Manlab 4.0 starting documentation

Louis GUILLOT, Bruno COCHELIN, Christophe VERGEZ

LMA - UMR 7031 Aix–Marseille Universités - CNRS - École Centrale de Marseille,
F-13402 MARSEILLE Cedex 20, France

April 19, 2018

Contents

Introduction

Manlab is a reasearch software suite devoted to the development and the promotion of the so-called Asymptotic Numerical Method (ANM), a continuation algorithm (or pathfollowing method) based on high order Taylor series expansions.

This software is written in MATLAB language using Object oriented programming. It can be downloaded on the dedicated website : `http://manlab.lma.cnrs-mrs.fr/`.

The graphical interface, the data base and the code structuration have been improved at the time where Automatic differentiation version of the ANM was introduced. The new software Diamanlab (written by Isabelle Charpentier and Bruno Cochelin) does not however support the continuation of periodic solution with the Harmonic Balance Method (HBM).

The present Manlab 4.0 (written by Louis Guillot) now includes a full re-implementation of the traditional ANM as in Manlab 2.0 and Manlab 3.0.

The base class `@Syst` of Diamanlab has been kept.

Two new classes `@SystAQ` and `@SystHBQ` were then implemented so as to fit previous research and developments on quadratic formulation of algebraic systems and differential algebraic equations (DAE) solved in combination with a HBM. Using a quadratic formulation is harder for the user but makes the Taylor series evaluation on which the asymptotic numerical method (ANM) relies far much faster.

This new implementation step gave us the opportunity to clean the code and to adopt some principles for these two `@SystAQ` and `@SystHBQ` classes:

- a clear definition of the original (main) variables of the system and the auxiliary variables used to obtain a quadratic formulation;
- the systematic use of a tensor formalism;
- an explicit construction of the Jacobian matrix using tensor rather than the evaluation of its columns one after the other, which was very time consuming;
- an optimised right hand side terms evaluation: the sum from $r = 1$ to $p - r$ of $Q(U_r, U_{p-r})$ has to be done at the lowest possible level to get the maximal efficiency;
- the bordering technique was re-introduced to solve linear systems so as to not destroy the band structure of the Jacobian matrices;
- the condensation of the auxiliary variables to improve the speed of the algorithm;

- a new order of the unknowns when using HBM or collocation techniques;
- taking advantage of what MATLAB does best, vector products, avoid as much as possible `for` loops, keep the number of code lines low.

This re-implementation is the result of many contributions due to Rémi Arquier (first Manlab version), Bruno Cochelin and Christophe Vergez (first version including HBM processing), Sami Karkar (tensor formalism, HBM, Collocation), El Hadi Moussi (Fast EHMAN, FFT, HBM Selective), Isabelle Charpentier (DA, Taylor series), Daniel Lampoh (Collocation+DA), Pernelle Marone-Hitz (BVP,Collocation), Olivier Thomas et Arnaud Lazarud (Stability), Emmanuelle Sarrouy (Continuation), Bruno Cochelin and Marc Médale (Bifurcation), Louis Guillot (Condensation, transcendental non–linearities).

To help the reader know which class should be used, here are typical situations coming from mechanical engineering and the classes which match the need.

Notations

- λ: parameter which varies. $\lambda \in \mathbb{R}$.
- $\mathbf{u}$: vector of unknowns of the primary (initial) of the problem. $\mathbf{u} \in \mathbb{R}^{N_{\text{eq}}}$.
- $\mathbf{U}$: augmented vector of unknowns. $\mathbf{U}^{\text{T}} = \{\mathbf{u}^{\text{T}}, \lambda\}^{\text{T}} \in \mathbb{R}^{N_{\text{eq}}+1}$.
- $\mathbf{U}_{\text{aux}}$: vector of auxiliary unknowns. $\mathbf{U}_{\text{aux}} \in \mathbb{R}^{N_{\text{eqaux}}}$.
- $\mathbf{U}$: vector containing all the unknowns. $\mathbf{U}_{\text{tot}}^{\text{T}} = \{\mathbf{u}^{\text{T}}, \lambda, \mathbf{U}_{\text{aux}}^{\text{T}}\}^{\text{T}} \in \mathbb{R}^{N_{\text{eqtot}}+1}$.
- $R(\mathbf{U}_{\text{tot}}) = 0$: set of N_{eq} equations depending on $N_{\text{eqtot}} + 1$ unknowns to be solved.
- $Raux(\mathbf{U}_{\text{tot}}) = 0$: set of N_{eqaux} equations depending on $N_{\text{eqtot}} + 1$ unknowns defining the auxiliary variables $\mathbf{U}_{\text{aux}}$.

