A TOOLBOX FOR COMPUTATIONAL ANALYSIS OF PIECEWISE LINEAR SYSTEMS

Sven Hedlund, Mikael Johansson

Department of Automatic Control
Lund Institute of Technology, Box 118, 221 00 Lund, Sweden.
Phone:(+46)-46-222 42 87, Fax:(+46)-46-138118, Email:sven@control.lth.se

Keywords piecewise linear system, LMIs, toolbox

Abstract

This paper reports the development of a Matlab toolbox for computational analysis of piecewise linear systems. The analysis is based on piecewise quadratic Lyapunov functions, which are computed via convex optimization. In this way, exponential stability and system performance can be assessed. The toolbox also supports efficient simulation of systems with discontinuous dynamics and sliding modes. A set of intuitive commands for describing piecewise linear systems is included, making the analysis routines easily accessible also for the inexperienced user.

1. Introduction

As performance demands on modern control systems increase, controllers are required to work over large operating ranges where assumptions on linear dynamics are no longer valid. Successful design and tuning of such controllers are strongly dependent on the possibility of analyzing the effects that arise away from equilibrium conditions. An interesting class for studying such problems is the class of piecewise linear systems. It captures the effects of saturations and state constraints, and is also a good candidate for studying hybrid control systems (cf. [Son96]). Moreover, many popular control schemes, such as gain scheduling and fuzzy logic controllers, can be well modeled by piecewise linear systems (cf. [ÅJB98]).

Recently, it has been shown how stability and performance of piecewise linear systems can be assessed using Lyapunov functions that are piecewise quadratic [JR96]. Such Lyapunov functions can be computed via convex optimization in terms of linear matrix inequalities (LMIs). The approach gives a drastic reduction of conservatism compared to approaches based on a single quadratic Lyapunov function[Cor94], while computations remain comparatively efficient.

This paper gives an overview of a MATLAB toolbox for computational analysis of piecewise linear systems. The main purpose of the paper is to show how simple the toolbox, PWLTOOL, makes experimenting with piecewise linear systems. For a detailed description of usage, the reader is referred to the manual [HJ99].

PWLTOOL is available free of charge upon request from the authors.

2. Model Representation

The toolbox handles piecewise affine systems on the form

$$\begin{cases} \dot{x} = A_i x + a_i + B_i u \\ y = C_i x + c_i + D_i u \end{cases} \qquad \text{for } x \in X_i. \tag{1}$$

Here, $\{X_i\}_{i\in I} \subseteq \mathbf{R}^n$ is a partition of the state space into a number of closed (possibly unbounded) polyhedral cells (cf. e.g. Fig. 1), and I is the index set of the cells. In order to allow rigorous analysis of smooth nonlinear systems, the toolbox allows the system dynamics to lie in the convex hull of a set of piecewise affine systems, see [Joh99]. This is e.g. useful for the analysis of fuzzy Takagi-Sugeno systems.

For convenient notation, we introduce

$$\bar{A}_i = \begin{bmatrix} A_i & a_i \\ 0 & 0 \end{bmatrix} \qquad \bar{C}_i = \begin{bmatrix} C_i & c_i \end{bmatrix} \qquad \bar{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

A large part of the analysis results will be concerned with (global) properties of equilibria. We therefore let $I_0 \subseteq I$ be the set of indices for the cells that contain the origin, and $I_1 \subseteq I$ be the set of indices for cells that do not contain the origin. We will assume that $a_i = 0$, $c_i = 0$ for $i \in I_0$.

The cells are represented by matrices $\bar{G}_i$ that satisfy

$$\bar{G}_i \bar{x} \succeq 0, \qquad \text{if and only if } x \in X_i \tag{2}$$

Here, the vector inequality $z \succeq 0$ means that each entry of z is non-negative. We recognize this as the halfspace

command	description
setpwl	initialize PWL object
addregion	define polyhedral region
addynamics	define system dynamics
getpwl	extract PWL object

Table 1 Commands for building a PWL system.

representation of a polyhedron. It is also necessary to specify matrices $\bar{F}_i = [\,F_i \quad f_i\,]$ with $f_i = 0$ for $i \in I_0$ that satisfy

$$\bar{F}_i \bar{x} = \bar{F}_j \bar{x} \qquad \text{for } x \in X_i \cap X_j. \tag{3}$$

These matrices are used to parameterize the Lyapunov function candidate to be continuous across cell boundaries. The PWLTOOL handles a piecewise linear (PWL) system as an object. The basic commands for building a PWL system are listed in Table 1. Having partitioned the state space and used the functions for entering data into MATLAB, the system is aggregated into a single record that is passed on to functions for analysis and simulations.

