Université de Liège

NNM Computation in MATLAB

M. Peeters, L. Renson, G. Kerschen

October 2011

Useful preliminary reading

About NNMs:

Kerschen et al., Nonlinear normal modes, Part I: A useful framework for the structural dynamicist, Mechanical Systems and Signal Processing 23 (2009), 170-194.

About the algorithm:

Peeters et al., Nonlinear normal modes, Part II: Toward a practical computation using numerical continuation, Mechanical Systems and Signal Processing 23 (2009), 195-216.

MATLAB code sources

"**NNMcont**" directory:

Add this folder to the Matlab path

- "**NNMsources**"

  Matlab source functions of the algorithm for NNM computation

- "**CLASS_NL**"

  Different Matlab classes for defining nonlinearities

- "**Newmark**"

  Matlab source functions for numerical time integration using Newmark method

MATLAB code sources

"**NNMsources**" directory:

- "**NNMcontinuation_... .p**"

  Sources of the computation algorithm combining shooting and continuation methods

- "**NNMcont.p**"

  Graphical user interface (GUI) to launch and control the computation

Running MATLAB code

In the working directory, create two **user m-files**:

- "*defsys.m*" function

  Defining the dynamical system to study

- "*paramdisp.m*" script

  Setting the result display properties

Type "**NNMcont**" in the Matlab command window to start the GUI for NNM computation. The system defined by the "defsys.m" file is then automatically loaded.

System definition: defsys.m file

The system of interest is described in the “defsys.m” function by defining the **sys** struct array with the **specified fields**:

- **sys.Klin** and **sys.Mlin** fields (array nxn)

  stiffness and mass matrices of the underlying linear system

- **sys.nl** filed (nonlinear object)

  object of *nonlinear class* defining the nonlinearities present in the system

- Different **optional fields**

  sys.norm, sys.lGcont, sys.vitfix

System definition: defsys.m file

```
function  sys = defsys()

sys.Klin= …
sys.Mlin= …
sys.nl= …
…
```

Field	Description
sys.Klin	Stiffness matrix (ndof x ndof) of the linear system
sys.Mlin	Mass matrix (ndof x ndof) of the linear system
sys.nl	Nonlinear class object defining the nonlinear terms of the system. This object is defined by means of the class constructor function (see below).
sys.norm	Norm of the linear mode used as initial guess for determining a first NNM motion at low energy (if the result MAT-file does not exist) [**OPTIONAL**]
sys.IGcont	Use of another initial guess for continuation specified by: sys.IGcont.x0 (vector of initial displacements, velocities being assumed to be equal to 0) and sys.IGcont.w (pulsation of the initial guess) [**OPTIONAL**]
sys.vitfix	Indexes of the velocities to set to zero as phase condition [**OPTIONAL**]

System definition: defsys.m file

Duffing example:

$$\ddot{m}y + ky + k_{nl}y^3 = 0$$

```
function  sys = defsys()
m=…;k=…;knl=…;

sys.Klin= [k];
sys.Mlin= [m];

coeffnl=knl;expnl=3;
nl_law=NL_LAW_POLY(coeffnl,expnl);
pos1=0;pos2=1;
sys.nl= NL_SPRING(pos1,pos2,nl_law);
```

System definition: defsys.m file

2DOF example:

$$\begin{bmatrix} m1 & 0 \\ 0 & m2 \end{bmatrix} \begin{Bmatrix} \ddot{x}1 \\ \ddot{x}2 \end{Bmatrix} + \begin{bmatrix} k1 & k12 \\ k12 & k2 \end{bmatrix} \begin{Bmatrix} x1 \\ x2 \end{Bmatrix} + \begin{Bmatrix} knl1.x1^3 + knl12(x1-x2)^3 \\ knl2.x2^3 + knl12(x2-x1)^3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

```
function  sys = defsys()
m1=…;m2=…;k1=…;k2=…;k12=…;knl1=…;knl2=…;knl12=…;

sys.Klin= [k1 k12;k12 k2];
sys.Mlin= diag([m1 m2]);

coeffnl=knl1;expnl=3;nl_law=NL_LAW_POLY(coeffnl,expnl);
pos1=1;pos2=0;
sys.nl(1)= NL_SPRING(pos1,pos2,nl_law);

coeffnl=knl2;expnl=3;nl_law=NL_LAW_POLY(coeffnl,expnl);
pos1=2;pos2=0;
sys.nl(2)= NL_SPRING(pos1,pos2,nl_law);

coeffnl=knl12;expnl=3;nl_law=NL_LAW_POLY(coeffnl,expnl);
pos1=1;pos2=2;
sys.nl(3)= NL_SPRING(pos1,pos2,nl_law);
```

