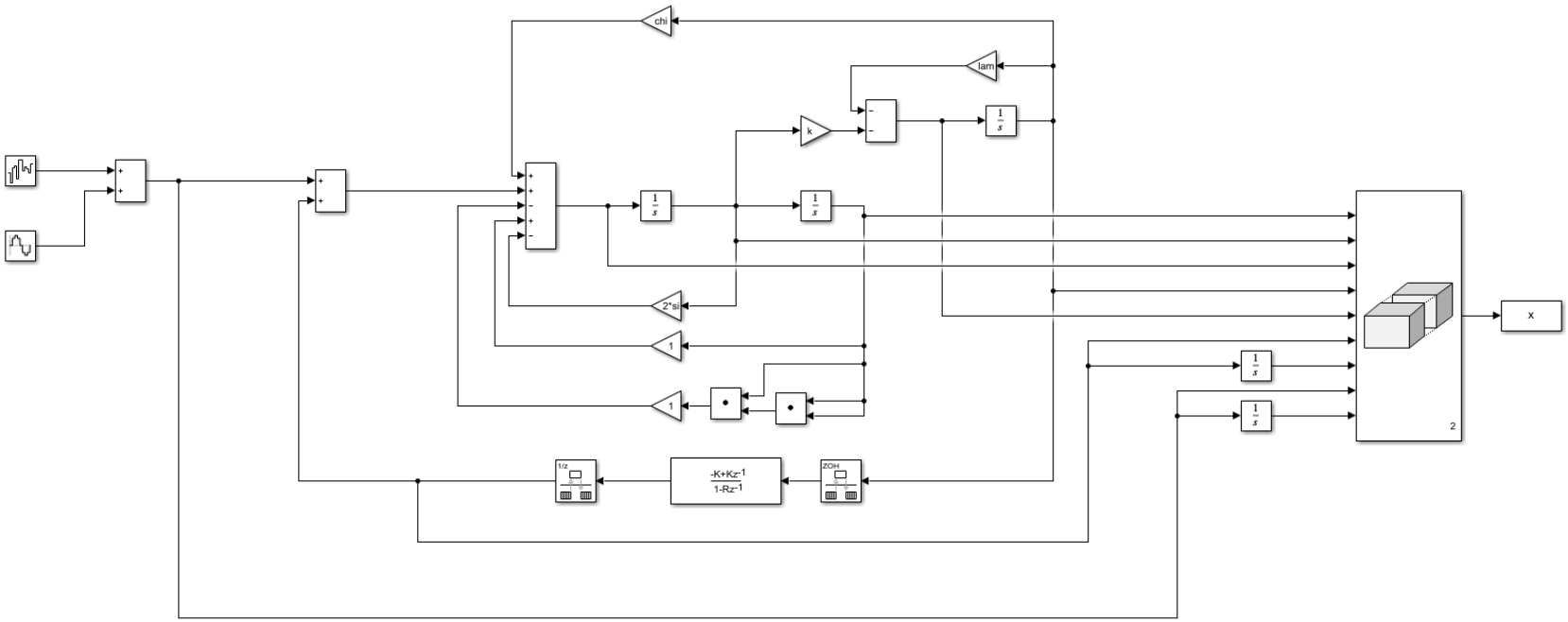
1. Harvest_DuffingCL.slx

Main-file

1. main_ETDF.m

Harvest_DuffingCL.slx

The Simulink model is designed as follows:



main_ETDF.m

This is the main file for a program for the piezo-magneto-elastic beam with control chaos by the Extended time-delayed feedback method.