The command `setpwl` initializes the PWL object and should be run first. When this is done, one will typically define the entire system by repeatedly calling `addynamics` and `addregion`. The command `addynamics` is used to specify the matrix variables (as given by (1)) corresponding to the dynamics in a certain region of a PWL system. An identifier is returned for future reference to the dynamics. The command `addregion` lets the user enter the region specific data ($\bar{G}_i$- and $\bar{F}_i$-matrices) and via the references returned by `addynamics` specify the dynamics in the region. By specifying several system matrices in one region, one indicates that the dynamics lies in the convex hull of these systems. When all matrices are entered, the PWL object is extracted by `getpwl`. In addition to linking several dynamics to one region, it is also possible to link several regions to the same dynamics. (This could sometimes be useful to save some data space and typing effort.)

EXAMPLE 1—THE FLOWER SYSTEM
The following system, whose partition is illustrated in Figure 1, has been used in [JR96] in order to demonstrate the flexibility of piecewise quadratic Lyapunov functions.

$$\dot{x} = \begin{cases} A_1 x = \begin{bmatrix} -0.1 & 1 \\ -5 & -0.1 \end{bmatrix} x & x \in X_1 \cup X_3 \\ A_2 x = \begin{bmatrix} -0.1 & 5 \\ -1 & -0.1 \end{bmatrix} x & x \in X_2 \cup X_4 \end{cases}$$

The following lines of code defines the "flower system".

```
% Initialize the PWL object
  setpwl([]);
% Enter A-matrices
  A1 = [-0.1 1; -5 -0.1];
  A2 = [-0.1 5; -1 -0.1];
% Set up dynamics
  d1 = addynamics(A1);
  d2 = addynamics(A2);
% Enter G- and F-matrices
  G1 = [ 1  1 0; -1  1 0];
  G2 = [ 1 -1 0;  1  1 0];
  G3 = [-1 -1 0;  1 -1 0];
  G4 = [-1  1 0; -1 -1 0];
  F1 = ...
  F2 = ...
  ...
% Define cells
  addregion(G1, F1, d1);
  addregion(G2, F2, d2);
  addregion(G3, F3, d1);
  addregion(G4, F4, d2);
% Extract PWL object
  pwlsys = getpwl;
```

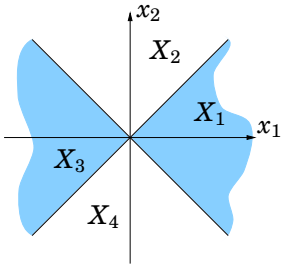

Figure 1. Partitions of the flower example.

We will return to this example later to assess global exponential stability of the origin. □

3. Describing Polyhedral Partitions

Defining all the data that the computational engine of PWLTOOL needs can be far from easy for the inexperienced user. It is therefore desireable to relieve the user from this task. In this section, we describe a set of user-friendly commands for specifying piecewise linear systems that automatically computes the constraint matrices, G_i and F_i, used by PWLTOOL.

The toolbox currently supports partitions induced by global hyperplanes and simplex partitions (see [Joh99] for precise definitions, and more elaborate explanations), but the layered structure of the toolbox makes it easy to add support for other types of partitions.

3.1 Describing Hyperplane Partitions

Specifying a hyperplane partition essentially consists of defining the generating hyperplanes, introducing the cells by stating which generating hyperplanes that bound the cell, and giving the affine dynamics valid within each region. Table 2 specifies a number of commands that support these steps.

The command `setpart` intializes a new partition, and should be issued prior to defining the partition

Command	Description
setpart	Initialize partition data structure
addhp	Add hyperplane
addati	Specify affine dynamics
addhcell	Define hyperplane cell
getpart	Retrieve partition data structure
part2pwl	Convert to generic data structure

Table 2 Commands for defining hyperplane partitions.

components. In order to indicate the type of partition, `setpart` takes the argument `'h'` for hyperplane partitions and `'s'` for simplex partitions.