Nonlinear object class

Nonlinear class = *nonlinear element* defined in "**CLASS_NL**" directory:

- Oriented object style of Matlab
- Each class corresponds to a type of nonlinearity
- More general formulation
- Easy implementation of other kinds of nonlinearities by developing a new nonlinear class (in a specified directory)

Nonlinear object class

Principle ?

Defining different object classes in Matlab corresponding to different nonlinear elements

$$M\ddot{x} + Kx + f_{nl}(x) = 0$$

In each nonlinear object class, we implement the computation of:

- the relating nonlinear restoring forces $\mathbf{f}_{nl}(\mathbf{x})$
- the first derivative of the restoring force with respect to the displacements $\frac{\partial \mathbf{f}_{nl}(x)}{\partial x}$
- the corresponding nonlinear energy $E_{nl}(\mathbf{x})$

Nonlinear object class

In Matlab, each class is defined in a directory with the @ **symbol**. The name of the directory is the name of the class.

A class is then described by:

- a **constructor** function

```
>> help NAME_CLASS
```

- several **methods** functions

```
>> methods NAME_CLASS
```

Nonlinear object class

Existing classes for defining **nonlinear laws**

(in 📁 CLASS_NL\NL_LAW directory):

- 📁 @NL_LAW_POLY — *polynomial law*
  - nl_law_poly.m
  - f_nl.m, df_nl.m, Energy_nl.m
- 📁 @NL_LAW_LINPIECEWISE — *piecewise law*
  - nl_law_linpiecewise.m
  - f_nl.m, df_nl.m, Energy_nl.m
- 📁 @NL_LAW_UNILAT — *unilateral law*
  - nl_law_unilat.m
  - f_nl.m, df_nl.m, Energy_nl.m

Nonlinear object class

NL_LAW_POLY illustration:

```
>> help NL_LAW_POLY
  Constructor of polynomial nonlinear law objects (NL_LAW_POLY class)
  [nlobj] = NL_LAW_POLY(coeffnl,expnl)
  - coeffnl: coefficient of the polynomial law
  - expnl: exponent of the polynomial law
  f(x)=coeffnl*x^(expnl)

>> methods NL_LAW_POLY

 Methods for class NL_LAW_POLY:
 Energy_nl   NL_LAW_POLY  df_nl  f_nl

>> help NL_LAW_POLY.f_nl
  Computation of the nonlinear function relating to NL_LAW_POLY
  object:
  [f_nl] = f_nl(obj,x)
  - obj: NL_LAW_POLY object
  - x: function argument
```

Nonlinear object class

NL_LAW_POLY illustration:

Cubic law example:

```
>> obj_nl_law = NL_LAW_POLY(1,3);

>> f_nl(obj_nl_law,1)

ans =

     1

>> f_nl(obj_nl_law,2)

ans =

     8
```

Nonlinear object class

Nonlinear law object illustration

example_nl_law.m file:

```
clear all;

% quadratic law
obj1=NL_LAW_POLY(1,2);
% cubic
obj2=NL_LAW_POLY(1,3);
% unilateral law (e.g., Hertz contact)
obj3=NL_LAW_POLY(1,3/2);
obj3=NL_LAW_UNILAT(obj3);
% linear piecewise law
obj4=NL_LAW_LINPIECEWISE(1,10,1.5);
```

