MATLAB-BASED TOOLS FOR NONLINEAR SYSTEMS

M. Ondera

Department of Automation and Control, Faculty of Electrical Engineering and Information Technology
Slovak University of Technology in Bratislava

Abstract

Custom tools for MATLAB supporting analysis and design of nonlinear control systems are introduced in this paper. The paper demonstrates their use to solve sample nonlinear control problems and presents some of the MATLAB algorithms involved. A lot of attention is dedicated to a rather unusual collaboration of Simulink and Symbolic Math Toolbox that was employed in creation of the tools.

1 Introduction

Nonlinear control systems have one principal disadvantage in comparison with their linear counterparts – there is no general nonlinear control theory, which means that it is impossible to find universal methods valid for analysis and/or synthesis of the whole class of nonlinear systems. Instead, techniques whose applicability is limited to a certain subgroup of systems with several properties in common are used. This "non-uniformity" of nonlinear techniques is probably one of the reasons why methods of nonlinear control do not have as good support in professional program products as linear ones. And yet, considering higher difficulties connected with nonlinear methods, some computer aid to support them would be more than convenient.

In order to cope with this situation (at least partially), we have created custom MATLAB[*]-based program tools supporting several methods of nonlinear control theory and grouped them into a toolbox named *NelinSys*. This way we tried to bridge the gap between what can be solved with MATLAB (in the field of nonlinear control) and what is offered by MATLAB as ready-made functions, blocks or toolboxes. The *NelinSys* toolbox contains tools for four methods – phase plane analysis, analysis of limit cycles via harmonic balance, exact linearization and gain scheduling. Other tools are being prepared, e.g. a tool for velocity linearization, which is in fact already made, although not included in the toolbox yet – see [3] for details.

The paper is organised as follows. §2 introduces the *NelinSys* toolbox by describing its four modules. It also contains a short description of each of the nonlinear methods that are supported by the toolbox – a more comprehensive description of the methods is available in [1]. §3 then shows how to apply the tools in order to solve sample problems of nonlinear control theory. §4 deals with the implementation of the tools and discusses the most important parts of MATLAB source code, especially those that ensure collaboration of Simulink and Symbolic Math Toolbox. Finally, §5 concludes the paper.

2 Description of the tools

The *NelinSys* toolbox consists of four modules, each of them providing support for different nonlinear method. All of the modules are described in this section.

First of the tools included in the *NelinSys* toolbox is dedicated to phase-plane analysis, which is a simple graphical method for analysis of 2nd-order nonlinear systems. Its main idea is to construct motion trajectories of a 2nd-order system corresponding to different initial conditions in phase plane and determine qualitative features of the system – number, types and stability of its equilibrium points and/or limit cycles – by their examination. Analysis of a nonlinear system via phase-plane consists of two basic steps – construction of a phase-plane portrait and its qualitative evaluation. From these two, usually the first step is more time-consuming than the other. Therefore, the aim of the *NelinSys* phase-plane analysis tool is to give a user an opportunity to construct phase-plane portraits easily. The tool is composed of several Simulink blocks (see Figure 1) from which a user chooses those he needs according to the system specifications (1st-order, 2nd-order, with or without hard nonlinearities) and

[*] In the whole paper, every reference to MATLAB relates to the version 5.2.

interconnects them into a simulation scheme – in the simplest case the scheme involves only two blocks, one for calculation of a numerical solution of the system and the other for graphical plot of the phase-plane portrait. Another advantage of using Simulink for obtaining phase-plane portraits is that while the simulation is running, user can observe how the portrait is being generated and get additional information about the nonlinear system this way.

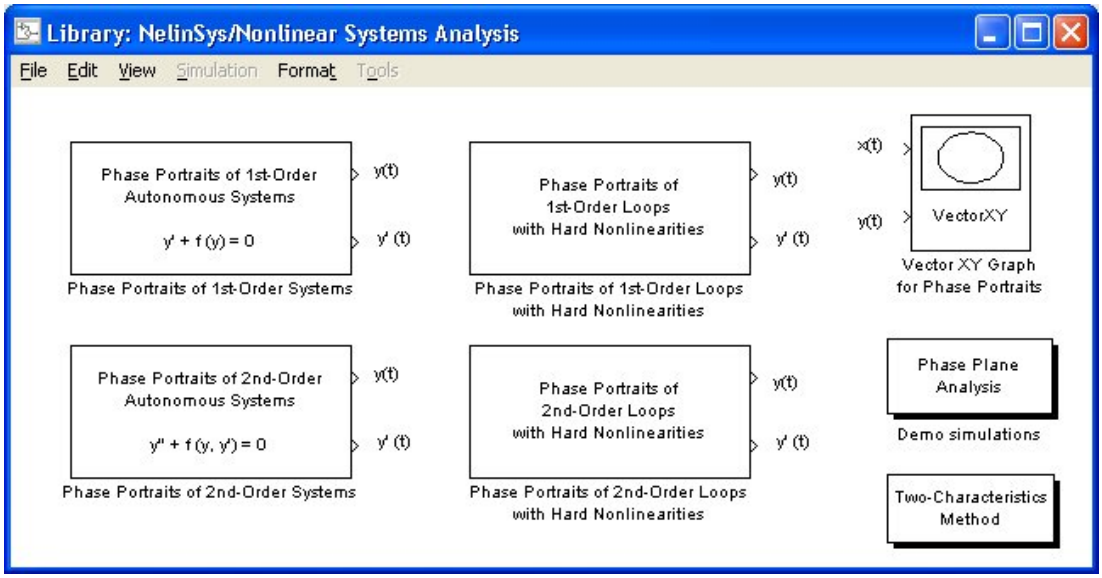

Figure 1: The *NelinSys* toolbox – Phase-Plane Analysis blockset

The second tool from the *NelinSys* toolbox is devoted to limit cycle analysis based on harmonic balance or, more precisely, to a so-called two-characteristics method. The method is suitable for analysis of nonlinear control loops consisting of a high-order linear part and an isolated static nonlinearity (Figure 2). There are two basic principles that the method takes advantage of: first, if the linear part of the system works as a low-pass filter, then (for calculation purposes) the nonlinearity can be substituted by a so-called describing function – see [1] or [4] for further details – and second, if there exist a limit cycle in the system, then its approximate amplitude a and approximate frequency ω can be determined by a solution of the harmonic balance equation

$$F_L(\mathrm{i}\omega) = -\frac{1}{F_e(a)} \tag{1}$$

where $F_L(\mathrm{i}\omega)$ stands for the frequency transfer function of the linear part and $F_e(a)$ for the describing function of the nonlinearity. Equation (1) can be solved either analytically, or graphically. In the latter case, which is sometimes referred to as the Two-Characteristics Method, two parametric curves are plotted in the complex plane, one that corresponds to the Nyquist plot of the linear part $F_L(\mathrm{i}\omega)$ and the other to the graphical plot of the right-hand side of the equation (1) i.e. to the term $-1/F_e(a)$. If the two curves have an intersection, then a limit cycle exists in the system and its quantitative parameters are the values of a and ω corresponding to the intersection. In case of more intersections, there are more limit cycles with different amplitudes and/or frequencies.

