ISSN 0280–5316
ISRN LUTFD2/TFRT--7582--SE

PWLTOOL

A Matlab toolbox for analysis of Piecewise Linear Systems

Sven Hedlund
Mikael Johansson
Department of Automatic Control
Lund Institute of Technology
March 1999

1. Introduction

This manual describes a MATLAB toolbox for computational analysis of piecewise linear systems. Key features of the toolbox are modeling, simulation, analysis, and optimal control for piecewise linear systems. The simulation routines detect sliding modes and simulate equivalent dynamics [2]. The analysis and design are based on computation of piecewise quadratic Lyapunov functions [6]. The computations are performed using convex optimization in terms of linear matrix inequalities (LMIs). This version of the toolbox requires the LMI control toolbox [1].

The structure of this manual is as follows. Section 2 describes the model representation, i.e. how a piecewise linear (PWL) system is defined in this toolbox. Certain structures of the PWL systems allow the systems to be defined in a more automated fashion. These systems, in the sequel referred to as Structured PWL (sPWL) systems, are handled by an additional set of commands described in Section 3. Section 4 lists all the commands (with explanations) of the PWLTOOL in two groups. The first subsection contains the generic PWL commands, the second subsection describes the additional sPWL commands. Section 5 contains some examples of how to use the toolbox.

Appendix A describes the data structure of a PWL object in MATLAB.

2. Piecewise Linear (PWL) Systems: Model Description

The toolbox is based on piecewise linear 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, see Figure 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 [5]. This is e.g. useful for the analysis of fuzzy Takagi-Sugeno systems [8].

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 $i \in I_0$.

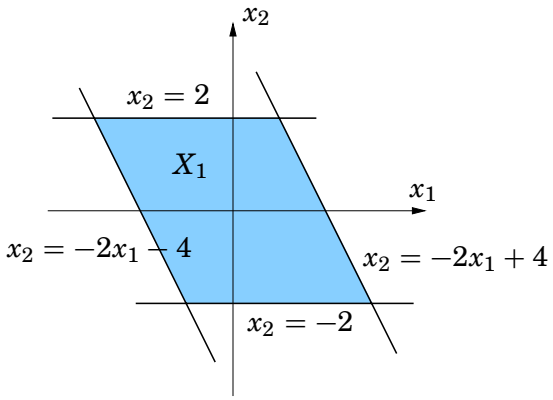

Figure 1 Example of a polyhedron in $\mathbf{R}^2$.

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 representation of a polyhedron, where each row of $\bar{G}_i$ corresponds to one halfspace. The $\bar{G}$-matrix for the polyhedron of Fig. 1 e.g. would be

$$\bar{G}_1 = \begin{bmatrix} 0 & -1 & 2 \\ -2 & -1 & 4 \\ 0 & 1 & 2 \\ 2 & 1 & 4 \end{bmatrix}$$

In addition to defining the regions of different dynamics, the G-matrices tell the PWLTOOL how to partition the Lyapunov functions that are used for the system analysis. A consequence of this is that one will sometimes divide the state space in to smaller cells than the ones implied by the system dynamics in order to increase the flexibility of the Lyapunov function candidate [6].

For the analysis of PWL systems, 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 Lyapunov functions that are continuous across cell boundaries. [1]

Note that the F-matrices are not a part of the PWL system definition itself, they are merely a computational aid in the system analysis such as stability, input output gain etc. Consequently, the *simulation* of a PWL system does not require these matrices.

Also note that Eq. (3) does not uniqely define the F-matrices. A more detailed description of the matrices can be found in [4]. For the inexperienced user, who might find it difficult to create appropriate F-matrices, Section 3 presents means to overcome this problem.

[1] The computations in [3, 7] use an additional matrix E_i. This matrix is derived directly from the corresponding G_i-matrix, and is therefore not requested from the user.

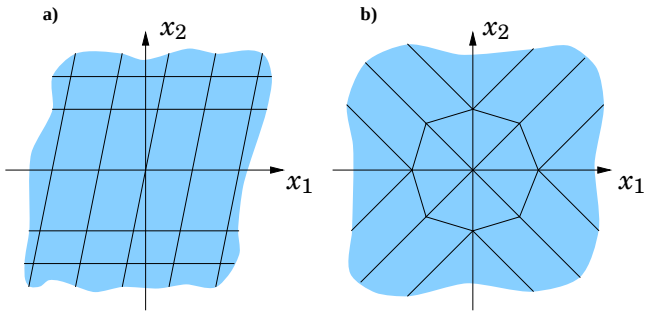

Figure 2 The special structures that are supported in sPWL package: **a)** Hyperrectangle partitions, **b)** Simplex partitions

3. Structured Piecewise Linear (sPWL) Systems — a user-friendly concept

As described in the previous section, it may be non-trivial to find the F-matrices for a system to be analyzed. Moreover, even if one can find matrices that satisfy the definition (3), they might not be the best ones in utilizing the piecewise structure of the Lyapunov functions. Thus it is desirable to be able to generate a good choice of F-matrices automatically.

When making the first attempts to analyze a PWL system, it is often natural to partition the state space in certain ways. For example when there is a need for approximating a general nonlinear function without considering a particular structure of the nonlinearity, one might as a first "quick and dirty" attempt grid the statespace using a set of hyperrectangles. Doing this, it is desirable to generate the G-matrices in a more automated fashion, since all the cells are similar in nature.

The toolbox supports automatic generation of region dependent matrices for two classes of PWL systems. These systems, called Structured Piecewise Linear (sPWL) systems, are constrained in the kind of state space partitions that are allowed, but cover many cases and have the advantage of making the construction of G- and F-matrices easy.

The classes that are supported in the sPWL package are called Hyperrectangle partitions and Simplex partitions. In a hyper rectangle partition, each state is split by a number of parallel hyperplanes, cf Fig. 2a. These planes build a set of hyperrectangles, the outermost rectangles extending to infinity. In a simplex partition, cf Fig. 2b, all cells are simplices, i.e. polyhedra that in an n-dimensional space are bounded by $n + 1$ vertices, some of which extend to infinity.

4. Command Reference

4.1 The PWL Package

The PWL package consists of the functions listed in Table 1-4:

Table 1 Model Construction

Command	Description
setpwl	Initialize the PWL system
addynamics	Add system dynamics
addregion	Add system region
getpwl	Extract the PWL system

Table 2 Model Analysis and Control Design

Command	Description
qstab	Quadratic stability (global)
pqstab	Piecewise quadratic stability
pqstabs	pqstab with sliding mode
pqobserv	observability
optcstlb	optimal cost, lower bound
optcstub	optimal cost, upper bound
iogain	input output gain

Table 3 Graphic Visualization

Command	Description
pwleval	evaluate PWL function
pwllevel	plot PWL function
pwqeval	evaluate PWQ function
pwqlevel	plot PWQ function

Table 4 Simulation

Command	Description
pwlsim	Simulate PWL system
findnb	Find neighbouring cells
findsm	Find possible sliding modes

4.2 The Added Structured PWL (sPWL) Package

Many of the commands of the sPWL package have a corresponding command in the model construction part of the generic PWL package. It is important not to mix up the two packages, however, since they use different data structures in MATLAB. The link between them is `part2pwl`, that converts the entered sPWL system to a generic PWLTOOL obejct. The interconnection between the packages is shown in Fig. 3.

