Matlab–Based Toolbox for Analysing Differential Linear Repetitive Processes

Artur Gramacki, Jaroslaw Gramacki, Krzysztof Galkowski, Eric Rogers, David H. Owens

Abstract— **A short overview of a Matlab–based** *toolbox* **developed to support the control-related analysis of a subclass of** *nD* **systems called differential linear repetitive processes (LRP) is the subject of this paper. Its main functionality covers two different areas. First the** *toolbox* **allows one to build discrete approximations of continuous–time LRPs and then perform analysis/simulation verification studies. All tasks use a graphical environment, with typical** *Windows* **components. In the second area continuous–time linear repetitive processes can be numerically solved using numerical methods for solving ordinary differential equations. This required the development of some Matlab–based functions (so called** *M–files***) which extensively use the Matlab** *ODE Suite* **mechanizm. The two mentioned areas are fully integrated together. In this paper we also document some details of the data format specifications used in the** *toolbox* **and on the adoption of original Matlab** *ODE solvers* **for solving LRP's. Moreover we show many (not presented so far in print) implementation details concerning ODE solvers used in the context of repetitive processes.**

Keywords— **Linear repetitive processes, Matlab toolbox, Matlab** *ODE Suite***, Ordinary differential equations**

I. Introduction

THE essential unique characteristic of a repetitive, or multi–pass, process (LRP) is a series of sweeps or passes through a set of dynamics defined over a fixed and finite duration termed the pass profile. On each pass, an output, termed the pass profile, is produced which acts as a forcing function on, and hence contributes to the dynamics of the next pass profile.

The explicit interaction between successive pass profiles is the source of the novel control (and numerical) problem for these processes in that the output sequence of pass profiles can contain oscillations that increase in amplitude in the pass to pass direction. Details can be found in [1].

The 2D systems structure of a repetitive process arises from information propagation in (i) the pass to pass direction, and (ii) along a given pass. By definition, the pass length is of finite duration and does not change value from pass to pass. In a so–called differential linear repetitive process, the dynamics along the pass evolve as a continuous function of the (temporal or spatial) independent variable and the evolution from pass to pass is, in effect, discrete. So–called discrete linear repetitive processes differ from differential processes only in the fact that the evolution of the dynamics along a pass is also discrete. Differential and discrete linear repetitive processes are (arguably) the most important sub–class of repetitive processes from both the theoretical and applications standpoints and are the subject of this paper. The notation for variables in this paper is of the form $y_k(t)$, $0 < t < \alpha$, where y is the (possibly vector valued) variable under consideration, k is the pass index or number, and α is the finite pass length. Figure 1 gives schematic illustration of the evolution of the dynamics of a repetitive process. The simplest possible case

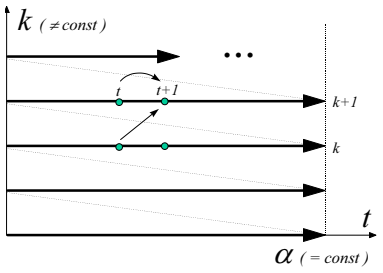

Fig. 1. Schematic illustration of the dynamics of a repetitive process.

is where only the previous pass profile contributes to the current one and such processes are termed unit memory. For an extension to so–called non–unit memory LRP's see [1].

Clear links exist between differential repetitive processes and so–called discrete–continuous 2D systems which have recently been the subject of detailed investigations, see, for example, [2]. A key difference, however, arises from the fact that the pass length of a repetitive process is always finite (α on Figure 1) and this is the basic reason why most results for the latter area either do not transfer at all or only after substantial modifications to the repetitive process setting.