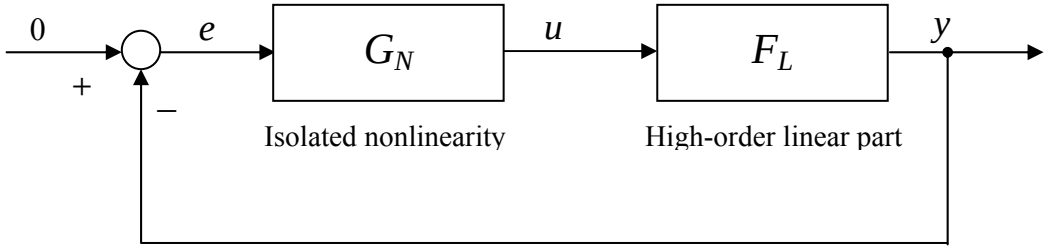

Figure 2: Nonlinear system suitable for harmonic-balance analysis of limit cycles

The Two-Characteristics Method tool included in the *NelinSys* toolbox gives a hand with the decision if there exists a limit cycle in a user-specified nonlinear system (of the Figure 2 structure) by plotting

the two curves in the complex plane (amplitude and frequency ranges are defined by user). Moreover, besides GUI version of the tool (Figure 3) there is also a text-based version, which is able to calculate the parameters of the limit cycle(s) analytically as well, by employing Symbolic Math Toolbox. The tool supports four types of nonlinearities so far – ideal relay, relay with dead-zone, relay with hysteresis and saturation.

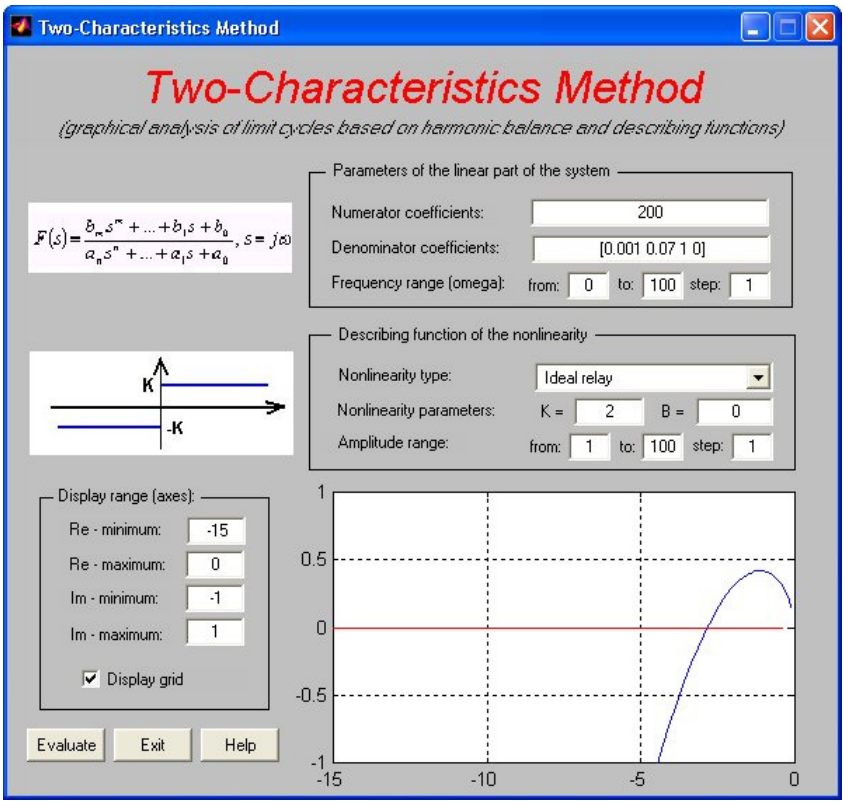

Figure 3: The *NelinSys* toolbox – Two-Characteristics Method tool (GUI version)

Next of the *NelinSys* tools deals with exact linearization. The objective of the method is to control a nonlinear system given in the form

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{2}$$

by turning it into a linear and controllable one i.e. such that can be described by linear state-space equations

$$\begin{aligned} \dot{q} &= Aq + Bv \\ y &= Cq \end{aligned} \tag{3}$$

with a help of nonlinear state-space transformation $q = q(x)$ and nonlinear state feedback $u = u(x,v)$. Once this is done, we can design a stabilizing linear controller for the linearized input v using arbitrary suitable linear control design method, e.g. pole placement. If (2) is a controllable SISO system, then the objective of the method can be met by applying nonlinear transformation

$$q = q(x) = \begin{pmatrix} h(x) \\ L_f h(x) \\ \dots \\ L_f^{r-2} h(x) \\ L_f^{r-1} h(x) \end{pmatrix} \tag{4}$$

and nonlinear state feedback

$$u = \frac{1}{L_g L_f^{r-1} h(x)} \left(- L_f^{r} h(x) + v \right) \tag{5}$$

The L is a so-called Lie derivative operator defined as

$$L_f h(x) = \sum_{i=1}^{n} \frac{\partial h(x)}{\partial x_i} f_i(x) \tag{6}$$

and r is the relative degree of the system; it has to be equal to the system order n for the transformation (4) to be complete. A more detailed description of exact linearization can be found in [1] or [4].

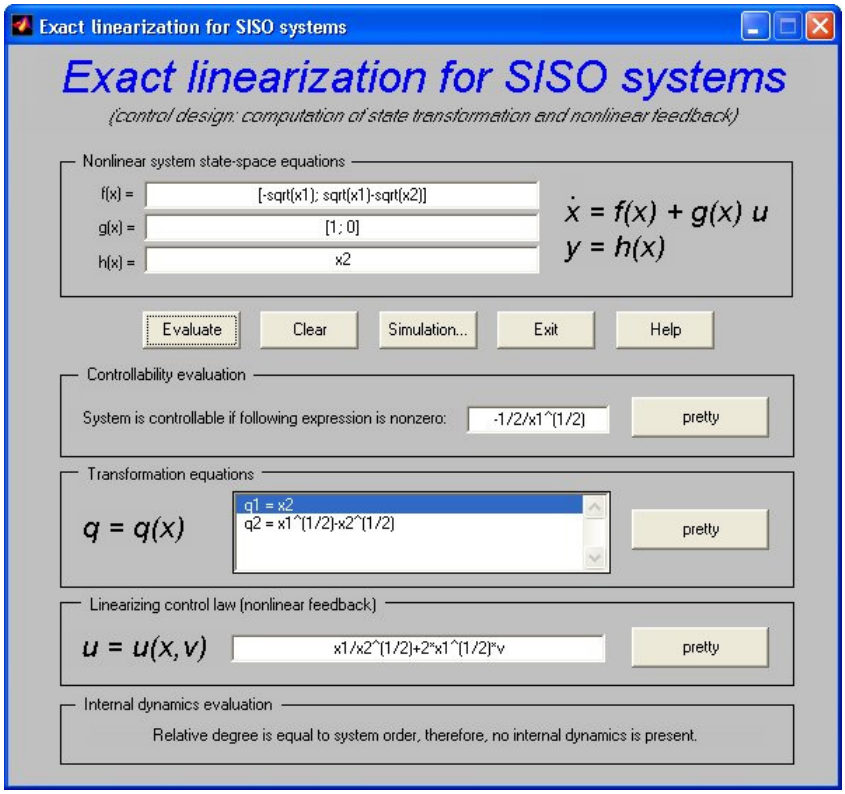

Figure 4: The *NelinSys* toolbox – Exact linearization controller design tool (SISO GUI version)