Nonlinear object class

Nonlinear law object illustration

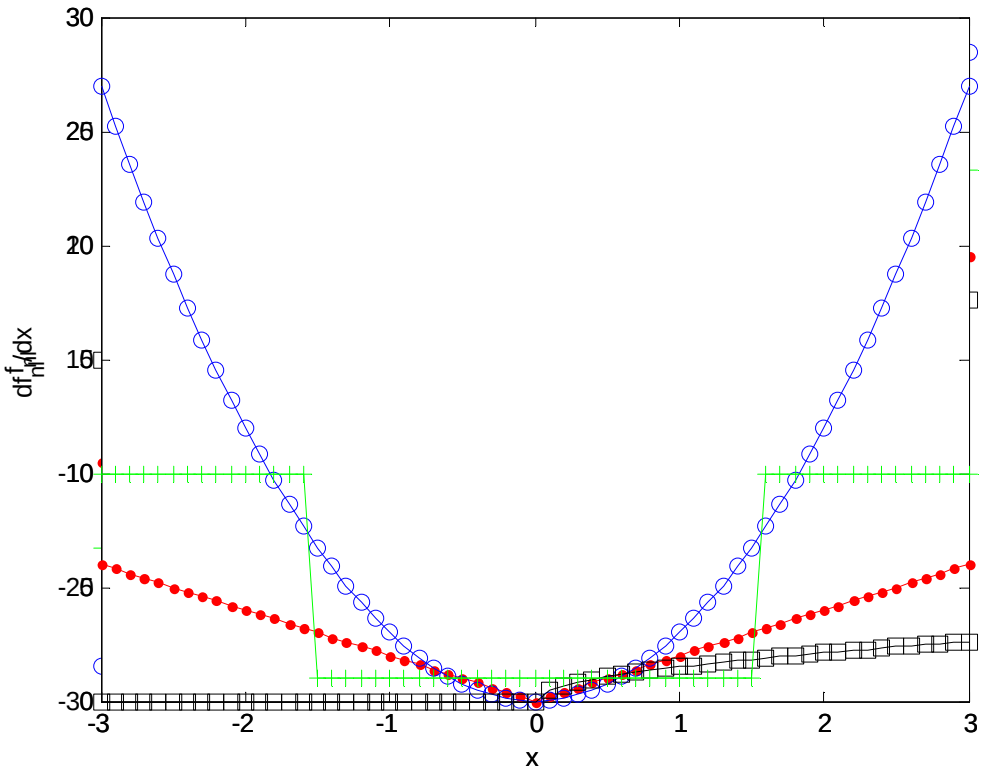
example_nl_law.m file

```
x=[-3:0.1:3];
f=zeros(length(x));
df=zeros(length(x));
k=1;
for y=x
    f1(k)=f_nl(obj1,y);df1(k)=df_nl(obj1,y);
    f2(k)=f_nl(obj2,y);df2(k)=df_nl(obj2,y);
    f3(k)=f_nl(obj3,y);df3(k)=df_nl(obj3,y);
    f4(k)=f_nl(obj4,y);df4(k)=df_nl(obj4,y);
    k=k+1;
end

figure
plot(x,f1,'.r-',x,f2,'ob-',x,f3,'sk-',x,f4,'+g-')
figure
plot(x,df1,'.r-',x,df2,'ob-',x,df3,'sk-',x,df4,'+g-')
```

Nonlinear object class

Nonlinear law object illustration

example_nl_law.m file



Nonlinear object class

Existing classes of **structural nonlinearties**

(in CLASS_NL\NL_ELEMENT directory):

