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 2D 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 LRPCs 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 mechanism. 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 \( \alpha \) is the finite pass length. Figure 1 gives schematic illustration of the evolution of the dynamics of a repetitive process. The simplest possible case 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 (\( \alpha \) 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]

\[
\begin{align*}
\dot{x}_{k+1}(t) &= A x_{k+1}(t) + B u_{k+1}(t) + B_0 y_k(t) \\
y_{k+1}(t) &= C x_{k+1}(t) + D u_{k+1}(t) + D_0 y_k(t).
\end{align*}
\]

(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@iie.uz.zgora.pl, 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 Zielona Gora, Poland, K.Galkowski@issi.uz.zgora.pl

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

$$
x_{k+1}(0) = d_{k+1}, \quad k \geq 0
$$

$$
y_0(t) = f(t), \quad 0 \leq t \leq \alpha
$$

where $d_{k+1}$ is an $n \times 1$ vector with constant entries and $f(t)$ is an $n \times 1$ 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

$$
x_{k+1}(p+1) = Ax_{k+1}(p) + Bu_{k+1}(p) + B_0y_k(p)
$$

$$
y_{k+1}(p) = Cx_{k+1}(p) + Du_{k+1}(p) + D_0y_k(p)
$$

with boundary conditions

$$
x_{k+1}(0) = d_{k+1}, \quad k = 0, 1, \ldots
$$

$$
y_0(p) = f(p), \quad p = 0, 1, \ldots, \alpha - 1
$$

where $p = \{0, 1, \ldots, \alpha - 1\}$ 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, verify a discretized model of (3) and hence of an approximate solution generated by (3) and (4).

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 format, 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 $\alpha$,
- the number of passes, say $K$, over which the simulation is to run,
- the sequence of input vectors $u_k(p), \quad k = \{0, 1, \ldots, K\}, \quad 0 \leq p \leq \alpha$,
- the initial state vector sequence $x_k(0), \quad k = \{0, 1, \ldots K\}$,
- the initial pass profile $y_0(p), \quad 0 \leq p \leq \alpha$,
- 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
\( x_k(p), \ k = \{0, 1, \ldots, K\}, \ 0 \leq p \leq \alpha, \)

- the pass profile at each instant along each pass
\( y_k(p), \ k = \{0, 1, \ldots, K\}, \ 0 \leq p \leq \alpha. \)

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 \( \alpha/T \) 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 \\textit{toolbox} appeared, MATLAB (version 4.2) did not support multidimensional (that is for \( n \geq 3 \)) 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.

\[
\begin{array}{c|c|c}
\text{u} & \text{Pass 0} & \text{Pass 1} \\
\text{y} & \text{Pass 0} & \text{Pass 1} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{x} & \text{Pass 0} & \text{Pass 1} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{y} & \text{Pass 0} & \text{Pass 1} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{yn} & \text{Pass 0} & \text{Pass 1} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{x} & \text{Pass 0} & \text{Pass 1} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{y} & \text{Pass 0} & \text{Pass 1} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{yn} & \text{Pass 0} & \text{Pass 1} \\
\end{array}
\]

![Figure 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.](image)

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

\[
A = \begin{bmatrix} 1 & -2 & -1 \\ 3 & -5 & 1 \\ -1 & 2 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 3 & -1 \end{bmatrix},
\]

\[
B_0 = \begin{bmatrix} 1 & 1 \\ 2 & 2 \\ -1 & -1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix},
\]

\[
D = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 1 & 1 \end{bmatrix}, \quad D_0 = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix}.
\]

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

\[
x = \begin{bmatrix} 4N & 4N & 4N \\ 4N & 4N & 4N \\ 4N & 4N & 4N \end{bmatrix}, \quad y = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \\ 1 & 1 & 1 \\
1 & 1 & 1 \\ -1 & -1 & -1 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \\ 1 & 1 & 1 \\ -1 & -1 & -1 
\end{bmatrix}
\]

where \( 4N \) (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)).

\[C. \text{ Solving Linear Repetitive Processes with Matlab ODE Solvers} \]

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

\[
y' = F(t, y)
\]

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 \( M\text{-file} \) 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 \( M\text{-file} \) as an input parameter \( ode \) file.

Now consider the special case of (1)

\[
\tilde{A} = \begin{bmatrix} -0.5 & -0.7 \\ 0.7 & -0.3 \end{bmatrix}, \quad \tilde{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \tilde{B}_0 = \begin{bmatrix} 1.9 \\ 6.4 \end{bmatrix},
\]

\[
\tilde{C} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \tilde{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \tilde{D}_0 = \begin{bmatrix} -0.7 \end{bmatrix}.
\]
the 1D differential linear systems state space model
\[
\begin{align*}
\dot{x}(t) &= \dot{Ax}(t) + \dot{Bu}(t) + \dot{B}_0y(t) \\
y(t) &= \dot{Cx}(t) + \dot{Du}(t) + \dot{D}_0y(t).
\end{align*}
\] (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 \textit{ode} file for solving (10):

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

and then from \textit{MATLAB} prompt we write:

```matlab
tode=[0 6]; x0ode=[1;0];
[tout,xout]=ode45('lrpodefile',tode,x0ode);
plot(tout(:,1), xout(:,2),'-*');
hold on, grid on
function plots the computed time re-
plot (Figure 3). Note the ‘condensation’ of the points at the
end 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 \textit{3D} plot (Figure
4) of the second stage 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

\[ \text{ MATLAB M-files} \]

Function \textit{rodefile} – this is a function \textit{M-file} that models
the first equation in (1). Note for lines with \textit{interp1}
MATLAB built-in function call where the interpolation of
lack vector elements is performed.
for i=1:total_passes
%Calculate for all passes
yode=y0';
stat2=cell(1,total_passes);
total_passes=size(x0,2)-1;

RelTol=1e-3; AbsTol=1e-6;

% Make output results compatible with the LRP Toolbox
% (see rm2d_3d.m)
xout = rm3d_2d([ones(1,size(xout3d,2)),size(xout3d,3)]);%N=3; xout;
yout = rm3d_2d([m3(1,1,:)]; yout3d(1,:)); yode; stat(1,:)=%;

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

function [m3] = rm2d_3d(m2,p);
% Conversion 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 j=1:row
m3(:,j,:)=m2(i,p*(j-1)+1 : p*(j-1)+p); end
end

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

function [m3] = rm3d_2d(m2);
% Conversion from 3d-style LRP matrices to the 2d-style.
if nargin~=2, error('Wrong number of input arguments.'); end
[row,colm]=size(m3);
for i=1:colm
m3(:,i,:)=m2(1,p*(j-1)+1 : p*(j-1)+p); 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