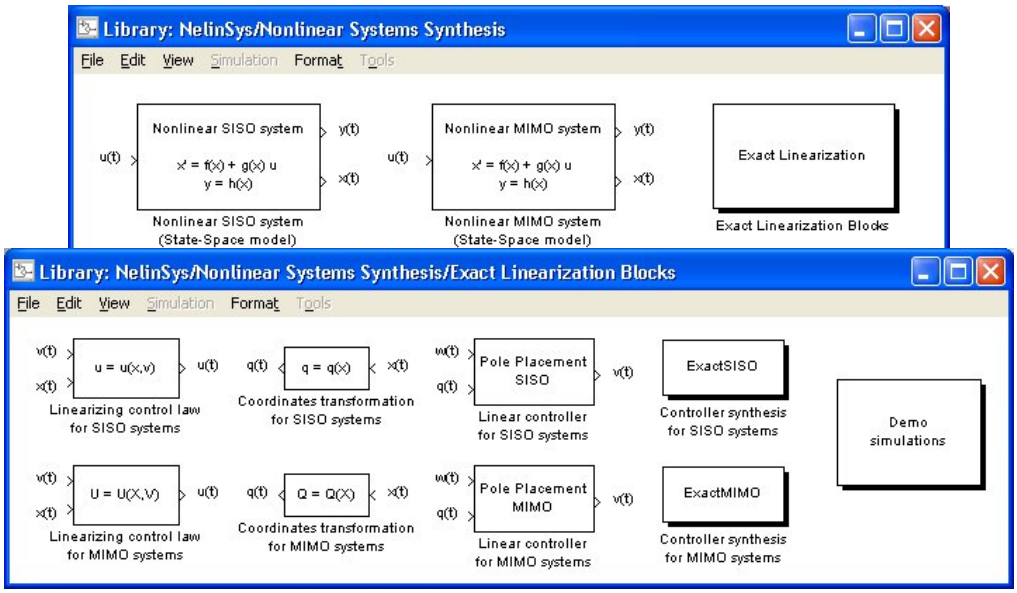

Figure 5: The *NelinSys* toolbox – Exact linearization blockset for Simulink

The *NelinSys* exact linearization module is composed of two submodules, one dedicated to calculation of a linearizing controller (Figure 4) and the other to performance evaluation of thus designed control loop (Figure 5). The use of the tool is very easy – to a user-specified nonlinear system (2) it calculates necessary transformations (4) and (5), puts them as parameters into corresponding Simulink blocks, creates a ready-to-run Simulink simulation model supplemented with

a linear pole placement controller and lets the user judge the performance of the loop. Besides SISO systems, the *NelinSys* toolbox can also handle exact linearization of MIMO systems with the same number of inputs and outputs.

The last of the *NelinSys* modules provides support for gain scheduling. The basic idea of gain scheduled control of a nonlinear system is to design multiple linear controllers (instead of designing a single nonlinear one) based on different operating points and to switch between them according to current operating point. This idea can be further extended by parameterization of the controllers, which practically turns switching into continuous adjustments of controller parameters. A more detailed description of the method together with some consequences and recommendations can be found in [1]. In the following, design of a gain scheduled state-space controller (also called extended linearization) will be considered. The method assumes a nonlinear system given by nonlinear state-space equations

$$\begin{aligned} \dot{x} &= f(x,u) \\ y &= h(x) \end{aligned} \tag{7}$$

If (x_0, u_0) is an equilibrium point of the system, then by a Taylor expansion of the nonlinear functions f and h and by neglecting higher-order terms one can get a linear approximation of the system dynamics

$$\begin{aligned} \Delta\dot{x} &= A_0\Delta x + B_0\Delta u \\ \Delta y &= C_0\Delta x \end{aligned} \quad \text{where } \Delta x = x - x_0,\ \Delta u = u - u_0,\ \Delta y = y - y_0 \tag{8}$$

As this approximation is valid only in a close neighbourhood of the equilibrium point, so is the control design based upon it. And here come the ideas of gain scheduling – let us design different controllers for different equilibrium points and switch between them or, better than that, let us design a controller whose parameters will change according to current operating point continuously. In order to do that, first we have to describe equilibrium points (x_0, u_0) of the system (7) with as few independent parameters as possible. These parameters, so-called scheduling variables, are chosen so that they are equal to one or more measurable system variables, e.g. the desired value w, the system output y or the control action u. The reason for that is obvious – once the controller is implemented, scheduling parameters are replaced with actual measured values. In SISO case, one parameter is usually enough to describe the equilibrium points. If we denote the parameter as α then (8) can be rewritten as

$$\begin{aligned} \frac{d}{dt}\left[x - x_0(\alpha)\right] &= A_0(\alpha)\left[x - x_0(\alpha)\right] + B_0(\alpha)\left[u - u_0(\alpha)\right] \\ y - y_0(\alpha) &= C_0(\alpha)\left[x - x_0(\alpha)\right] \end{aligned} \tag{9}$$

A state-space controller designed to control this system can be described by the equation

$$u = u(w, x) = u_0(\alpha) + k_0(\alpha)\,[w - w_0(\alpha)] - K(\alpha)\,[x - x_0(\alpha)] \tag{10}$$

Feedback part of the control law $K(\alpha)$ can be computed via pole placement from

$$\det\left[sI - \left(A_0(\alpha) - B_0(\alpha)K(\alpha)\right)\right] = \prod_{i=1}^{n}\left(s - p_i\right) \tag{11}$$

and feedforward part as

$$k_0(\alpha) = -\frac{1}{C_0(\alpha)\left[A_0(\alpha) - B_0(\alpha)K(\alpha)\right]^{-1}B_0(\alpha)} \tag{12}$$

This will assure that the closed loop system will have desired poles p_i, $i = 1, 2, ..., n$, for arbitrary α i.e. for any of its equilibrium points.