II. Background

The state space model of a linear differential repetitive process with constant pass length has the following, commonly known, form [1]

$$\begin{aligned} \dot{x}_{k+1}(t) &= \hat{A}x_{k+1}(t) + \hat{B}u_{k+1}(t) + \hat{B}_0 y_k(t) \\ y_{k+1}(t) &= \hat{C}x_{k+1}(t) + \hat{D}u_{k+1}(t) + \hat{D}_0 y_k(t). \end{aligned} \quad (1)$$

Here, on pass k, $x_k(t)$ is the $n \times 1$ state vector, $y_k(t)$ is the $m \times 1$ pass profile vector, $u_k(t)$ denotes the $r \times 1$ vec-

Artur Gramacki and Jaroslaw Gramacki are with the Department of Electronics and Computer Science,University of Zielona Gora, Poland, A.Gramacki[J.gramacki]@iie.uz.zgora.pl

Krzysztof Galkowski is with the Department of Control and Computation Engineering, University of Zielona Gora, Poland, K.Galkowski@issi.uz.zgora.pl

Eric Rogers is with Department of Electronics and Computer Science, University of Southampton, UK etar@ecs.soton.ac.uk

David H. Owens is with Department of Automatic Control and Systems Engineering, University of Sheffield, UK

tor of control inputs and $\hat{A}, \hat{B}, \hat{B}_0, \hat{C}, \hat{D}, \hat{D}_0$ are matrices of appropriate dimensions.

To complete the process description, it is necessary to specify the state and pass initial conditions, i.e. the initial state vector on each pass $x_{k+1}(0)$, $k \geq 0$ and the initial pass profile (i.e. on pass number 0) $y_0(t)$, $0 \leq t \leq \alpha$. The simplest possible choice is

$$\begin{aligned} x_{k+1}(0) &= d_{k+1},\ k \geq 0 \\ y_0(t) &= f(t),\ 0 \leq t \leq \alpha \end{aligned} \tag{2}$$

where d_{k+1} is an $n \times 1$ vector with constant entries and $f(t)$ is an $m \times 1$ vector whose entries are known functions of t.

Clearly the first requirement of a systems theory for these processes is a stability theory and associated computationally feasible tests. Such a theory already exists and details can, for example, be found in [3], [4], [5], [1], [6], [7].

Two different approaches can be used in order to evaluate equations (1). The first one of these is to discretize the continuous–time model to obtain its discrete–time equivalent in the form

$$\begin{aligned} x_{k+1}(p+1) &= Ax_{k+1}(p) + Bu_{k+1}(p) + B_0 y_k(p) \\ y_{k+1}(p) &= Cx_{k+1}(p) + Du_{k+1}(p) + D_0 y_k(p) \end{aligned} \tag{3}$$

with boundary conditions

$$\begin{aligned} x_{k+1}(0) &= d_{k+1},\ k = 0, 1, \ldots \\ y_0(p) &= f(p),\ p = 0, 1, \ldots, \alpha - 1 \end{aligned} \tag{4}$$

where $p = \{0, 1, \cdots, \alpha - 1\}$ and the matrices A, B, B_0, C, D, D_0 are computed from those of (1) by formulas determined by the particular numerical approximation (i.e. discretization) method used. The approximate solution generated from (3) and (4), should be as close as possible (in a well defined sense) to the exact solution obtained from (1) and (2) (assuming that it is known or may be calculated with negligible errors. A process described by these last two equations is termed a discrete linear repetitive process and for further details on the numerical approximation of linear repetitive processes see for example [8], [9], [10], [11], [12]. (These references also make clear why a discrete linear repetitive process is the most natural approximation of a differential process.)

The second approach uses the fact that the first equation in (1) is a commonly encountered ordinary differential equation. A novel feature here is that the last factor $y_k(t)$ in (1) can be interpreted (in computational terms) as an additional input sequence (similar as the second factor u_{k+1}(t) in (1)). Moreover this term has boundary value condition and is specified for the first pass as $y_0(t)$. Note also that the presence of the second variable k in (1) means that in numerical terms the problem here reduces to solving a 1D differential equation iteratively k times.

The well known MATLAB package includes a very powerful mechanism, termed the *Matlab ODE Suite*, which may be easily used in this task. In particular, after some programming work it can be successfully adopted for solving differential repetitive processes. More details on the *Matlab ODE Suite* can, for example, be found in [13], [14].

The two approaches mentioned above are fully supported by a MATLAB–based *toolbox* specially designed and implemented as a computer tool for the analysis of linear repetitive processes and some of its features are detailed in section III.