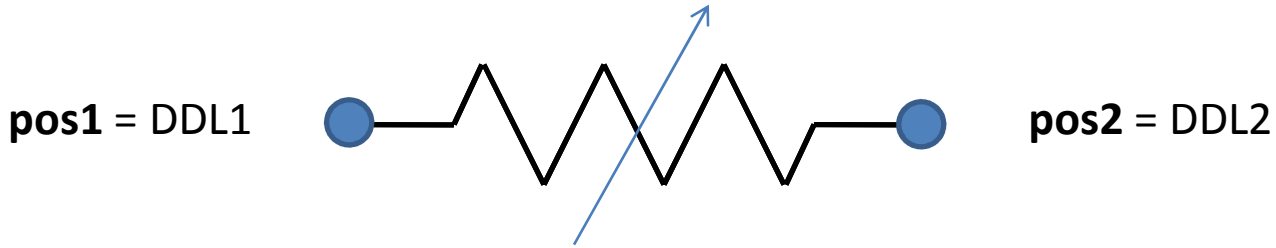
- @NL_SPRING — *nonlinear spring*
- @NL_USER — *nonlinearities directly defined by the user*

with the common methods functions:

- fint_nl.m — *nonlinear internal force vector*
- dfint_nl.m — *first derivative of internal forces*
- Energy_nl.m — *energy relating to the nonlinear element*

Nonlinear object class: NL_SPRING

1) NL_SPRING element



nl_law = object defining the nonlinear law of the spring (e.g., NL_POLY_LAW class object)

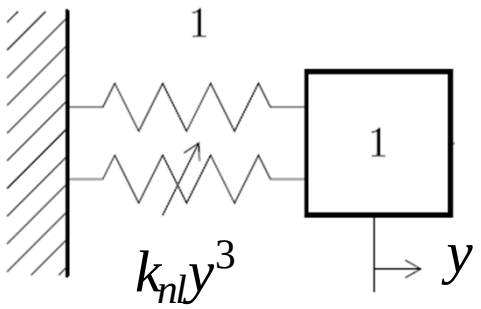
Constructor (see “help NL_SPRING”):

obj_nl=NL_SPRING(pos1,pos2,nl_law)

Nonlinear object class: NL_SPRING

Duffing example:

$$\ddot{m}y + ky + k_{nl}y^3 = 0$$




```
function  sys = defsys()
m=…;k=…;knl=…;

sys.Klin= [k];
sys.Mlin= [m];
```

```
coeffnl=knl;expnl=3;
nl_law=NL_LAW_POLY(coeffnl,expnl);
pos1=1;pos2=0;
sys.nl= NL_SPRING(pos1,pos2,nl_law);
```

Nonlinear object class: NL_SPRING

2DOF example:

$$\begin{bmatrix} m1 & 0 \\ 0 & m2 \end{bmatrix} \begin{Bmatrix} \ddot{x}1 \\ \ddot{x}2 \end{Bmatrix} + \begin{bmatrix} k1 & k12 \\ k12 & k2 \end{bmatrix} \begin{Bmatrix} x1 \\ x2 \end{Bmatrix} + \begin{Bmatrix} knl1.x1^3 + knl12(x1-x2)^3 \\ knl2.x2^3 + knl12(x2-x1)^3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

```
function  sys = defsys()
m1=…;m2=…;k1=…;k2=…;k12=…;knl1=…;knl2=…;knl12=…;

sys.Klin= [k1 k12;k12 k2];
sys.Mlin= diag([m1 m2]);
```

```
coeffnl=knl1;expnl=3;nl_law=NL_LAW_POLY(coeffnl,expnl);
pos1=0;pos2=1;
sys.nl(1)= NL_SPRING(pos1,pos2,nl_law);

coeffnl=knl2;expnl=3;nl_law=NL_LAW_POLY(coeffnl,expnl);
pos1=2;pos2=0;
sys.nl(2)= NL_SPRING(pos1,pos2,nl_law);

coeffnl=knl12;expnl=3;nl_law=NL_LAW_POLY(coeffnl,expnl);
pos1=1;pos2=2;
sys.nl(3)= NL_SPRING(pos1,pos2,nl_law);
```

Nonlinear object class: NL_USER

2) NL_USER element

Nonlinear terms specified by the functions

- fint_nl_user.m
- dfint_nl_user.m
- Energy_nl_user.m

in the working directory.