First, it defines the parameters:

```
% Simulation parameters
tf=10000;
dt=0.001;
t=0:dt:tf-dt;

% Input signal
A=1.3*9.81;
u=(0.03*A)*1.4;
w=0.8;

% initial conditions
x0=[-1.63 0.78 0 0];
x0=[1 0 0];

% Duffing Parameters
si=0.01;
k=0.51;
chi=0.51;
lam=0.04;
```

main_ETDF.m

Then is defined the parameters of the control employed.

```
% Digital controller parameters
T=0.01;
K=-1.3;
R=0.95;
```

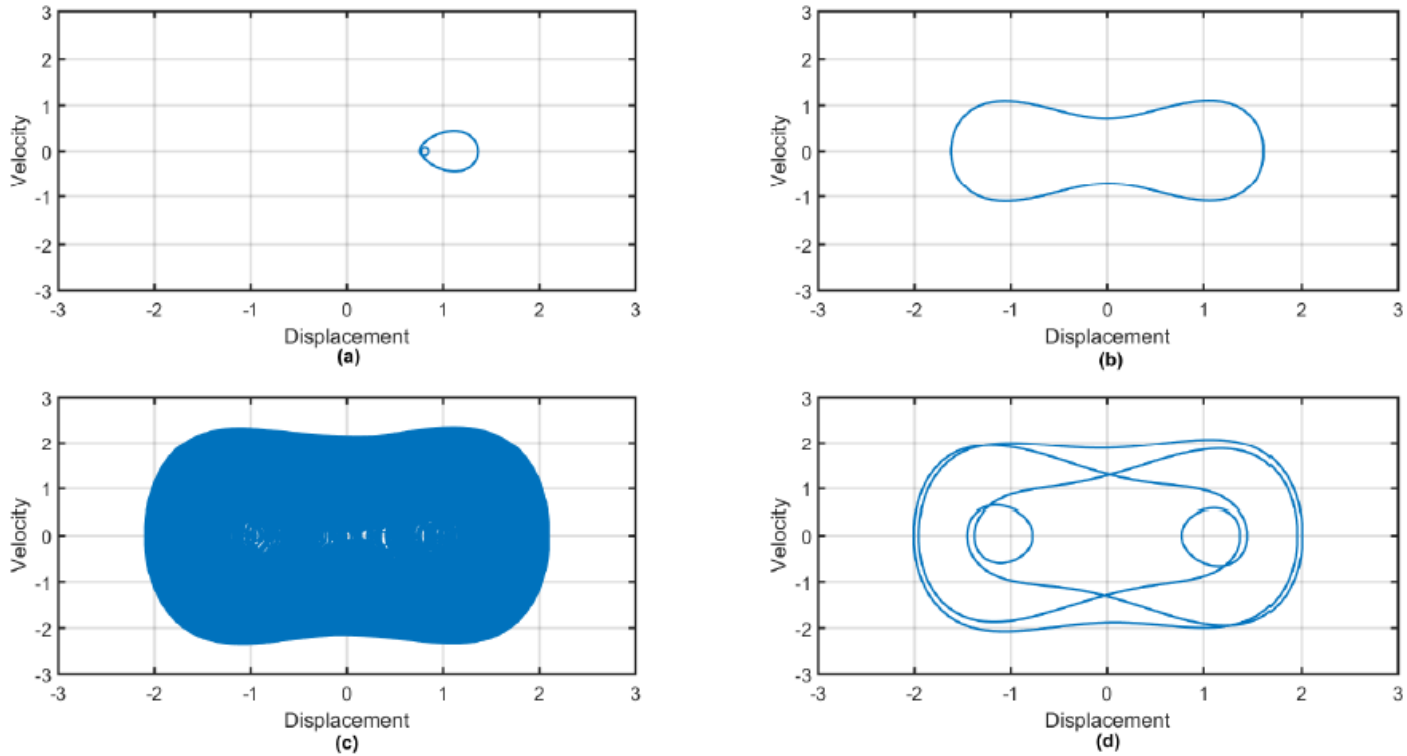
The Simulink model is runned

```
% Run Simulink
sim Harvest_DuffingCL
```