III. The Matlab Toolbox

A. Toolbox Overview

The core of the MATLAB *toolbox* provides the following classes of functions:

- Generally Applicable Functions (written as typical MATLAB *M–files*) – essentially all the necessary inputs (e.g. control inputs, initial conditions, initial pass profile, the matrices defining a state space model, discretization period, etc.) are prepared manually by the user using a specially developed collection of functions. These functions are, of course, very similar to those found in standard MATLAB toolboxes, such as *Control Toolbox* but they differ in some obvious details which are dictated by the structure of LRP. In the *toolbox* there are also functions for numerical solving the equations of 1 which are, from pure mathematical point of view, ordinary differential equations, and as such can be solved by commonly known numerical methods. Next, these solutions can be treated as reference ones to, for example, verify a discretized model of (3) and hence of an approximate solution generated by (3) and (4).
- A User Friendly Graphical Interface – this has been designed to run from within MATLAB . During operation, it is possible, for example, to modify parameters of the model being simulated to view 2D and 3D plots of, say, the resulting sequence of pass profiles etc.

Additional information on the *toolbox* can be found in [15].

B. Data Format Specification

In its current form, the *toolbox* can, amongst other tasks, simulate and display the response of differential and discrete linear repetitive processes and construct, using a user specified numerical integration technique, a discrete approximation to the dynamics of a differential process. Here we describe the data structures used and related tasks necessary to simulate a discrete model defined by (3) and (4). The basic user supplied data required is as follows:

- the matrices which define the LRP model,
- the pass length α,
- the number of passes, say K, over which the simulation is to run,
- the sequence of input vectors $u_k(p),\ \ k = \{0, 1, \ldots K\},\ \ 0 \leq p \leq \alpha$,
- the initial state vector sequence $x_k(0),\ \ k = \{0, 1, \ldots K\}$,
- the initial pass profile $y_0(p),\ \ 0 \leq p \leq \alpha$,
- the sampling period T.

Note: According to the convention adopted in the development stage, the first pass is numbered 0 (zero).

Assuming this data has been supplied, the *toolbox* calculates:

- the state vector at each instant along each pass $x_k(p),\ \ k = \{0, 1, \dots K\},\ \ 0 \le p \le \alpha$,
- the pass profile at each instant along each pass $y_k(p),\ \ k = \{0, 1, \dots K\},\ \ 0 \le p \le \alpha$.

Given T, the number of points P at which computations are performed along any pass is $P = (\alpha/T) + 1$ subject to the requirement that the remainder on evaluating α/T is zero (since P must also be an integer).

Consider now the storage of the sequence of control vectors u for each pass. A natural approach would be to store values of control sequence for a given pass in an array of r rows (number of inputs) and P columns (number of points). Hence there are K passes and one should simply add a third dimension to the array. Unfortunately, when the first release of the *toolbox* appeared, MATLAB (version 4.2) did not supported multidimensional (that is for $n \ge 3$) arrays. Hence the control sequences for each pass are stored in one (potentially 'large') two–dimensional array where each pass occupies P respective columns. The same method is used for the state initial state sequence x_0, the initial pass profile y_0, the computed sequence of state vectors x, and the computed sequence of pass profiles y. Figure 2 gives a schematic illustration of the format of these matrices.

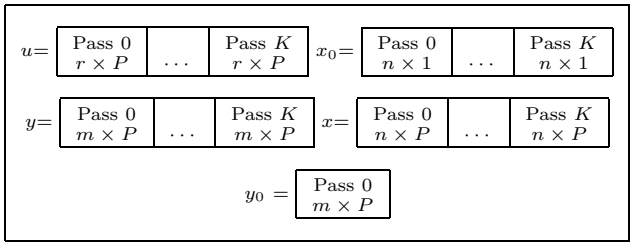

Fig. 2. Format details of input and output vectors for linear repetitive processes (vectors u, x_0, y_0, x, y). r – number of inputs, n – number of states, m – number of pass profiles (outputs), P – number of points on a given pass, K – number of passes.