Constructor (see “help NL_USER”):

obj_nl=NL_USER()

Nonlinear object class: NL_USER

2DOF example using NL_USER class

$$\begin{bmatrix} m1 & 0 \\ 0 & m2 \end{bmatrix} \begin{Bmatrix} \ddot{x}1 \\ \ddot{x}2 \end{Bmatrix} + \begin{bmatrix} k1 & k12 \\ k12 & k2 \end{bmatrix} \begin{Bmatrix} x1 \\ x2 \end{Bmatrix} + \begin{Bmatrix} knl1.x1^3 + knl12(x1-x2)^3 \\ knl2.x2^3 + knl12(x2-x1)^3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$$

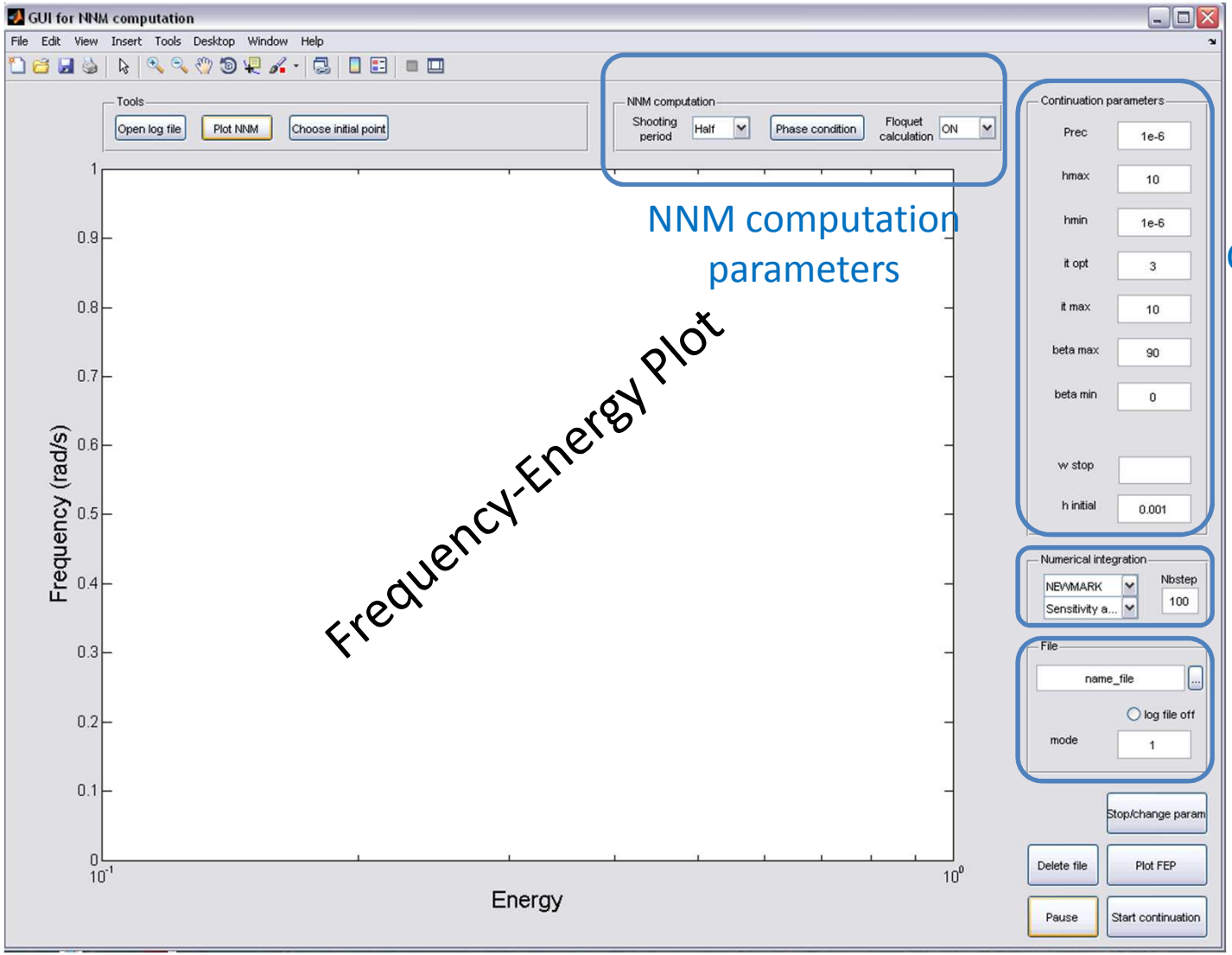
```
function  sys = defsys()
m1=…;m2=…;k1=…;k2=…;k12=…;

sys.Klin= [k1 k12;k12 k2];
sys.Mlin= diag([m1 m2]);

sys.nl= NL_USER();
```