The commands `addhp` and `addati` define generating hyperplanes and affine dynamics respectively. Both commands return an identifier for later reference. Cells are subsequently defined using the command `addhcell`, which takes two arguments. The first argument specifies the bounding hyperplanes (using their identifiers returned by `addhp`), and the second argument specifies the dynamics valid in the region (using the identifiers returned by `addati`). The sign of the hyperplane reference indicates on "what side" of the hyperplane the cell is located.

The command `getpart` returns a data structure that describes the partition. Finally, the command `part2pwl` computes the data required by the computational engine `pwltools`. The computations performed by `part2pwl` are explained in [Joh99]

We illustrate the commands on a simple relay feedback system.

EXAMPLE 2—A RELAY FEEDBACK SYSTEM
Consider a linear system under relay feedback

$$\begin{cases} \dot{x} &= Ax + Bu \\ y &= Cx \\ u &= -\text{sign}(y). \end{cases}$$

The relay feedback induces a piecewise linear system with two regions, separated by the switching hyperplane $Cx = 0$. The following lines of MATLAB code define the relay system using hyperplane partitions.

```
% Initialize hyerplane partition
  setpart('h');
% Define boundary hyperplanes
  switch_plane = addhp([C 0]);
% Dynamics \dot{x}=Ax+B and \dot{x}=Ax-B
  d_on = addati(A,-B);
  d_off= addati(A,B);
% Introduce cells
  X_1 = addhcell(switch_plane, d_on);
  X_2 = addhcell(-switch_plane, d_off);
% Retrieve data structure
  part = getpart;
% Transform to pwl data structure
  pwlsys = part2pwl(part);
```

□

Command	Description
setpart	Initialize partition data structure
addvtx	Add vertex
addray	Add ray
addati	Specify affine dynamics
addscell	Define simplex cell
getpart	Retrieve partition data structure
part2pwl	Convert to generic data structure

Table 3 Commands for defining simplex partitions.

3.2 Describing Simplex Partitions

The specification of a simplex partition is very similar to the definition of a hyperplane partition. The main difference is that (generalized) simplices are defined by vertices ("points") and rays ("directions") rather than the equations for its bounding hyperplanes (cf. [Joh99]). The commands for building simplex partitions are shown in Table 3. A new simplex partition is initialized by the command `setpart('s')`. A (generalized) simplex in R^n is defined by $n+1$ vertices or rays. Vertices and rays are defined by the commands `addvtx` and `addray`. Both commands return an identifier for later reference. As in the hyperplane case, the dynamics are defined by the command `addati`. The cells of the partition are defined by the command `addscell`, which takes three arguments. The first two arguments are lists of vertex and ray references respectively, while the last argument specifies the dynamics valid within the region. The total number of vertex and ray references sums to $n+1$, and at least one extreme point of the cell is a vertex. Once all cells are defined, the command `getpart` retrieves a data structure describing the partition, and the command `part2pwl` transform this information into the data required by `pwltools`. We return to Ex. 1 to demonstrate the commands.

EXAMPLE 3—FLOWER SYSTEM – SIMPLEX DESCRIPTION

```
% Initialize simplex partition
  setpart('s');
% Define vertices and rays
  v1 = addvtx([0 0]);
  r1 = addray([1 1]);
  r2 = addray([-1 1]);
  r3 = addray([-1 -1]);
  r4 = addray([1 -1]);
% Set-up dynamics
  d1 = addati(A1);
  d2 = addati(A2);
% Define cells
  X_1 = addvcell([v1],[r1 r2],d1);
  X_2 = addvcell([v1],[r2 r3],d2);
  X_3 = addvcell([v1],[r3 r4],d1);
  X_4 = addvcell([v1],[r4 r1],d2);
% Retrieve partition data structure
  part = getpart;
% Transform into pwltools data structure
  pwlsys = part2pwl(part);
```

□

4. Simulation of Piecewise Linear Systems

Simulation is one of the most important tools for evaluating new control strategies, in academia as well as in industry. Although there has been a strong development of general-purpose simulation environments during the last 20 years, simulation of systems with switching and discontinuous dynamics is still poorly supported by most software packages. In the context of piecewise

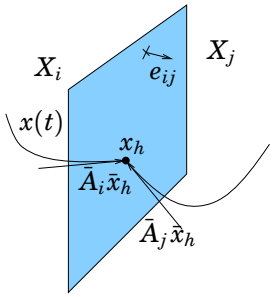

Figure 2. Sliding mode surface.