Finally, the space-state is plotted

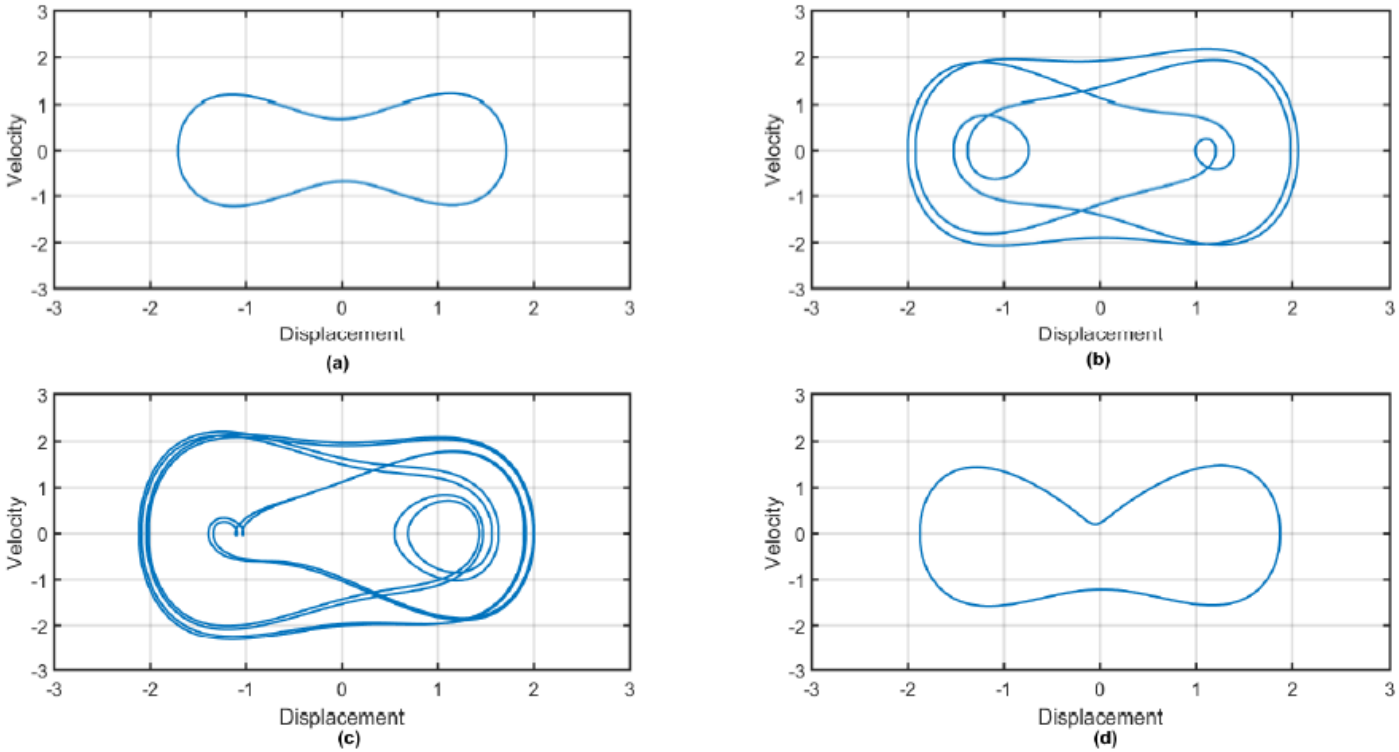
main_ETDF.m

Uncontrolled



main_ETDF.m

Controlled



Publications

 J V L L Peterson and V G Lopes and A Cunha Jr **On the nonlinear dynamics of bi-stable piezoelectric energy harvesting device**. Proceedings of the 24th International Congress of Mechanical Engineering - COBEM 2017, Curitiba, Brazil, 2017. DOI: 10.26678/ABCM.COBEM2017.COB17-1570