To illustrate the computations, consider the discrete–time process (3) defined by the following matrices

$$
\begin{aligned}
A &= \begin{bmatrix} 1 & -2 & -1 \\ 3 & -5 & 1 \\ 1 & -2 & 0 \end{bmatrix} B = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 3 & -1 \end{bmatrix} \\
B_0 &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & -1 \end{bmatrix} C = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\
D &= \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ -1 & -2 \\ -1 & -2 \end{bmatrix} D_0 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix}
\end{aligned} \tag{5}
$$

here $r = 2$, $n = 3$, $m = 4$. Suppose also that $\alpha = 2$, $T = 1$ and hence $P = (\alpha/T) + 1 = 3$ and

$$
\begin{aligned}
x_0 &= \left[\begin{array}{c|c|c} NaN & 1 & 2 \\ NaN & 1 & 2 \\ NaN & 1 & 2 \end{array}\right] \begin{array}{l} \leftarrow x_0^1 \\ \leftarrow x_0^2 \\ \leftarrow x_0^3 \end{array} \\
y_0 &= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{array}{l} \leftarrow y_0^1 \\ \leftarrow y_0^2 \\ \leftarrow y_0^3 \\ \leftarrow y_0^4 \end{array} \\
u &= \left[\begin{array}{ccc|ccc|ccc} NaN & NaN & NaN & 1 & 1 & 1 & 2 & 2 & 2 \\ NaN & NaN & NaN & 1 & 1 & 1 & 2 & 2 & 2 \end{array}\right] \begin{array}{l} \leftarrow u^1 \\ \leftarrow u^2 \end{array}
\end{aligned} \tag{6}
$$

where the superscripts are used to denote the entries in the corresponding vector. Here we have 3 states, 4 pass profiles and 2 inputs. Then the resulting state and pass profile vectors for the case of $K = 3$ are as follows

$$
\begin{aligned}
x &= \left[\begin{array}{ccc|ccc|ccc} NaN & NaN & NaN & 1 & 7 & 16 & 2 & 7 & 19 \\ NaN & NaN & NaN & 1 & 2 & 10 & 2 & 5 & 4 \\ NaN & NaN & NaN & 1 & -4 & 0 & 2 & -7 & 12 \end{array}\right] \begin{array}{l} \leftarrow x^1 \\ \leftarrow x^2 \\ \leftarrow x^3 \end{array} \\
y &= \left[\begin{array}{ccc|ccc|ccc} 1 & 1 & 1 & 12 & 14 & 35 & 17 & 20 & 68 \\ 1 & 1 & 1 & 1 & 1 & 1 & -1 & -5 & -47 \\ 1 & 1 & 1 & 2 & 4 & 25 & 7 & 10 & 58 \\ 1 & 1 & 1 & -9 & -9 & -9 & -11 & -15 & -57 \end{array}\right] \begin{array}{l} \leftarrow y^1 \\ \leftarrow y^2 \\ \leftarrow y^3 \\ \leftarrow y^4 \end{array}
\end{aligned} \tag{7}
$$

where NaN (*not a number*) denotes entries in the relevant matrices which are only necessary for computational information purposes (the control sequence u and initial state x_0 are not defined for pass number 0 for (1) and (2)).

C. Solving Linear Repetitive Processes with Matlab ODE Solvers

The MATLAB *ODE solver* is a collection of MATLAB–based functions for the solution of initial value problems of the commonly encountered form

$$y' = F(t, y) \tag{8}$$

over the time interval $[t_0, t_f]$ with given initial values $y(t_0) = y_0$. To solve (8) numerically, it is obviously necessary to use the most appropriate numerical method for the particular data encountered. A very extensive description of such solvers can be found in, for example, [13] or the original MATLAB documentation [14].

To illustrate the MATLAB *ODE solvers* in the repetitive process setting we now detail how to solve numerically a very simple example. This consists of 2 steps, the first of which is to write a function *M–file* that models the desired initial value problem (we call this function *ode file*). In the second step we choose one of the MATLAB's solvers (the one very commonly used is *ode45*) and pass our previously written function *M–file* as an input parameter *ode file*.