linear systems, problems may occur when the vector fields are discontinuous across cell boundaries. If the flow in two neighboring cells point toward their common boundary, cf. Fig. 2, the state goes through a number of infinitely fast mode changes that cause most simulators to 'get stuck'. The nature of these fast mode changes has been studied by several researchers, see [Fil88, Utk77]. In general, the net effect of the fast mode switches is a constrained motion along the swithing surface, referred to as a sliding motion. The dynamics of the sliding motion can be uniquely defined for simple boundaries, while intersecting boundaries may cause uniqueness problems. Figure 3 gives an overview of how PWLTOOL handles simulations. Before starting, some preparatory computations are made. During the initialization phase, 1, each region is assigned a number of pointers to the neighboring regions to allow for efficient switching. In addition, each surface separating the regions undergo sliding mode analysis. Define e_{ij} to be the normal vector of the hyperplane between X_i and X_j directed from X_i to X_j, cf. Fig. 2. The surface then contains a sliding mode if there exist an x such that

$$\begin{aligned} \bar{G}_i\bar{x} &\succeq 0 \\ \bar{G}_j\bar{x} &\succeq 0 \\ e_{ij}^T\bar{A}_i\bar{x} &> 0 \\ -e_{ij}^T\bar{A}_j\bar{x} &> 0 \end{aligned} \tag{4}$$

This is an LP problem, the result of which is patched up for each boundary into one single matrix. The first step of the actual simulation is to find the initial region, 2, i.e. if starting in x_0, find i such that $\bar{G}_i\bar{x}_0 \succeq 0$. During the first visit to 2, the G_i-matrices have to be

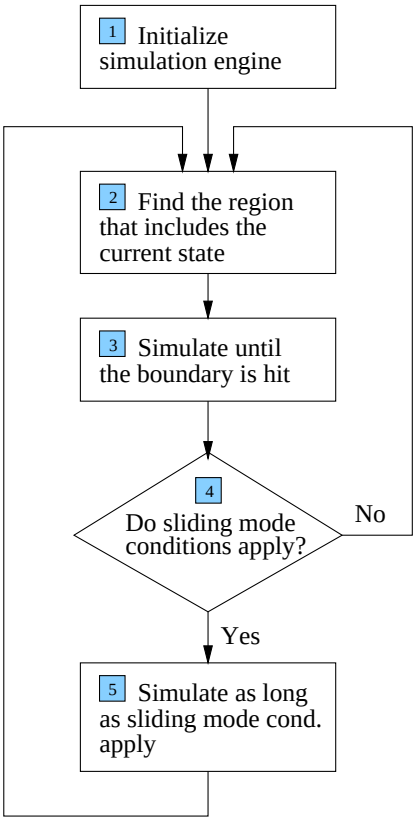

Figure 3. Schematical description of simulation algorithm for systems with sliding modes.

tested one by one. Thanks to the initialization phase, however, this is avoided when entering next time. Having found the right region, the simulation is started, 3, and proceeded until the boundary is hit. When a boundary is hit, one must check whether to enter the sliding mode state, 4. This is done by first looking up into the sliding mode matrix whether the surface contains a sliding mode. If it does, the conditions (4) are checked for the specific entry point. Having entered the sliding mode state, 5, The resulting equivalent dynamics is computed according to Filippov's convex definition [Fil88]:

$$\dot{x} = \begin{cases} \bar{A}_i\bar{x}, & x \in X_i \\ \bar{A}_j\bar{x}, & x \in X_j \\ \lambda(x)\bar{A}_i\bar{x} + (1-\lambda(x))\bar{A}_j\bar{x}, & x \in X_i \cap X_j \end{cases}$$

where $\lambda(x)$ is the solution to

$$e_{ij}^T\left(\lambda(x)\bar{A}_i\bar{x} + (1-\lambda(x))\bar{A}_j\bar{x}\right) = 0$$

Currently, PWLTOOL does not support sliding mode on intersecting hyperplanes.

Table 4 lists the commands that are available for detecting sliding modes and simulating pwl systems with sliding modes. The command `findsm` searches all the boundaries between cells and informs the user of

command	description
findsm	detect sliding modes
pwlsim	simulate PWL system

Table 4 Simulation related commands.