GUI for NNM computation

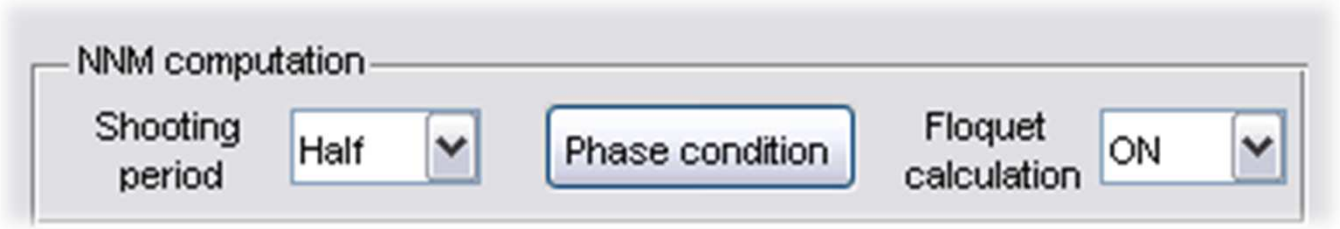
To launch the GUI for NNM computation:

```
>> NNMcont
```

GUI for NNM computation



GUI: NNM computation parameters



Time period of the shooting:
- Half period (symetric motions)
- Full period

Phase condition: Selection of the velocity to set to zero

Calculation of Floquet multipliers during the computation (ON / OFF)

GUI: Continuation parameters

Relative precision (residue) of the shooting

Maximum continuation stepsize

Optimal number of iterations for stepsize control

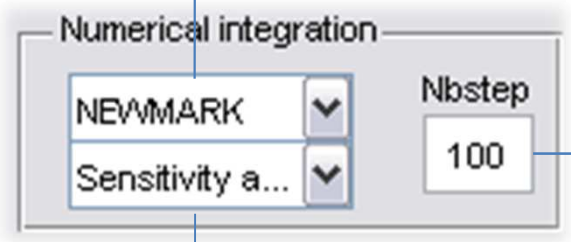
Maximum number of iteration

Maximum angle (deg) to avoid branch switching

Minumum angle (deg) for stepsize control

Initial continuation stepsize

GUI: Numerical integration parameters



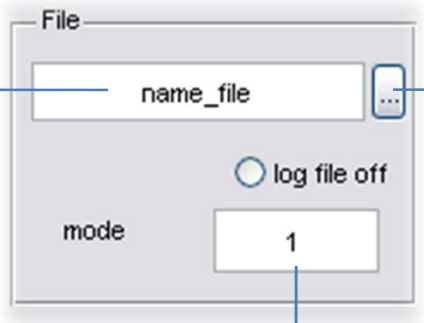
Scheme for numerical time integration:
- Newmark
- Runge-Kutta (ode45)

Number of time steps along one shooting period

Jacobian computation:
- Sensitivity analysis
- Finite differences

GUI: File parameters

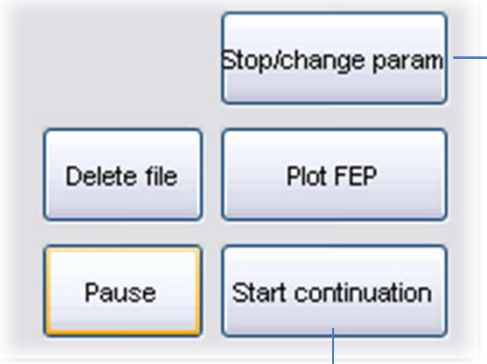
Name of the result MAT-file



Load a existing result MAT-file

Mode number to compute (NNM extension of this linear mode)