Typical situations

- Static non linear problem: $\mathbf{K}(\lambda)\mathbf{u} + \mathbf{f}_{\text{nl}}(\mathbf{u}, \lambda) = \mathbf{f}_{\text{e}}(\lambda)$
  Such a problem can be solved using `@SystAQ` class. It needs a quadratic formulation.
- Dynamic non linear problem: $\mathbf{f}_{\text{nl}}(\mathbf{u}, \dot{\mathbf{u}}, \ddot{\mathbf{u}}, \lambda) = \mathbf{f}_{\text{e}}(t)$ where $\mathbf{u}$ is an unknown periodic function of the time.
  Such a problem can be solved using a HBM method via `@SystHBQ` class. It needs a quadratic formulation.

Chapter 1

Algebraic systems in quadratic format for efficiency: the `@SystAQ` class

Let us consider the ECrea example. The unknown is $\mathbf{U} = [u_1, u_2, \lambda]^{\mathrm{T}}$ and the equations are

$$\begin{array}{rl} r_1(u_1,u_2,\lambda) = & 2u_1 - u_2 + 100\frac{u_1}{1+u_1+u_1^2} - \lambda \quad = 0 \\ r_2(u_1,u_2,\lambda) = & 2u_2 - u_1 + 100\frac{u_2}{1+u_2+u_2^2} - (\lambda+\mu) \quad = 0 \end{array} \tag{1.1}$$

with μ a given parameter.

To use the `@SystAQ` class, this problem must be rewritten in a quadratic format. This is achieved by defining auxiliary variables: $v_1 = 1 + u_1 + u_1^2$, $v_2 = 1 + u_2 + u_2^2$, and $v_3 = 1/v_1$, $v_4 = 1/v_2$. One gets an augmented vector of unknown $\mathbf{U}_{\mathrm{tot}} = [u_1, u_2, \lambda, v_1, v_2, v_3, v_4]^T$ which should verify

$$\begin{array}{rcccccc} 0 & + & 2u_1 - u_2 - \lambda & + & 100u_1v_3 & = 0 \\ -\mu & - & u_1 + 2u_2 - \lambda & + & 100u_2v_4 & = 0 \\ -1 & + & v_1 - u_1 & - & u_1u_1 & = 0 \\ -1 & + & v_2 - u_2 & - & u_2u_2 & = 0 \\ -1 & + & 0 & + & v_1v_3 & = 0 \\ -1 & + & 0 & + & v_2v_4 & = 0 \end{array} \tag{1.2}$$

Remark: Here we have, $N_{\mathrm{eq}} = 2$ main equations, $N_{\mathrm{eqaux}} = 4$ auxiliary equations, $N_{\mathrm{eqtot}} = N_{\mathrm{eq}} + N_{\mathrm{eqaux}} = 6$ equations in total and $N_{\mathrm{inc}} = 7$ unknowns. The main N_{eq} equations are given first. Then the auxiliary variables are defined in the last N_{eqaux} equations.

The basic directory structure for a user defined system which is called "MySystem" and defined using the `@SystAQ` class should be

\somewhereinmytree	
MySystemLaunchSheet.m	Sheet launching Manlab 4.0 with options
equations.m	Sheet defining the equations of the system
point_display.m	Sheet defining user defined display of a point
global_display.m	Sheet defining user defined display of a solution–branch

The MySystemLaunchSheet.m file This files define the parameter of the System and creates the System before launching Manlab 4.0 with possible options. A typical `MySystemLaunchSheet.m` is:

```
% Some comments about the system studied

% Definition of the parameters of My System
parameters.prop1value = 30;
parameters.prop2value = 0.1;

% Creation of the SystAQ object
sys = SystAQ(neq,neq_aux,@equations, ...
    @point_display,@global_display,parameters)

% Starting point
lambda0 = 0;
u0 = zeros(neq,1);
U0 = [u0 ; lambda0];

v0 = zeros(neq_aux,1);

U0tot = [U0 ; U0tot];

% Running Manlab
Manlab(  'sys'     ,sys, ...
         'U0value' ,U0tot, ...
         'displayvariables',[1, 2]);
```

A `@SystAQ` class object is created when calling `SystAQ(neq,neq_aux,..)`. It is given to Manlab 4.0 as a first argument `'sys'`.
The arguments of SystAQ define the system. More precisely its number of equations before adding auxiliary variables, its number of auxiliary equations, the function defining the equations of the system, the two point–display and global–display functions and the parameters. See below for more precision about the handles.
The `'U0value'` argument is the starting point (column vector with size of $\mathbf{U}_{\text{tot}}$).
Finally `'displayvariables'` variables to be displayed in the projected bifurcation diagram are defined; here `[1,2]` means that $U(2)$ vs $U(1)$ will be drawn.