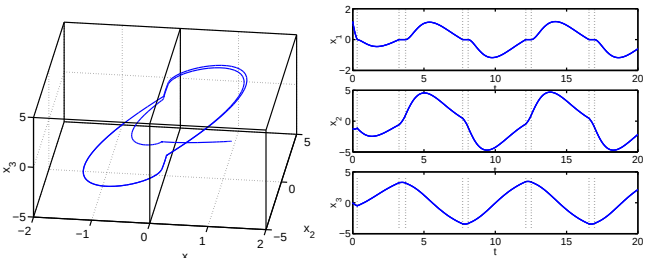

Figure 4. Limit cycle with sliding trajectory. The vertical dashed lines in the right part indicate time instances for the mode selection.

between which cells sliding modes are possible. This is of course interesting from a stability analysis point of view. Knowing that most uncontrolled systems do not exhibit sliding modes, however, this command can sometimes give a first warning if the system is not modelled in an appropriate way.

A trajectory can be simulated from a given initial state with `pwlsim`. The outputs from this function are, in addtition to the time vector and matching state vectors, the times when cell switching occured and the corresponding cells that have been visited.

EXAMPLE 4—RELAY SYSTEM WITH SLIDING MODE
Returning to the relay feedback system of Ex. 2, we now consider the following nonminimum phase system from [Joh97]:

$$\dot{x} = \begin{bmatrix} -3 & 1 & 0 \\ -3 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} u$$
$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x$$

It is assumed that `A` and `B` have been entered before executing the code of Ex. 2.

```
% Search for sliding modes
  findsm(pwlsys);
    Sliding mode detected on boundary between
    cell 1 and 2.
% Simulate the system
  x0 = [1 -1 0]';
  [t, x, te] = pwlsim(pwlsys, x0, [0 20]);
```

The above code establishes that the system exhibits a sliding mode on the switching surface. Simulating the system using the command `pwlsim`, one can see how the system tends to a limit cycle with sliding mode, see Figure 4. □

command	description
qstab	quadratic stability analysis
pqstab	piecewise quadratic analysis
pqstabs	d.o. taking sliding into accout

Table 5 Commands for stability analysis.

5. Computation of Piecewise Quadratic Lyapunov Functions

In PWLTOOL , stability of pwl systems is proved with the aid of piecewise quadratic (pwq) Lyapunov functions. This is less conservative than the commonly used global quadratic approach and the toolbox makes it possible to prove stability for pwl systems that do not admit quadratic Lyapunov functions.

The F-matrices as defined by Eq. (3) are used to force continuity of the Lyapunov function. It is parameterized by a symmetric matrix, T, as follows

$$V(x) = \bar{x}^T \bar{F}_i^T T \bar{F}_i \bar{x} \qquad x \in X_i, \qquad i \in I.$$

This structure allows the usual constraints on $V(x)$ (positive definiteness and decrement along the system trajectories) to be expressed as a set of LMIs [Joh99].

The commands provided for stability analysis are shown in Table 5. The command `pqstab` searches for a pwq Lyapunov function as described above. If there exist a piecewise quadratic Lyapunov function, `pqstab` returns a three dimensional array, a vector of matrices, where matrix no. i corresponds to $\bar{F}_i^T T \bar{F}_i$ of eq. (5). The command `qstab` tries to find a *global* quadratic Lyapunov function ($V(x) = x^T P x$). This is of course conservative, but `qstab` uses the state space partitioning structure to relax the constraints on the Lyapunov function. In addition, the simplicity of a globally quadratic function often makes it a natural choice for a first attempt.

The LMI:s stated in `pqstab` for the decreasing condition are only valid for systems without any sliding modes. The command `pqstabs` is slightly modified to be able to handle the sliding mode case.