 V. G. Lopes and J. V. L. L. Peterson and A. Cunha Jr **Numerical study of parameters influence over the dynamics of a piezo-magneto-elastic energy harvesting device**. XXXVII Congresso Nacional de Matemática Aplicada e Computacional (CNMAC 2017), São José dos Campos, Brazil, 2017. DOI: doi.org/10.5540/03.2018.006.01.0407

 V.G. Lopes and J.V.L.L. Peterson and A. Cunha Jr **Nonlinear characterization of a bistable energy harvester dynamical system**. Topics in Nonlinear Mechanics and Physics, 228: 71-88, 2019. DOI: 10.1007/978-981-13-9463-8_3

 L de la Roca and J V L L Peterson and M P and A Cunha Jr **Control of chaos via OGY method on a bistable energy harvester**. Proceedings of the 25th International Congress of Mechanical Engineering - COBEM 2019, Uberlândia, Brazil, 2019. DOI: 10.26678/ABCM.COBEM2019.COB2019-1970

 V. G. Lopes and J. V. L. L. Peterson and A. Cunha Jr **Exploring the nonlinear dynamics of bistable energy harvester**. Proceedings of the XVIII International Symposium on Dynamic Problems of Mechanics, Buzios, RJ, Brazil, 2019

 V. G. Lopes and J. V. L. L. Peterson and A. Cunha Jr **The nonlinear dynamics of a bistable energy harvesting system with colored noise disturbances**. Journal of Computational Interdisciplinary Sciences, 10, 125, 2019

 J.P. Norenberg and A. Cunha Jr and S. da Silva and P. S. Varoto **Global sensitivity analysis on bistable energy harvester**. In: 3rd International Congress on Engineering Vibration (ICoEV 2020). Proceedings of ICoEV, 2020

A. Cunha Jr **Enhancing the performance of a bistable energy harvesting device via the cross-entropy method**. Nonlinear Dynamics, 103, 137-155, 2021. DOI: 10.1007/s11071-020-06109-0

J.P. Norenberg and A. Cunha Jr and S. da Silva and P. S. Varoto **Exploring the behavior of a bistable energy harvester via global sensitivity analysis**. In: International Conference on Advances in Energy Harvesting Technology (ICAEHT-2021), 2021, Lublin. Book of Abstracts of the International Conference on Advances in Energy Harvesting Technology (ICAEHT-2021), 2021

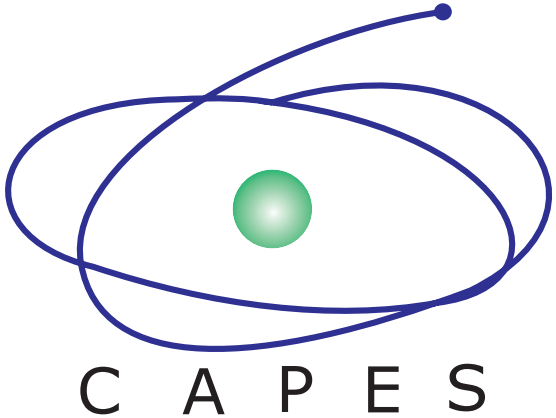
J.P. Norenberg and A. Cunha Jr and S. da Silva and P. S. Varoto **Sobol Global Sensitivity Analysis on a Bistable Energy Harvester**. In: 25th International Congress of Theoretical and Applied Mechanics (ICTAM 2020 + 1). Abstract book, 293:294, 2021

J. P. Norenberg and R. Luo and A. Cunha Jr and S. da Silva and P. Varoto **Remarks on the dynamic behavior of an asymmetric bistable energy harvester**. Proceedings of the 26th International Congress of Mechanical Engineering - COBEM 2021, Florianopolis, Brazil, 2021

J.P. Norenberg and A. Cunha Jr and S. da Silva and P. S. Varoto **Global sensitivity analysis of (a)symmetric energy harvesters**. arXiv, 2107.04647, 2021 (submitted)

Acknowledgment

Financial support



Institutional support