Now consider the special case of (1)

$$
\begin{aligned}
\hat{A} &= \begin{bmatrix} -0.5 & -0.7 \\ 0.7 & -0.3 \end{bmatrix} \hat{B} = \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} \hat{B}_0 = \begin{bmatrix} 1.9 \\ 6.4 \end{bmatrix} \\
\hat{C} &= \begin{bmatrix} 0 & 0 \end{bmatrix} \hat{D} = \begin{bmatrix} 0 & 0 \end{bmatrix} \hat{D}_0 = \begin{bmatrix} -0.7 \end{bmatrix}.
\end{aligned} \tag{9}
$$

and for simplicity lets calculate only the first pass. Then we can simple "remove" the variable k from (1) an obtain

the 1D differential linear systems state space model

$$\begin{aligned} \dot{x}(t) &= \hat{A}x(t) + \hat{B}u(t) + \hat{B}_0 y(t) \\ y(t) &= \hat{C}x(t) + \hat{D}u(t) + \hat{D}_0 y(t). \end{aligned} \tag{10}$$

Now let the desired time span be [0 6], with initial conditions: $x_1(0) = 1$, $x_2(0)$, initial pass profile: $y(t) = \sin(t)$, control sequence: $u_1(t) = 0$, $u_2(t) = 1$ for $0 \leq t \leq 6$. The *M–file* code below details our *ode file* for solving (10):

```
function dxdt = lrpodefile(t,x);
================================
  % The model - the first eq. in (1)
  % x' = ax + bu + b0y
  a=[-0.5  -0.7; 0.7  -0.3];
  b=[1 -1; 0 2];
  b0=[1.9; 6.4];
  u=[0; 1];
  y0=sin(t);
  dxdt = a*x + b*u + b0*y0;
```

and then from MATLAB prompt we write:

```
tode=[0 6];
x0ode=[1;0];
[tout,xout]= ode45('lrpodefile',tode,x0ode);
plot(tout(:,1), xout(:,1),'-o');
hold on, grid on
plot(tout(:,1), xout(:,2),'-*');
xLabel('Time (sec.)');
yLabel('State values');
```

Here `tout` is a column vector containing the time points where the solution was computed, and `xout` is a matrix having two columns and `length(tout)` rows. The `tout` vector is usually not equally spaced because *ode45* method implements a variable step–size variant of original Runge–Kutta method of orders 4 and 5 with error control on each step.

Finally, the `plot` function plots the computed time response for both two states (Figure 3). The small circles and stars on the figure are the time points chosen by the *ODE solver*. Note the 'condensation' of the points at the beginning of time interval. In numerical methods we use for solving differential equations, the algorithm employed is usually a variable–step one where the local discretization period h is adjusted to give the desired accuracy. (In the example given here it usually alternates very significantly on over the interval [0 6].)

For illustrative purposes we also show a *3D* plot (Figure 4) of the second state now treated as a regular repetitive process which consists of $K = 8$ passes (plus pass number 0 which is the initial pass profile $y_0(k)$). Note that horizontal axis on Figure 3 shows continuous time (in seconds), while on Figure 4 the axis labeled "points on pass" is scaled in discrete time points ($\alpha = 3$, $T = 0.2 \rightarrow P = (\alpha/T)+1 = 31$ – from point number 0 to point number 30). Nevertheless, Figures 3 and 4 are fully comparable. (A non–zero value of matrix $\hat{D}_0$ for numerical convenience only.)

As noted in Section II, (1) will be solved k times (for each pass) as a consequence of its 2–dimensionality. Here we have two independent variables t and k, where the first one t is continuous–time while the second one k is discrete in nature and receives integer values. During an iterative procedure for solving equation (1), where the former affects the latter, some numerical stability problems may be

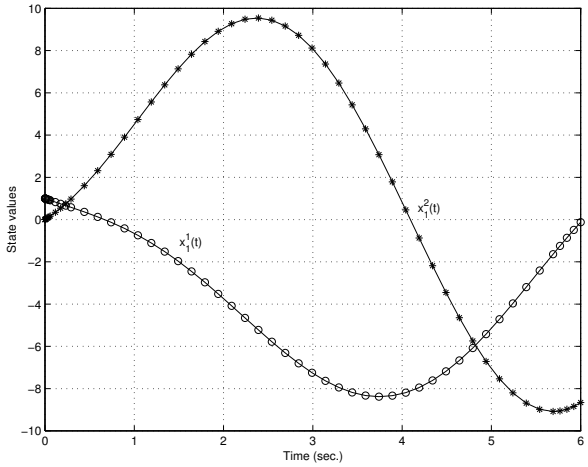

Fig. 3. Response of the model described by (9) computed by numerical solution of the equation (1).