Similarly as the *NelinSys* exact linearization module, the gain-scheduling tool also incorporates two parts – a controller design part (Figure 6) and a simulation part. While the first one lets a user calculate a gain scheduled controller based on state-space equations of a nonlinear system, desired poles of the closed loop and desired scheduling variable, the other serves as a verification tool and lets him decide if and how the control design satisfies desired performance criteria by means of Simulink simulation. And again, the use of the tool is very simple, leaving no unnecessary calculation to the user.

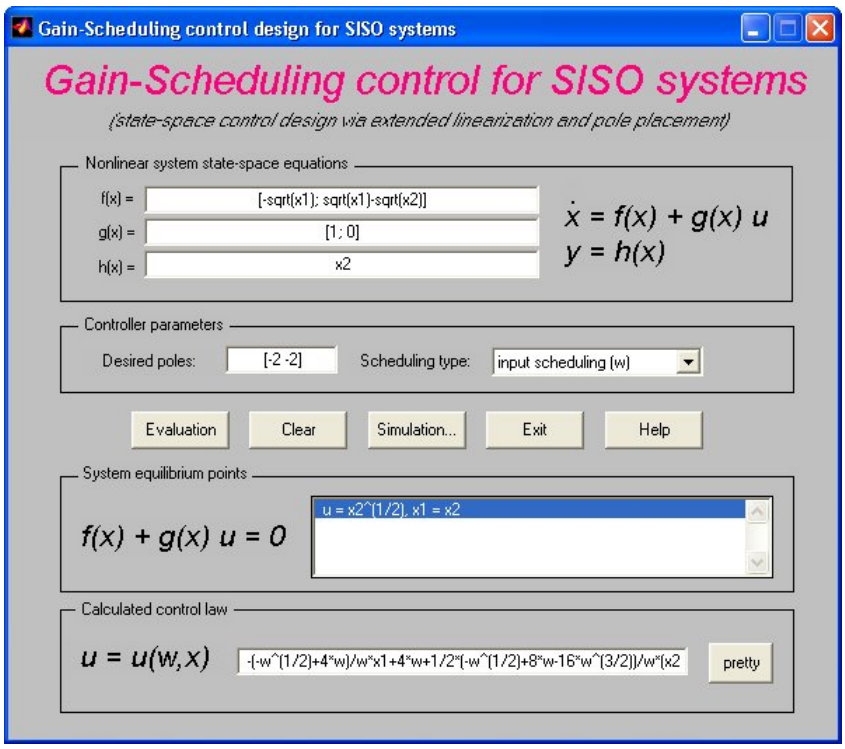

Figure 6: The *NelinSys* toolbox – Gain-scheduling controller design tool (SISO GUI version)

3 Application of the tools

This section contains two examples demonstrating application of the *NelinSys* tools to solution of sample problems of nonlinear analysis and nonlinear control design.

Example 1

Let us consider a nonlinear system with the structure according to Figure 2, where the transfer function of the linear part is

$$F_L(s) = \frac{2}{s^2 + 0{,}5s} \tag{13}$$

and the nonlinearity G_N is a relay with hysteresis with gain $K = 2$ and hysteresis width $B = 0{,}5$. The task is to find whether there exists a limit cycle in the system and if so to determine its parameters.

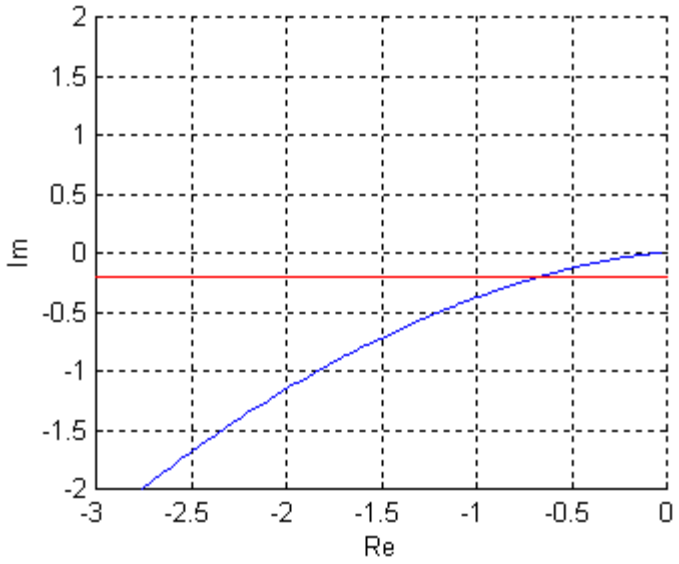

Figure 7: Analysis of limit cycles with a help of Two-Characteristics method

Since the nonlinear system is of required type, we can use the Two-Characteristics-method tool to solve the problem. If we provide the tool with input data, we obtain the plot (Figure 7) from which it is possible to conclude that there is one limit cycle in the system – the two curves have one intersection. Approximate parameters of this limit cycle – its amplitude $a = 1{,}7$ and frequency $\omega = 1{,}7$ – can be determined analytically by a solution of the harmonic balance equation (1). In most cases, they can be calculated with a help of the text-based version of the tool that incorporates Symbolic Math Toolbox as a solver. However, there are also cases in which Symbolic Math Toolbox fails to find an explicit analytical solution of (1) so it has to be calculated manually.

As (13) is a 2nd-order linear system and relay with hysteresis is a static nonlinearity, we can alternatively analyze the same system using *NelinSys* Phase-Plane Analysis tool. This analysis is very easy, because all we need to do is just to build a simulation (Figure 8) from the blocks included in the *NelinSys* toolbox (Figure 1) and standard Simulink *Scope* block, specify parameters of the blocks and run it. During the simulation we can observe how the phase-plane portrait (Figure 9) is being generated and afterwards we can accordingly judge the limit cycles.

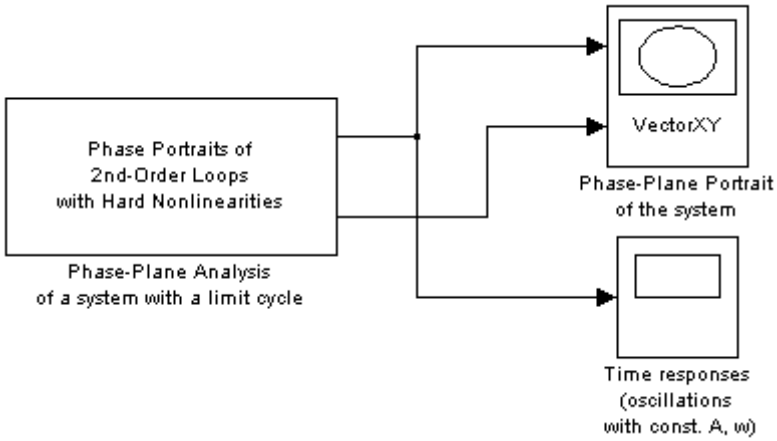

Figure 8: Analysis of limit cycles in phase plane – simulation scheme for Simulink