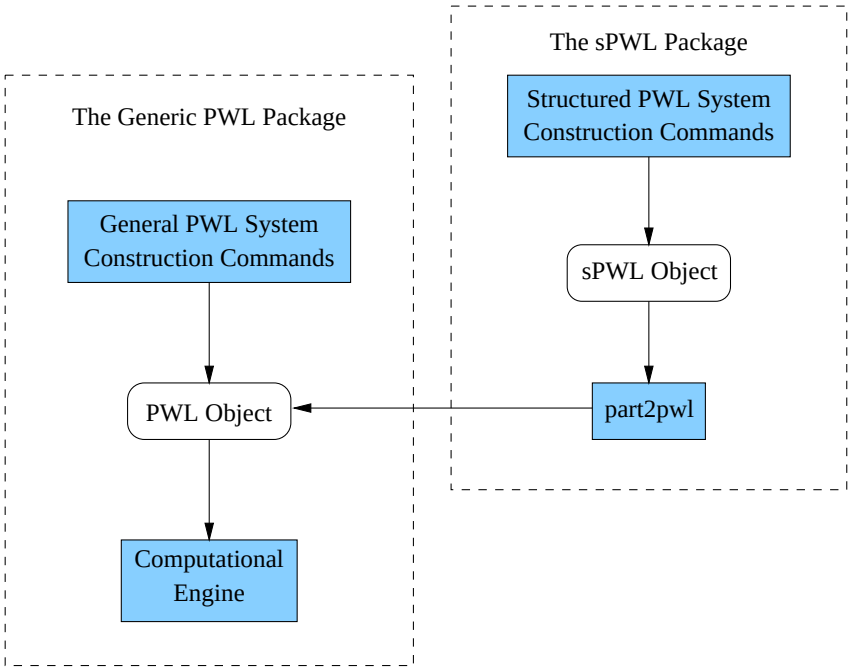

Figure 3 How the extra package for generating sPWL systems relates to the rest of the toolbox

As told in section 3, there are two kinds of sPWL systems. The commands that are in common for both structures are listed in Table 5, while Table 6 lists the

Table 5 The general commands of the sPWL package

Command	Description
setpart	Initialize partition data structure
addati	Specify affine dynamics
getpart	Retrieve partition data structure
part2pwl	Convert to pwltools data structure

commands that are specific to a certain structure.

Table 6 The structure specific commands of the sPWL package

Hyperplane description		Simplex description	
Command	Description	Command	Description
addhp	Add hyperplane	addvtx	Add vertex
		addray	Add ray
addhcell	Define hyperplane cell	addscell	Define simplex cell

All of the above commands will be described in detail on the following pages. Not to mix up the model construction commands, the sPWL commands have been marked with an (s).

4.3 The input vector "options"

Several of the commands included in PWLTOOL use the LMI Control Toolbox to solve the *feasibility problem* (find a solution to the LMI system $A(x) < 0$) or the *linear objective minimization problem* (Minimize $c^T x$ subject to $A(x) < 0$). The commands of the LMI Control Toolbox use a general structure to give access to certain control parameters, which consists of a five-entry vector. Every command in PWLTOOL that (as a part of its task) solves LMI:s like these also has this input parameter, which is passed to the corresponding LMI Control Toolbox function. The parameter is named `options` and consists of the following elements [1]:

- `options(1)` sets the desired relative accuracy on the optimal value ($c^T x$) when addressing the linear objective minimization problem. It is not used for the feasibility problem.
- `options(2)` sets the maximum number of iterations allowed to be performed by the optimization procedure (100 by default).
- `options(3)` sets the *feasibility radius*. Setting `options(3)` to a value $R > 0$ further constrains the decision vector $x = (x_1, \ldots, x_N)$ to lie within the ball

$$\sum_{i=1}^{N} x_i^2 < R^2,$$

  i.e. the Euclidean norm should not exceed R. The feasibility radius is a simple means of controlling the magnitude of solutions. The default value is $R = 10^9$.
- `options(4)` helps speed up termination. When set to an integer value $J > 0$, the code terminates when a certain minimizer (eg. $c^T x$ for the linear objective minimization problem) does not decrease by more than one percent in relative terms during the last J iterations. This parameter, whose default value is 10, trades off speed vs. accuracy.
- `options(5) = 1` turns off the trace of execution of the optimization procedure. Resetting `options(5)` to zero (default value) turns it back on.

Setting `options(i)` to zero is equivalent to setting the corresponding control parameter to its default value. Consequently, there is no need to redefine the entire vector when changing just one control parameter.

In each command that accepts `options` as an input, this input is optional. If the vector `options` is omitted, PWLTOOL searches for the function `pwloptions` that should return a vector on the format described above. Writing one's own `pwloptions` is useful when doing many different computations requiring the same accuracy. If there exists no `pwloptions`, the default vector of the LMI Control Toolbox will be used.

addati (s)

Purpose

Specify the matrix variables corresponding to the dynamics in a certain region of an sPWL system.

Synopsis

```
dyn = addati(A, a, B, C, c, D)
```

Description

`addati`[2] defines new dynamics in the piecewise linear system currently described. The output `dyn` is an identifier that can be used for subsequent reference to this dynamics as for instance when connecting it to the corresponding region using `addhcell` or `addscell`. The arguments are matrices (and vectors) in the affine system

$$\begin{cases} \dot{x} = Ax + a + Bu \\ y = Cx + c + Du \end{cases}$$

All arguments except A can be omitted. If there is a specified argument that appears to the right of omitted arguments in the list, the omitted arguments must be replaced by empty matrices (`[]`) as place holders.

See Also

`setpart, addhcell, addscell, getpart`

[2] addati is an abbreviation of "add affine time invariant dynamics". This is of course what is done by the command addynamics as well. Another name must, however, be chosen to avoid name conflicts.

addhcell (s)

Purpose

Add a new cell to the hyperrectangle partition currently described.

Synopsis

```
reg = addhcell(hprefs, UsedDynamics)
```

Description

`addhcell` defines a new cell in the hyperrectangle partition by combining those (previously entered) hyperplanes that should bound the cell.

Parameters

- `hprefs` is a vector of indices to the hyperplanes that bounds the cell, each index being an identifier returned by the function `addhp`. Each of the indices could be either positive or negative depending on which side of the hyperplane the cell is situated. Using `hp` (from `addhp`) without a minus sign as an index in the vector means that the cell lies at the same side of the hyperplane as the normal of the plane, i.e. if the plane was defined as

$$\text{hpeq}\begin{bmatrix} x \\ 1 \end{bmatrix} = 0, \qquad \text{then} \qquad \text{hpeq}\begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0$$

  should hold for all points, x, that belong to the cell. Using `-hp` as an index in the vector means that the cell is on the opposite side of the hyperplane normal.

- `UsedDynamics` is a reference to one or several dynamics specifications that shall be used in the region. This corresponds to the identifier `dyn` that is returned from `addati`. If several dynamics specifications shall be used in one region, `UsedDynamics` is a vector of corresponding identifiers.

- `reg` is a label for future reference to the cell.

See Also

`addhp, setpart, addati, getpart`

addhp (s)

Purpose

Add a hyperplane that shall be used as a boundary of one or several cells.

Synopsis

```
hp = addhp(hpeq)
```

Description

`addhp` defines a hyperplane that shall be used as a cell boundary. The input parameter `hpeq` is an $(n+1)$-dimensional vector (in the n-dimensional space) containing the coefficients for the equation of the hyperplane such that

$$\text{hpeq} \begin{bmatrix} x \\ 1 \end{bmatrix} = 0$$

on the surface.

The output `hp` is an identifier that is used for subsequent reference to the plane when connecting several planes to cells, using `addhcell`

See Also

`addhcell, setpart, addati, getpart`

addray (s)

Purpose

Add a ray that shall be used as a boundary for several cells in a simplex partition.