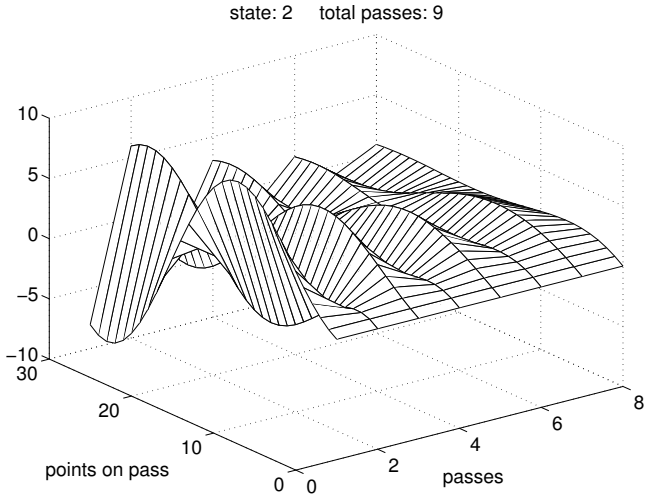

Fig. 4. Second state for linear repetitive process described by (9).

expected together with a loss of accuracy in the final result. Also the possibility of obtaining stiff problems is much higher, hence the model and its all initial values are known only for the first stage (first pass) of computations.

The output results generated by the*ODE solver* must match exactly the LRP *toolbox* data specification described in Section III-B and *vice versa*. To achieve this requirement a form of adjustment (interpolation in this context) of the data is required. The source files included in the next Subsection illustrate technical details here.

D. Matlab M–files

Function `rodefile` – this is a function *M–file* that models the first equation in (1). Note for lines with `interp1` MATLAB built-in function call where the interpolation of lack vector elements is performed.

```
function [dx] = rodefile(t,x,flag,varargin);
=========================================
% The function solves the following equation:
% dx/dt = A*x(t) + B*U(t) + B0*Y(t)
A= varargin{1}; B= varargin{2}; B0=varargin{3};
T=varargin{4}; % T must be monotonic! Regularly spacing not required.
U=varargin{5}; % Values of U at time points of T.
Y=varargin{6}; % Values of Y at time points of T.
% Local interpolation required.
low=find(T<=t); high=find(T>=t); p1=size(low,1);
test=0;
if ~isempty(high)
  p2=high(1,1);
else p2=size(T,1);
  if (t-T(size(T,1)))==eps; test=1; end
end
% Make interpolation to find the missing U(t).
% In these time points interpolation is not needed.
if t~=0 & t~=T(size(T,1)) & any(t==T)==0 & test==0
  ut=interp1([T(p1) T(p2)],[U(p1,:); U(p2,:)], t);
else
  ut=U(p1,:);
end
% Make the same for Y.
low=find(T<=t); high=find(T>=t); p1=size(low,1); test=0;
if ~isempty(high)
  p2=high(1,1);
else
  p2=size(T,1);
  if (t-T(size(T,1)))==eps; test=1; end
end
if t~=0 & t~=T(size(T,1)) & any(t==T)==0 & test==0
  yt=interp1([T(p1) T(p2)],[Y(p1,:); Y(p2,:)], t);
else
  yt=Y(p1,:);
end
dx = zeros(size(A,1),1);
dx = A*x + B*ut' + B0*yt';  % note for transpose
```