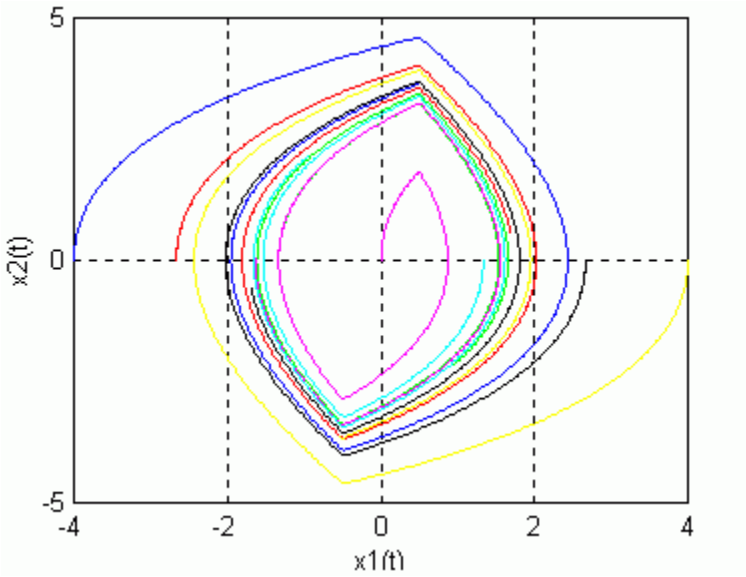

Figure 9: Analysis of limit cycles in phase plane – phase-plane portrait of the system

Utilizing knowledge from phase-plane analysis theory and the simulation result in Figure 9, we come to the same conclusion as before – there is one limit cycle in the system because there is one isolated closed path in the phase-plane plot. Moreover, we can add that the limit cycle is stable because all neighbouring system trajectories converge to the closed path. The quantitative parameters of the limit cycle can be determined by exploring the time-plot in Figure 10 – the amplitude of sustained oscillations is approximately $a = 1{,}716$ and the period is $T = 3{,}8$s, which corresponds to the frequency $\omega = 1{,}653$.

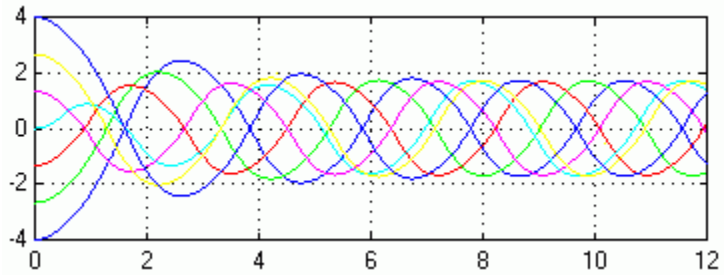

Figure 10: Analysis of limit cycles in phase plane – time response

Example 2

The task is to design a nonlinear controller to a *Two-tanks without interaction* system described by nonlinear state-space equations

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix}=\begin{pmatrix}-\sqrt{x_1}\\ \sqrt{x_1}-\sqrt{x_2}\end{pmatrix}+\begin{pmatrix}1\\ 0\end{pmatrix}u \qquad (14)$$
$$y=x_2$$

Since (14) suits both the system description required by exact linearization (2) and by gain scheduling (7), we can use either of the methods. Let us choose exact linearization first and let us employ *NelinSys* exact linearization module (Note: for better readability of the program output, text-based version of the controller design tool is used in the following example, user input is written in *italics*).

```
Exact linearization for SISO systems
------------------------------------
                          .
State-space equations: x = f(x) + g(x) u
                       y = h(x)

System order: 2

System matrix: f(x) = [-sqrt(x1); sqrt(x1)-sqrt(x2)]
Input matrix: g(x) = [1; 0]
Output matrix: h(x) = x2

Controllability: system is controllable if the following expression is nonzero:

                                    1
                              - 1/2 -----
                                    1/2
                                   x1
Transformation equations:
 q1 = x2
 q2 = x1^(1/2)-x2^(1/2)

Relative degree of the system: 2

Nonlinear feedback: u =
                                  x1          1/2
                                ----- + 2 x1    v
                                  1/2
                                x2
```

This way we have calculated the state-space transformation (4) and the linearizing control law (5) necessary to turn the nonlinear system (14) into the form

$$\begin{pmatrix}\dot{q}_1\\ \dot{q}_2\end{pmatrix}=\begin{pmatrix}0 & 1\\ 0 & 0\end{pmatrix}\begin{pmatrix}q_1\\ q_2\end{pmatrix}+\begin{pmatrix}0\\ 1\end{pmatrix}v$$
$$y=\begin{pmatrix}1 & 0\end{pmatrix}\begin{pmatrix}q_1\\ q_2\end{pmatrix} \qquad (15)$$

i.e. to a linear and controllable form. In order to complete the controller synthesis, we need to design a linear controller for (15). The linear controller can be designed using arbitrary linear technique, however, if pole placement is chosen, special blocks from the *NelinSys* exact linearization blockset (Figure 5) can be utilized and performance-evaluating simulation can be started immediately. Simulink scheme used for the simulation is in Figure 11, simulation result in Figure 12.