The equations.m file This files defines the quadratic formulation of the equations of the user system. Contrary to Manlab previous versions, there is no need to specify the operator C, L and Q one by one. Manlab 4.0 computes directly the operator from the quadratic recast of the equations, using polarization formulaes. It should look like :

```
function [Rtot,dRaux] = equations(sys,Utot,dUtot)
% The auxiliary variables are defined first and then the residue of the
% system considered.

%% Variables :  Utot=[ U  ; Uaux ]   with U=[ u ; lambda ]

u      = Utot(1:sys.neq);       % Main variables
lambda = Utot(sys.neq+1);       % Continuation parameter
Uaux   = Utot(sys.neq+2:end);   % Auxiliary variables

du     = dUtot(1:sys.neq);      % differential of Main variables
dUaux  = dUtot(sys.neq+2:end); % differential of Auxiliary variables

%% Residues
R     = zeros(sys.neq,1);       % Main residue
Raux  = zeros(sys.neq_aux,1);   % Auxiliary residue

% Differential form of non-quadratic part of the auxiliary residue
dRaux = zeros(sys.neq_aux,1);

% Definition of the auxiliary variables | differentiation of the non-quadratic equations
Raux(1) =  ;                              dRaux(1) =
Raux(2) =  ;
...

% Equations of the main system
R(1) =  ;
R(2) =  ;
...

% Concatenation of the two residues
Rtot=[R ; Raux];

end
```

The equations defining the auxiliary variables are given after the equations of the system. It is important not to change this order.

The point_display.m file The `point_display.m` file which defines the display of a solution should look like

```
function [] = point_display(sys,Utot)
% This is the function used to display during the computation of the
% branches of solution.

u      = Utot(1:sys.neq);
lambda = Utot(sys.neq+1);
Uaux   = Utot(sys.neq+2:end);

%%% Plot the value of the main variables but can do anything
figure(11)
bar(u)

end
```

The global_display.m file The `global_display.m` file which defines the solution–branch display at the end of each section should look like

```
function [] = global_display(sys,Section)
% This is the function used to display during the computation of the
% branches of solution.

% Section.U0  (Taylor series) contains the Taylor series used to describe the section.
% Section.Amax (real positive number) is the domain of validity of the series.
% Section.Upp (ninc,nb_pts) is the point representation of the series.

u1 = Section.Upp(1,:);
lambda = Section.Upp(sys.neq+1,:);

%%% plot the value of the first variable u1 with respect to the value of
%%% the continuation parameter lambda along the section
figure(5)
plot(lambda,u1);hold on;

end
```

Chapter 2

Differential algebraic systems in quadratic format solved using HBM: the `@SystHBQ` class

It is possible to continue periodic solutions of differential-algebraic equations using the Harmonic Balance Method (HBM) in Manlab 4.0 by providing the system to be solved. The system originally looks like :

$$\mathbf{f}(t, \mathbf{Y}(t), \dot{\mathbf{Y}}(t), \ddot{\mathbf{Y}}(t), \lambda) = \mathrm{F}(t) \tag{2.1}$$

Where the vector $\mathbf{Y}$ is of size nz and where $\mathbf{f}$ has its values in $\mathbb{R}^{nz}$ *i.e.* there are nz equations in total.

λ is here explicitly considered as a "special" parameter compared to $\mathbf{Y}$ components; indeed, $\mathbf{Y}$ components, denoted y_i will be further expanded into a Fourier series but λ as well as the circular frequency ω will not. Hence, these two variables have to be identified and processed separately.