Function `rsolve` – this is a function *M–file* that makes use of the previously shown `rodefile` function. Note for the loop: `for i=1:total_passes`, where the solutions for every pass profiles are calculated and printed out to the screen. Usage of functions `rm2_d3d` and `rm3d_2d` simplifies the codes and also make the final resulting matrices compatible with the data format specification detailed in Section III-B.

```
function [xout,yout,stat,stat2] = rsolve(a,b,b0,c,d,d0,u,x0,y0,T);
=================================================================
% Numerical solution of a k-th pass of a given LRP
% described by a differential equation of the form:
% x'(k+1,t) = a*x(k+1,t) + b*u(k+1,t) + b0*y(k,t)
% y(k+1,t) = c*x(k+1,t) + d*u(k+1,t) + d0*y(k,t)
odesolver ='ode45';
% Accuracy. See the Matlab ODE Solvers guide.
RelTol=1e-3;   AbsTol=1e-6;
if nargin~=10 error('Wrong number of input arguments.'); end
options=odeset('RelTol',RelTol, 'AbsTol',AbsTol,'MaxStep',1000000);
disp(' ');
disp(['RelTol=',num2str(RelTol),'  AbsTol=',num2str(AbsTol)]);
disp(' ');
% The last element in vector 't' is assumed to be the pass length.
alpha=T(size(T,1));
p=size(y0,2);
total_passes=size(x0,2)-1;
stat2=cell(1,total_passes);
yode=y0';
% Calculate for all passes
for i=1:total_passes
  x0ode=x0(:,i+1)'; u3d=rm2d_3d(u,p);
  uode=u3d(i+1,:,:); uode=permute(uode,[3,2,1]); uode=uode';
  % Call the solver for the first time.
  [tout,xode,s]=
  feval(odesolver,'rodefile',T,x0ode,options,a,b,b0,T,uode,yode);
  % Call the solver for the second time to
  % determine the minimal and maximal step size used by the solver.
  [ttemp,xtemp]=
  feval(odesolver,'rodefile',
  [0 alpha],x0ode,options,a,b,b0,T,uode,yode);
  minStepSize= min(abs(ttemp(1:size(ttemp)-1,1) -
  ttemp(2:size(ttemp),1)));
  maxStepSize= max(abs(ttemp(1:size(ttemp)-1,1) -
  ttemp(2:size(ttemp),1)));
  % Save the ode solver solution.
  stat2{1,i}=[ttemp,xtemp];
  yode=c*xode' + d*uode' + d0*yode';
  yode=yode';
  if i<10, ii=[num2str(i),' ']; else, ii=[num2str(i)]; end
  disp(['Step ',ii,' done. Min step: ',num2str(minStepSize),...
        '   Max step: ',num2str(maxStepSize),...
        '   Statistics:  ',num2str(s')]);
  xout3d(i,:,:)=xode; yout3d(i,:,:)=yode; stat(i,:)=s';
end
% Make output results compatible with the LRP Toolbox
% (see rm2d_3d.m)
xout = rm3d_2d([ones(1,size(xout3d,2),size(xout3d,3))*NaN; xout3d]);
yout = [ones(1,size(yout3d,2),size(yout3d,3))*NaN; yout3d];
yout(1,:,:)=y0'; yout = rm3d_2d(yout);
```

Function `rm2d_3d` – has an auxiliary meaning. Used in `rsolve` function.

```
function [m3] = rm2d_3d(m2,p);
==============================
% Convertion from 2d-style LRP matrices to the 3d-style one.
if (nargin~=2), error('Wrong number of input arguments.'); end
[row,col]=size(m2);
for i=1:row
  for j=1:(col/p)
    m3(j,:,i)=m2(i,p*(j-1)+1 : p*(j-1)+p);
  end
end
```

Function `rm3d_2d` – has an auxiliary meaning. Used in `rsolve` function.

```
function [m2] = rm3d_2d(m3);
============================
% Convertion from 3d-style  LRP matrices to the 2d-style.
% Converts a matrix created by rm2d_3d.m back to its 2d-style.
if (nargin~=1), error('Wrong number of input arguments.'); end
[row,col,dim3]=size(m3);
for i=1:dim3
  for j=1:row
    m2(i,col*(j-1)+1 : col*(j-1)+col)=m3(j,:,i);
  end
end
```

IV. Conclusions

In the paper we have first given a short introduction to differential linear repetitive processes (details can be found in the attached references list). Then we have presented the MATLAB–based *toolbox* for supporting analysis of these processes with particular emphasis on a module of the *toolbox* for direct solving the repetitive equations of (1) with intensive use of the MATLAB *ODE Suite* mechanism. Simplified (but fully functional) MATLAB *M–files* which realize this task are included. A problem not investigated here is the accuracy and numerical stability of calculated results. This problem is currently under investigation and results will be available in due course.