Synopsis

```
ray = addray(rdir)
```

Description

`addhp` defines a ray that shall be used as a cell boundary. The input parameter `rdir` is an n-dimensional vector (in the n-dimensional space) pointing in the direction of infinite extension.

The output `ray` is an identifier that is used for subsequent reference to the ray when connecting several rays and vertices to cells, using `addscell`

See Also

`addvtx`, `addscell`, `setpart`, `addati`, `getpart`

addregion

Purpose

Specify the matrix variables corresponding to a certain region of a PWL system.

Synopsis

```
addregion(G, F, UsedDynamics)
```

Description

`addregion` defines a new region in the piecewise linear system currently described and links it to some dynamics.

Parameters

The matrix `G`, corresponding to $\bar{G}$ in (2), specifies the boundaries of the region. It is an $(m \times n + 1)$-matrix, such that the inequality

$$\bar{G} \begin{bmatrix} x \\ 1 \end{bmatrix} \succeq 0$$

holds for all x within the region (cf. Eq. 2). Each row of these matrices corresponds to a hyperplane on the region boundary.

`F`, corresponding to $\bar{F}$ in (3), is constructed in a way such that

$$\bar{F} \begin{bmatrix} x \\ 1 \end{bmatrix}$$

is continuous between all regions. The $\bar{F}$-matrices are not needed for simulation (cf. Section 2). When only doing simulations, the input `F` can be replaced with an empty matrix (`[]`) as a place holder.

`UsedDynamics` is a reference to one or several dynamics specifications that shall be used in the region. This corresponds to the identifier `dyn` that is returned from `addynamics`. If several dynamics specifications shall be used in one region, `UsedDynamics` is a vector of corresponding identifiers.

See Also

`adddynamics, getpwl, setpwl`

addscell (s)

Purpose

Add a new cell to the simplex partition currently described.

Synopsis

```
reg = addscell(vtxrefs, rayrefs, UsedDynamics)
```

Description

`addscell` defines a new cell in the simplex partition by combining those (previously entered) vertices and rays that should bound the cell.

Parameters

- `vtxrefs` is a vector of indices to the vertices that bound the cell, each index being an identifier returned by the function `addvtx`.
- `rayrefs` is a vector of indices to the rays that bound the cell, each index being an identifier returned by the function `addray`.
- `UsedDynamics` is a reference to one or several dynamics specifications that shall be used in the region. This corresponds to the identifier `dyn` that is returned from `addati`. If several dynamics specifications shall be used in one region, `UsedDynamics` is a vector of corresponding identifiers.
- `reg` is a label for future reference to the cell.

As shown in section 3, the number of entries in `vtxrefs` and `rayrefs` must sum up to $(n + 1)$ in an n-dimensional space. There must be at least one entry in `vtxrefs`.

See Also

`addray`, `addvtx`, `setpart`, `addati`, `getpart`

addvtx (s)

Purpose

Add a vertex that shall be used as a corner of several cells in a simplex partition.

Synopsis

```
vtx = addvtx(vtxcor)
```

Description

`addvtx` defines a vertex that shall be used as a corner of several cells. The input parameter `vtxcor` is the coordinate of the vertex.

The output `vtx` is an identifier that is used for subsequent reference to the ray when connecting several vertices and rays to cells, using `addscell`

See Also

`addray, addscell, setpart, addati, getpart`

addynamics

Purpose

Specify the matrix variables corresponding to the dynamics in a certain region of a PWL system.

Synopsis

```
dyn = addynamics(A, a, B, C, c, D)
```

Description

`addynamics` defines new dynamics in the piecewise linear system currently described. The output `dyn` is an identifier that is used for subsequent reference to this dynamics when specifying the corresponding region using `addregion`. The arguments are matrices (and vectors) in the affine system

$$\begin{cases} \dot{x} = Ax + a + Bu \\ y = Cx + c + Du \end{cases}$$

All arguments except A can be omitted. If there is a specified argument that appears to the right of omitted arguments in the list, the omitted arguments must be replaced by empty matrices (`[]`) as place holders. Those of the arguments that are omitted, will be replaced by zero-matrices of appropriate dimensions.

See Also

`addregion, getpwl, setpwl`

findnb

Purpose

Find the neighbors of the regions of a PWL system.

Synopsis

```
whereto = findnb(pwlsys)
```

Description

`findnb` searches all the $\bar{G}$ matrices of `pwlsys` and generates the matrix `whereto` such that `whereto(i,j)` contains the number of the region that lies behind the boundary defined by the i:th row of G_j.

See Also

`findsm`

findsm

Purpose

Find possible sliding modes of a PWL system.

Synopsis

```
slide = findsm(pwlsys, whereto)
```

Description

`findsm` searches the piecewise linear system `pwlsys` for possible sliding modes. `findsm` returns a square matrix, `slide`, where `slide(i,j)` $= 1$ iff there exist a sliding mode for any x on the boundary between region i and j. The input matrix `whereto`, which contains information about neighboring regions as given by `findnb`, is optional. If it is already computed by `findnb`, those calculations that have already been made can be avoided in this function.

See Also

`findnb`

getpart (s)

Purpose

Get the internal description of an sPWL system

Synopsis

```
part = getpart
```

Description

Having entered the description of a given structured piecewise linear system using the commands for defining the dynamics and the state partition, the internal representation is obtained with the command

```
part = getpart
```

This MATLAB representation of the sPWL system can be converted to the generic PWLTOOL format using `part2pwl`. The system can also be extended by calling `setpart` and iterating the system building commands again.

See Also

`setpart`, `part2pwl`, `getpwl`[3]

[3]The command getpwl is an generic PWL command, described in section 4.1.

getpwl

Purpose

Get the internal description of a PWL system

Synopsis

```
pwlsys = getpwl
```

Description

After completing the description of a given piecewise linear system with `addynamics` and `addregion`, its internal representation pwlsys is obtained with the command

```
pwlsys = getpwl
```

This MATLAB representation of the piecewise linear system can be forwarded to PWLTOOL functions for subsequent processing.

See Also

`setpwl, addynamics, addregion`

iogain

Purpose

Compute an upper bound on the L_2 induced input output gain of a PWL system.

Synopsis

```
[gamma, P, NoLMIs, NoVars] = iogain(pwlsys, inp, outp, options)
```

Description

`iogain` computes an upper bound on the L_2 induced input output gain of the piecewise linear system `pwlsys`, by finding a minimal γ that satisfies the inequality

$$\int_0^\tau |y|^2 dt \leq \gamma^2 \int_0^\tau |u|^2 dt \qquad \forall \tau > 0 \tag{4}$$

For a MIMO system `inp` and `outp` allows the user to specify the input and output signals of interest. The default values are 1. *options* is an optional five-entry vector of control parameters (cf. section 4.3).

`iogain` returns `gamma` $= \gamma$. `P` is a matrix resulting from the LMI calculations (as outlined in [6]). *NoLMIs* is the number of LMI:s needed to solve the problem. *NoVars* is the number of decision variables needed for the LMI:s.

optcstlb

Purpose

Compute a lower bound on the optimal cost.