To solve this system with the `@SystHBQ` class, we rewrite this implicit system in the following quadratic form :

$$dd(\ddot{\mathbf{Y}}_{\mathrm{tot}}) + \lambda d_1(\dot{\mathbf{Y}}_{\mathrm{tot}}) + d(\dot{\mathbf{Y}}_{\mathrm{tot}}) + c_0 + \lambda c_1 + l_0(\mathbf{Y}_{\mathrm{tot}}) + \lambda l_1(\mathbf{Y}_{\mathrm{tot}}) + q(\mathbf{Y}_{\mathrm{tot}}, \mathbf{Y}_{\mathrm{tot}}) = \mathrm{F}(t) \tag{2.2}$$

Where $\mathbf{Y}_{\mathrm{tot}} = [\mathbf{Y}; \mathbf{Y}_{\mathrm{aux}}]$ contains the original degrees of freedom of the system and some auxiliary variables needed to write the system (2.1) in the quadratic form (2.2). $\mathbf{Y}_{\mathrm{tot}}$ is of size $nz_{\mathrm{tot}} = nz + nz_{\mathrm{aux}}$ where nz_{aux} is the size of $\mathbf{Y}_{\mathrm{aux}}$ which is the number of (additional) auxiliary variables needed to write the quadratic recast. $\mathrm{F}(t)$ is a periodic forcing term that can be null.

The basic directory structure for a user defined system which is called MySystem and defined using the `@SystHBQ` class should be

\somewhereinmytree	
MySystemLaunchSheet.m	Sheet launching Manlab 4.0 with options
equations.m	Sheet defining the equations of the system
point_display.m	Sheet defining user defined display of a point
global_display.m	Sheet defining user defined display of a solution–branch

It is exactly the same structure as to define a system using the @SystAQ class.

The MySystemLaunchSheet.m file It defines the parameters of MySystem and creates the System before launching Manlab 4.0 with possible options. A typical MySystemLaunchSheet.m is:

```
% Write here the description of your system, the auxiliary variables used,
% and its final quadratic recast

global U Section Diagram   % Global variables to export point from the diagram.

% Path of the SRC file.
addpath('../../SRC')

%% Parameters of the system
nz= 1;                % number of main equations of the DAE
nz_aux =2;            % number of auxiliary equations of the DAE

H = 20;               % number of harmonics used to compute the solution-branch

type = 'forced';      % type of system ('autonomous' or 'forced')
forcing_pulsation=1; % if the system is forced at a constant pulsation,
                     % forcing_pulsation contains its value.
                       % Otherwise, forcing_pulsation = 'omega'.

%%% Parameters specific to your system
parameters.param1 = 0.5;
parameters.param2 = 1;

%% initialization of the system
sys=SystHBQ(nz,nz_aux,H,@equations, ...
    @point_display,@global_display,parameters,type,forcing_pulsation);

%% starting point
lambda0=0;    % Continuation parameter initial value
omega0=1;     % Pulsation initial value

Z0 =zeros(2*H+1,sys.nz_tot);
% The matrix Z0 contains in column the Fourier development of all the
% variables of your system :
% Z0 = [ u , v ]
% where u = [ u_0 ; u_c1 ; u_c2 ; ... ; u_cH ; u_s1 ; u_s2 ; ... ; u_sH ];
% u_0 is the constant Fourier coefficient, u_c1 the first cosine Fourier
% coefficient, u_s1 the first sine Fourier coefficient, etc...
Z0(1,1)      = u0;
Z0(2,1)      = uc1;
Z0(1+j,1)    = ucj; % with 1 <= j <= H
Z0(2+H,1)    = us1;
```

```
Z0(1+H+j,1) = usj; % with 1 <= j <= H

%%% Vector U0 containing all the unknowns.
U0 = sys.init_U0(Z0,omega0,lambda0);

%% Launch of Manlab with options
Manlab('sys'       ,sys , ...
    'U0value'         ,U0, ...
    'displayvariables',[ sys.neq+1  2 ]);     % MANLAB run
%                         lambda   uc1 - first cosine of the first variable
```

A `@SystHBQ` class object is created when calling `SystHBQ(nz,nz_aux,H,..)`. It is given to Manlab 4.0 as a first argument `'sys'`.
The arguments of SystHBQ define the system. More precisely :

- `nz` is the number of equations of MySystem before adding auxiliary variables;
- `nz_aux` is the number of auxiliary variables needed to have the form (2.2) for MySystem;
- `H` is the number of harmonics to resolve MySystem;
- `@equations` is the handle of the function defining all the equations of MySystem;
- `@point_display` is the handle of the user-defined display function of a solution;
- `@global_display` is the handle of the user-defined display function of a solution–branch;
- `parameters` is a structure containing the parameters of MySystem;
- `type` is a string containing the type of MySystem which can be autonomous or forced;
- `forcing_pulsation` is an optional argument containing the value of the pulsation of the oscillation of MySystem when it is fixed by the forcing terms;

The `'U0value'` argument is the starting point (column vector with size of $\mathbf{U}_{\text{tot}}$). In `@SystHBQ` class, the user has to give the value of the auxiliary variables of the system.
Finally `'displayvariables'` variables to be displayed in the projected bifurcation diagram are defined; here `[sys.neq+1,2]` means that *lambda* vs $u_{c1}^{(1)}$, the first cosine of the first variable, will be drawn.