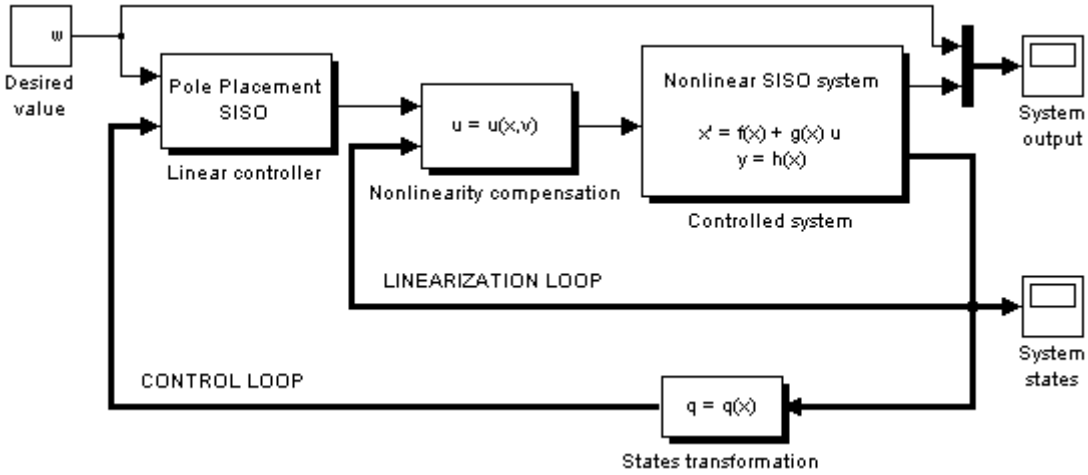

Figure 11: Exact linearization – simulation scheme for Simulink

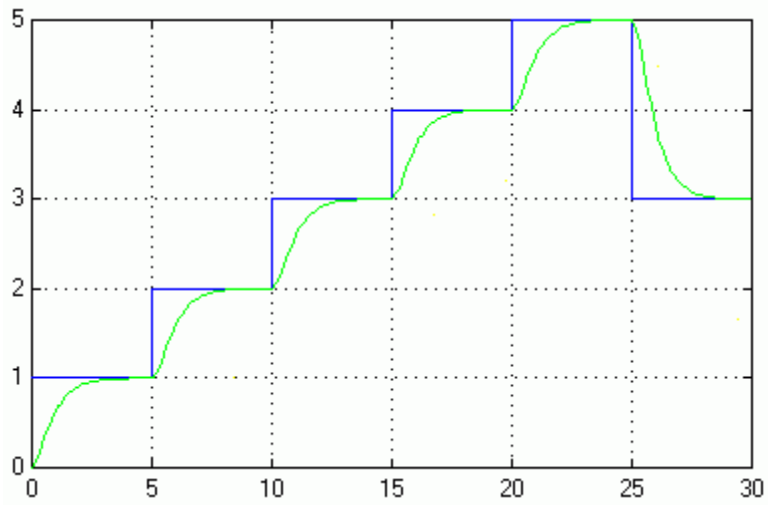

Figure 12: Exact linearization – simulation results

Now let us try gain scheduling and the *NelinSys* gain-scheduling module. Again, in order to improve the readability of the output, the use of the text-based version of the controller design tool will be illustrated instead of the GUI version.

```
Gain Scheduling control for SISO systems
----------------------------------------
                        .
Nonlinear system: (1) x = f(x,u)
                      y = h(x,u)
                        .
                  (2) x = f(x) + g(x) u
                      y = h(x)

Choose one of the options: 2

System order: 2

System matrix function: f(x) = [-sqrt(x1); sqrt(x1)-sqrt(x2)]
Input matrix function:  g(x) = [1; 0]
Output matrix function: h(x) = x2
                        .
System linearization: /|x = A /|x + B /|u
                      /|y = C /|x + D /|u
```

```
A =
                          [        1                 ]
                          [- 1/2 -----         0     ]
                          [        1/2               ]
                          [      x1                  ]
                          [                          ]
                          [        1               1 ]
                          [ 1/2 -----     - 1/2 -----]
                          [        1/2             1/2]
                          [     x1              x2   ]
B =
                                    [1]
                                    [ ]
                                    [0]
C =
                                  [0    1]
D =
                                     0

System equilibrium points: u = x2^(1/2), x1 = x2

Desired poles of the closed-loop system: [-2 -2]

Controller for the system linearization: /|u = f0 /|w - K /|x
f0 =
                                       1/2
                                  8 x2
K =
                    [    1                 1             1/2]
                    [- ----- + 4    1/2 ----- - 4 + 8 x2   ]
                    [    1/2               1/2             ]
                    [  x2                x2               ]

Gain Scheduling type: (1) no scheduling (fixed operating point)
                      (2) input scheduling (w)
                      (3) output scheduling (y)
                      (4) mixed scheduling (m*w+(1-m)*y)
Selected scheduling type: 2

Control action: u =

             1/2                            1/2             3/2
       (-w    + 4 w) x1              (-w    + 8 w - 16 w   ) (x2 - w)
     - ---------------- + 4 w + 1/2 --------------------------------
              w                                  w
```

After we are ready with the calculation of the control law, we can evaluate performance of the closed-loop system by means of a Simulink simulation. The scheme, composed of blocks from the *NelinSys* gain-scheduling blockset, is in Figure 13, simulation results in Figure 14. Although in the example above input scheduling was considered, with a different scheduling variable the control law would be different as well. Therefore, Figure 14 demonstrates two cases – the green line corresponds to input-type whereas the red line to output-type scheduling; the blue line is the desired value *w*.

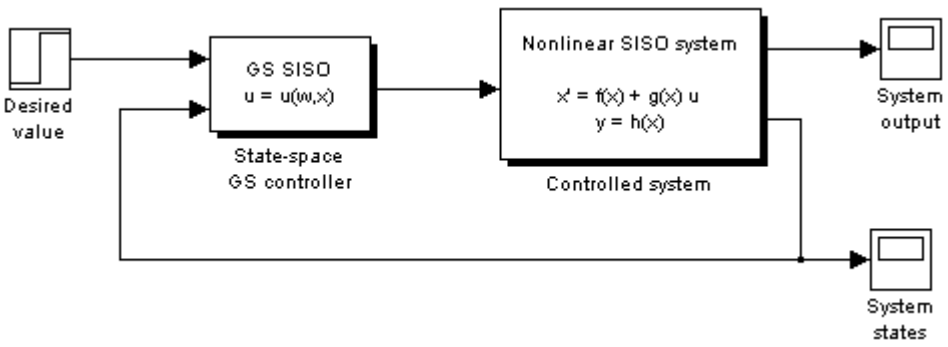

Figure 13: Gain scheduling – simulation scheme for Simulink

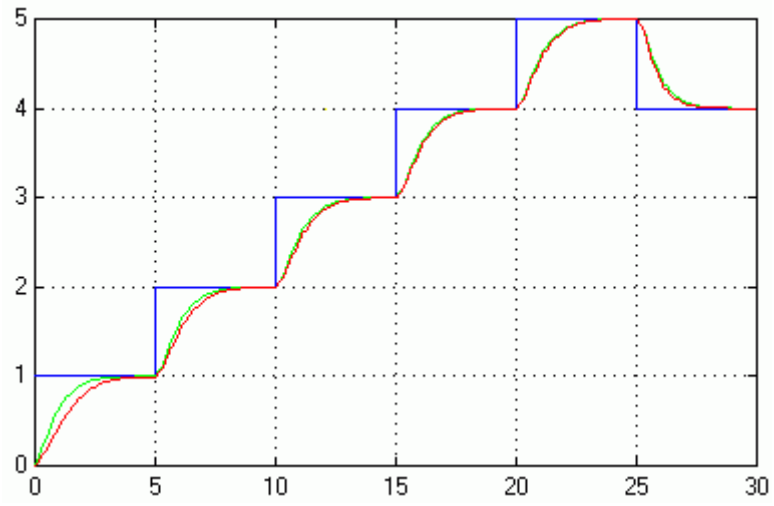

Figure 14: Gain scheduling – simulation results

4 Implementation details

The *NelinSys* toolbox was designed with the intention to be a user-friendly and intuitive "plug-in" for MATLAB – or rather for Simulink (in fact, the toolbox is encapsulated as a Simulink library, although it contains links to pure-MATLAB applications). Therefore, from users' point of view it behaves just like any other Simulink blockset and anybody who wants to use it needs no further knowledge than basic MATLAB/Simulink navigation and, of course, at least a basic level of nonlinear control theory. However, *NelinSys* can be also interesting from developers' point of view, because its MATLAB implementation involves a few unusual features; some of them will be discussed in this section.