Synopsis

```
[lb, P, NoLMIs, NoVars] = optcstlb(pwlsys, Q, R, x0, options)
```

Description

The optimal control problem for piecewise linear systems is (while bringing the system to $x(\infty) = 0$ from an arbitrary initial state, $x(0)$) to minimize the cost

$$J(x_0, u) = \int_0^\infty \left(\begin{bmatrix} x \\ 1 \end{bmatrix}^T \bar{Q}_{i(t)} \begin{bmatrix} x \\ 1 \end{bmatrix} + u^T R_{i(t)} u \right) dt \tag{5}$$

Here $i(t)$ is defined so that $x(t) \in X_{i(t)}$ and

$$\bar{Q}_{i(t)} = \begin{bmatrix} Q_{i(t)} & 0 \\ 0 & 0 \end{bmatrix} \text{ if } i(t) \in I_0 \tag{6}$$

`optcstlb` computes a lower bound, `lb`, on the minimum achievable value of $J(x_0, u)$. `optcstlb` also returns `P` which is a matrix resulting from the LMI calculations (as outlined in [6]). *NoLMIs* is the number of LMI:s needed to solve the problem. *NoVars* is the number of decision variables needed for the LMI:s.

Parameters

- `pwlsys` is the piecewise linear system.
- `Q`, `R` are three dimensional matrices defining the cost function such that $\bar{Q}_i =$ `Q(:,:,i)` and $R_i =$ `R(:,:,i)`.
- `x0` is the initial state, $x(0)$.
- `options` is an optional five-entry vector of control parameters (cf. section 4.3).

See Also

`pwlctrl`, `optcstub`

optcstub

Purpose

Compute an upper bound on the optimal cost when applying a PWL control law to a PWL system.

Synopsis

```
[ub, O, NoLMIs, NoVars] = optcstub(pwlsys, Qub, x0, options)
```

Description

`optcstub` computes and returns an upper bound, `ub`, on the optimal cost when applying a piecewise linear control law computed by `optcstlb` and `pwlctrl`. The result is an upper bound on the minimum achievable value of the cost function (cf. Eqs. (5) and (6) on the previous page) applying the control law given by Eq. (7) on page 26. The function `optcstub` also returns `O` that is a matrix resulting from the LMI calculations (as outlined in [6]). *NoLMIs* is the number of LMI:s needed to solve the problem. *NoVars* is the number of decision variables needed for the LMI:s.

Parameters

- `pwlsys` is the piecewise linear system.
- `Qub` is a matrix defining a cost function. This matrix is computed by `pwlctrl`.
- `x0` is the initial state, $x(0)$.
- `options` is an optional five-entry vector of control parameters (cf. Section 4.3).

See Also

`optcstlb`, `pwlctrl`

part2pwl (s)

Purpose

Derive a PWL description from partition data.

Synopsis

```
pwlsys = part2pwl(part)
```

Description

`part2pwl` converts the structured PWL system `part` to an generic PWL representation, `pwlsys`, that can be forwarded to PWLTOOL functions for subsequent processing.

See Also

`setpart, getpart, getpwl`[4]

[4]The command getpwl is an "ordinary" PWL command, described in section 4.1.

pqobserv

Purpose

Compute bounds on the observability of a PWL system.

Synopsis

```
[observ, O, P, NoLMIs, NoVars] = pqobserv(pwlsys, x0, outp, options)
```

Description

`pqobserv` computes a lower and an upper bound on the integral of the output energy,

$$\int_0^\infty |y|^2 dt,$$

when $u = 0$ and the initial state of the piecewise linear system `pwlsys` is given by `x0`. For systems with multiple output signals, the optional parameter `outp` specifies the output signal of interest. The default value is 1. *options* is an optional five-entry vector of control parameters (cf. section 4.3).

`observ = [lower upper]` is a vector consisting of two entries: the lower and the upper bound. `O`, and `P` are matrices resulting from the LMI calculations (as outlined in [6]). *NoLMIs* is the number of LMI:s needed to solve the problem. *NoVars* is the number of decision variables needed for the LMI:s.

pqstab

Purpose

Search for a piecewise quadratic lyapunov function to verify stability of a PWL system, assuming that there are no sliding modes.

Synopsis

```
[P, NoLMIs, NoVars] = pqstab(pwlsys, options)
```

Description

`pqstab` tries to find a piecewise quadratic lyapunov function to verify stability of the piecewise linear system, `pwlsys`. If there exist a piecewise quadratic lyapunov function, it can be written

$$V(x) = \begin{bmatrix} x \\ 1 \end{bmatrix}^T \bar{P}_i \begin{bmatrix} x \\ 1 \end{bmatrix}, \quad x \in X_i, \qquad \text{where } \bar{P}_i = \begin{bmatrix} P_i & 0 \\ 0 & 0 \end{bmatrix} \text{ if } i \in I_0$$

`P` will in that case be a vector of matrices such that `P(:,:,i)` $= \bar{P}_i$. If no lyapunov function exist, `pqstab` will return an empty matrix, `P = []`. *NoLMIs* is the number of LMI:s needed to solve the problem. *NoVars* is the number of decision variables needed for the LMI:s. *options* is an optional five-entry vector of control parameters (cf. section 4.3).

See Also

`qstab, pqstabs`

pqstabs

Purpose

Search for a piecewise quadratic lyapunov function, taking the possibility of sliding modes into account, to verify stability of a PWL system.

Synopsis

```
[P, NoLMIs, NoVars] = pqstabs(pwlsys, options)
```

Description

`pqstabs` tries to find a piecewise quadratic lyapunov function to verify stability of the piecewise linear system, `pwlsys`. If there exist a piecewise quadratic lyapunov function, it can be written

$$V(x) = \begin{bmatrix} x \\ 1 \end{bmatrix}^T \bar{P}_i \begin{bmatrix} x \\ 1 \end{bmatrix}, \quad x \in X_i, \qquad \text{where } \bar{P}_i = \begin{bmatrix} P_i & 0 \\ 0 & 0 \end{bmatrix} \text{ if } i \in I_0$$

`P` will in that case be a vector of matrices such that `P(:,:,i)` $= \bar{P}_i$. If no lyapunov function exist, `pqstabs` will return an empty matrix, `P = []`. *NoLMIs* is the number of LMI:s needed to solve the problem. *NoVars* is the number of decision variables needed for the LMI:s. *options* is an optional five-entry vector of control parameters (cf. section 4.3).

`pqstabs` first uses `findsm` to check whether there exist any sliding modes. If there are no possible sliding modes, `pqstabs` calls `pqstab` and return the result. Otherwise it extends the LMI:s to also include sliding modes.

See Also

`qstab, pqstab, findsm`

pwlctrl

Purpose

Create a PWL controller based on the results from a minimization of a cost function as given by `optcstlb`.

Synopsis

```
[pwlc, L, Qub] = pwlctrl(pwlsys, Q, R, P)
```

Description

Having split the state space into certain regions, `optcstlb` uses that partition to give a lower bound on the optimal cost for *any* control law. `pwlctrl` uses information from `optcstlb` to compute a piecewise linear control law that achieves a low cost.

The control law is

$$u(t) := \bar{L}_i \begin{bmatrix} x \\ 1 \end{bmatrix} \qquad x \in X_i \tag{7}$$

and `pwlctrl` returns a representation of the closed loop system, `pwlc`, and a three dimensional matrix, `L`, such that `L(:,:,i)` $= L_i$. `Qub` consists of data that is needed to compute an upper bound on the optimal cost (using `optcstub`) when implementing this control law.

If several dynamics are linked to one region, the controller will be based on the nominal (first linked) dynamics of each region.

Parameters

- `pwlsys` is the piecewise linear system.
- `Q`, `R` are three dimensional matrices defining the cost function such that $\bar{Q}_i$ = `Q(:,:,i)` and R_i = `R(:,:,i)`.
- `P` is the `P` matrix resulting from `optcstlb`.