The equations.m file This files defines the quadratic formulation of the equations of the user system. Contrary to Manlab previous versions, there is no need to specify the operator C, L and Q one by one. Manlab 4.0 computes directly the operator from the quadratic recast of the equations, using polarization formulaes. It should look like :

```
function [Rtot,dRaux,Forced] = equations(sys,t,Utot,dUtot,d2Utot)
% Equations of the system of the form
% R(U) = C + L0(U) + lambda L1(U) + D0(dU) +
% lambda D1(dU) + DD(d2U) + Q(U,U) + f(U).
```

```
%%% parameters of the system
param1 = sys.parameters.param1;
param2 = sys.parameters.param2;
param3 = ...

%% Variables :

u      = Utot(1:sys.nz);          % Main variables
Uaux   = Utot(sys.nz+1:end-1);    % Auxiliary variables
lambda = Utot(end);               % Continuation parameter

du     = dUtot(1:sys.nz);         % first order derivated of Main variables
dUaux  = dUtot(sys.nz+1:end-1);   % first order derivated of Auxiliary variables

d2u    = d2Utot(1:sys.nz);        % second order derivated of Main variables
d2Uaux = d2Utot(sys.nz+1:end-1); % first order derivated of Auxiliary variables

%% Residues
R     = zeros(sys.nz,1);          % Main residue
Raux  = zeros(sys.nz_aux,1);      % Auxiliary residue
% Differential form of non-quadratic part of the auxiliary residue
dRaux = zeros(sys.nz_aux,1);

% Definition of the auxiliary variables | differentiation of the non-quadratic equations
Raux(1) =  ;                              dRaux(1) =
...

% Equations of the main system
R(1) =  ;
...

% Concatenation of the two residues
Rtot=[R ; Raux];

%% Forcing terms
% Should be written as if the forcing pulsation value is 1
% i.e. the forcing period is 2*pi
Forced = zeros(2*sys.H+1,sys.nz_tot); % DO NOT CHANGE this line.

% if the equation number k is forced, write the forcing in Forced(:,k) :
Forced(:,k) = forcing_function(t);

end
```

The equations defining the auxiliary variables are given after the equations of the system. It is important not to change this order.

The point_display.m file An example of a `point_display.m` file which defines the display of a solution looks like

```
function [] = point_display(obj,Utot)

%Temporal reconstruction of Ut(nt,nz)
H=obj.H;
% Get the matrix of Fourier coefficients, omega and lambda
[Ztot,omega,lambda]=obj.get_Ztot(Utot);
T=2*pi/omega;           % period of the oscillation
time=0:T/(4*H+1):T;     % time vector (over one period)
nt=length(time);        % number of points per period
```

```
%%% Construction of the matrix that transform the frequency domain vector
%%% Ztot in the time domain vector Ut
VectCos=cos(omega*time'*(1:H));
VectSin=sin(omega*time'*(1:H));
MAT=[ ones(nt,1) , VectCos , VectSin ];  % MAT(nt,2*H+1)

Ut =MAT*Ztot;                % Ut(nt,nz_tot)
dUt=MAT*omega*obj.D(Ztot);   % dUt/dt(nt,nz_tot)

%%% Time evolution of the first variable and its derivative
figure(11);
plot(time,Ut(:,1),'b',time,dUt(:,1),'r')
title(['Time evolution of u and du/dt with ' ...
    num2str(H) ' harmonics'],'Fontsize',16)
xlabel('time (s)');legend(['u'],['du/dt'],'Location','SouthEast');grid on;

%%% Phase diagram of the first variable
figure(12)
plot(Ut(:,1),dUt(:,1))
title(['Phase diagram (u, du/dt) with ' num2str(H) ' harmonics'],'Fontsize',16)
xlabel('u');ylabel('du/dt');

end
```