Probably the most important feature is the collaboration of Simulink and Symbolic Math Toolbox. The need for this collaboration results from the fact that all *NelinSys*' modules (perhaps except for the Two-Characteristics Method tool) require both symbolic and numerical calculations. Exact linearization for instance – in the control design process, large amount of symbolic computation is used (e.g. calculation of analytical partial derivations of functions, algebraic operations with functional matrices, etc.) whereas consequential simulation of the closed-loop system is a numerical task. Thanks to Simulink + Symbolic Math Toolbox collaboration it is possible to use the control law in simulation in symbolic form. That means a lot because without the collaboration, after calculation of the control law a user would need to create a simulation scheme manually by converting the control law from the symbolic form to a chain of standard Simulink blocks (of course, this would also mean the necessity to modify the scheme each time the system description of the original nonlinear system is changed).

So far, Simulink is completely numerical i.e. there is not a single block able to work with symbolic variables. Nevertheless, since it contains user-definable blocks such as *MATLAB Fcn* and *S-function*, symbolic capabilities can be incorporated into Simulink by a developer. This was done as a part of implementation of *NelinSys*. Besides *S-function* blocks, there were two other services utilized in making Simulink understand symbolic data – the *Subsystem* block from standard *Simulink / / Connections* library and the *Mask Subsystem* item from Simulink's *Edit* menu. S-functions allow a developer to create custom Simulink blocks whose behaviour is determined by a program code consisting of arbitrary MATLAB commands (i.e. including commands from Symbolic Math Toolbox). *Mask Subsystem*, on the other hand, helps to create a user-friendly interface between S-function code and simulation scheme, processes user-specified input parameters and transforms them into the form required by the S-function. The transformation is carried out via block initialisation commands that can again take advantage of Symbolic Math Toolbox. Although symbolic expressions cannot be used as block parameters, string expressions can, which means that with a help of sym-string conversion commands *char* and *sym* symbolic values can be specified to the blocks.

The principle described in previous paragraph was more or less used in programming of almost every Simulink block contained in the *NelinSys* toolbox. In the following, the *Nonlinear SISO system*

(State-Space model) block's implementation is analyzed as a representative example. Internal structure of the block is depicted in Figure 15 and its parameters setup in Figure 16.

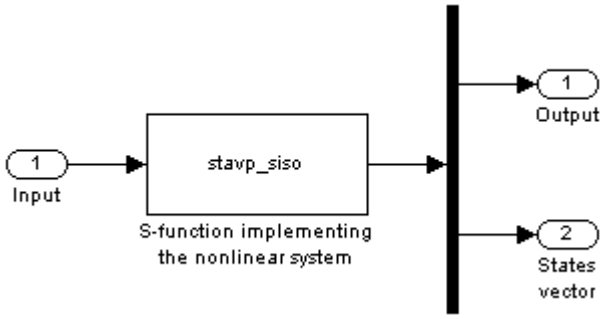

Figure 15: Internal structure of the *Nonlinear SISO system (State-Space model)* block

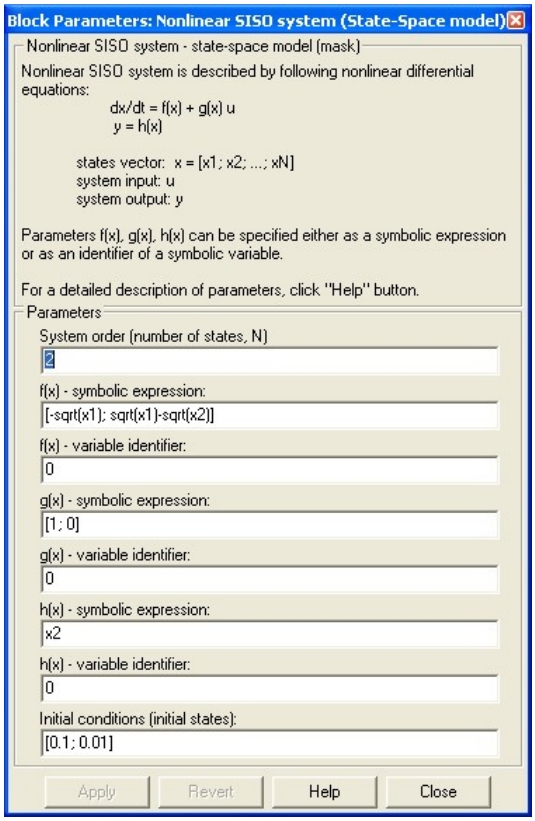

Figure 16: Parameters set-up screen of the *Nonlinear SISO system (State-Space model)* block

User has to specify three things – system order, system matrix-functions $f(x)$, $g(x)$, $h(x)$ and initial conditions. While there are no problems with the first and the third one (they are numerical entries), the matrix functions have to be treated in a different way – the block's mask treats them as strings, block initialization commands transform them and pass them to the S-function, which finally works with them as with symbolic objects. As can be seen in Figure 16, each of the matrix functions can be specified either as a symbolic expression or as an identifier of a symbolic variable. The latter allows the block to use variables from MATLAB workspace – typically those that were put there by a controller design tool (Figure 4, Figure 6) – so that a simulation can be started immediately after the control law is calculated, while the former is usually used when a stand-alone simulation is needed. It is also possible to combine the two options, e.g. $f(x)$, $g(x)$ can be specified as a symbolic expression and $h(x)$ as an identifier, however, this is hardly ever necessary.