See Also

`optcstlb`, `optcstub`

pwleval

Purpose

Evaluate a vector field and an output vector of a piecewise linear function.

Synopsis

```
[xd, y, reg] = pwleval(pwlsys, x, u)
```

Description

`pwleval` finds the region, X_i, that x belongs to and evaluates `xd` and `y` according to

$$x_d = A_i x + a_i + B_i u \qquad \text{for } x \in X_i \tag{8}$$

$$y = C_i x + c_i + D_i u \tag{9}$$

It also returns `reg`, which is the number of the region where `x` is located. `pwlsys` contains the PWL system to be evaluated. `x` and `u` is the state vector and input vector respectively.

See Also

`pwllevel, pwqeval`

pwllevel

Purpose

Plot the level surfaces of a second order piecewise linear function.

Synopsis

```
[Z, x1, x2] = pwllevel(pwlsys, A, K, parea, resol, linespec, V)
```

Description

`pwllevel` plots the level surfaces of a second order global or piecewise quadratic function. It returns a matrix containing the function values

$$Z = K\bar{A}_i \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{10}$$

The vectors `x1` and `x2`, that specify the grid points used for x_1 and x_2 respectively, can be used with the MATLAB function `mesh` to plot the entire PWL function.

Parameters

- For a piecewise linear function `pwlsys` contains a description of those regions that correspond to different linear functions. When plotting a global linear function, `pwlsys` can bet set to [ ].
- `A` represents the function to be plotted. If A is a three dimensional array, then for each region, i, the function `A(:,:,i)*[x; 1]` is evaluated. For a global linear function, `A` is a (two dimensional) matrix such that the function `A*x` (or `A*[x; 1]`) is evaluated. If A is an empty matrix, `[]`, the first dynamics of each region in `pwlsys` will be used.
- If `K` is a scalar, vector component number `K` (of `A*x`) will be plotted. If K is a row vector, `[k1 k2]`, several state variables can be weighted together, such that the resulting plot is `K*A(:,:,i)*[x; 1]`. If `K` is an empty matrix, the first vector component will be plotted.
- `parea = [xmin, xmax, ymin, ymax]` sets scaling for the x- and y-axes on the plot.
- `resol = [resx1 resx2]` is an optional parameter that specifies the resolution of the grid that is used when evaluating the linear function. These numbers specify at how many instances the state variables x_1 and x_2 respectively will be used. If any of the parameters `linespec` or `V` are specified though `resol` is not, `resol` must be replaced by an empty matrix (`[]`) as place holder.
- The level surfaces are normally drawn black and solid. The optional character string `linespec` allows you to specify another color and line type in the same format as the MATLAB `plot` function. To omit the plot (when using this function to get the function values in Z), use the color 'n' (none).
- `V` is an optional parameter that is used to plot `length(V)` contour lines at the values specified in vector `V`

See Also

`pwleval, pwqlevel`

pwlsim

Purpose

Simulate a PWL system.

Synopsis

```
[t, x, te, regidx] = pwlsim(pwlsys, x0, tspan)
```

Description

`pwlsim` simulates the piecewise linear system, `pwlsys`, from the initial state `x0`. The system will be simulated from time t_0 to t_{final} which is specified in `tspan = [t0 tfinal]`.

`pwlsim` returns data as follows. Each row in the solution array `x` corresponds to a time returned in column vector `t`. `regidx` is a vector that contains the regions entered during simulation and `te` contains the corresponding entry times.

Additional information on the simulation can be found in [2].

pwqeval

Purpose

Evaluate a piecewise quadratic function.

Synopsis

```
[y, reg] = pwqeval(pwlsys, Q, x)
```

Description

`pwqeval` finds the region, X_i, that x belongs to and evaluates

$$y = \begin{bmatrix} x \\ 1 \end{bmatrix}^T \bar{Q}_i \begin{bmatrix} x \\ 1 \end{bmatrix} \qquad \text{for } x \in X_i \tag{11}$$

It also returns `reg`, which is the number of the region where `x` is located.

Parameters

- For a piecewise quadratic function `pwlsys` contains the regions that correspond to different functions. To plot a global quadratic function, `pwlsys` can be set to `[]`.
- `Q` represents the function to be evaluated according to Eq. 11, i.e. `Q(:,:,i)` $= \bar{Q}_i$. For a global quadratic function, Q is a matrix such that the function values are given by $x^T Q x$.
- `x` is the state vector

See Also

`pwqlevel, pwleval`

pwqlevel

Purpose

Plot the level surfaces of a second order quadratic function.

Synopsis

```
[Z, x1, x2] = pwqlevel(pwlsys, Q, parea, resol, linespec, V)
```

Description

`pwqlevel` plots the level surfaces of a second order global or piecewise quadratic function. It returns a matrix containing the function values

$$Z = \begin{bmatrix} x \\ 1 \end{bmatrix}^T \bar{Q}_i \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{12}$$

The vectors `x1` and `x2` specify the grid points used for x_1 and x_2 respectively. These vectors can be used with the MATLAB function `mesh` to plot the entire quadratic function.

Parameters

- For a piecewise quadratic function `pwlsys` contains a description of those regions that correspond to different quadratic functions. When plotting a global quadratic function, `pwlsys` can bet set to [ ].
- `Q` represents the function to be plotted. For a global quadratic function, `Q` is a matrix such that the function values are given by $x^T Q x$. For a piecewise quadratic function, `Q` is a vector of matrices such that `Q(:,:,i)` $= \bar{Q}_i$ and the function values of region i are given by Eq. 12.
- `parea = [xmin, xmax, ymin, ymax]` sets scaling for the x- and y-axes on the plot.
- `resol = [resx1 resx2]` is an optional parameter that specifies the resolution of the grid that is used when evaluating the quadratic function. These numbers specify at how many instances the state variables x_1 and x_2 respectively will be used. If any of the parameters `linespec` or `V` are specified though `resol` is not, `resol` must be replaced by an empty matrix (`[]`) as place holder.
- The level surfaces are normally drawn black and solid. The optional character string `linespec` allows you to specify another color and line type in the same format as the MATLAB `plot` function. To omit the plot (when using this function to get the function values in Z), use the color 'n' (none).
- `V` is an optional parameter that is used to plot `length(V)` contour lines at the values specified in vector `V`

See Also

`pwqeval`, `pwllevel`

qstab

Purpose

Search for a global quadratic lyapunov function to verify stability of a PWL system.

Synopsis

```
[P, NoLMIs, NoVars] = qstab(pwlsys, options)
```

Description

`qstab` tries to find a global quadratic lyapunov function to verify stability of the piecewise linear system, `pwlsys`. If there exist a global quadratic lyapunov function, $V(x)$, then `P` is the stability matrix such that $V(x) = x^T Px$. If no Lyapunov function exist, the function will return an empty matrix, `P = []`. *NoLMIs* is the number of LMI:s needed to solve the problem. *NoVars* is the number of decision variables needed for the LMI:s. *options* is an optional five-entry vector of control parameters (cf. section 4.3).

See Also

`pqstab`

setpart (s)

Purpose

Initialize the description of an sPWL system

Synopsis

```
setpart(part)
```

Description

`setpart` is called before starting the description of a structured piecewise linear system. The function could be called in three ways

- `setpart('h')` creates a new hyperrectangle partition.
- `setpart('s')` creates a new simplex partition.

To add on to an existing structured piecewise linear system, use the syntax

```
setpart(part)
```

where `part` is the internal representation of the existing system. Subsequent system building commands will then add new dynamics and partitions to `part`.