GUI: Starting computation



Interrupt the computation to change parameters

Start the computation:

- If the result MAT-file does not exist, the algorithm uses the linear mode as initial guess for starting the continuation procedure (or the sys.IGcont field defined in the defsys.m file).

- If the result MAT-file exists, the continuation is restarted from the last computed point.

Computation process and results

- The computed NNM is represented in the FEP of the GUI.
- Convergence information is displayed in the Matlab command window
- The computed results are saved in the MAT-file with the following variables:
  - **Pulsation**: vector containing the pulsation (rad/s) of the computed NNM motions
  - **Energy**: vector containing the conserved total enegy of the computed NNM motions
  - **Sol**: matrix composed of the initial states (displacements and velocities) inducing the different computed NNM motions
  - **Floq**: vector containing the Floquet multipliers of the computed NNM motions

Result display in GUI

In the FEP of the GUI, right click:

Frequency in rad/s or Hz in the FEP

Representation of unstable motions (using red markers)

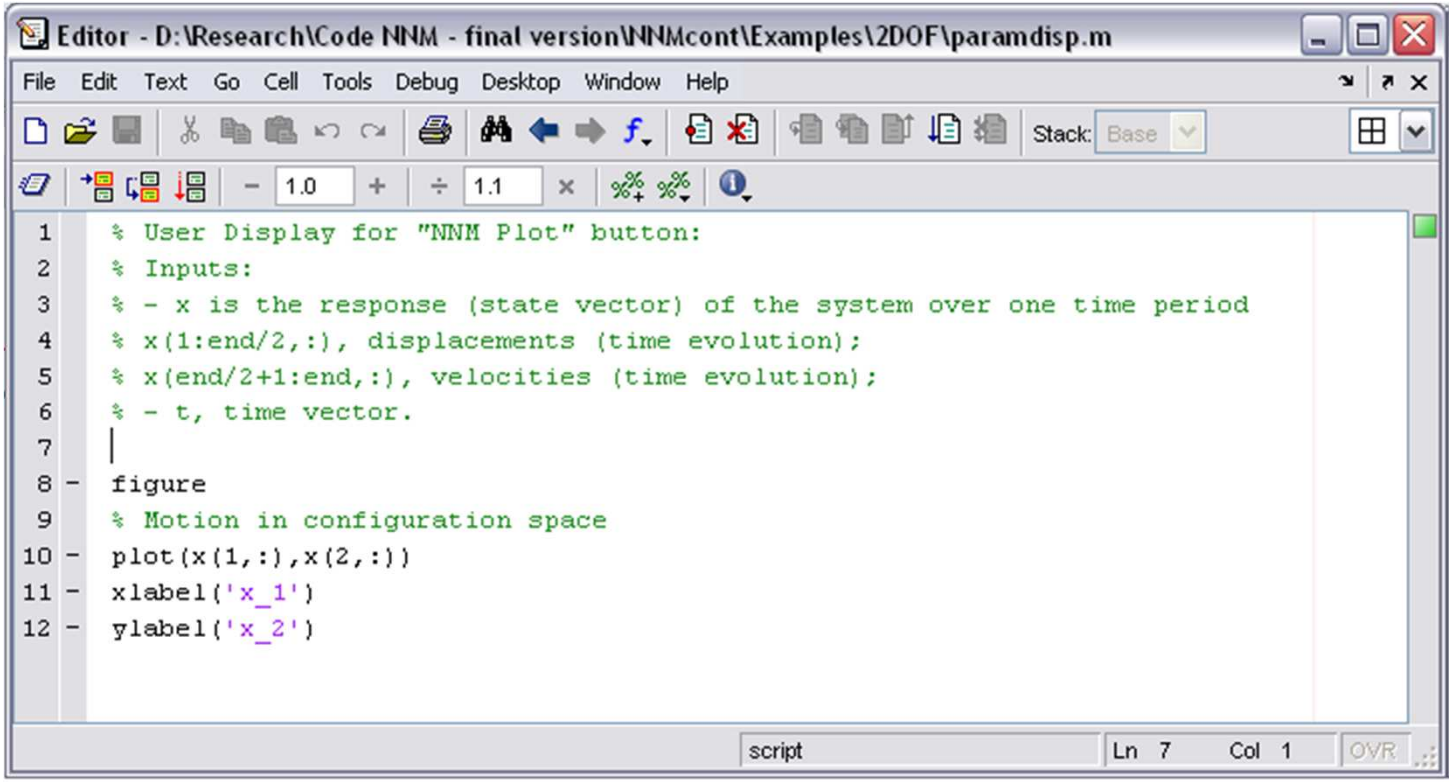
Result display in GUI

Display of computed NNM motions:

- Choose one computed point in the FEP
- The NNM motion is computed
- Some results are plotted by default (time series, Floquet multipliers…)
- Next, the user script *paramdisp.m* (in the working directory) is called

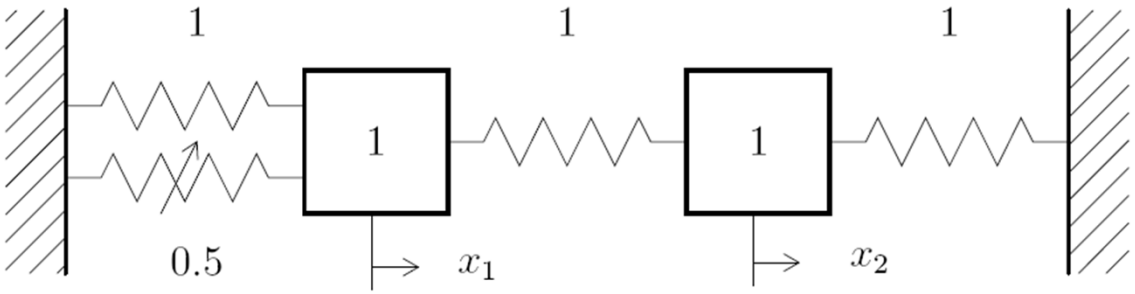
➔ You can configure the display by editing this file

Result display in GUI



Illustrations

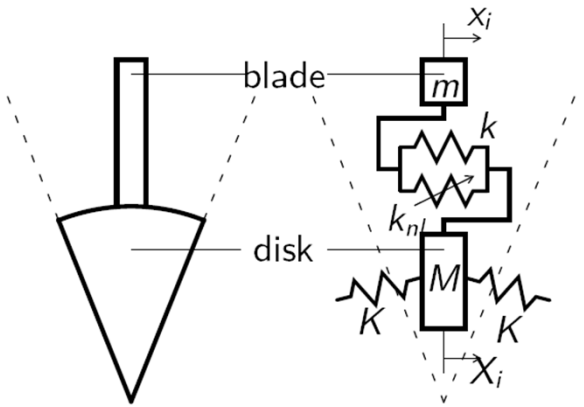
2DOF example (MSSP papers)



Directory “Examples\2DOF”

Illustrations

Simplified bladed disk (AIAA paper)



Simplified discrete model of bladed disk assembly (30 sectors)

Directory “Examples\BladedDisk”

References

Georgiades et al., Modal analysis of a nonlinear periodic structure with cyclic symmetry, AIAA Journal 47 (2009), 1014-1025.

Kerschen et al., Nonlinear normal modes, Part I: A useful framework for the structural dynamicist, Mechanical Systems and Signal Processing 23 (2009), 170-194.

Peeters et al., Nonlinear normal modes, Part II: Toward a practical computation using numerical continuation, Mechanical Systems and Signal Processing 23 (2009), 195-216.