The initialization commands of the block (see the code listed below) first check which of the two representations for $f(x)$, $g(x)$ and $h(x)$ (symbolic expression or identifier) is to be used, search the expressions for unknown symbols (the execution is stopped if there are any), check if matrix dimensions are appropriate according to system order and, finally, adjust the parameters to the form convenient for the S-function.

```
% Overenie spravnosti zadania parametrov bloku %
if ~isempty(F) & (Fp == 0)
   F = sym(F);
elseif isempty(F) & (any(Fp ~= 0))
   F = sym(Fp);
else
   error(['Matrix f(x) unspecified or specified more than once - cannot
continue!']);
end

if ~isempty(G) & (Gp == 0)
   G = sym(G);
elseif isempty(G) & (any(Gp ~= 0))
   G = sym(Gp);
else
   error(['Matrix g(x) unspecified or specified more than once - cannot
continue!']);
end

if ~isempty(H) & (Hp == 0)
   H = sym(H);
elseif isempty(H) & (any(Hp ~= 0))
   H = sym(Hp);
else
   error(['Matrix h(x) unspecified or specified more than once - cannot
continue!']);
end

% Zisti si symbolicke premenne, ktore vystupuju vo vyrazoch %
premF = strrep(strrep(findsym(sym(F)),', ',''),',',' ');
premG = strrep(strrep(findsym(sym(G)),', ',''),',',' ');
premH = strrep(strrep(findsym(sym(H)),', ',''),',',' ');

% Premenne x1, x2, ..., xN su korektne, preskoc ich %
for k = 1 : n
   premF = strrep(premF, sprintf('x%d',k), '');
   premG = strrep(premG, sprintf('x%d',k), '');
   premH = strrep(premH, sprintf('x%d',k), '');
end

% Ak niektory retazec nezostal prazdny, vyhlas chybu %
if ~isempty(premF)
   error(['Unknown symbol in f(x) expression - cannot continue!']);
end
if ~isempty(premG)
   error(['Unknown symbol in g(x) expression - cannot continue!']);
end
if ~isempty(premH)
   error(['Unknown symbol in h(x) expression - cannot continue!']);
end

% Otestovanie spravnosti rozmerov matic F,G,H (podla "n") %
if ~prod(size(F) == [n,1])
   error(['Invalid matrix dimensions: f(x) - cannot continue!']);
end
if ~prod(size(G) == [n,1])
   error(['Invalid matrix dimensions: g(x) - cannot continue!']);
end
if ~prod(size(H) == [1,1])
   error(['Invalid matrix dimensions: h(x) - cannot continue!']);
end

% Kontrola vektora pociatocnych podmienok (pociatocnych stavov) %
if ~prod(size(pp) == [n,1])
   error(['Invalid dimensions: initial conditions vector - cannot
continue!']);
end

% Nahradenie identifikátorov "xI" identifikátormi "x(I)" %
for k = 1 : n
   F = subs(F, sprintf('x%d',k), sprintf('x(%d)',k));
```

```
   G = subs(G, sprintf('x%d',k), sprintf('x(%d)',k));
   H = subs(H, sprintf('x%d',k), sprintf('x(%d)',k));
end
```

The code of the *stavp_siso* S-function (see below) implements the behaviour of the nonlinear SISO system specified by the block parameters. There are *n* continuous states (where *n* is the system order), one input and *n*+1 outputs – besides the output *y* of the nonlinear system itself, the block also outputs the whole state-space vector *x*. In every time instant a calculation according to system state-space equations is performed. As the equations are nonlinear and, therefore, their result might not remain in real domain for some values of system states and/or inputs, it is also checked whether the result is real (if not, an error message is generated).

```
function [sys,x0,str,ts] = stavp_siso(t,x,u,flag,n,F,G,H,pp)

switch flag,

% ================= Inicializacna cast s-funkcie  ================= %
case 0

   % Definicia stavov, vstupov a vystupov nelinearneho systemu %
   sizes = simsizes;
   sizes.NumContStates = n;
   sizes.NumDiscStates = 0;
   sizes.NumOutputs = 1+n;
   sizes.NumInputs = 1;
   sizes.DirFeedthrough = 0;
   sizes.NumSampleTimes = 1;
   sys = simsizes(sizes);

   % Inicializacia pociatocnych podmienok %
   x0 = pp;

   % Inicializacia "str" ako prazdna matica %
   str = [];

   % Sample times in TS %
   ts = [0 0];

% ================= Vypocet podla 1. stavovej rovnice ================= %
case 1
   sys = eval(F) + eval(G) * u;

   for k = 1 : n
      if ~isreal(sys(k,:))
         error(['At the time instant t = ',num2str(t),'s, the mathematical
                model of the system is no longer valid! Cannot continue the
                simulation!']);
      end
   end

% ================= Vypocet podla 2. stavovej rovnice ================= %
case 3
   sys(1,:) = eval(H); % Ako vystup bloku sa berie nielen vystup systemu...  %
   sys(2:n+1,:) = x;   % ... ale aj cely stavovy vektor nelinearneho systemu %

% ======================== Nepouzite priznaky ======================== %
case {2, 4, 9}
   sys = [];

% ======================== Spracovanie chyby ======================== %
otherwise
   error(['Unknown flag = ',num2str(flag)]);
end
```

Besides the need for collaboration of Simulink and Symbolic Math Toolbox, which was the biggest challenge resulting from creation of the *NelinSys* toolbox, there were also other minor problems to cope with during programming. One of them was the necessity to use different sets of initial conditions for calculation of different phase-plane trajectories (Phase-Plane Analysis tool).

However, according to S-function template, for an n^{th}-order system it is possible to define only n initial conditions i.e. only one vector of the length n, which would mean that during one simulation it is possible to calculate (and to plot) only one phase-plane trajectory. This would be a great disadvantage because it would be almost impossible to read qualitative features of a system from only one trajectory. Of course, there would be a possibility to draw several trajectories one after another, but that way we would lose the synchronization i.e. the advantage of seeing how the phase-plane portrait is being gradually generated (after all, we would have only a static picture). Therefore, for *NelinSys* we used another approach: pretending a higher-order system to an S-function. For example, if we need to have a phase-plane portrait of a 2^{nd}-order system containing 5 trajectories, instead of 5 different pairs of initial conditions we will specify to the S-function a 10^{th}-order system with only one initial condition vector composed of 10 elements (obviously, the 10^{th}-order system will consist of equations of the original 2^{nd}-order system repeated in pairs). The approach can be illustrated by a source code of any of the phase-plane analysis blocks' S-functions (for the sake of brevity the code is omitted here, but it can be located in the *NelinSys* folder; see e.g. the *fazrov_auton2.m* file).

5 Conclusion

In this paper *NelinSys* – a custom toolbox for nonlinear control systems – was introduced. It was shown how to use it in order to solve certain tasks from nonlinear control theory and how some of its modules were implemented in MATLAB/Simulink. Probably the most interesting side effect of the programming of the toolbox was the creation of Symbolic Math Toolbox – Simulink collaboration concept. It has turned out to be very useful in this case and it is possible that it could be also utilized in other applications where both symbolic calculations and numerical simulations are necessary.

Although even now it provides solid support for nonlinear control, the *NelinSys* toolbox is being continually improved. Its first version (with Slovak language interface) was presented in [2] in 2004; first English version was introduced in February 2005 and since June 2005 it is available via MATLAB Central – File Exchange. Besides, there were several bug fixes and minor improvements in the meantime. A new version of the tool, bringing support for other nonlinear methods (e.g. velocity linearization [3]), is planned to be released soon.

References

[1] M. Huba. *Nonlinear Systems [in Slovak]*. Vydavateľstvo STU, Bratislava, 2003.

[2] M. Ondera. *Program Tools for Analysis and Synthesis of Nonlinear Processes [in Slovak]*. Master thesis. Supervisor: Assoc. Prof. A. Jadlovská. KKUI FEI TU Košice, 2004.

[3] M. Ondera. *MATLAB-Based Tool for Velocity Linearization.* In: *CEEPUS Summer School „Intelligent Control Systems"*, pp. 55-60, Brno University of Technology, Brno, 2005.

[4] J. J. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, New Jersey, 1991.

Martin Ondera
Katedra automatizácie a regulácie FEI STU Bratislava
Ilkovičova 3, 812 19 Bratislava, Slovak Republic
Tel.: +421 2 60291458
E-mail: Martin.Ondera@stuba.sk