See Also

`addhcell, addscell, addati, getpart`

setpwl

Purpose

Initialize the description of a PWL system

Synopsis

```
setpwl(pwlsys0)
```

Description

Before starting the description of a new piecewise linear system with `addynamics` and `addregion`, type

```
setpwl([])
```

to initialize its internal representation.

To add on to an existing piecewise linear system, use the syntax

```
setpwl([pwlsys0])
```

where `pwlsys0` is the internal representation of this piecewise linear system. Subsequent `addynamics` and `addregion` will then add new dynamics and regions to the initial piecewise linear system pwlsys0.

See Also

`getpwl, addynamics, addregion`

5. Examples of Usage

In order to clarify the usage of the PWLTOOL commands, two examples are presented in this section. These examples contain the complete code, i.e. one should be able to reproduce the results (when having access to PWLTOOL) by entering the lines marked with the MATLAB prompt (>>) into MATLAB.

Having presented two complete examples using the general PWL package, we will show a simpler way to enter some of the system matrices of the first example using the sPWL package

5.1 The Flower System

In this example, we will study the piecewise linear system

$$\dot{x}(t) = \begin{cases} A_1 x(t), & x_1^2(t) - x_2^2(t) \geq 0 \\ A_2 x(t), & x_1^2(t) - x_2^2(t) < 0 \end{cases}$$
$$A_1 = \begin{bmatrix} -\varepsilon & \alpha\omega \\ -\omega & -\varepsilon \end{bmatrix} \qquad A_2 = \begin{bmatrix} -\varepsilon & \omega \\ -\alpha\omega & -\varepsilon \end{bmatrix}$$
$$C_1 = C_2 = [\,1 \quad 0\,] \tag{13}$$

where $\alpha = 5$, $\omega = 1$, and $\varepsilon = 0.1$. We will do simulations and analyze the stability and observability of the system.

PWL System Initialization First we must enter the PWL system according to Eqs. (1) - (3).

```
>> A1 = [-0.1, 5; -1, -0.1];   % Enter matrices describing the
>> A2 = [-0.1, 1; -5, -0.1];   % dynamics

>> a = [];

>> B = [];

>> C1 = [1 0];
>> C2 = [1 0];

>> G1 = [1 -1; 1 1];           % Enter the regions
>> G2 = [-1 1; 1 1];
>> G3 = [-1 1; -1 -1];
>> G4 = [1 -1; -1 -1];

>> F1 = [G1; eye(2)];
>> F2 = [G2; eye(2)];
>> F3 = [G3; eye(2)];
>> F4 = [G4; eye(2)];

>> setpwl([]);                 % Set up PWL system

>> dyn1 = addynamics(A1, a, B, C1);
>> dyn2 = addynamics(A2, a, B, C2);
```

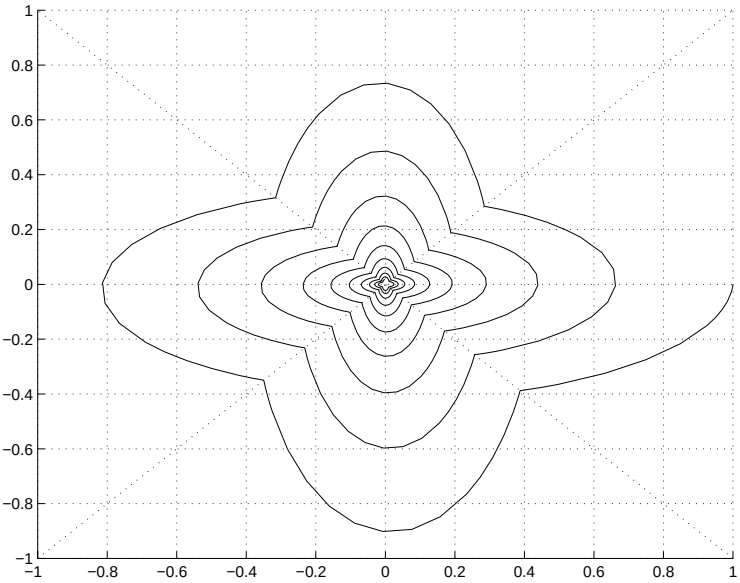

Figure 4 Simulation and cell partition of flower system

```
>> addregion(G1, F1, dyn1);
>> addregion(G2, F2, dyn2);
>> addregion(G3, F3, dyn1);
>> addregion(G4, F4, dyn2);

>> pwlsys = getpwl;            % Extract PWL system
```

Simulation Having entered the system properly, we can make a simulation. In this example we will simulate a trajectory starting in $x(0) = (1 \;\; 0)'$.

```
>> [t, xv] = pwlsim(pwlsys, [1 0]', [0 40]);   % Simulate

>> hold on;                                    % Plot phase plane
>> plot(xv(:,1), xv(:,2));
>> plot([-1 1],[-1 1],'k:');
>> plot(-[-1 1],[-1 1],'k:');
>> grid on
```

The result of this is shown in Fig. 4.

Stability Analysis Judging from the simulation, it seems as if the PWL system is stable. We will now try to prove the stability of this system. Let us first try to find a global quadratic Lyapunov function:

```
>> P = qstab(pwlsys)
```

The P matrix returned from this function is an empty matrix, which indicates the nonexistence of a *global* quadratic Lyapunov function. Our next move is to look for a *piecewise* quadratic Lyapunov function.

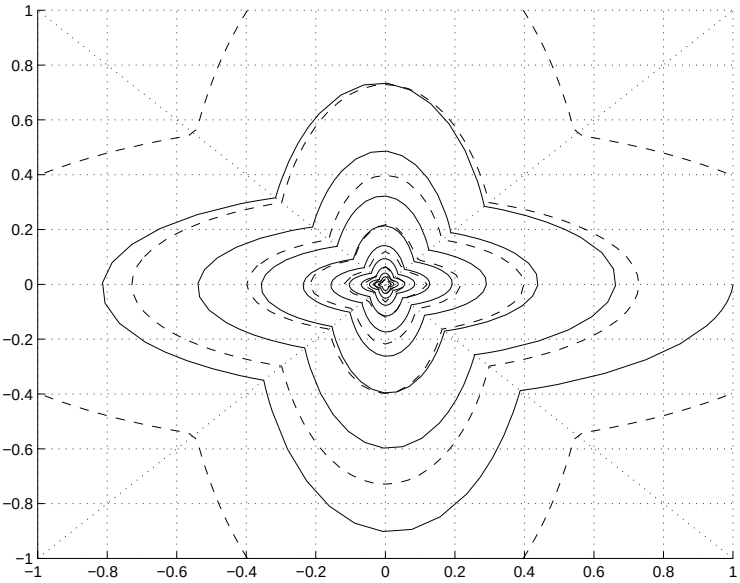

Figure 5 Level surfaces of the Lyapunov function

```
>> P = pqstab(pwlsys)
```

This time we succeed and the P structure returned by the function contains a set of four matrices, where each matrix corresponds to one of the four regions that build our system. We can now plot the level surfaces of the Lyapunov function

```
>> pwqlevel(pwlsys, P, [-1 1 -1 1], [], 'k--');
```

and the result is shown in fig 5

Observability Analysis The "degree of observability" can be measured by the amount of output energy $\int_0^\infty |y|^2 dt$ that is generated for different values of the initial state $x(0)$. This amount can be estimated from a set of LMI:s thanks to the structure of the systems under consideration. PWLTOOL allows us to compute bounds on the integral of the output energy corresponding to a trajectory from a given initial state:

```
>> x0 = [1 0]';
>> observ = pqobserv(pwlsys, x0)
```

The function returns

```
observ =

    0.6025    2.5060
```

which is a lower and an upper bound respectively on the output energy when using an initial state $x(0) = (1, 0)'$. This is a valid but very coarse estimation, which depends on the state space being divided into (too) few regions. Splitting up the state space more will lead to narrower bounds (e.g. 32 regions will confine the estimation to $[1.78, 1.88]$.), cf. [6].