EXAMPLE 5—FLOWER SYSTEM — STABILITY ANALYSIS
We now try to prove stability of the flower system (Ex. 1).

```
% Search for sliding modes
  findsm(pwlsys);
  There are no sliding modes.
% Since there are no sliding modes, use pqstab
  pqstab(pwlsys);
  Lyapunov function was found.
```

Level surfaces of the lyapunov function are plotted together with a simulated trajectory (using `pwlsim` as shown in Ex 4) in Fig. 5. □

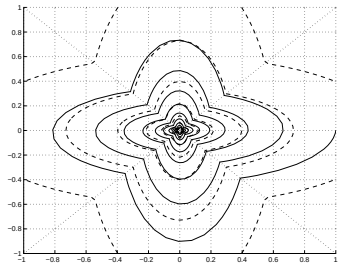

Figure 5. Simulation of the flower system (solid line) and level curves of a pwq lyapunov function (dashed).

command	description
iogain	L_2 gain computation
pqobserv	Output energy estimation
optcstlb	Lower cost for LQG problem
pwlctrl	Derive piecewise LQG controller
optcstub	Estimate cost achieved by pwlctrl

Table 6 Commands for performance analysis and control design.

6. Performance Analysis and Control Design

Having utilized the Lyapunov function machinery for assessing stability, it can be used in a similar way for other computations. PWLTOOL supports performance analysis and control design. Table 6 lists the commands available. All of these commands estimate an upper and/or a lower bound on a certain performance property. If the estimates are too coarse, the results can be refined by further refinement of the state space partitions. The command `iogain` computes an upper bound on the L_2 induced input output gain of a pwl system. The command `pqobserv` computes a lower and an upper bound on the integral of the output energy for a given initial state, $x(0)$.

There are three commands related to controller synthesis. The optimal control problem for piecewise linear systems (while bringing the system to $x(\infty) = 0$ from an arbitrary initial state, $x(0)$) can be defined to minimize the cost

$$J(x_0, u) = \int_0^\infty \left(\bar{x}^T \bar{Q}_{i(t)} \bar{x} + u^T R_{i(t)} u\right) dt$$

(Here $i(t)$ is defined so that $x(t) \in X_{i(t)}$.) A lower bound, on the minimum achievable J is computed by `optcstlb` The command `pwlctrl` creates a pwl controller based on the results from `optcstlb`. A vector of matrices representing the state feedback used in different regions is returned. Having applied the controller given from above, `optcstub` returns an upper bound on the resulting optimal cost.

7. Summary

This paper has presented a MATLAB toolbox for analysis of piecewise linear systems, a class of nonlinear systems that appears frequently in control theory, e.g. in hybrid systems and linear systems with various constraints. The analysis is based on piecewise quadratic Lyapunov functions, which are computed via convex optimization. In this way, exponential stability and system performance can be assessed for this class of nonlinear systems. The toolbox also supports efficient simulation of systems with discontinuous dynamics and sliding modes.

PWLTOOL makes it simple to experiment with piecewise linear systems. The authors provide it free of charge upon request, with a reference manual [HJ99] and additional examples.

Acknowledgements

This work was supported by the Esprit LTR project FAMIMO and TFR project 95-759.

8. References

[ÅJB98] Karl-Erik Årzén, Mikael Johansson, and Robert Babuska. A survey on fuzzy control. Technical report, Esprit LTR project FAMIMO Deliverable D1.1B, (1998).

[Cor94] M. Corless. Robust stability analysis and controller design with quadratic Lyapunov functions. In Alan S.I. Zinober, editor, *Variable Structure and Lyapunov Control*, Lecture notes in Control and Information Sciences, chapter 9, pages 181–203. Springer Verlag, (1994).

[Fil88] A. F. Filippov. *Differential Equations with Discontinuous Righthand Sides*. Kluwer, Dordrecht, (1988).

[HJ99] Sven Hedlund and Mikael Johansson. PWLTool — A Matlab toolbox for analysis of piecewise linear systems. Report TFRT-7582, Department of Automatic Control, Lund Institute of Technology, Lund,Sweden, March (1999).

[Joh97] Karl Henrik Johansson. *Relay Feedback and Multivariable Control*. PhD thesis, September (1997).

[Joh99] Mikael Johansson. *Piecewise linear control systems*. PhD thesis, Lund institute of technology, march (1999).

[JR96] Mikael Johansson and Anders Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. Report ISRN LUTFD2/TFRT–7459–SE, Department of Automatic Control, Lund Institute of Technology, Lund,Sweden, June (1996).

[Son96] E.D. Sontag. Interconnected automata and linear systems: A theoretical framework in discrete time. In R. Alur, T. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III: Verification and Control*, pages 436–448. Springer, NY, (1996).

[Utk77] V. I. Utkin. Variable structure systems with sliding modes: A survey. *IEEE Transactions on Automatic Control*, 22(2):212–222, (1977).