References

[1] E. Rogers and D.H. Owens, *Stability analysis for linear repetitive processes*, vol. 175 of *Lecture Notes in Control and Information Science*, Springer Verlag, Berlin, 1992.

[2] T. Kaczorek, "Singular 2D continuous-discrete linear systems," *Bulletin Polish Academy of Science. Technical Sciences. Electronics and Electrotechnics*, vol. 42, pp. 417–422, 1994.

[3] K. Galkowski, E. Rogers, A. Gramacki, J. Gramacki, and D.H. Owens, "Stability and dynamic boundary condition decoupling analysis for a class of 2–D discrete linear systems," *IEE Proceedings - Circuits, Devices and Systems*, vol. 148, no. 3, pp. 126–134, 2001.

[4] J. Gramacki, *Methods of testing stability and stabilization of linear discrete repetitive processes*, Ph.D. thesis, Technical University of Zielona Gora, Computer Eng. and Electronics Dept., 1999, (in Polish).

[5] D.H. Owens and E. Rogers, "Stability analysis for a class of 2D continuous–discrete linear systems with dynamic boundary

conditions," *Systems and Control Letters*, vol. 37, pp. 55–60, 1999.

[6] E. Rogers, K. Galkowski, A. Gramacki, J. Gramacki, and D.H. Owens, "Stability and controllability of a class of 2–D linear systems with dynamic boundary conditions," *IEEE Transactions on Circuits and Systems – I. Fundamental Theory and Applications*, vol. 49, no. 2, pp. 181–195, 2002.

[7] E. Rogers, J. Gramacki, K. Galkowski, and D.H. Owens, "Stability theory for a class of 2D linear systems with dynamic boundary conditions," *Proceedings of the CDC–98, Tampa, USA*, pp. 2800–2805, 1998.

[8] K. Galkowski, E. Rogers, A. Gramacki, J. Gramacki, and D.H. Owens, "Higher order discretization methods for a class of 2–D continuous–discrete linear systems," *IEE Proceedings – Circuits, Devices and Systems*, vol. 146, no. 6, pp. 315–320, 1999.

[9] A. Gramacki, *Discretization of linear, differential repetitive processes*, Ph.D. thesis, Technical University of Zielona Gora, Computer Eng. and Electronics Dept., 1999, (in Polish).

[10] A. Gramacki, "On a new method of discretization of differential linear repetitive processes," *Bulletin of the Polish Academy of Science, Technical Sciences*, vol. 48, no. 14, pp. 539–560, 2000, (in Polish).

[11] A. Gramacki, K. Galkowski, E. Rogers, and D.H. Owens, "Methods for the discretization of a class of 2D continuous–discrete linear systems," *Proceedings of the CDC–99, Phoenix, USA*, December 1999, (CD–ROM).

[12] A. Gramacki, J. Gramacki, K. Galkowski, E. Rogers, and D.H. Owens, "From continuous to discrete models of linear repetitive processes," *Archives of Control Sciences*, vol. 12, no. 1–2, pp. 151–185, 2002.

[13] L. F. Shampine and M. W. Reichelt, "The MATLAB ODE Suite," *SIAM Journal on Scientific Computing*, vol. 18-1, pp. 1–22, 1997.

[14] Inc. The MathWorks, *Using MATLAB, Version 5*, 24 Prime Park Way, Natick, Massachusets 01760, 1998.

[15] K. Galkowski, E. Rogers, A. Gramacki, J. Gramacki, and D.H. Owens, "Development of a MATLAB toolbox for a class of 2D linear systems," *Systems Analysis–Modelling–Simulation*, vol. 38, pp. 313–324, 2000.