5.2 Sliding mode system

In this example we will show the capability of PWLTOOL to handle sliding modes. The system that is used for this purpose is

$$\dot{x}(t) = \begin{cases} A_1 x(t), & x_1(t) > 0 \\ A_2 x(t), & x_1(t) \leq 0 \end{cases}$$

$$A_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & -2 & -1 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 0 & -2 & 1 \\ 0 & -1 & -1 \\ -1 & -2 & -3 \end{bmatrix} \tag{14}$$

PWL System Initialization We start by entering the system as in the former example.

```
>> A1 = [0 1 0; 0 0 1; -1 -2 -1];
>> A2 = [0 -2 1; 0 -1 -1; -1 -2 -3];

>> G1 = [1 0 0];
>> G2 = [-1 0 0];

>> F1 = [0 0 0; eye(3)];
>> F2 = [G2(1,:); eye(3)];

>> setpwl([]);

>> dyn1 = addynamics(A1);
>> dyn2 = addynamics(A2);

>> addregion(G1, F1, dyn1);
>> addregion(G2, F2, dyn2);

>> pwlsys = getpwl;
```

Simulation Before simulating the system we try to find a piecewise quadratic Lyapunov function. Being aware of possible sliding surfaces of this system we use `pqstabs` this time. One could of course *always* use use this function instead of `pqstab`. When the system is known not to exhibit sliding modes, however, one can save some computational load by using `pqstab`.

```
>> [P, NoLMIs, NoVars] = pqstabs(pwlsys);
```

When the function is called with the PWL system as the only input parameter, `pqstabs` will display its computations. Among other text we will find

```
Possible sliding mode between region(s) no
2 - 1
```

which indicates that the vector fields of the system are such that sliding modes are possible. A nonempty P is returned and we conclude that the system is stable. We simulate the system when starting in $x(0) = (1 \;\; 2 \;\; 3)'$.

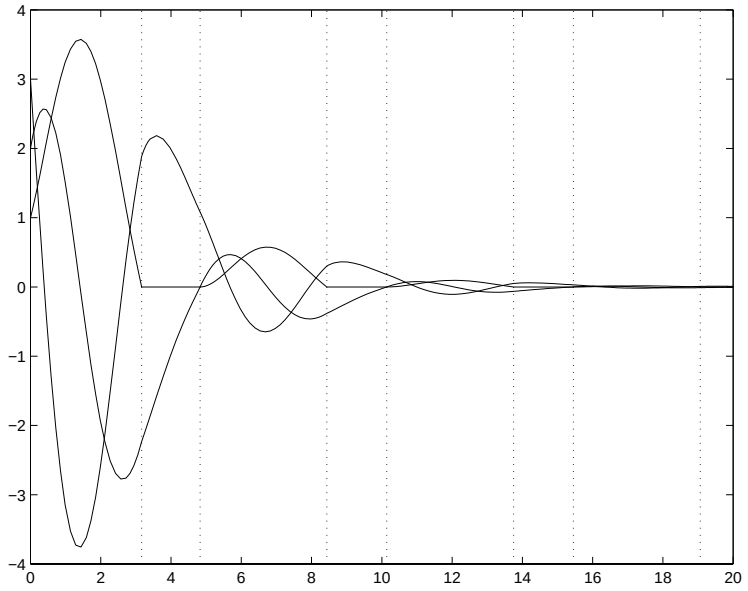

Figure 6 Trajectories from sliding mode simulation

```
>> x0 = [1 2 3]';                                % initial state for
                                                 % simulation
>> [t, x, te] = pwlsim(pwlsys, x0, [0 20]);      % simulate for 20
                                                 % time units
>> plot(t, x);                                   % plot the results
>> hold on;
>> V = axis;                                     % mark region transitions
>> for lp = 1:length(te);
>>   plot([te(lp) te(lp)], [V(3) V(4)], 'k:');
>> end
```

and the result is shown in Fig. 6. The function `pwlsim` also returns the points of time where region transitions have occurred. Looking into Fig. 6, one can easily see when the system has been sliding (when $x_1(t)$ is zero, e.g. around four time units). Let us examine the state space trajectory in a three dimensional plot as well.

```
>> plot3(x(:,1), x(:,2), x(:,3), 'LineWidth', 2);

>> hold on;
>> patch([0 0 0 0 0]-0.05,[-1 -1 1 1 -1]*5, [-1 1 1 -1 -1]*5, [0.95
0.95 0.95]);
>> ee = 0.05;
>> plot3([0 0]+ee, [0 0], [-1 1]*5, 'k--');
>> plot3([0 0]+ee, [-1 1]*5, -min(5, max(-5, 2*[-1 1]*5)), 'k--');
```

The result is shown in Fig. 7. To be able to see where the system is sliding, we have added a wall between the two regions of this system. One can see that the

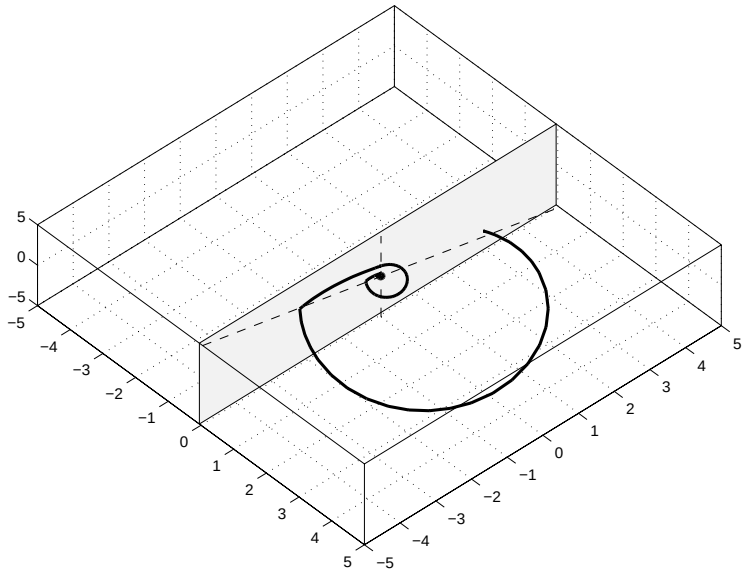

Figure 7 Three dimensional plot from sliding mode simulation

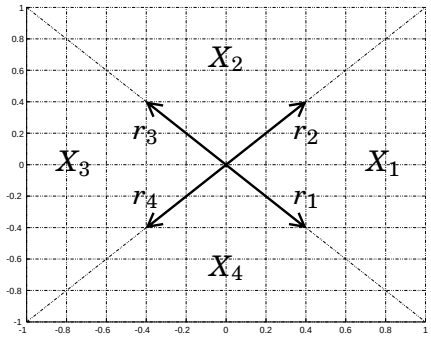

Figure 8 Simplex partitioning of the flower example.

trajectory gets stuck on this surface at a couple of points and slide for a while before escaping.