The global_display.m file An example of a `global_display.m` file which defines the solution–branch display at the end of each section looks like

```
function [] = global_display(obj,Section)
% Section is a 'CheckPoint class' object
% containing all the information about the series:
% discrete representation, bifurcation analysis,...

Upp = Section.Upp;           % Point representation of the series

H=obj.H;                     % Number of harmonics
nz_tot=obj.nz_tot;           % Total number of variable (main and auxiliary)
tol_h = 1e-4;                % Tolerance for the convergence of the Fourier series

nb_pt = size(Upp,2);         % Number of points per Section

h = zeros(nb_pt,nz_tot);

%% Loop on the point representation of the series
for k = 1:nb_pt
    % k-th point of the point representation of the series
    Uk = Upp(:,k);
    % Get the matrix of Fourier coefficients, omega and lambda
    [Zk,omega(k),lambda(k)] = obj.get_Ztot(Uk);

    %Computation of the number of harmonics needed to reach the Fourier
    % norm of the i-th variable with tol_h tolerance.
    for i=1:nz_tot
        norm_Z = norm(Zk(:,i));
        partial_norm_Z = norm(Zk(1,i));
        while (partial_norm_Z/norm_Z<1-tol_h)
            h(k,i) = h(k,i)+1;
            ind = [ (1:h(k,i)+1) (H+2:H+1+h(k,i)) ];
            partial_norm_Z = norm(Zk(ind,i));
```

```
        end
    end

end

figure(6);
% plot the number of harmonics needed to have a "good" representation of the solution
for i=1:nz_tot
    if i>obj.nz; color = 'b'; else; color = 'r'; end
    pl(i)=plot(lambda,h(:,i),[color '-*']);hold on;grid on;
end
title(['Number of harmonics needed to reach the ...
    Fourier norm with ' num2str(tol_h) ' precision.'])
legend([pl(1) pl(end)],'Main variables', ...
    'Auxiliary variables','Location','SouthEast');

end
```

Appendix A

Manlab 4.0 command options

When launching Manlab 4.0, several pairs of arguments can be passed to the function in the following way:

```
Manlab('argument1_name',argument1_value,...
          'argument2_name',argument2_value,...
                ... etc. ...
          'argumentN_name',argumentN_value);
```

Here is a list of the mandatory and optional ones:

Mandatory arguments

- `'sys'`: an instance of user-defined class
- `'U0value'`: column vector giving U value used as start point; if $R(U)$ does not satisfies the accuracy criterion, Newton iterations are applied.
- `'displayvariables'`: a $p \times 2$ array of index of U components to be displayed in the projected bifurcation diagram.

Optional arguments

- `'order'`: order of the Taylor Series used to build sections of branches. Default is `20`.
- `'ANMthreshold'`: threshold above which the Taylor series is considered inaccurate (limits a_{max} value). Default is `1e-6`. It can be modified at any time via the GUI interface.
- `'Amax_max'`: Maximum value for a_{max}. Default is `1e6`. It can be modified at any time via the GUI interface.
- `'NRthreshold'`: threshold under which Newton iterations stop. Default is `2e-5`. It can be modified at any time via the GUI interface.
- `'NRmethod'`: indicates whether Newton iterations should be applied to satisfy `NRthreshold` each time a new section starts (based on start point or end point of previous section). Value is `0` (no Newton iteration) or `1` (Newton iteration). It can be modified at any time via the GUI interface. Default is `1`.

- `'NRitemax'`: Maximum number of Newton iteration for correction. Default is `10`.
- `'BifDetection'`: indicates whether Manlab 4.0 computes the simple bifurcation informations when one is detected. Value is `0` (bifurcation computation off) or `1` (bifurcation computation on). It can be modified at any time via the GUI interface. Default is `1`.
- `'PointDisplay'`: indicates whether Manlab 4.0 plots the solution at then end of each computation of a solution–branch. Default value is `1` (point display on). It can be modified at any time via the GUI interface.
- `'GlobalDisplay'`: indicates whether Manlab 4.0 plots the solution–branch at then end of each computation. Default value is `1` (global display on). It can be modified at any time via the GUI interface.
- `'StabilityCheck'`: indicates whether Manlab 4.0 computes the stability of the solution–branch. Default value is `0`. It can be modified at any time via the GUI interface.
- `'StabTol'`: Tolerance used in the stability computation. Default value is `1e-6`.

Warning. The stability computation should not be used if you are not familiar with Manlab 4.0. It requires a special writing of the system of equations that is explained in some of the examples that goes with the starting package of Manlab 4.0. It uses Hill's method for the stability of periodic solutions. The equations must come from an ODE system written in the form $\mathbf{f}(y(t)) - \mathbf{M}\dot{y}(t) = \mathrm{F}(t)$.