5.3 The Flower System using the sPWL package

As seen from the examples of this section, the system initialization in the generic PWL package requires the user to enter F-matrices that are used for Lyapunov computations. In these examples, as well as in many others, this effort can be avoided by using the sPWL package. We will show below how to apply the package on the flower system.

The Simplex Interpretation The state space partitioning, with regions $X_1 - X_4$, of the flower system is shown in Figure 8.

Instead of defining those regions by entering the G-matrices, we will now use the simplex notation. Each simplex of Fig. 8 is unbounded and can be represented by one vertex (the origin) and two rays pointing in the directions of the region boundaries (denoted $r_1 - r_4$ in the figure). Thus, the model construction code of the flower example can be replaced by the following code. (It is assumed that the dynamics matrices already have been defined.)

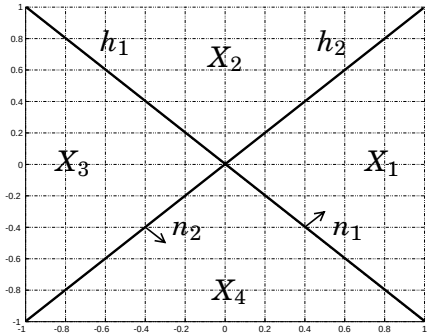

Figure 9 Hyperplane partitioning of the flower example.

```
>> setpart('s');                    % Set up sPWL system
                                    % using simplices

>> v1 = addvtx([0 0]);              % Enter the origin as a vertex

>> r1 = addray([ 1 -1]);            % Enter the rays
>> r2 = addray([ 1  1]);
>> r3 = addray([-1  1]);
>> r4 = addray([-1 -1]);

>> d1 = addati(A1, a1, B1, C1);     % Add the dynamics
>> d2 = addati(A2, a2, B2, C2);

>> addscell([v1],[r1 r2],d1);       % Connect the dynamics to regions
>> addscell([v1],[r2 r3],d2);
>> addscell([v1],[r3 r4],d1);
>> addscell([v1],[r4 r1],d2);

>> part = getpart;                  % Extract sPWL system
>> pwlsys = part2pwl(part);         % Transform to PWL system
```

The Hyperplane Interpretation In addition to the simplex interpretation, the partitions of the flower system can be seen as consisting of hyperplane intersections. Figure 9 shows how the system can be defined. Hyperplane h_1 has the normal direction n_1, i.e. $n_1 x > 0$ on the upper right side of the plane. The normal direction of hyperplane h_2 is n_2.

```
>> setpart('h');                    % Set up sPWL system
                                    % using hyperplanes

>> h1 = addhp([1  1 0]);            % Enter the hyperplanes
>> h2 = addhp([1 -1 0]);

>> d1 = addati(A1, a1, B1, C1);     % Add the dynamics
>> d2 = addati(A2, a2, B2, C2);

>> addhcell([ h1  h2], d1);         % Connect the dynamics to regions
>> addhcell([ h1 -h2], d2);
```

```
>> addhcell([-h1 -h2], d1);
>> addhcell([-h1  h2], d2);

>> part = getpart;                  % Extract sPWL system
>> pwlsys = part2pwl(part);         % Transform to PWL system
```

Note that the flower example is rather special — the simplex description and the hyperplane description are in general not applicable to the same problems.

6. References

[1] P. Gahine, A. Nemirovski, A. Laub, and M. Chilali. *LMI Control toolbox user's guide*. The MathWorks, inc, 1995.

[2] S. Hedlund and M. Johansson. "A toolbox for computational analysis of piecewise linear systems." In *Proceedings of European Control Conference*, Karlsruhe, 1999.

[3] K. H. Johansson and A. Rantzer. "Global analysis of third-order relay feedback systems." Technical Report TFRT–7542, Department of Automatic Control, Lund Institute of Technology, 1996. Submitted for journal publication.

[4] M. Johansson. *Piecewise Linear Control Systems*. PhD thesis TFRT-1052, Dept. of Automatic Control, Lund Institute of Technology, Box 118, S-221 00 Lund, SWEDEN, 1999.

[5] M. Johansson and A. Rantzer. "Piecewise quadratic Lyapunov functions for hybrid systems." In *Proceedings of European Control Conference*, Brussels, 1997.

[6] A. Rantzer and M. Johansson. "Piecewise linear quadratic optimal control." In *Proceedings of American Control Conference*, Albuquerque, 1997. Submitted for journal publication.

[7] A. Rantzer and M. Johansson. "Piecewise linear quadratic optimal control." Technical Report ISRN LUTFD2/TFRT--7569--SE, Department of Automatic Control, November 1997. To appear in IEEE Transactions on Automatic Control.

[8] T. Takagi and M. Sugeno. "Fuzzy identification of systems and its applications to modeling and control." *IEEE Transactions on Systems, Man and Cybernetics*, **15:1**, pp. 116–132, 1985.

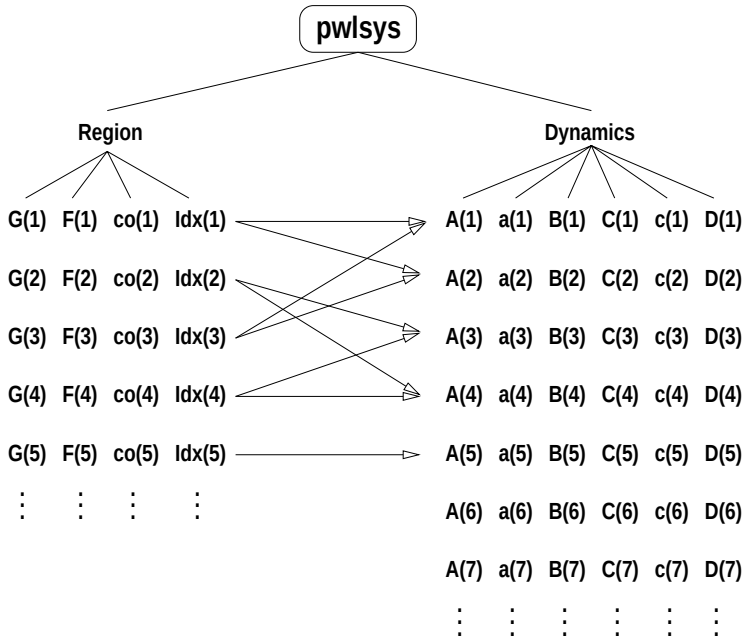

Figure 10 Schematic view of the data structure used for representing a PWL system in MATLAB

A. Data Structure

The PWL systems that this toolbox was designed for can be quite complex, i.e. they can contain many regions with different dynamics. Regions and dynamics can also be interconnected in several ways: one type of dynamics can appear in several regions (the flower system in Section 5 is a simple example of this), but one might also want to specify several dynamics for one region. The latter situation would typically appear when bounding a nonlinearity between two piecewise linear functions.

Since it is not desirable to store the same information in two different places, pointers are used to link dynamics to regions. A schematic view of the data structure used in PWLTOOL is shown in Fig. 10. The total piecewise linear system is represented as a MATLAB-struct that is called `pwlsys` for future reference. The matrices describing the dynamics (A_i, a_i, B_i, C_i, c_i, and D_i) are collected into one struct. The struct `pwlsys` contains an array of such structs that holds all the dynamics of the system. In a similar manner the matrices connected to the state space partition (G_i, and F_i) are collected into one struct. In addition, this struct contains two other elements. It contains a vector, Idx, that points to the dynamics-array, and thus tells which dynamics (possibly several dynamics sets) that is valid in this particular region. I also contains a flag, co, that is set if the region contains the origin. The struct `pwlsys` contains a vector of all these regionstructs of the system.